65C/735/CDV

COMMITTEE DRAFT FOR VOTE (CDV)
PROJET DE COMITÉ POUR VOTE (CDV)

	Project number Numéro de projet	IEC 62734/Ed.1
	IEC/TC or SC: **SC65C** CEI/CE ou SC:	Secretariat / Secrétariat **FRANCE**
☒ Submitted for parallel voting in CENELEC Soumis au vote parallèle au CENELEC	Date of circulation Date de diffusion **2013-07-05**	Closing date for voting (Voting mandatory for P-members) Date de clôture du vote (Vote obligatoire pour les membres (P)) **2013-09-13**

Also of interest to the following committees Intéresse également les comités suivants SC17B, SC22G, TC57, ISO TC184/SC5	Supersedes document Remplace le document 65C/714/CDV & 65C/733/RVC

Proposed horizontal standard
Norme horizontale suggérée

☐ Other TC/SCs are requested to indicate their interest, if any, in this CDV to the TC/SC secretary
Les autres CE/SC sont requis d'indiquer leur intérêt, si nécessaire, dans ce CDV à l'intention du secrétaire du CE/SC

Functions concerned
Fonctions concernées

☐ Safety Sécurité	☐ EMC CEM	☐ Environment Environnement	☐ Quality assurance Assurance qualité

CE DOCUMENT EST TOUJOURS À L'ÉTUDE ET SUSCEPTIBLE DE MODIFICATION. IL NE PEUT SERVIR DE RÉFÉRENCE.

LES RÉCIPIENDAIRES DU PRÉSENT DOCUMENT SONT INVITÉS À PRÉSENTER, AVEC LEURS OBSERVATIONS, LA NOTIFICATION DES DROITS DE PROPRIÉTÉ DONT ILS AURAIENT ÉVENTUELLEMENT CONNAISSANCE ET À FOURNIR UNE DOCUMENTATION EXPLICATIVE.

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

RECIPIENTS OF THIS DOCUMENT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Titre : CEI 62734/Ed.1: Réseaux de communication industriels – Réseau de communication sans fil et profils de communication – ISA100.11a

Title : IEC 62734/Ed.1:Industrial communication networks – Wireless communication network and communication profiles – ISA 100.11a

Note d'introduction

Introductory note

This 2CDV has been drafted according to the comment resolution prepared during the 65C/WG16 meeting end January 2013 and issued as 65C/733/RVC. Please note that in order to take into account summer vacation schedule in some National Committees, and allow them sufficient time for review, the circulation of this 2CDV has been extended until September 13th, 2013. In addition, to facilitate review by National Committees, an auxiliary document (65C/739/INF) is circulated at the same time, showing the changes made between the first CDV (65C/714/CDV) and this second CDV. Any comments on this second CDV will be solved during the next meeting scheduled on September 25th-27th, 2013 in Switzerland.

ATTENTION **VOTE PARALLÈLE** **CEI – CENELEC** L'attention des Comités nationaux de la CEI, membres du CENELEC, est attirée sur le fait que ce projet de comité pour vote (CDV) de Norme internationale est soumis au vote parallèle. Un bulletin de vote séparé pour le vote CENELEC leur sera envoyé par le Secrétariat Central du CENELEC.	**ATTENTION** **IEC – CENELEC** **PARALLEL VOTING** The attention of IEC National Committees, members of CENELEC, is drawn to the fact that this Committee Draft for Vote (CDV) for an International Standard is submitted for parallel voting. A separate form for CENELEC voting will be sent to them by the CENELEC Central Secretariat.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

Industrial communications networks – Wireless communication network and communication profiles – ISA 100.11A

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

International Standard IEC 62734 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This International Standard is based on ISA100.11a, ISBN: 978-1-936007-96-7.

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/XX/FDIS	65C/XX/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

The National Committees are requested to note that for this publication the stability date is 2019.

THIS TEXT IS INCLUDED FOR THE INFORMATION OF THE NATIONAL COMMITTEES AND WILL BE DELETED AT THE PUBLICATION STAGE.

0 Introduction

0.1 General

This standard provides specifications in accordance with the OSI Basic Reference Model, ISO/IEC 7498–1, (e.g., PhL, DL, etc.), and also provides security and management (including network and device configuration) specifications for wireless devices serving Annex C's usage classes 1 through 5, and potentially class 0, for fixed, portable, and moving devices.

This standard is intended to provide reliable and secure wireless operation for non-critical monitoring, alerting, supervisory control, open loop control, and closed loop control applications. This standard defines a protocol suite, including system management, gateway considerations, and security specifications, for low-data-rate wireless connectivity with fixed, portable, and slowly-moving devices, often operating under severe energy and power constraints. The application focus is the performance needs of process automation monitoring and control where end-to-end communication latencies on the order of at least 100 ms can be tolerated.

To meet the needs of industrial wireless users and operators, the technology specified in this document provides robustness in the presence of interference found in harsh industrial environments or caused by wireless systems not covered by this international standard. As described in Clause 4, this standard addresses coexistence with other wireless devices anticipated in the industrial workspace, such as cell phones and devices based on IEC 62591 (based on WirelessHART™[1]), IEC 62601 (based on WIA-PA), IEEE 802.11 (WiFi), IEEE 802.15, IEEE 802.16 (WiMax), and other relevant standards. Furthermore, this standard supports interoperability of devices compliant with this international standard, as described in Clause 5, in those aspects of operation that are covered by this international standard.

This standard does not define or specify plant infrastructure or its security or performance characteristics. However, it is important that the security of the plant infrastructure be assured by the end user.

0.2 Document structure

This document is organized into clauses focused on unique network functions and protocol suite layers. The clauses describe system, system management, security management, physical layer, data-link layer, network layer, transport layer, application layer, and provisioning. Generic considerations that apply to protocol gateways are also included, though specifications of specific protocol gateways are not. Each clause describes a functionality or protocol layer and dictates the behavior required for proper operation. When a clause describes behaviors related to another function or layer, a reference to the appropriate other clause is supplied for further information.

The mandatory and optional communication protocols defined by this document are referred to as native protocols, while those protocols used by other networks such as legacy fieldbus communication protocols are referred to as foreign protocols.

0.3 Potentially relevant patents

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of multiple patents:

a) concerning elliptic curve (asymmetric) cryptography, given in 7.4.6 and 7.2.2.3;

---

[1] Property of the HART Communication Foundation. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holder or any related products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

b) concerning synchronizing clocks and assessing link quality, given in 9.1.9.3 and 9.1.15;

c) concerning unspecified subject areas;

d) concerning wireless provisioning, and selection and routing among multiple gateways.

IEC takes no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the IEC that they are willing to negotiate licences either free of charge (free) or under reasonable and non-discriminatory terms and conditions (RAND) with applicants throughout the world. In this respect, the statements of the following holders of those patent rights are registered with IEC.

Information on these patent rights and their licensing may be obtained from:

a)	Certicom Corporation 4701 Tahoe Blvd, Bldg A L4W 0B5 Mississauga, ON CANADA Attn: Patent licensing Licensing terms: presumably RAND Relevant patents: unknown; not stated by patent holder	b)	NIVIS LLC 1000 Circle 75 Pkwy, Suite 300 Atlanta, GA 30339-6051 USA Attn: Patent licensing Licensing terms: RAND Relevant patents: – US 20100027437 – US 20100098204
c)	General Electric 1 Research Cir Schenectady, NY 12309-1027 USA Attn: Patent licensing Licensing terms: presumably RAND, reciprocity Relevant patents: unknown; not stated by patent holder	d)	Yokogawa Electric Corporation 2-9-32 Nakachou, Musashina-shi Tokyo JAPAN Attn: Patent licensing Licensing terms: RAND, reciprocity Relevant patents: – JP 4129749 – US 8005514 – US 8031727 – US 8305927 – US 2009080394
The above patent holders, patents, and licensing terms are those declared to the IEC as relevant to IEC 62734, as of the date of preparation of this text.			

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

ISO (http://www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult these databases for the most up-to-date information concerning patents.

Industrial communications networks – Wireless communication network and communication profiles – ISA 100.11A

1 Scope

This International Standard specifies a method of reliable and secure wireless operation for non-critical monitoring, alerting, supervisory control, open loop control, and closed loop control applications. This standard defines a protocol suite, including system management, gateway considerations, and security specifications, for low-data-rate wireless connectivity with fixed, portable, and slowly-moving devices, often operating under severe energy and power constraints. The application focus of this standard is the performance needs of process automation monitoring and control, where end-to-end communication delays on the order of 100 ms can be tolerated.

This standard specifies the following:

- physical layer service definition and protocol specification;
- data-link layer service definition and protocol specification;
- network layer service definition and protocol specification;
- transport layer service definition and protocol specification;
- application layer service definition and protocol specification, including support for protocol tunneling and gateways;
- security and security management;
- provisioning and configuration;
- network management; and
- additive communication role profiles (i.e., one or more can be selected concurrently).

Functionality above the application layer of the OSI Basic Reference Model, such as the so-called User Layer and different profiles for functionality at that layer, is not addressed specifically. However, it is discussed briefly in Annex A.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE 1 See the Bibliography for non-normative references.

ISO/IEC 646, Information technology – ISO 7-bit coded character set for information interchange

ISO/IEC 10731, Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services

ISO/IEC 18033-3, Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers

ISO/IEC 19772, Information technology – Security techniques – Authenticated encryption

IETF RFC 2460, Internet Protocol, Version 6 (IPv6) Specification

IETF RFC 2464, Transmission of IPv6 Packets over Ethernet Networks

IETF RFC 2529, Transmission of IPv6 over IPv4 Domains without Explicit Tunnels

IETF RFC 3168, The Addition of Explicit Congestion Notification (ECN) to IP

IETF RFC 4213, Basic Transition Mechanisms for IPv6 Hosts and Routers

IETF RFC 4291:2006, IP Version 6 Addressing Architecture

IETF RFC 4944, Transmission of IPv6 Packets over IEEE 802.15.4 Networks

IETF RFC 6282, Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks

IETF RFC 6298, Computing TCP's Retransmission Timer

IEEE Std 802.15.4™:2011[2], IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements – Part 15-4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)

ANSI X9.63:2001, Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography

SEC 1:2009, *Elliptic Curve Cryptography, version 2*, available at http://www.secg.org

SEC 4, *Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV), version 0.97*, available at http://www.secg.org

ISA Handbook of Measurement Equations and Tables, 2nd Edition, ISBN 978-1-55617-946-4

3 Terms, definitions, abbreviated terms, acronyms, and conventions

For the purposes of this document, the following terms, definitions, abbreviations, acronyms and conventions apply.

3.1 Terms and definitions

3.1.1 (N)-layer and other terms and definitions from the open systems interconnection Basic Reference Model

3.1.1.1
abstract syntax
specification of (N)-PDUs by using notation rules which are independent of the encoding technique used to represent them

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.1.1.2, generalized to any layer]

[2] Property of IEEE, http://www.ieee.org

3.1.1.2
accountability
property that ensures that the actions of an entity may be traced uniquely to the entity

[SOURCE: ISO 7498-2:1989, 3.3.3]

3.1.1.3
acknowledgment
function of the (N)-layer which allows a receiving (N)-entity to inform a sending (N)-entity of the receipt of an (N)-PDU

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.16]

3.1.1.4
application-entity
active element, within an application process, embodying a set of capabilities that is pertinent to OSI and that is defined for the AL, that corresponds to a specific application-entity-type (without any extra capabilities being used)

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.1.1.1]

Note 1 to entry: This is a slight specialization of (N)-entity, because the AL includes non-OSI-relevant application functions. Each application-entity represents one and only one process in the open system interconnection environment.

3.1.1.5
application-management
functions in the AL related to management of OSI application-processes

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 8.1.1]

3.1.1.6
association
cooperative relationship between system entities, usually for the purpose of transferring information between them

[SOURCE: IEC/TS 62443-1-1:2009, 3.2.7]

3.1.1.7
(N)-association
cooperative relationship among (N)-entity-invocations

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.1]

3.1.1.8
authorization
granting of rights, which includes the granting of access based on access rights

[SOURCE: ISO 7498-2:1989, 3.3.10]

3.1.1.9
availability
property of being accessible and useable upon demand by an authorized entity

[SOURCE: ISO 7498-2:1989, 3.3.11]

3.1.1.10
blocking
function performed by an (N)-entity to map multiple (N)-SDUs into one (N)-PDU

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.11]

3.1.1.11
centralized-multi-endpoint-connection
multi-endpoint-connection where data sent by the entity associated with the central-connection-endpoint is received by all other entities, while data sent by the other entities is received by only the central entity

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.2]

3.1.1.12
ciphertext
data produced through the use of encipherment so that the semantic content of the resulting data is not available

[SOURCE: ISO 7498-2:1989, 3.3.1]

Note 1 to entry: See cleartext, plaintext.

Note 2 to entry: Relative to a PDU, ciphertext is information in a PDU that is subject to obscuration by encryption, in its post-encryption pre-decryption obscured form.

3.1.1.13
cleartext
<generic> intelligible data, the semantic content of which is available

[SOURCE: ISO 7498-2:1989, 3.3.15]

3.1.1.14
cleartext
<communications-protocol-specific> information in a PDU that is not subject to obscuration by encryption

Note 1 to entry: Relative to a PDU, cleartext is information in the PDU that is not subject to obscuration by encryption, that when present in the PDU is always present in its unobscured form.

3.1.1.15
compromise
violation of computer security whereby programs or data may have been modified, destroyed, or made available to unauthorized entities

[SOURCE: ISO/IEC 2382-8:1998, 08.05.11]

3.1.1.16
concatenation
function performed by an (N)-entity to map multiple (N)-PDUs into one (N-1)-SDU

Note 1 to entry: Blocking and concatenation, though similar (they both permit grouping of data-units) serve different purposes. For instance, concatenation permits the (N)-layer to group one or several acknowledgment (N)-PDUs with one (or several) (N)-PDUs containing user-data. This would not be possible with the blocking function only. Note also that the two functions are combinable so that the (N)-layer performs blocking and concatenation.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.13]

3.1.1.17
concrete syntax
those aspects of the rules used in the specification of data which embody a specific representation of that data

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.2.1.1]

3.1.1.18
confidentiality
property that information is not made available or disclosed to unauthorized individuals, entities, or processes

[SOURCE: ISO 7498-2:1989, 3.3.16]

Note 1 to entry: In a general information security context, confidentiality preserves authorized restrictions on information access and disclosure, including means for preserving personal privacy and proprietary information.

3.1.1.19
(N)-connection
association requested by an (N+1)-layer entity for the transfer of data between two or more (N+1)-entities

Note 1 to entry: The association is established by the (N)-layer and provides explicit identification of a set of (N)-data-transmissions and agreement concerning the (N)-data-transmission services to be provided for the set.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.2]

3.1.1.20
(N)-connection endpoint
terminator at one end of an (N)-connection within an (N)-service-access-point

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.3]

3.1.1.21
(N)-connection-endpoint-identifier
identifier of an (N)-connection-endpoint which can be used to identify the corresponding (N)-connection at an (N)-SAP

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.4.1.5]

3.1.1.22
(N)-connection-endpoint-suffix
that part of an (N)-connection-endpoint-identifier which is unique within the scope of an (N)-SAP

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.4.1.6]

3.1.1.23
(N)-connection-mode-transmission
(N)-data-transmission in the context of an (N)-connection

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.17]

3.1.1.24
(N)-connectionless-mode-transmission
(N)-data-transmission not in the context of an (N)-connection and not required to maintain any logical relationship between (N)-SDUs

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.18]

3.1.1.25
correspondent-(N)-entities
(N)-entities with an (N-1)-connection between them

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.5]

3.1.1.26
cryptanalysis
analysis of a cryptographic system and/or its inputs and outputs to derive confidential variables and/or sensitive data including cleartext

[SOURCE: ISO 7498-2:2009, 3.3.18]

3.1.1.27
data integrity
property that data has not been altered or destroyed in an unauthorized manner

[SOURCE: ISO 7498-2:1989, 3.3.21]

3.1.1.28
data-origin authentication
corroboration that the source of data received is as claimed

[SOURCE: ISO 7489-2:1989, 3.3.22]

3.1.1.29
(N)-data transmission
(N)-facility that conveys SDUs from one (N+1) layer entity to one or more (N+1) entities

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.9]

3.1.1.30
deblocking
function performed by an (N)-entity to identify multiple (N)-SDUs which are contained in one (N)-PDU

Note 1 to entry: In the absence of error, deblocking is the reverse function of blocking.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.12]

3.1.1.31
decentralized-multi-endpoint-connection
multi-endpoint-connection where data sent by an entity associated with a connection-endpoint is received by all other entities

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.3]

3.1.1.32
decryption
reversal of a corresponding reversible encipherment

[SOURCE: ISO 7498-2:1989, 3.3.23]

3.1.1.33
demultiplexing
function performed by an (N)-entity which identifies (N)-PDUs for more than one (N)-connection within an (N-1)-connection

Note 1 to entry: In the absence of error, demultiplexing is the reverse function of multiplexing.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.5]

3.1.1.34
digital signature
data appended to, or a crypto graphic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery e.g. by the recipient

[SOURCE: ISO 7498-2:1989, 3.3.26]

3.1.1.35
(N)-entity
active element within an (N)-subsystem embodying a set of capabilities defined for the (N)-layer that corresponds to a specific (N)-entity-type (without any extra capabilities being used)

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.11]

3.1.1.36
(N)-entity-invocation
specific utilization of part or all or all of the capabilities of a given (N)-entity (without any extra capabilities being used)

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.12]

3.1.1.37
(N)-entity-type
description of a class of (N)-entities in terms of a set of capabilities defined for the (N)-layer

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.10]

3.1.1.38
(N)-interface-control-information
information transferred locally between an (N+1)-entity and an (N)-entity to coordinate their joint operation

3.1.1.39
(N)-layer
subdivision of the OSI architecture, constituted by subsystems of the same rank (N)

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996,5.2.1.2]

3.1.1.40
(N)-layer-management
functions related to the management of the (N)-layer partly performed in the (N)-layer itself according to the (N)-protocol of the layer (activities such as activation and error control) and partly performed as a subset of systems-management

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 8.1.6]

3.1.1.41
multi-endpoint-connection
connection with more than two connection-endpoints

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.4]

3.1.1.42
multiplexing
function performed by an (N)-entity in which one (N-1)-connection is used to support more than one (N)-connection

Note 1 to entry: The term multiplexing is also used in a more restricted sense to the function performed by the sending (N)-entity while the term demultiplexing is used to the function performed by the receiving (N)-entity.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.4]

3.1.1.43
password
confidential authentication information, usually composed of a string of characters

[SOURCE: ISO 7498-2:1989, 3.3.39]

3.1.1.44
peer-(N)-entities
entities within the same (N)-layer

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.3]

3.1.1.45
peer-entity authentication
corroboration that a peer entity in an association is the one claimed

[SOURCE: ISO 7489-2:1989, 3.3.40]

3.1.1.46
(N)-protocol
set of rules and formats (semantic and syntactic) that determines the communication behavior of (N)-entities in the performance of (N)-functions

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.9]

3.1.1.47
(N)-protocol-addressing-information
those elements of (N)-PCI which contain addressing information

[SOURCE: ISO/IEC 7498-3:1997, 3.4.20]

3.1.1.48
(N)-protocol-control-information
information exchanged between (N)-entities to coordinate their joint operation

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.6.1.1]

3.1.1.49
(N)-protocol-data-unit
unit of data specified in an (N)-protocol and consisting of (N)-protocol-control-information and possibly (N)-user-data

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.6.1.3]

3.1.1.50
(N)-protocol-version-identifier
identifier conveyed between correspondent (N)-entities which allows the selection of the version of an (N)-protocol

Note 1 to enry: The definition of a new (N)-protocol-version-identifier presupposes a minimal common knowledge of the (N)-protocol identified by the preceding (N)-protocol-version-identifier. When such a minimal common knowledge cannot be achieved, the (N)-protocols are considered to be independent and different.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.18]

3.1.1.51
quality of service
<generic> collective effect of service performance which determines the degree of satisfaction of a user of the service

[SOURCE: IEC 61907:2009, 3.1.15]

Note 1 to entry: The quality of service is characterized by the combined aspects of service support performance, service operability performance, serveability performance, service integrity and other factors specific to each service.

Note 2 to entry: ISO defines quality as the ability of a product or service to satisfy users' needs.

3.1.1.52
quality of service
<data link service> negotiated parameters for a link, including

- priority;
- time windows for control messaging;
- acceptability of out-of-order message delivery; and

- acceptability of message delivery in partial increments

3.1.1.53
reassembling
function performed by an (N)-entity to map multiple (N)-PDUs into one (N)-SDU

Note 1 to entry: In the absence of error, reassembling is the reverse function of segmenting.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.10]

3.1.1.54
recombining
function performed by an (N)-entity which identifies (N)-PDUs for a single (N)-connection in (N-1)-SDUs received on more than one (N-1)-connection

Note 1 to entry: In the absence of error, recombining is the reverse function of splitting.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.7]

3.1.1.55
(N)-relay
(N)-function by means of which an (N)-entity forwards data received from one peer (N)-entity to another peer (N)-entity

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.3.1.6]

3.1.1.56
reset
function that sets the corresponding (N)-entities to a predefined state with a possible loss or duplication of data

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.17]

3.1.1.57
security label
marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource

[SOURCE: ISO 7498-2:1989, 3.3.49]

3.1.1.58
segmenting
function performed by an (N)-entity to map one (N)-SDU into multiple (N)-PDUs

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.9]

3.1.1.59
(N)-selector
that part of an (N)-address that is specific to the addressed (N)-subsystem, i.e., which identifies one or more (N)-SAPs within an end open system once that end open system is unambiguously identified

Note 1 to entry: Since the end open system is implicitly known at the Network layer, (N)-selectors are used above the Network layer, along with local information, to address the desired (N+1)-entity within the open system. (N)-selector values are exchanged between open systems as part of the (N)-PAI.

[SOURCE: ISO/IEC 7498-3:1997, 6.2.3]

3.1.1.60
separation
function performed by an (N)-entity to identify multiple (N)-PDUs which are contained in one (N-1)-SDU

Note 1 to entry: In the absence of error, separation is the reverse function of concatenation.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.14]

3.1.1.61
sequencing
function performed by the (N)-layer to preserve the order of (N)-SDUs that were submitted to the (N)-layer

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.15]

3.1.1.62
(N)-service
capability of the (N)-layer and the layers beneath it, which is provided to (N+1) entities at the boundary between the (N)-layer and the (N+1) layer

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.5]

3.1.1.63
(N)-service access point
point at which (N)-services are provided by an (N)- entity to an (N+1)-entity

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.2.1.8]

3.1.1.64
(N)-service-access-point-address
(N)-address that is used to identify a single (N)-SAP

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.4.1.2]

3.1.1.65
(N)-service-data-unit
amount of information whose identity is preserved when transferred between peer (N+1)-entities and which is not interpreted by the supporting (N)-entities

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.6.1.4]

3.1.1.66
splitting
function within the (N)-layer by which more than one (N-1)-connection is used to support one (N)-connection

Note 1 to entry: The term splitting is also used in a more restricted sense to see the function performed by the sending (N)-entity while the term recombining is used to see the function performed by the receiving (N)-entity.

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.8.1.6]

3.1.1.67
system management
functions in the AL related to management of various OSI resources and their status across all layers of the OSI architecture

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 8.1.4]

3.1.1.68
transfer syntax
abstract and concrete syntax used in the transfer of data between open systems

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.2.1.2]

3.1.1.69
user application process
active process within the highest portion of the AL that is the user of OSI services

Note 1 to entry: The aspects of a UAP that need to be taken into account for the purpose of OSI are represented by one or more application-entities, of one or more application-entity-types, defined in ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.1.2.2 and 7.1.2.3.

Note 2 to entry: The collection of UAPs is sometimes referred to as the user layer, even though ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 7.1.2.1 states that the AL has no boundary with a higher layer. In the OSI Basic Reference Model, the AL includes the UAPs.

3.1.1.70
(N)-user-data
data transferred between (N)-entities on behalf of the (N+1)-entities for which the (N)-entities are providing services

[SOURCE: ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, 5.6.1.2]

3.1.2 Other terms and definitions

NOTE Sources of definitions that are otherwise unreferenced by this standard can be found in the Bibliography.

3.1.2.1
access control
means to ensure that access to assets is authorized and restricted based on business and security requirements

[SOURCE: ISO/IEC 27000:2009, 2.1]

3.1.2.2
alarm
condition that maintains a state until the condition clears, reported on change of state

EXAMPLE The occurrence of an alarm or of a return-to-normal condition that is of potential significance to a correspondent UAP.

3.1.2.3
alert
action of reporting an event condition or an alarm condition

3.1.2.4
(end) application, noun
system or problem to which a computer is applied

3.1.2.5
application (program), noun
program that provides functionality to end users

3.1.2.6
application (layer), noun
highest protocol layer in the ISO/IEC Basic Reference Model; types of applications according to criticality

3.1.2.7
application process
element that performs the information processing for a particular application

3.1.2.8
asymmetric-key (cryptographic) algorithm
public key cryptographic algorithm
<information security> algorithm for performing encipherment or the corresponding decipherment in which the keys used for encipherment and decipherment differ

[SOURCE: ISO/IEC 10181-1:1996, 3.3.1]

3.1.2.9
authentication
<information security> verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system

Note 1 to entry: See data-origin authentication (3.1.1.28) and/or peer-entity authentication (3.1.1.45).

3.1.2.10
authentication code
<information security> full or truncated cryptographic checksum based on an appropriate security function (see 3.1.2.93)

Note 1 to entry: This is also known as a message authentication code (MAC). It is called a message integrity code (MIC) when used in contexts where the acronym MAC has an alternate definition, such as in local area network standards.

[SOURCE: ISO/IEC 19790:2006, 3.8, modified by the “full or truncated” prefix]

3.1.2.11
backbone network
backbone subnet
network not specified by this standard, generally using IPv6 or IPv4 network technology, that is used for routing between the wireless network (WISN) of this standard and

a) connected back-end devices that are specified in part by this standard, such as system managers, security managers, and protocol gateways;
b) devices that natively support the wireless TL, AL and management protocols of this standard; and
c) other backbone routers on the same backbone subnet.

3.1.2.12
backbone router
router that forwards between the wireless network of this standard and a higher-speed backbone network

3.1.2.13
backup
procedure, technique, or hardware used to help recover lost or destroyed data or to keep a system operating

[SOURCE: ISO 2382-12:1988, 12.01.17]

3.1.2.14
bandwidth
<analog domain> numerical difference between the upper and lower frequencies of a band of frequencies

Note 1 to entry: Analog bandwith is expressed in Hz.

3.1.2.15
bandwidth
<digital domain> amount of data that can be passed along a communications channel within a given period of time

Note 1 to entry: Digital bandwith is expressed in bit/s.

3.1.2.16
black channel
communication channel of a safety system that provides no safety functionality in addition to its basic communication capability

3.1.2.17
blackList

list of RF channels upon which transmission is prohibited

Note 1 to entry: A blacklist is temporary or permanent, local or network-wide.

3.1.2.18
block cipher

<information security> cryptographic primitive that uses a symmetric key to create a key-dependent pseudorandom permutation of a fixed-size bit string

3.1.2.19
broadcast

transmission intended for all nodes

Note 1 to entry: Broadcast reception often is limited to specific layers, e.g., MAC or network layer.

Note 2 to entry: Many lower layer protocols do not provide an acknowledgment for broadcasts.

3.1.2.20
canonical transfer syntax
full transfer syntax

full encoding of an object for transfer between devices, before any compression

3.1.2.21
cipher

<information security> cryptographic technique used to protect the confidentiality of data, consisting of three component processes: an encryption algorithm, a decryption algorithm, and a method for generating keys

[SOURCE: attributed in other ISO/IEC standards to an unknown edition of ISO/IEC 18033-1, slightly edited for readability]

3.1.2.22
coexistence

ability of multiple systems to perform their tasks in a given environment where they may or may not be using a similar set of rules

3.1.2.23
compressed transfer syntax

encoding of an object for transfer between devices, after any compression

3.1.2.24
construction option

set of features that a device designer may choose to include in, or exclude from, a device

3.1.2.25
contract

agreement between the system manager and a device in the network involving the allocation of network resources by the system manager to support a particular communication need of that device

3.1.2.26
cryptographic algorithm

<information security> algorithm based upon the science of cryptography, including encryption algorithms, cryptographic hash algorithms, digital signature algorithms, and key agreement algorithms

[SOURCE: IEC/TS 62443-1-1:2009, 3.2.34]

Note 1 to entry: Examples of cryptographic algorithms are block and stream ciphers and keyed hashes. An unkeyed hash is not formally a cryptographic algorithm, although it often is a one-way function that has similar resistance to attack, and often is constructed from a cryptographic algorithm with a fixed key. The SHA family of hashes is so constructed.

3.1.2.27
cryptographic key
key
<information security> mathematical value that is used

a) in an algorithm to generate ciphertext from plaintext or vice versa, and
b) to determine the operation of a cryptographic function (e.g., the synchronized generation of keying material), or a digital signature computation or validation

[SOURCE: IEC/TS 62351-2:2008, 2.2.64]

3.1.2.28
cryptographic key component
key component
<information security> parameter(s) used in a security function to perform a cryptographic function

[SOURCE: ISO/IEC 19790:2006, 3.16]

3.1.2.29
cryptographic module
<information security> set of hardware, software, and/or firmware that implements appropriate security functions and is contained within the cryptographic boundary

[SOURCE: ISO/IEC 19790:2012, 3.25]

3.1.2.30
cryptoperiod
<information security> time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect

[SOURCE: ISO/IEC 11568-4:2007, 3.9]

3.1.2.31
data authenticity
<information security> assurance about the source of information

[SOURCE: IEEE 802-15-4:2011, 3.1]

3.1.2.32
data key
data authenticating key
data encrypting key
<information security> cryptographic key used for the encipherment, decipherment or authentication of data

[SOURCE: ISO/IEC 11568-2:2012 , 3.5]

3.1.2.33
deployment option
set of features that a device designer includes in a device, but which the end-user or their agent (e.g., a network security manager) can elect to employ or not employ

3.1.2.34
derived key
<information security> symmetric key that is derived from a prior symmetric key

Note 1 to entry: Such keys are usable to limit the cryptoperiod of any single key while meeting key archive requirements, provided that the independent key from which the derived key was derived (perhaps through many generations of derivation) has previously met those archive requirements.

3.1.2.35
(key) destruction
<information security> zeroisation or physical destruction of keying material so that it cannot be recovered

3.1.2.36
deterministic random bit generator
<information security> process used to generate an unpredictable series of bits that are random in the sense that there is no way to describe the generator’s output that is more efficient than simply listing each entire output string

Note 1 to entry: Deterministic random bit generators have provable properties. The unpredictability of their output depends on the unpredictability of their initial seed and, for hybrid generators, the rate at which new unpredictable (high entropy) input is included relative to the number of output bits generated. See note to entry 1 of 3.1.2.98, non-deterministic random bit generator, for common sources of such unpredictability.

3.1.2.37
device security management object
application software within a device that acts as a local peer of a security manager

3.1.2.38
duocast
variant of unicast, wherein a second receiver is scheduled to overhear the DPDU and provides a second acknowledgment within a single D-transaction

Note 1 to entry: Duocast is shown graphically in Figure 84.

3.1.2.39
encrypted key
<information security> cryptographic key that has been encrypted using an approved security function with a key encryption key

[SOURCE: ISO/IEC 19790:2012, 3.36]

Note 1 to entry: This process is used in order to disguise the value of the underlying plaintext key.

3.1.2.40
encryption
<information security> reversible operation by a cryptographic algorithm converting data into ciphertext so as to hide the information content of the data

[SOURCE: ISO/IEC 9798-1:2010, 3.13]

3.1.2.41
entity
individual (person), organization, device or process

3.1.2.42
ephemeral key
<information security> cryptographic key that is generated for each execution of a key establishment process and that meets other requirements of the key type (e.g., unique to each message exchange or session)

Note 1 to entry: In some cases ephemeral keys are used more than once, within a single session (e.g., broadcast applications) where the originator generates only one ephemeral key pair per message and the private key (of that pair) is combined separately with each recipient’s public key.

3.1.2.43
event
transient (i.e., stateless) condition, used to report when something happened

EXAMPLE The occurrence of an alarm or a return-to-normal condition that is of potential significance to a correspondent UAP

3.1.2.44
field device
physical device designed to meet the rigors of plant operation that communicates via DPDUs and higher-layer protocols conforming to this standard

Note 1 to entry: These include routing devices, sensors, and actuators.

3.1.2.45
field network
configuration of two or more field devices interconnected by the wireless protocol defined by this standard

3.1.2.46
field router
router that is also a field device (i.e., not a backbone router), existing within a field network

3.1.2.47
foreign protocol application communication
optimized conveyance of PDUs or portions of PDUs from a first protocol within a second protocol by selective usage of caching, compression, address translation and proxy techniques

3.1.2.48
fragment
segment
(verb) segmenting

(noun) one of the (N)-protocol-data-units resulting from the operation of segmenting

Note 1 to entry: Fragment and fragmentation are the terms used by the IETF in internet protocol specifications to describe the OSI concepts of segment and segmentation. In this standard the terms fragment and segment are essentially synonymous, with fragment usually used at the network layer and sometimes at other protocol layers, while segment is usually used at the application layer and sometimes at other protocol layers.

3.1.2.49 gateway
role (of a device) that acts as a protocol translator between an AE conforming to this standard and other, different AEs

3.1.2.50
hash-based message authentication code
<information security> message authentication code that uses an appropriate keyed-hash function

[SOURCE: ISO/IEC 9797-2:2011, generalized]

Note 1 to entry: This definition is generalized from the second construction of ISO/IEC 9797-2, which refers to the HMAC construction that is also specified in [US] FIPS 198A.

3.1.2.51
hash function
<information security> function which maps strings of bits to fixed-size strings of bits, satisfying the following two properties: for a given output, it is computationally infeasible to find an input which maps to this output; for a given input, it is computationally infeasible to find a second input which maps to the same output

[SOURCE: ISO/IEC 9796-2:2010, 3.6]

Note 1 to entry: Computational feasibility depends on the specific security requirements and environment.

Note 2 to entry: See the note of 3.1.2.26.

3.1.2.52
hash value
<information security> full or truncated result of applying a hash function to information

3.1.2.53
identity
distinguishing character or personality of an individual or entity

3.1.2.54
independent key
<information security> symmetric key that is derived from a high entropy bit source and not from a prior key

3.1.2.55
infrastructure
technical structures that support data communications within a facility

EXAMPLE Parts of a plant's IT network, perhaps using IEEE 802.3 or IEEE 802.11, or IEC 61158 Type 10 (PROFInet) or IEC 61158 Type 9 (FOUNDATION™ Fieldbus HSE).[3]

3.1.2.56
initialization vector
<information security> block of bits that is required to allow a cryptographic cipher in a streaming mode of operation to produce a unique stream independent from other streams produced under the same encryption key

3.1.2.57
interconnectable
using the same communication protocols, communication interface and data access

[SOURCE: IEC/TR 62390:2005, 6.2.2, adapted]

3.1.2.58
interoperable
able to work together to perform a specific role in one or more distributed application programs

Note 1 to entry: In this case parameters and their application-related functionality fit together both syntactically and semantically. Interoperability is achieved when the devices support complementary sets of parameters and functions belonging to the same profile.

[SOURCE: IEC/TR 62390:2005, 6.2.2, adapted]

3.1.2.59
Interworkable
able to transfer parameters among correspondents

Note 1 to entry: In addition to the communication protocol, communication interface and data access, the parameter data types are the same.

[SOURCE: IEC/TR 62390:2005, 6.2.2, adapted]

---

3 PROFInet and FOUNDATION Fieldbus are the trademarks of various trade organizations. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

3.1.2.60
Kerberos protocol
specific network authentication protocol that allows individuals communicating over an insecure network to prove their identity to one another in a secure manner

3.1.2.61
key agreement
<information security> process of establishing a shared secret key between entities in such a way that neither of them can predetermine the value of that key

[SOURCE: ISO/IEC 11770-1:2010, 2.13]

3.1.2.62
key archive
key management archive
<information security> encryption system with a backup decryption capability that allows authorized persons, under certain prescribed conditions, to decrypt ciphertext with the help of information supplied by one or more trusted parties which hold special data recovery keys

3.1.2.63
key center
<information security> centralized key distribution process, usually a separate computer system, that uses key-encrypting keys (master keys) to encrypt and distribute T-keys needed by a community of users

Note 1 to entry: Key centers generally are certified, traceable to an accredited independent testing agency, as meeting the requirements of ISO/IEC 19790 (similar to FIPS 140-2) for a Level 3 or Level 4 cryptographic module.

3.1.2.64
key confirmation
<information security> assurance for one entity that another identified entity is in possession of the correct key

[SOURCE: ISO/IEC 11770-1:2010, 2.16]

3.1.2.65
key de-registration
<information security> marking of all keying material records and associations to indicate that the key is no longer in use

3.1.2.66
key derivation
<information security> process by which one or more keys are derived from a shared secret and other information

3.1.2.67
key distribution
<information security> transport of a key and other keying material from an entity that either owns the key or generates the key to another entity that is intended to use the key

3.1.2.68
key distribution center
entity that is trusted to generate or acquire keys and to distribute the keys to communicating parties and that shares a unique symmetric key with each of the parties

[SOURCE: ISO/IEC 11770-1:2010, 2.22]

3.1.2.69
key encrypting key
<information security> cryptographic key that is used for the encryption or decryption of other keys

Note 1 to entry: Best practice is to limit use of symmetric (secret) KEKs to key wrapping and not use them for key transport or session (i.e., data) keys.

Note 2 to entry: KEKs may form a hierarchy. In this standard KEKs are often referred to as “master keys”, although the two concepts are not synonymous.

3.1.2.70
key escrow
<information security> process of recording keys and any related essential key recovery information in a key archive

3.1.2.71
key establishment
<information security> process by which cryptographic keys are securely distributed among cryptographic modules using manual transport methods (e.g., key loaders), automated methods (e.g., key transport and/or key agreement protocols), or a combination of automated and manual methods (consisting of key transport plus key agreement)

3.1.2.72
keying material installation
<information security> installation of keying material for operational use

3.1.2.73
key management

<information security> administration and use of generation, registration, certification, deregistration, distribution, installation, storage, archiving, revocation, derivation and destruction of keying material in accordance with a security policy

[SOURCE: ISO/IEC 11770-1:2010, 2.28]

3.1.2.74
key management infrastructure
<information security> framework and services that provide for the generation, production, distribution, control, accounting, and destruction of all cryptographic material, including symmetric keys, as well as public key signing and generation of its own static and ephemeral asymmetric-key pairs

Note 1 to entry: This includes all elements (hardware, software, other equipment, and documentation); facilities; personnel; procedures; standards; and information products that form the system that distributes, manages, and supports the delivery of cryptographic products and services to end users.

Note 2 to entry: Key management services include key ordering, distribution, re-key, update of keying material attributes, certificate revocation, key recovery and the distribution, accounting, tracking, and control of software that performs either keying material security or cryptographic functions.

3.1.2.75
key manager
<information security> key management infrastructure device that provides key management services

3.1.2.76
(asymmetric) key pair
<information security> pair of related keys where the private key defines the private transformation and the public key defines the public transformation

[SOURCE: ISO/IEC 11770-1:2010, 2.2]

Note 1 to entry: A key pair is used with asymmetric-key cryptographic algorithms.

3.1.2.77
key recovery
<information security> mechanisms and processes that allow authorized entities to retrieve keying material from secure key backup or archive storage

3.1.2.78
key registration
<information security> process of officially recording the keying material by a registration authority

3.1.2.79
key revocation
<information security> process whereby a notice is made available to affected entities that keying material should be removed from operational use prior to the end of the established cryptoperiod of that keying material

3.1.2.80
key transport
<information security> process of transferring a key from one entity to another entity, suitably protected

[SOURCE: ISO/IEC 11770-1:2010, 2.33]

Note 1 to entry: When used in conjunction with a public key (asymmetric) algorithm, the keying material is encrypted using the public key of the receiver and subsequently decrypted using the private key of the receiver. When used in conjunction with a symmetric algorithm, the keying material is wrapped with a key encrypting key shared by the two parties.

3.1.2.81
key update
<information security> function performed on a cryptographic key in order to compute a new but related key

3.1.2.82
key usage period
<information security> either the originator usage period or the recipient usage period of a symmetric key

3.1.2.83
key wrapping
<information security> method of encrypting keys (along with associated integrity information) that provides both confidentiality and integrity protection using a symmetric key

3.1.2.84
key wrapping key
<information security> (symmetric-key) key encryption key

3.1.2.85
keying material
<information security> data necessary to establish and maintain cryptographic keying relationships

EXAMPLE Keys, initialization values, periods of validity.

[SOURCE: ISO/IEC 11770-1:2010, 2.27]

3.1.2.86
latency
delay from when data is created at a data source device to when it is available to be consumed at the destination device)

Note 1 to entry: The designated points of measurement are a) physical devices, or b) layer boundaries within multi-layer software (e.g., from sending transport to receiving transport functionality, or from sending application to sending modem).

3.1.2.87
lease
per-session fine-grained communication resource allocation occurring at a GIAP

3.1.2.88
least privilege
<information security> security principle that restricts the access privileges (e.g., program execution privileges, file modification privileges) of authorized personnel and their cyber agents to the minimum necessary to perform their jobs

3.1.2.89
link
momentary or persistent interconnecting path between two or more devices for the purpose of transmitting and receiving messaging

3.1.2.90
master key
<information security> cryptographic key that is used for deriving other keys

Note 1 to entry: Best practice prohibits using master keys as session (i.e., data) keys, which would ease their cryptanalysis. They may be used as KEKs, often at the top of a KEK hierarchy.

3.1.2.91
mesh topology
network topology in which redundant physically-diverse routing paths are available between each pair of network nodes

Note 1 to entry: Wireless mesh topology is usable to extend coverage via multi-hop capability and/or to facilitate communication reliability by providing redundant paths between devices.

3.1.2.92
message authentication
PDU authentication
<information security> process of establishing that a message was formed by a member of an authorized group of communicants and that the message is unchanged since it was formed

3.1.2.93
message authentication code
message integrity code
<information security> cryptographic checksum generated using a symmetric key that is typically appended to data in order to provide data integrity and source authentication similar to a digital signature

[SOURCE: ISO/IEC 26907:2009, 4.16]

3.1.2.94
message authentication code algorithm
<information security> algorithm for computing a function which maps strings of bits and a secret key to fixed-size strings of bits, satisfying the following two properties:

- for any key and any input string, the function can be computed efficiently;
- for any fixed key, and given no prior knowledge of the key, it is computationally infeasible to compute the function value on any new input string, even given knowledge of a set of input strings and corresponding function values, where the value of the ith input string might have been chosen after observing the value of the first i-1 function values (for integers i> 1)

[SOURCE: ISO/IEC 9797-1:2011, 3.10, modified by deletion of notes judged not relevant to this standard]

3.1.2.95
MIC-computation syntax
<information security> concrete representation of an N-SDU associated protocol information, usually added via a prefix pseudo-header, that is used to bind selective N-addresses and N-PCI, and sometimes selective (N-1)-addresses and (N-1)-PCI, to the N-SDU before computing an integrity check code (MIC) over the assemblage

3.1.2.96
multicast
messaging from a source to a set of intended recipients

Note 1 to entry: The set membership is either indeterminate or determinate, where the latter includes the null set.

Note 2 to entry: Broadcast is a special form of multicast, usually to an indeterminate set of intended recipients. See also unicast.

Note 3 to entry: Multicast, other than broadcast, is not supported in this standard.

3.1.2.97
network management object
application software within a device that acts as a local peer of a network manager

3.1.2.98
non-deterministic bit generator
<information security> random bit generator whose security depends upon sampling an entropy source

[SOURCE: ISO/IEC 18031:2011, 3.23]

Note 1 to entry: Sources of such bits include avalanche breakdown of a Zener diode, shot noise, thermal noise, radioactive decay, cosmic rays, etc.

Note 2 to entry: Post-processing of such noise sources is required to whiten their output and to detect failures in the circuit providing the randomness. Only the post-processed bit stream is suitable for seeding a deterministic bit generator.

3.1.2.99
non-repudiation
<information security> ability to prove the occurrence of a claimed event or action and its originating entities, in order to resolve disputes about the occurrence or non-occurrence of the event or action and involvement of entities in the event

[SOURCE: ISO/IEC 27000:2009, 2.27]

Note 1 to entry: In a general information security context, non-repudiation provides assurance that the originator of information is provided with durable proof of delivery or the recipient is provided with durable proof of the originator's identity, so that the party that provided the non-repudiable proof has no credible later denial of having processed the information.

3.1.2.100
nonce
<information security> number used once, or a value that has (at most) a negligible chance of repeating

3.1.2.101
operational phase
operational use
<information security> phase in the lifecycle of keying material whereby keying material is used for standard cryptographic purposes

3.1.2.102
operational storage
<information security> normal storage of operational keying material during its cryptoperiod

3.1.2.103
originator usage period
<information security> period of time during the cryptoperiod of a symmetric key during which cryptographic protection may be applied to data

3.1.2.104
period of protection
<information security> period of time during which the integrity and/or confidentiality of a key needs to be maintained

3.1.2.105
plaintext
<information security> unencrypted information (relative to an encryption or decryption process)

[SOURCE: ISO/IEC 10116:2006, 3.11, modified by parenthetical comment]

Note 1 to entry: Usually, the plaintext input to an encryption operation is not already enciphered, but in some cases the input is itself the output of another cryptographic operation.

3.1.2.106
policy-based management
administrative (managerial) approach used to simplify the management of a given system via the establishment of policies in order to deal with situations that are understood to be likely to occur

3.1.2.107
private key
<information security> (cryptographic) key of an entity's asymmetric-key pair that is kept private

Note 1 to entry: The security of an asymmetric system depends on the privacy of this key.

[SOURCE: ISO/IEC 11770-1:2010, 2.35]

Note 2 to entry: In an asymmetric (public) cryptosystem, the private key is associated with a public key. The private key is known only by the owner of the key pair and is used to:

- compute the corresponding public key;
- compute a digital signature that is verifiable by the corresponding public key;
- decrypt data that was encrypted by the corresponding public key; or
- compute a piece of common shared data, together with other information.

3.1.2.108
pseudo-header
information that is logically prepended to a PDU before computing a MIC for the PDU, but which is not explicitly conveyed by the PDU

3.1.2.109
public key
<information security> key of an entity's asymmetric-key pair which can usually be made public without compromising security

[SOURCE: ISO/IEC 11770-1:2010, 2.36]

3.1.2.110
public key certificate
<information security> public key information of an entity signed by the certification authority

[SOURCE: ISO/IEC 11770-1:2010, 2.37]

Note 1 to entry: Additional information in the certificate is able to specify how the key is used and its cryptoperiod.

3.1.2.111
recipient usage period
<information security> period of time during the cryptoperiod of a symmetric key during which the protected information is processed

Note 1 to entry: This period frequently extends beyond the originator's period of permitted usage.

3.1.2.112
resilience
ability of a functional unit to continue to perform a required function in the presence of faults or errors

[SOURCE: ISO/IEC 2382-14:1997, 14.04.06]

3.1.2.113
retention period
<information security> minimum amount of time that a key or other cryptographic related information should be retained in an archive

3.1.2.114
robustness
degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions

[SOURCE: ISO/IEC/IEEE 24765:2010, 3.2601]

3.1.2.115
router
device that forwards NPDUs within a computer network based on network-layer information

3.1.2.116
short control signaling
short MAC messaging, sent by a nominal receiver of MAC messaging as an immediate response to the originator of that MAC messaging, used to send control information such as ARQ ACK/NAK status, received signal quality and level, etc. as a way of informing the originating device of reception status and of instantaneous conditions on the medium

3.1.2.117
secret key
<information security> key used with symmetric cryptographic techniques by a specified set of entities

[SOURCE: ISO/IEC 11770-3:2008, 3.35]

3.1.2.118
secure communications protocol
<information security> communication protocol that provides the appropriate confidentiality, authentication, content integrity and message timing protection

3.1.2.119
security association
<information security> relationship between two or more entities for which there exist attributes (state information and rules) to govern the provision of security services involving those entities

[SOURCE: ISO/IEC 10745:1995, 3.8]

3.1.2.120
security domain
<information security> set of assets and resources subject to a common security policy

[SOURCE: ISO/IEC 18028-3:2005, 3.19]

Note 1 to entry: Security domains often are organized (e.g., hierarchically) to form larger domains.

3.1.2.121
security manager
<information security> application software that supervises various operational security aspects of a multi-device network, usually through interaction with device security management objects (DSMO) in the supervised device(s)

Note 1 to entry: A network security manager often is a dedicated device that is protected both physically and by construction. (See ISO/IEC 19790 (similar to FIPS 140-2) and the NIST/CSE Cryptographic Module Validation Program, http://csrc.nist.gov/cryptval/cmvp.htm.)

3.1.2.122
security services
<information security> mechanisms used to provide confidentiality, data integrity, authentication and/or non-repudiation of information

3.1.2.123
separation of duties
<information security> security principle that divides critical functions among different staff members in an attempt to ensure that no one individual has enough information or access privilege to perpetrate damaging fraud

3.1.2.124
session
T-association

3.1.2.125
session key
T-key
temporary data key used by TLEs

3.1.2.126
signature generation
<information security> use of a digital signature algorithm and a private key to generate a digital signature on data

3.1.2.127
signature authentication
<information security> use of a digital signature algorithm and a public key to verify a digital signature on data

3.1.2.128
source authentication
<information security> process of corroborating that the source of data is as claimed

3.1.2.129
split knowledge
<information security> process by which a cryptographic key is split into multiple key components, individually disclosing no knowledge of the original key other than possibly its size, which subsequently can be combined in any of a number of predefined groupings of the multiple specific keys to recreate the original key

3.1.2.130
static key
<information security> key that is not an ephemeral key, which is intended for use for a relatively long period of time, typically in successive invocations of a cryptographic key establishment scheme

3.1.2.131
stream cipher
<information security> cryptographic primitive that uses a symmetric key and an initialization vector and that works with continuous streams of input rather than fixed blocks

3.1.2.132
subnet
(N)-subnet
D-subnet
N-subnet
sub-network, a subset of a full network, either at data-link or network layer (layers 2 or 3 of the OSI Basic Reference Model), comprised of multiple end-point and relay nodes that are interconnected via a frequently-homogeneous (N-1)-layer

3.1.2.133
superframe
collection of timeslots with a common repetition period and possibly other common attributes

3.1.2.134
symmetric key
<information security> secret key shared between two or more parties that may be used for both encryption and decryption as well as for message integrity code computation and verification

[SOURCE: ISO/IEC 26907:2009, 4.27]

3.1.2.135
symmetric-key (cryptographic) algorithm
<information security> cryptographic algorithm that that uses the same (usually secret) key for an operation and its inverse (e.g., encryption and decryption)

3.1.2.136
system initialization
<information security> setting up and configuring a system for secure operation

3.1.2.137
system manager
application software that supervises various operational aspects of a multi-device network other than security, usually through interaction with network management objects in the supervised device(s)

Note 1 to entry: A network manager either supervises an entire multi-device network or acts as a subordinate to another network manager, thereby supervising only a subset of the entire network.

Note 2 to entry: A network manager is not always a dedicated device.

3.1.2.138
system/security manager
system manager functionality acting on behalf of security manager functionality

3.1.2.139
threat
<information security> circumstance or event with the potential to adversely impact plant operations (including function, image, or reputation), assets or individuals through an

information system via unauthorized access, destruction, disclosure, modification of data, message delay, and/or denial of service

3.1.2.140
D-transaction
MPDU that is not an immediate acknowledgment MPDU, plus any sequence of zero or more immediate acknowledgment MPDUs that immediately follow and are a consequence of the first MPDU (and optionally each other), all on the same channel and in the same time slot

3.1.2.141
tunneling
encapsulation of a first protocol within a second communication protocol to convey PDUs from the first protocol

3.1.2.142
type-of-service
collective name given to a set of protocol elements and associated quality-of-service attributes that together form a subprotocol (e.g., real-time voice, time-critical data, and non-time-critical data) with distinct functionality

3.1.2.143
unauthorized disclosure
<information security> event involving the exposure of information to entities not authorized access to the information

3.1.2.144
unicast
messaging from a source to a single intended recipient

Note 1 to entry:See also multicast and broadcast.

3.1.2.145
user initialization
process whereby a user prepares the cryptographic application for use (e.g., installing and configuring software and hardware)

3.1.2.146
user registration
process whereby an entity becomes a member of a security domain

3.1.2.147
zeroisation
<information security> method of erasing electronically stored data, cryptographic keys, and critical stored parameters by altering or deleting the contents of storage in a manner that prevents recovery of the data

3.1.3 Symbols for symmetric keys, and for asymmetric keys and certificates

NOTE Symmetric keys defined by this standard are 128-bit keys. Asymmetric keys defined by this standard have a similar hypothesized cryptographic bit strength of 128 bits or greater. See Clause 7 for the exact description of the cryptographic material.

3.1.3.1
K_DL
current data key for all devices in the local D-subnet

3.1.3.2
K_global
well-known key whose value is static and published, which is used to provide uniformity of PDU structure and processing when a shared-secret key is inappropriate or unknown

3.1.3.3
K_open
well-known limited-utility key, different from K_global, whose value is static and published, which can be used as the value of K_join to provision a device via its over-the-air interface

Note 1 to entry: Use of this key in an environment where eavesdropping could occur can compromise the security of the device and its relationships to the rest of the wireless system.

3.1.3.4
K_join
key used to bootstrap a new device securely into the network

3.1.3.5
K_join_wrapped
wrapped version of the join key, used to recover from a failed security manager

3.1.3.6
K_master
master key used as a KEK for key distribution and security management of a single device

3.1.3.7
K_session_AB
current data key for a session between device A and device B, identical with K_session_BA

3.1.3.8
CA_root
public key of a certificate authority which signed a device's public-key certificate

Note 1 to entry: This key is commonly referred to as a root key and is used to assist in verifying the true identity of the device communicating the certificate, as well as some related keying information.

3.1.3.9
Cert-A
public-key certificate of device A, used to evidence the true identity of the device, as well as related keying information, during execution of an authenticated public-key key establishment protocol

3.1.4 Terms used to describe device behavior

3.1.4.1
capability
ability to perform actions, including attributes on qualifications and measures of the ability as capacity

[SOURCE: IEC 62264-1:2003, 3.6]

EXAMPLE The number of connected devices that a router can support.

Note 1 to entry: Profiles specify a minimum capability.

3.1.4.2
configuration
set of parameters that 1) alter behavior and 2) can be set by a system manager

EXAMPLE network-layer hop limit

Note 1 to entry: Configurations where defaults are appropriate state those defaults (e.g., NL hop limit = 64).

3.1.4.3
configure
act of specifying and administering configuration parameters

3.1.4.4
feature
notable characteristic of a device

EXAMPLE Battery-powered.

3.1.4.5
mandatory
required element for any claim of compliance to this standard

EXAMPLE Support for symmetric-key cryptography.

3.1.4.6
optional
element that is not required to claim comformance with this standard but which, if present, is required to behave as specified in this standard

EXAMPLE Support for asymmetric-key cryptography.

3.2 Abbreviated terms and acronyms

6LoWPAN	IPv6 over a low power personal area network (PAN)
6TSCH	time-slotted channel hopping (TSCH) with 6LoWPAN
ACK	positive acknowledgment
AE	application-entity
AES	advanced encryption standard (see ISO/IEC 18033-3)
AID	attribute ID
AL	application layer
ALE	application layer entity
AME	application layer management entity
APDU	application layer protocol data unit
ARMO	alert reporting management object
ARO	alert-receiving object
ARQ	automatic repeat request
ASAP	application layer service access point
ASDU	application layer service data unit
ASL	application sublayer
ASLMO	application sublayer management object
ASMSAP	application sublayer management service access point
ASM	asset management
ASN.1	abstract syntax notation one
ATT	address translation table
BBR	backbone router
BECN	backward explicit congestion notification
BRPT	backbone router peer table
C/S	client/server
CBC	cipher-block-chaining stream cipher mode (see NIST SP 800-38C)
CCA	clear channel assessment
CCM	counter with cipher-block-chaining message authentication mode
CCM*	CCM enhanced

CIP	Common Industrial Protocol™ [4]
CON	concentrator object
CoS	class of service
CoSt	change of state
CSMA	carrier sense multiple access
CSMA/CA	carrier sense multiple access with collision avoidance
dBm	dB (1 mW)
DADDR	data-link address
DAUX	data-link auxiliary header
DBP	device being provisioned
DCS	distributed control system
DD	device description
DDE	data-link layer data (sub-)entity
DDL	device description language
DDSAP	data-link layer data service access point
DE	discard eligible
DIS	dispersion object
DK	(symmetric) data key, data authentication key, data encryption key
DL	data-link layer
DLE	data-link layer entity
DLMO	data-link layer management object
DMAP	device management application process
DME	data-link layer management (sub-)entity
DMIC	data-link layer message integrity code
DMO	device management object
DMSAP	data-link layer management service access point
DMSO	device management service object
DMXHR	data-link layer management extension header
DPDU	data-link layer protocol data unit
DPO	device provisioning object
DPSO	device provisioning service object
DROUT	data-link layer routing (subheader)
DSAP	data-link layer service access point
DSC	data-link layer security component
DSDU	data-link layer service data unit
DSMO	device security management object
DSO	directory service object
DSSS	direct sequence spread spectrum
DWT	data-link layer protocol data unit wait time
EC	European community
ECB	electronic code book

[4] Property of ODVA, http://www.odva.org

ECC	elliptic curve cryptography standard (see ISO/IEC 18033-2)
ECMQV	Menezes-Qu-Vanstone algorithms using elliptic curve cryptography
ECN	explicit congestion notification
ECPVS	Pintsov-Vanstone algorithms using elliptic curve cryptography
ECQV	Qu-Vanstone algorithms using elliptic curve cryptography
ED	energy detection
EDDL	extended device description language
EMA	exponential moving average
ETSI	European Telecommunications Standards Institute
EUI-64™[5]	64-bit extended unique identifier as specified by the IEEE
FDT/DTM	field device tool / device type manager
FCC	U.S. Federal Communications Commission
FCS	frame check sequence
FEC	forward error correction
FECN	forward explicit congestion notification
FF-H1	Foundation Fieldbus – H1 protocol
FF-HSE	Foundation Fieldbus – high speed ethernet
FHSS	frequency hopping spread spectrum
FHSSM	frequency hopping spread spectrum modulation
FIFO	first-in first-out (queuing discipline)
FIPS	[US] federal information processing standard (issued by NIST)
FPAC	foreign protocol application communication
FPT	foreign protocol translator
FPT-PAI	foreign protocol translator – protocol address information
FPT-PCI	foreign protocol translator – protocol control information
FPT-PDU	foreign protocol translator – protocol data unit
FSK	frequency shift keying
GIAP	gateway interface access point
GPS	global positioning system
GUC	number of supportable gateway-UAP connections
GUI	graphical user interface
HART	highway addressable remote transducer
HC	header compression
HCF	HART Communication Foundation
HMAC	(keyed)-hash message authentication code
HMI	human-machine interface
HRCO	health reports concentrator object
(N)-ICI	(N-layer) interface control information
ICMP	internet control messaging protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical & Electronics Engineers

[5] Property of the trademark owner

IFO	interface object
INF	infinity
I/O	input/output
IPv4	internet protocol version 4
Ipv6	internet protocol version 6
ISA	International Society of Automation
ISM	industrial, scientific, medical
IV	initialization vector
JT_n	join timer n
KDC	key distribution center
KEK	key encryption key
LAN	local area network
LBT	listen before talk
LH	last hop
LLC	logical link control sublayer (in upper DL)
LP	low power
LQI	link quality indicator
LSB	least significant bit
MAC	media access control sublayer (spanning lower DL and upper PhL)
MAN	manual
MIB	management information base
MIC	message integrity code
MICI	media access control interface control information
MK	(symmetric) master key
MP	management process
MPCI	media access control sublayer protocol control information
MPDU	media access control sublayer protocol data unit
MSB	most significant bit
NAK	negative acknowledgment
NaN	not-a-number
NDE	network layer data (sub-)entity
NDSAP	network layer data service access point
NFC	near-field communications
NICI	network interface control information
NIDS	network intrusion detection system
NIST	[US] National Institute of Standards and Technology
NL	network layer
NLE	network layer entity
NME	network layer management (sub-)entity
NMSAP	network layer management service access point
NO	native object
NPCI	network layer protocol control information
NPDU	network layer protocol data unit

NPDU.F128	final destination IPv6Address in the expanded network header
NPDU.F16	final destination DL16Address in the expanded network header
NPDU.O128	originator IPv6Address in the expanded network header
NPDU.O16	originator DL16Address in the expanded network header
NSAP	network layer service access point
NSD	number of system devices
NSDU	network layer service data unit
OBJ	generic application layer object
ODVA	Open Device Vendor Association (now legally known only by its acronym)
OOB	out-of-band
OPC	open connectivity in industrial automation
OSI	Open Systems Interconnection
OTA	over-the-air
P/S	publish/subscribe
PA	process automation
(N)-PAI	(N-layer) protocol addressing information
PAN	personal area network
PCH	PHY coding header
(N)-PCI	(N-layer) protocol control information
PD	provisioning device
(N)-PDU	(N-layer) protocol data unit
PhD	physical layer data service
PhICI	physical layer interface control information
PhL	physical layer
PhLE	physical layer entity
PhPDU	physical layer protocol data unit
PhSAP	physical layer service access point
PhSDU	physical layer service data unit
PHY	physical layer (as used in IEEE 802 standards)
PIB	policy information base
PICS	protocol implementation conformance statement
PKI	public key infrastructure
PNO	PROFIBUS Nutzerorganisation (PROFIBUS User Organization)
PSH	PHY synchronization header
PSMO	proxy security management object
PWT	PDU wait time
QoS	quality of service
R&TTE	radio and telecommunications terminal equipment
RDP	reliable datagram protocol
RF	radio frequency
RFC	request for comments
RFP	request for proposal
RSSI	received signal strength indicator

RSQI	received signal quality indicator
RT	routing table
RTO	retry time-out interval
RTT	round-trip time
RTU	remote terminal unit
RTTV	round-trip time variation
S/S	source/sink
(N)-SAP	(N-layer) service access point
SCADA	supervisory control and data acquisition
SCS	short control signaling
SCO	system communication configuration object
(N)-SDU	(N-layer) service data unit
SIFS	short inter-frame separation
(U)SIM	(universal) subscriber identity module
SINR	signal to interference plus noise ratio
SK	(symmetric) T-key used for authentication and confidentiality
SKG	secret key generation
SL	session layer
SMIB	structured management information base
SMAP	system manager application process
SM	system manager
SMAP	system management application process
SMO	system monitoring object
SOE	sequence of events
SRTT	smoothed round-trip time
STSO	system time service object
TAI	international atomic time; temps atomique international
TCP	transmission control protocol
TDMA	time division multiple access
TDE	transport layer data (sub-)entity
TDSAP	transport layer data service access point
TFRC	TCP-friendly rate control, IETF RFC 5348
TICI	transport interface control information
TL	transport layer
TLE	transport layer entity
TME	transport layer management (sub-)entity
TMIB	transport layer management information base
TMIC	transport layer message integrity code
TMSAP	transport layer management service access point
ToS	type of service
TPCI	transport layer protocol control information
TPDU	transport layer protocol data unit
TSAP	transport layer service access point

TSC	transport layer security component
TSCH	time-slotted channel hopping
TSDU	transport layer service data unit
TUN	tunnel object
TUN-Data	tunnel data
UAL	upper application layer
UAP	user application process
UAPMO	user application process management object
UDO	upload/download object
UDP	user datagram protocol
UFO	unified field object
UTC	universal coordinated time; temps universel coordonné
WBM	wideband modulation
WISN	wireless industrial sensor (and actuator) network

3.3 Conventions

3.3.1 Service interfaces

Portions of this standard use the descriptive conventions given in ISO/IEC 10731.

Service primitives, used to represent service user/service provider interactions (see ISO/IEC 10731), convey parameters that indicate information available in the user/provider interaction.

This standard uses a tabular format to describe the component parameters of the NS (N layer SAP or entity SAP) primitives. The parameters that apply to each group of NS primitives are set out in tables. Each table consists of up to six columns, containing the name of the service parameter, and a column each for those primitives and parameter-transfer directions used by the NS:

- the request primitive’s input parameters;
- the indication primitive’s output parameters;
- the response primitive’s input parameters; and
- the confirm primitive’s output parameters.

NOTE 1 The request, indication, response, and confirm primitives are also known as requestor.submit, acceptor.deliver, acceptor.submit, and requestor.deliver primitives, respectively (see ISO/IEC 10731).

One parameter (or part of it) is listed in each row of each table. Under the appropriate service primitive columns, a code is used to specify the type of usage of the parameter on the primitive and parameter direction specified in the column:

M: parameter is mandatory for the primitive;

S: selection from defined set of two or more parameters;

U: parameter is a user option and may or may not be provided depending on the dynamic usage of the NS-user. When not provided, a default value for the parameter is assumed;

C: parameter is conditional upon other parameters or upon the environment of the NS-user;

(blank) or em-dash (“—“): parameter is never present.

Some entries are further qualified by items in brackets. These may be:

- a parameter-specific constraint:

  (=) indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table;

- an indication that some note applies to the entry:

  (n) indicates that the following note n contains additional information pertaining to the parameter and its use. Letter-enumerated notes are normative; digit-numbered notes are informative.

A summary table is provided to define the parameter usage within the primitive. Each cell defines whether each parameter is mandatory, optional, prohibited, or conditional.

The ordering of the parameters is also implicitly defined from top to bottom.

Complex parameters may also be shown. For example, a structure may be mandatory, but some of the elements of the structure may be optional. The structure elements are indented proportional to their level of hierarchy.

Input parameters for services are specified for request and response service primitives. Output parameters for services are specified for indication and confirmation service primitives.

The following abbreviations are used in the service tables:

- Request service request
- Indication service indication
- Response service response
- Confirmation service confirmation

NOTE 2 Intra-device handling of inter-layer error situations, such as situations where a lower layer queue overflows or a lower layer timeout occurs, is a local matter and hence is not addressed by this standard.

3.3.2 Table cells

For all tables, table entries that are irrelevant or unspecified may contain an em-dash (“—”). Table entries to be filled by suppliers may contain an ellipsis (“…”).

3.3.3 Italics

In some cases a generic term that is used for a specific purpose is italicized at first occurrence, to indicate to the reader that care in interpretation is suggested. The reasons for other uses of italics, such as in 4.5.5.2.1, should be apparent from their immediate context.

3.3.4 Bold face

In some cases a descriptive paragraph is preceded by a bold-faced term or summarizing descriptive phrase, where the bold-facing is used to assist the reader’s future memory.

NOTE Such use of bold-face is inessential, so loss of such distinction due to photocopying is not important.

3.3.5 Informal declarations of named constants

ASN.1 permits numeric constants to be assigned symbolic names. It also permits named constants of enumerations to be assigned specific numeric representations. In this standard the two are unified, when used as inline declarations of permissible choices for fields of data structures, through use of the syntax

numericValue “: ” explanatory text

which is intended as the equivalent to the ASN.1 declaration explanatory_text "(" numericValue ")"

where the explanatory text is converted into an alphanumeric identifer by replacing the spaces between words (if any) with underscores, and by adjusting any delimiting period, comma, or semicolon after the explanatory text to the required ASN.1 list element separator after the equivalent closing parenthesis of the numericValue.

4 Overview

4.1 General

This standard uses the OSI layer description methodology (see Annex C) to define protocol suite specifications, in addition to specifications for the functions of security, management, gateway, and provisioning for an industrial wireless network. The protocol layers supported are the physical layer (PhL), data-link layer (DL), network layer (NL), transport layer (TL), and the application layer (AL).

NOTE 1 Although this standard uses the concept of protocol layers, compliance to this standard does not mandate that implementations partition function similarly to that implied by these layers. Inter-layer interfaces are generally not exposed in a product, and hence are not suitable for conformance testing.

The wireless network defined by this standard consists of wireless devices serving usage classes 1 through 5 (described in Annex C) for non-critical applications of fixed, portable, and moving devices.

References to a network compliant to this standard will hereafter be referred to as a wireless industrial sensor network (WISN), even when that network contains actuators and other devices that are not logically classifiable as sensors.

Most devices that participate in a WISN are expected to implement just a single wireless PhLE and associated DLE. However, devices with multiple wireless PhLEs and associated DLEs are not precluded. Therefore this standard distinguishes between requirements for a device and requirements for a PhLE and associated DLE, even though the two are usually thought of as synonymous.

NOTE 2 It is suggested that the reader internalize this relationship, so that encountering the term DLE brings to mind the term device, even though in rare cases they are not one-to-one.

4.2 Interoperability and related issues

IEC/TR 62390 provides useful definitions for differing levels of interoperation. The following three definitions, each of which includes the former, are quoted exactly from that technical report.

NOTE 1 IEC/TR 62390, Figure 9 provides additional clarity on the relationship of these terms.

NOTE 2 The notes in subclause 4.2 were not in IEC/TR 62390.

a) **Interconnectability**: Two or more devices are interconnectable if they are using the same communication protocols, communication interface and data access.

b) **Interworkability**: Two or more devices are interworkable if they can transfer parameters between them, i.e, in addition to the communication protocol, communication interface and data access, the parameter data types are the same.

   NOTE 3 Interworkability implies interconnectability.

c) **Interoperability**: Two or more devices are interoperable if they can work together to perform a specific role in one or more distributed application programs. The parameters and their application-related functionality fit together both syntactically and semantically. Interoperability is achieved when the devices support complementary sets of parameters and functions belonging to the same profile.

NOTE 4 Interoperability implies interworkability, and thus also interconnectability.

4.3 Quality of service

To support multiple applications within a network along with diverse needs this standard supports multiple levels of quality of service (QoS). QoS describes parameters such as latency, throughput, and reliability. A device's application(s) requests the level of QoS needed. If the necessary resources can be made available for the requesting application, the system manager will allocate those resources in response to the requested QoS. See Clause 6 for additional information.

4.4 Worldwide applicability

This standard is intended to conform to established regulations in most world regions; however, its acceptability in any specific regulatory environment is not guaranteed and thus shall be evaluated. Annex U addresses this topic, including use in the EC under EN 300 328.

4.5 Network architecture

4.5.1 Interfaces

4.5.1.1 Defined interfaces

This standard defines service access points (SAPs) at the upper boundary of each protocol layer to decouple specifications of, and revisions to, each of those protocol layers from oher layers, to the extent feasible. For example, if a new PhL is defined that does not require corresponding changes in the employing DL, then it can be added to the specification with minimal (if any) impact to the other protocol layers defined by this standard.

In most cases these defined interfaces are internal to an implementation. As such they are not subject to standardization and conformance testing, since such testing can be applied only to external interfaces of a unit under test (i.e., black box testing). As a consequence, such internal interfaces are descriptive and informative, not normative. However, it is expected that implementations which partition software along the lines suggested by these interfaces will be easier to maintain and adapt to future revisions of this standard.

Conformance to this standard applies only to the observable behavior of an implementation, including the structure and encoding of any information exchanged at observable interfaces that are specified as such by this standard.

4.5.1.2 Interfaces that are not defined

The following interfaces are not addressed by this standard:

- System manager to security manager: The security manager and the system manager form two core roles in the network that are closely related. Since the security manager and the system manager are so dependent upon each other, and because the security manager communicates directly only with the security manager, it is expected that the one or more devices that provide these two roles will be procured from a single vendor, often realized as a single device supporting both roles. Given these expectations, it is deemed not necessary to standardize this interface.

NOTE This interface is a subject of potential future standardization.

- External interfaces: An important attribute of this standard is that it is designed to allow the wireless network to leverage or integrate into a plant's communication infrastructure. This standard defines specific roles to allow this network to interface to other networks, including both wired and wireless networks and both standard and proprietary networks. However, since those specific external networks cannot be identified by this standard, neither can the interfaces to such networks be identified or specified.

4.5.2 Data structures

4.5.2.1 Defined PDUs

This standard defines the structure of protocol data units (PDUs) used for inter-device communication at the following protocol layers: PhL, DL, NL, TL and AL. Most of these are based on other international standards: ISO/IEC/IEEE standards for the PhL and DL, and IETF standards for the NL and TL. All such PDU definitions are normative, subject to external examination and conformance testing.

Conceptually, each distinct class of PDU has

- an abstract transfer syntax, which describes the structure of the PDU, including order of fields and semantic meaning of each field and its alternative contents;
- one or more concrete transfer syntaxes, which describe the encoding within the PDU for each of those fields and content alternatives.

This standard specifies a *full* or *canonical* concrete transfer syntax for each PDU, and for DPDUs, NPDUs and TPDUs also specifies a *compressed* concrete transfer syntax that reduces the energy requirements for PDU transmission and reception, as well as the occupancy time of the wireless channel when the PDU is being transmitted.

NOTE 1 Decreased channel occupancy reduces interference with other wireless devices and systems, as well as increasing the probabilty that the PDU is successfully received. It also reduces the average power required for device operation, which is of particular importance for devices not connected to an external power supply.

For DPDUs and TPDUs, a third concrete syntax (i.e., encoding) is employed when computing a message integrity code for the PDU. That encoding typically prepends a *pseudo-header* to the PDU as transmitted/received, where the pseudo-header contains specific lower-layer PDU addressing information, thus serving to bind that lower-layer information to the PDU whose integrity is covered by the computed MIC. In this standard this third concrete syntax is called the *MIC-computation syntax*.

NOTE 2 The PDU descriptions in this standard often conflate the abstract syntax of the PDU (i.e., its logical contents) with the concrete transfer syntax (i.e., its encoding). It is anticipated that a future edition of this standard will correct this deficiency.

4.5.2.2 Defined management data structures

This standard defines the structure of management data objects at various protocol layers. Such definitions are normative with respect to the behavior of defined operations on those data structures, and with respect to the presentation of those data structures to the extent, and in the form, that they occur when conveyed by PDUs. However, the representation of those data structures internal to an implementation is beyond the scope of standardization, since such representation is unobservable and thus not subject to conformance testing.

4.5.3 Network description

Figure 1 depicts the communication areas addressed by this standard, as well as those areas (shaded in blue) that are not within the scope of this standard. In Figure 1, circular objects represent roles for field devices (sensors, valves, actuators, etc.) and rectangular objects represent roles for infrastructure devices that communicate to other network devices via an interface to the network infrastructure backbone network.

NOTE This standard defines roles that devices embody; for further information on these roles, see 5.2.6.

A backbone is a data network (preferably high data rate) that is not defined by this standard. This backbone could be an industrial Ethernet, IEEE 802.11, or any other network within the facility interfacing to the plant’s network. See Annex E for further information and assumptions about the characteristics of a backbone.

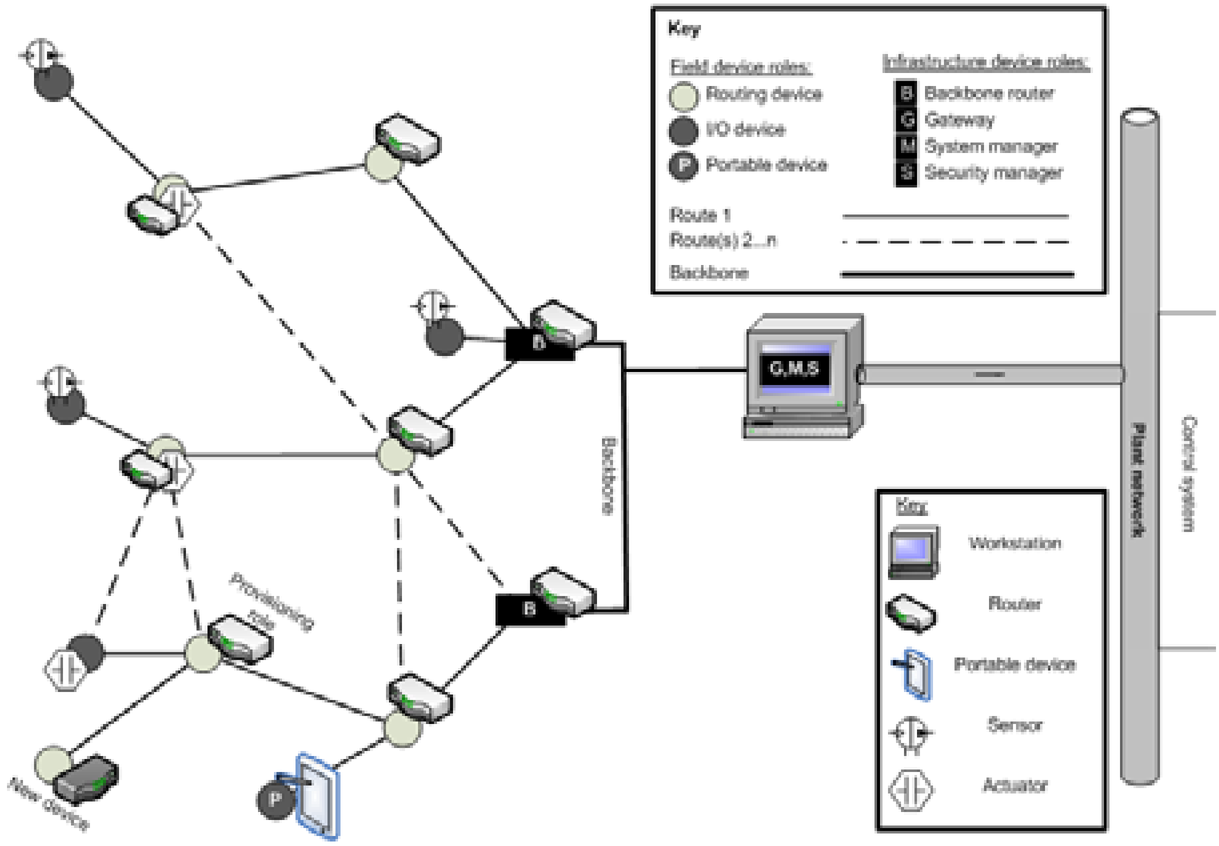


Figure 1 – Standard-compliant network

A complete network as defined in this standard includes all components and protocols required to route secure traffic, manage network resources, and integrate with host systems. A complete network consists of one or more field D-subnets that may be connected by an infrastructure device to a plant subnet.

A field D-subnet consists of a collection of field devices that wirelessly communicate using a protocol stack defined by this standard. As shown in Figure 1, some field devices may have routing capabilities, enabling them to forward messages from other devices.

A transit subnet consists of infrastructure devices on a backbone, such as backbone routers, gateways, system managers, and security managers. Since the backbone physical communication medium and its network protocol stack are outside the scope of this standard, they are not specified and may include tunneling compliant PDUs over external TL or AL protocols.

Devices that connect two disparate D-subnets have at least two DLE and PhLE interfaces. A backbone router connects a field D-subnet with a backbone D-subnet. A gateway connects a backbone subnet with a plant subnet; it may be collocated with a backbone router.

NOTE 1 The scope of a subnet depends on the uppermost communications layer used in construction of the subnet, which could be OSI layer 1, 2, 3 or 4.

NOTE 2 Since gateways are not defined in this standard, the nature of the interface that they provide to a “plant” subnet is strictly notional. Thus the term “plant subnet” in Figure 1 refers to whatever network or other communications means exists on the “far” side of the gateway, relative to the D-subnets that are covered by this standard, while the term “field subnet” refers to the D-subnet composed directly of field devices.

All addressing, routing, and transport are limited to the scope of the field D-subnet. Each DLE within such a D-subnet is identified by a local DL16Address as well as an IPv6Address with global scope.

4.5.4 Generic protocol data unit construction

This communication standard uses communication protocol layers modeled in accord with the OSI Basic Reference Model. A protocol layer typically encapsulates the data it is conveying for a higher protocol layer, and in turn uses a lower layer to convey the encapulated result.

The information conveyed across the network between peer entities is called a protocol data unit (PDU). The information conveyed at a layer boundary between layer entities of a single network node is called a service data unit (SDU). An SDU usually consists of a single PDU, but it can also be a group of concatenated PDUs (see concatenation, 3.1.1.16).

An SDU is considered to be an opaque octetstring or bitstring that is to be conveyed transparently (i.e., without interpretability or alteration) by the lower layer. The SDU can be conveyed by a single lower-layer protocol data unit (PDU), or be segmented (see segmenting, 3.1.1.58) to be conveyed piecemeal by many lower-layer PDUs, or be grouped with other SDUs (see blocking, 3.1.1.10) for conveyance as a group within a single lower-layer PDU. In the most common case, a header and footer are added to a single SDU to form a single protocol data unit (PDU), as shown in Figure 2.

NOTE The footer is usually either null or used for some form of message integrity code (MIC), such as an easily-spoofed computationally-simple checksum or a portion of a cryptographically-secured keyed hash.

The header and footer are often referred to as overhead with respect to the single or multiple or fractional conveyed SDU(s), with the amount of overhead depending upon how much additional information needs inclusion for the conveying protocol to function properly. Since one goal of this standard is to minimize energy consumption and channel occupancy when conveying PDUs, minimizing the amount of overhead at each protocol layer is a primary method of achieving that goal.

A complete description of each header and footer can be found in the appropriate protocol layer description. A full multi-layer PDU includes all headers and footers as shown in Figure 3. The amount of data (measured in octets) of an application PDU that can be sent in a single transmission is determined by the difference between the maximum permitted or supported PhL payload and the overhead imposed by all intermediary headers and footers.

Figure 2 – Typical single-layer PDU without fragmenting or blocking

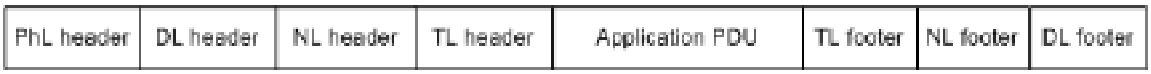


Figure 3 – Full multi-layer PDU structure used by this standard

4.5.5 Abstract data and concrete representations

4.5.5.1 Abstract data types

Each protocol layer of this standard defines the structure of the PDUs that it exchanges with peer protocol entities at the same layer, and of the SDUs and related interface control information (ICI) that it exchanges with adjacent layer entities within a local node of the network.

Additionally, each protocol layer defines management data structures that are exchanged by each layer entity (as conveyed data) with remote systems management entities.

Each of these data structures has an abstract form that requires a concrete representation. The abstract elements are either scalars, or composites composed of scalars and other composites. Composites can be homogeneous, in which case they are single or multi-dimensional arrays (often known as a “vector” or “matrix”, respectively), or heterogeneous (often known simply as a “data structure”).

The scalar elements used by the protocol entities of this standard are:

a) integers of a constrained range, where each abstract value is represented by the equivalent two’s-complement or unsigned concrete binary value, depending on whether the abstract range includes negative values;
b) enumerations, usually declared as an UnsignedN representation for N ≤ 8, where each abstract value generally is represented by the zero-origin ordinal index of the abstract value within the list of defined values;
c) Booleans, with two abstract values (FALSE, TRUE) to which the rules and operators of Boolean logic apply, with a concrete represention of FALSE as zero and TRUE as any non-zero binary value;

   NOTE 1 Booleans are named after the logician George Boole.

   NOTE 2 Although Booleans appear to be a special class of enumeration, the differences are that the value TRUE can be represented by any non-zero binary value, and that Boolean operators apply to this class.

d) IEEE floating point numbers of a specified range and precision, whose values are approximate real numbers or special non-numeric constants;
e) representations of TAI time as integer or scaled-fixed-point values modulo 2^{32} s.

The composite elements used by the protocol entities of this standard include three noteworthy classes of singly-dimensioned zero-origin arrays that are called *strings*:

f) characters, known as *visible strings*;
g) bits, known as *bit strings*; and
h) uninterpreted octets, known as *octet strings*.

Other composite elements include packed Boolean arrays, which are often (incorrectly) conflated with their underlying representation as bit strings.

4.5.5.2 Declarations of abstract data elements and their concrete representations

4.5.5.2.1 Simple declarations

Within this standard, an abstract type and its concrete representation are often declared in a unified form that indicates both the class of 4.5.5.1 a) through h) and the number of bits in the underlying binary representation.

- Integers are declared with an implicit range as *unsignedN*, implying a range of $0..2^{N}$-1, or *signedN*, implying a range of $-2^{N-1}..2^{N-1}$-1, where N is the number of bits of the representation, usually a multiple of 8.

  Integers that require a range other than that implied by their representation are declared as

  unsignedN range min..max

  where min and max are the minimum and maximum values of that integer element’s range, respectively.
- Booleans are declared as *BooleanN*, where N is the number of bits of the representation, usually either 1 (when within a packed data structure) or 8.
- Floating point numbers and their range and precision are declared as *float32* or *float64*.
- Fixed-size strings and their sizes are declared as *visibleStringN*, *octetStringN* and *bitStringN*, where N is the number of elements in the underlying array.

NOTE 1 An internal coding mechanism within some visible strings, usually a null (0x00) used as a content-end delimiter, often is used to truncate the effective size of the contained string. Many libraries of string operators presume such a coding.

- Varying-size strings are declared as *visibleString and octetString* without a concatenated declared size (i.e., the *N* of *visibleStringN*).
- Packed Boolean arrays and their size are declared as *BooleanArrayN*, where N is the number of elements in the array (and hence the number of bits in the representation).
- Named constants, usually of values for UnsignedN fields, may be declared in ASN.1 fashion as if they were ASN.1 enumerations:

  UnsignedN {name-1(integer-value-1), … , name-K(integer-value-K)}

  where N is the number of bits of the representation, and an explicit declaration of the constants are provided in the form of a bracketed, comma-separated list, each element of which is followed by "(K)" where K is the value assigned to that element.

  EXAMPLE 1 A declaration for the protocol layers defined in this standard might be:

  protocolLayers Unsigned3 {PhL(1), DL(2), NL(3), TL(4), AL(7)}

  NOTE 2 Example 1 demonstrate that the names of the elements of what amounts to an enumeration need not be disjoint from those used elsewhere in other places inthis standard, as might be required for an explicit programming language specification.

  Alternatively, these named constants may be declared as

  UnsignedN {integer-value-1:description-1; … ; integer-value-K:description-K}

  where N is the number of bits of the representation, and an explicit declaration of the named constants in the form of a bracketed, semicolon-separated list, each element of which is preceded by "K:" where K is the value assigned to that named constant. (See 3.3.5.)

  EXAMPLE 2 A declaration for the join_method defined in this standard might be:

  join_method Unsigned8{ 0:none; 1:join and start; 2:warm restart; 3:restart as provisioned; 4:reset to factory defaults }

  For this latter form of declaration, the bracketed list may be separated from the declaration of the field's representation.

  EXAMPLE 3 join_method Unsigned8

  Named constants:
0:none;
1:join and start;
2:warm restart;
3:restart as provisioned;
4:reset to factory defaults

  NOTE 3 This latter form of declaration often occurs in tabular descriptions of data structures in this standard where the first part of the declaration is in one column and the second part is in the same or a different column of the same row.

  Many times only a few elements of the range of an UnsignedN are named, such as may occur when the all-zero or all-ones value of the representation has a special interpretation. On some occasions, particularly when the description is "reserved", a value range may be specified rather than just a single value.

4.5.5.2.2 Declarations of compound objects, and of methods and their arguments

Within this standard. compound data structures are usually declared in tables, each row of which (after heading rows) describes a constituent element within the data structure.

Within this standard. method descriptions are also usually declared in tables. Each such description specifies a method name, a numeric method ID and a method description, followed by a series of descriptions of method input arguments, followed by a series of descriptions of method output arguments. As with data structure definitions, each argument is declared in its own row of the table and has a declared type and, where relevant, a declaration of the alternatives of the associated named constants.

4.6 Network characteristics

4.6.1 General

Characteristics of a WISN (i.e., a wireless network that conforms to this standard):

- scalable;
- extensible;
- support for simple operation;
- license-exempt operation;
- robustness in the presence of interference and with non-WISNs;
- determinism or contention-free media access;
- self-organizing network with support for redundant communications from field device to plant network;

  NOTE Redundancy support is not defined in this standard.
- IP-compatible network layer;
- coexistence with other wireless devices in the industrial workspace;
- security, including data authenticity, data confidentiality, data integrity, delay protection, and replay protection;
- system management of all communication devices;
- support for application processes using standard objects; and
- support for tunneling (i.e., transporting) other protocols through the wireless network.

4.6.2 Scalability

The architecture supports wireless systems that span the physical range from a single, small, isolated D-subnet, such as might be found in the vicinity of a gas or oil well or a very small machine shop, to integrated systems of many thousands of devices and multiple D-subnets that can cover a multi-square-kilometer plant. There is no technical limit on the number of devices that can participate in a network that is composed of multiple D-subnets. A D-subnet, a group of DLEs sharing some DL configuration aspects, may contain up to 30 000 DLEs (which is a limitation of the D-subnet addressing space). With multiple D-subnets, the number of DLEs (and thus devices) in the network can scale linearly.

The maximum amount of higher-layer or management data that can be conveyed in a single DPDU is limited by the PhL and the amount of required DL overhead. Therefore this standard supports fragmentation within the DL, enabling transmission of a much greater amount of data. In fragmentation, the data is segmented into appropriate-size portions at the originating DLE, encapsulated in DPDUs and transmitted through the D-subnet, then reassembled at the receiving DLE. One use of this mechanism is to update device firmware.

4.6.3 Extensibility

The protocols defined by this standard have fields and parameter value ranges that are reserved for future use, and version (edition) identifiers in headers that permit identification of the appropriate edition. These are intended to permit future revisions of this standard to offer additional or enhanced functionality without sacrificing backward compatibility, and without the encoding bloat that typically occurs through use of the ASN.1 declared extensibility mechanism.

4.6.4 Simple operation

Upon provisioning, as described in Clause 13, a DLE can automatically join the D-subnet and its superior N-network. Automatic device joining and D-subnet formation enables system configuration with minimal need for personnel who have specialized radio frequency (RF) training and tools.

Additionally, this standard supports the use of fully redundant and self-healing D-routing techniques to minimize D-subnet maintenance. (See 9.1.6 for further information.)

4.6.5 Site-license-exempt operation

This standard uses radios compliant with IEEE 802.15.4:2011, using 2,4 GHz DSSS channels 11..26 as specified in that standard.

NOTE 1 The 2,4 GHz ISM band is available and site-license-exempt in most countries, provided that the equipment has a type-license for such operation that is accepted in that country.

NOTE 2 ISA TR100.00.01 provides additional information on radio operation.

4.6.6 Robustness in the presence of interference, including from other wireless systems

This standard uses time-based channel hopping

- to provide a level of immunity against interference from other RF devices operating in the same band,
- to mitigate multipath interference effects,
- to facilitate coexistence with other RF systems, and
- to meet common regulatory requirements.

In some regulatory regimes, coexistence may be further enhanced through selective channel blacklisting, thereby avoiding otherwise-occupied channels within the band. Selective channel blacklisting can also enhance reliability by avoiding the use of channels with consistently poor performance.

4.6.7 Determinism and contention-free media access

This standard defines a time-division-multiple-access (TDMA) mechanism that allows a device to access the RF medium on a schedule, such that much of the competition for use of the channel has been pre-resolved by the scheduling agent. Time-synchronized communication is based on consecutive timeslots that have configurable durations, usually in the range of 10 ms to 12 ms. Scheduling and DLE operation are greatly simplified when all timeslots have a single, common duration, so WISNs usually are configured to have only one timeslot duration that is used for all timeslots.

NOTE 1 Because timeslots are assigned to logical channels that are then mapped cyclically to physical channels, use of a single common timeslot duration means that avoidance of contention on the logical channels automatically avoids that contention on the physical channels. Timeslots of differing durations lose this scheduling simplification.

A sending DLE is assigned a timeslot and channel unique to that device and the device to which it will communicate. These timeslot durations are configurable on a per-superframe basis. A superframe is a cyclic collection of timeslots. The ability to configure timeslot duration enables:

- shorter timeslots to take full advantage of optimized implementations;
- longer timeslots to accommodate:
  - extended DPDU wait times,
  - serial acknowledgment from two or more configured devices (e.g., duocast),

– CSMA/CA at the start of a timeslot (e.g., to implement listen-before-talk, or for prioritized access to shared timeslots);

- periods of extended duration for slow-channel-hopping.

NOTE 2 Local regulatory requirements may constrain the maximum duration of such a slow-channel-hopping period.

Support is provided for both dedicated time slots for predictable, regular traffic and shared time slots for bursty traffic such as alarms. Publishing/subscribing, client/server communications, alert reporting and bulk data transfer are also supported.

4.6.8 Self-organizing networking with support for redundancy

Fully redundant and self-healing routing techniques, such as mesh routing (see 9.1.6), support end-to-end network reliability in the face of changing RF and environmental conditions. Special characteristics that allow the network to adapt frequencies used (e.g., adaptive channel-hopping) along with mesh routing, can automatically mitigate coexistence issues without user intervention.

4.6.9 Internet-protocol-compatible NL

This standard's NL uses header formats that comply with the Internet Engineering Task Force's 6LoWPAN standards, thus facilitating potential use of IPv6-compatible networks as backbone networks in support of this standard. Use of headers that comply with 6LoWPAN does not imply either

- that a backbone network needs to be based on 6LoWPAN or IPv6, or
- that a network based on this standard is open to Internet hacking.

In fact, many networks based on this standard will not be directly connected to the Internet. Others may use the older IPv4 standard, at least during initial years of operation.

NOTE Use of IPv6 enables use of standard networking tools and software, as well as the potential for use of a wide variety of IETF standards in future revisions of or extensions to this standard.

4.6.10 Coexistence with other radio frequency systems

NOTE Coexistence with other radio frequency systems is the subject of concurrent IEC standardization. See IEC/TS 62657-2 for the expected form and direction of such standardization. It is likely that 4.6.10 will be reviewed and updated in the future, based in part on the progress of such concurrent standardization.

4.6.10.1 Coexistence overview

The system architecture specified by this standard is specifically designed to support coexistence with other

- WISNs (i.e., wireless systems conforming to this standard);
- other communication networks operating at 2,4 GHz that employ varying versions of IEEE 802.11, IEEE 802.15.1 and IEEE 802.15.4; and
- other devices that use the same radio frequency spectrum.

Operating with very short, time-synchronized communications tends to reduce congestion of RF bands and allow neighboring systems to recover quickly from lost or corrupted PhPDUs.

Due to the reduced dwell time on any one channel when channel-hopping, the impact on other radio systems is reduced and reliability in the face of interference is increased. For example, DPDUs may be resent on other, non-interfered channels. Selective channel blacklisting increases coexistence even further by avoiding those channels that are predetermined to be unusable or too congested.

This standard supports (but does not require) the use of clear channel assessment (CCA) to minimize collisions with non-synchronized systems, and also to provide CSMA/CA functionality within synchronized systems.

NOTE Some regulatory jurisdictions require use of CCA in certain modes of operation. Such required use is provided by this standard when a device is configured for operation in those jurisdictions.

The WISN architecture is designed to support operation in the presence of interference from unintentional radiators, such as microwave ovens, using channel-hopping and an automatic repeat-request (ARQ) protocol. ARQ is a common error control method for data transmission that uses acknowledgments for successful message reception, coupled with delayed retransmission in the case of erroneous reception, to achieve reliable data conveyance.

For additional information on diversity techniques that maximize coexistence, see 9.1.2.

4.6.10.2 Coexistence strategies

4.6.10.2.1 General

The following are examples of coexistence techniques that are not specific to any protocol. They improve coexistence with a wide range of devices sharing the 2,4 GHz band while optimizing the success of each communication attempt.

NOTE See IEC/TS 62657-2 for a more comprehensive discussion of wireless coexistence.

4.6.10.2.2 Leverage infrastructure for high data rate communication links

Multi-hop networks convey the same higher-layer data multiple times, once (or more) per hop. One basic capability of this standard is the ability to get the data to a DLE connected to a backbone subnet (preferably one offering a high data rate and low error rate) as directly as possible. This often reduces the use of the PhL specified by this standard to one or two D-transactions and D-subnet hops per conveyed higher-layer PDU.

4.6.10.2.3 Time-slotted operation

Time-slotted operation and scheduled transmissions serve to minimize collisions within the D-subnet, thus avoiding unnecessary use of the channel for retries.

4.6.10.2.4 Radio type selection

IEEE 802.15.4 was selected as the PhL for this standard because, under many conditions, overlapping similar radios, as well as IEEE 802.11 radios, can be active simultaneously without loss of conveyed data.

NOTE This standard is focused primarily on coexistence with the most recent versions of those standards, as older versions tend to be encountered less frequently as the years progress.

4.6.10.2.5 Low-duty cycle

Data conveyance for the focus applications described in 0.1 is infrequent, while added overhead from the conveying protocol layers is minimized.

4.6.10.2.6 Staccato transmissions

Expected transmissions are very short, which is a feature of the selected PhL. This enables co-located IEEE 802.11 networks to recover quickly in the event of inerference from the WISN.

4.6.10.2.7 Time diversity

Many of the focus applications have less stringent latency requirements than other users of the spectrum, providing more opportunity to use time diversity for coexistence. Configurable retry periods, potentially spanning hundreds of milliseconds, enable the system to coexist with other users that may require use of the same spectrum during higher-priority bursts of activity.

4.6.10.2.8 Channel diversity

The low-duty cycle of the radio is spread across up to sixteen IEEE 802.15.4 channels, further reducing the worst-case potential for interference to 1% of the time or less under many realistic scenarios.

4.6.10.2.9 Spectrum management

The user may configure superframes within the D-subnet to limit operation to certain radio channels.

4.6.10.2.10 Selective channel utilization

Where the regulatory regime permits, D-management can avoid problematic channels on a link-by-link basis, such as channels exhibiting IEEE 802.11 cross-interference or persistent multipath fades.

4.6.10.2.11 Collision avoidance

All DLEs support CSMA/CA, which allows a DLE to implement a "listen before talk" protocol to provide real-time detection of ongoing use of the channel and delay its own transmission, reducing interference to those other users.

NOTE Such avoidance is required in some regulatory regimes, at least in some modes of operation.

4.6.10.2.12 Varying PhPDUs

Due to the DL's built-in security measures, which include defense against same-channel and cross-channel replay attacks, PhPDUs vary from transmission to transmission even when retransmitting the same nominal DPDU information. With spread-spectrum modulation this results in time-varying interference even when otherwise-identical messages are being transmitted.

4.6.11 Time-slotted assigned-channel D-transactions as the basis for communication

4.6.11.1 Overview

Except during the interval when a DLE is soliciting the opportunity to join a D-subnet, each instance of DLE communication in accordance with this standard occurs

a) within a prespecified time window, known as a timeslot, relative to the DLE's sense of TAI time;
b) on a specific PhL channel at a power level that meets local regulations;
c) using a specific timeslot template for the intiator of a D-transaction, which specifies
    1) channel acquisition, configured in accord with local regulations and the timeslot template;
    2) transmission of a Data DPDU (i.e., the initial DPDU of a transaction) containing either higher-layer data or management data to a set of intended correspondents; and
    3) when so specified by the timeslot template, attempted reception of one or more ACK/NAK DPDUs (i.e., short control signaling) sent by intended correspondents;

       NOTE 1 Intentional reception of more than one ACK/NAK DPDU is useful for assessing network operation but is not essential for successful DPDU conveyance.

d) using a different specific timeslot template for an intended correspondent of a D-transaction, which specifies

   1) the duration of the channel acquistion phase, related to c)1);

   2) attempted reception of a Data DPDU containing either higher-layer data or management data addressed to either the DLE itself or to a specified other DL16Address; and

   NOTE 2 This latter capability is used for multicast/broadcast and duocast/N-cast

   3) when reception d)2) did occur and was error-free at the PhL with an error-free DLE FCS, and when so specified by the timeslot template, transmission of a single ACK/NAK DPDU (i.e., short control signaling) sent to the sending DL16Address of the DPDU received in d)2), occurring either

      i) at a specified delay after the end of receipt of the Data DPDU d)2), or

      ii) at a specified time before the scheduled end of the timeslot,

   as specified by the timeslot template.

When the corresponding timeslot template for the transaction initiator specifies more than one interval for ACK/NAK DPDU reception in c)3), then the timeslot templates for the transaction's responders differ in their assigned values for d)3)i) or d)3)ii), thus allocating those potential responses to disjoint time intervals within the timeslot.

The devices to which c)2) applies are known as *transaction initiators*; those devices to which d)2) applies are known as *transaction recipients* (which are the intended recipients, and not just eavesdroppers); those devices to which d)3) also applies are known preferentially as *transaction responders* (although they are also *transaction recipients*).

NOTE 3 IEEE 802.15.4e:2012 specifies mechanisms that are similar to, but not identical to, many of the DL mechanisms specified in this standard.

4.6.11.2 Channel acquisition phase

The channel acquisition phase of a transaction has two uses:

a) **ListenBeforeTalk (LBT)**: A mode of operation that is required in certain regulatory jurisdictions and optional in others, whose purpose is to reduce interference with other devices transmitting in the same frequency range, whether those devices use similar PhLs (e.g., other IEEE 802.15.4 systems on the same channel) or different PhLs that overlap in their frequency use (e.g., IEEE 802.11). In this mode of operation, intended transaction initiators sample the channel in a specified way (e.g., CCA mode 1) for a specified time interval, according to local regulatory requirements, and terminate the intended use of the timeslot if that sampling implies that the channel is in use by another device.

b) **CSMA/CA**: A means by which multiple DLEs conforming to this standard attempt to claim use of a designated shared-use timeslot. In this mode of operation each competing transaction initiator operates as in a) for a time interval that is uniformly chosen from a distribution of interval durations that increases exponentially with successive failures (up to some predetermined limit), thus providing exponential backoff in cases of congestion for use of the timeslot. When the smallest value of the selected interval is greater than zero, such operation supports prioritized access because DLEs implementing the CSMA/CA mode tend to defer to those that do not.

The two modes a) and b) can be combined to meet both sets of objectives. The timeslot templates used by intended recipients need to account for these initial delays, as in 4.6.11.1, d)1), so that such recipients do not terminate reception prematurely in an attempt to minimize the energy used by their PhL receivers when active.

Due to well-known properties of RF propagation, the above ListenBeforeTalk and CSMA/CA processes are not reliable in their ability to detect either channel use by other devices or the potential that channel use for one transaction will interfere with other, distant, ongoing communications. This issue has many names, often called the "hidden node" problem. Thus deferral of transactions due to a) and/or b) is always pessimistic with respect to projected interference, yet non-deferral is inadequate to avoid interference (which occurs at receivers) caused by concurrent RF emitters that fail to detect each other.

4.6.11.3 Communication phase

The communication phase of a transaction is used for three basic classes of transactions:

a) **Multicast/broadcast**: The timeslot template for the transaction initiator consists of 4.6.11.1, c)1) and c)2), with c)3) omitted, while the timeslot template for the transaction recipients consists of 4.6.11.1, d)1) and d)2), with d)3) omitted. For these transactions there are no transaction responders, since no opportunity for an immediate response of short control signaling is provided in the template.

b) **Unicast**: The timeslot template for the transaction initiator consists of all parts of 4.6.11.1, c), while the template for the recipients consists of all parts of 4.6.11.1, d). For these transactions there is exactly one intended transaction responder, with a single ACK/NAK immediate response opportunity provided in the template.

c) **Duocast/n-cast**: The timeslot template for the transaction initiator consists of all parts of 4.6.11.1, c), while the template for the recipients consists of all parts of 4.6.11.1, d). For these transactions there is a designated number (2 or n, respectively, for duocast and n-cast) of intended transaction responders, each with a different timeslot template that specifies a single response opportunity for an ACK/NAK DPDU, disjoint from all the other response opportunities for the same transaction.

   In this transaction class all, or all but the first, transaction responders sensitize themselves to receiving a DPDU whose destination DL16Address is not the DLE's own DL16Address. In such cases the timeslot template also specifies the DL16Address to be used for this purpose. This use of a distinct DL16Address applies only to the Data DPDU of the transaction; any ACK/NAK DPDU uses the actual explicit or implied DL16Addresses of the transaction responder and initatior.

4.6.12 Robust and flexible security

All compliant networks have a security manager to manage and authenticate cryptographic keys in transit. Security primitives defined by IEEE 802.15.4:2011 are used by the DLE and TLE, providing message originator authentication, message integrity, and optional message content privacy.

Device authentication is enabled by the use of symmetric keys and unique device IDs, with an option for use of asymmetric keys during the device provisioning process and some other security-related processes.

During normal operation, received data authenticity and data integrity is verifiable through the use of secret symmetric keys known to both the originator and the receiver(s).

During provisioning, the authenticity of the received device credentials from a new device may be verified by a system manager through the optional use of public keys shared openly by the new device, and a corresponding asymmetric private key kept secret within the new device.

PDUs are protected using the default AES-128 or an alternative, locally-mandated block cipher, using standard cryptographic modes. Secret symmetric keys that are known to communicating entites are used to secure device-to-device communication.

4.6.13 System management

This standard includes functions to manage communication resources on each individual device, as well as system resources that impact end-to-end performance. System management provides for policy-based management of the runtime configuration and also monitors and reports on configuration, performance, fault conditions, and operational status. The system management functions take part in activities such as:

- device joining and leaving the network;
- reporting of faults that occur in the network;
- communication configuration;
- configuration of clock distribution and the setting of system time;
- device monitoring;
- performance monitoring and optimization.

System security management works in conjunction with the system management function and, potentially, external security systems to enable secure system operation.

All management functions are accessible remotely via the gateway.

4.6.14 Application process using standard objects

This standard's application process is represented as a standard object which contains one or more communicating components drawing from a set of standard defined application objects. These objects provide storage for and access the data of an application process.

Defining standard objects provide an open representation of the capabilities of a distributed application in a definitive manner, thereby enabling independent implementations to interoperate. Objects are defined to enable not only interaction among field devices but also interoperation with different host systems.

The standard objects and services of this standard may be used to directly map existing legacy field device communications onto standard objects and application sublayer communication services, thereby providing a means to adapt legacy devices to communicate over the WISN.

4.6.15 Tunneling

The native protocols defined by this standard allow devices to encapsulate foreign PDUs and transport these foreign PDUs through the WISN to a destination device within the WISN, which usually is a gateway to legacy protocols. This encapsulation mechanism is referred to as tunneling. Successful application of tunneling depends upon how well the foreign protocol's technical requirements (e.g., timing, latency, etc.) are met by the instantiation of the WISN.

5 Systems

5.1 General

In this standard a system is defined to have an application focus and addresses applications and their needs. Networks, on the other hand, have a communication focus and are devoted to the task of device-to-device communication. For the purposes of this standard, a network is a component of a larger system.

Clause 5 describes how the various protocol layers and functions of this standard work together to form a system that achieves the goals of this standard. Specifically, Clause 5 describes the system aspects of devices, networks, protocol suite, data flow, a shared time base, and the applications need for firmware revisions.

5.2 Devices

5.2.1 General

A device implements a combination of protocol layers, usually including a PhLE, a DLE, a NLE, a TLE and an ALE, and may include functions such as the system manager role, the security manager role, one or more gateway roles, and support for provisioning other devices in the network.

NOTE Only system behaviors are specified in Clause 5.

5.2.2 Device interworkability

Device interworkability is the ability of devices from multiple vendors to communicate and maintain the complete network. Device interworkability requires control over the device's various options, configuration settings, and capabilities:

a) **Options**: To allow all devices to interwork, and to interoperate within a constrained domain of application, regardless of implemented options (those defined within this standard), devices shall be capable of disabling (i.e., not using) any options that are not mandatory for the device's configured role(s), as specified in the role profiles of Annex B.
b) **Configuration settings**: The system manager is responsible for configuring WISN devices and roles implemented by WISN devices. The system manager is described in Clause 6. Some configuration aspects are described in Annex D.
c) **Capabilities**: There are minimum capabilities to be met for devices based upon their role in the system. Annex B defines the baseline capabilities required for all devices.

5.2.3 Profiles

A profile can be described as a vertical slice through the protocol layers. It defines those options in each protocol layer that are mandatory for that profile. It also defines configurations and parameter ranges for each protocol. The profile concept is used to reduce the risk of device interworkability and interoperability problems between different manufacturers' products. Interoperability in areas outside the scope of this standard requires either use of profiles beyond those of this standard, or other extra-standard arrangements.

A role profile is defined as the baseline capabilities, including any optional features, settings, and configurations, that are required of a device to perform that role adequately. The roles are defined in 5.2.6.2.

All devices conforming to this standard include a default application profile that addresses many basic process automation needs. That default profie is used by the System Manager to manage the various protocol layer management entities in each device, including aspects related to reporting of protocol-layer-related events.

5.2.4 Quality of service

An application within a device is assumed to know the level of service that is necessary for its proper operation. The level of quality of service (QoS) is agreed upon via a contract between the system manager and the requesting device. When the application within a device desires to communicate at a certain QoS level, it sends a request to the system manager notifying it that it wishes to communicate with a specific destination and that it desires a given QoS level. This desired QoS level is indicated by a desired contract and message priority. In addition, a certain level of reliability, periodicity, phase, and deadline for periodic messages, and short-term burst rate, long-term burst rate, and maximum number of outstanding requests for client/server messages, may be indicated. See 6.3.11.2.7 for specific information on QoS.

5.2.5 Device worldwide applicability

The Type A field medium employs a widely used and accepted physical interface and is therefore appropriate for use in many geographic regions of the world. However, some regions have special considerations that may impose different regulatory requirements. Even if a product is designed to meet those regulatory requirements, its use is often not legally permitted until it has undergone compliance testing and any required local permits or certificates have been issued by country-specific regulatory agencies.

This standard does not specify what regulatory certifications or permits a product compliant with this standard needs; that is the responsibility of the product manufacturer and the end user to determine. A product may have certifications for operation in multiple countries or regions.

Specific design considerations within this standard support the transition between different regulatory regimes, of mobile wireless systems conforming to this standard. For example, tanker ships moving between ocean ports are subject to the local regulations of each port at which they dock. Annex U addresses specific provisions that have been made in this standard to support regulatory approval of both fixed-locale equipment and systems, and of those that move between regulatory regimes.

5.2.6 Device description

5.2.6.1 General

Within this standard, devices are the physical embodiment of the behaviors, configuration settings, and capabilities that are necessary to implement and operate a network. There are many different types of devices depending upon the application, environment, and function within the network. To fully describe necessary network behavior without defining specific device implementations this standard defines roles, protocol layers, and a field medium that devices may embody.

A role defines a collection of functions and capabilities. This standard defines all the roles necessary for the network to operate properly, including system manager, security manager, gateway, backbone router, system time source, provisioning, router, and I/O device. All devices conforming to this standard shall implement at least one role; however, a device may implement many roles. A device implementing a role shall implement all functions required for that role in 5.3.

The protocol layers describe required behaviors. Not all devices are required to implement all the protocol layers defined in this standard. However, all devices conforming to this standard shall implement the network and transport layers in addition to the DMAP functionality as described in 6.2. Every device shall contain a device management function and a device security management function that cooperate with the system processes to enable secure management of a device’s resources and the device’s usage of system resources.

A field medium is represented within a device a combination of a PhLE and a DLE, both as described in this standard. While not all devices need to implement a field medium, any device that implements the I/O, routing, or backbone routing roles shall directly support at least one field medium as specified by this standard.

Figure 4 illustrates the distinction between physical devices (e.g., as supplied by a manufacturer) and the roles that those devices can assume.

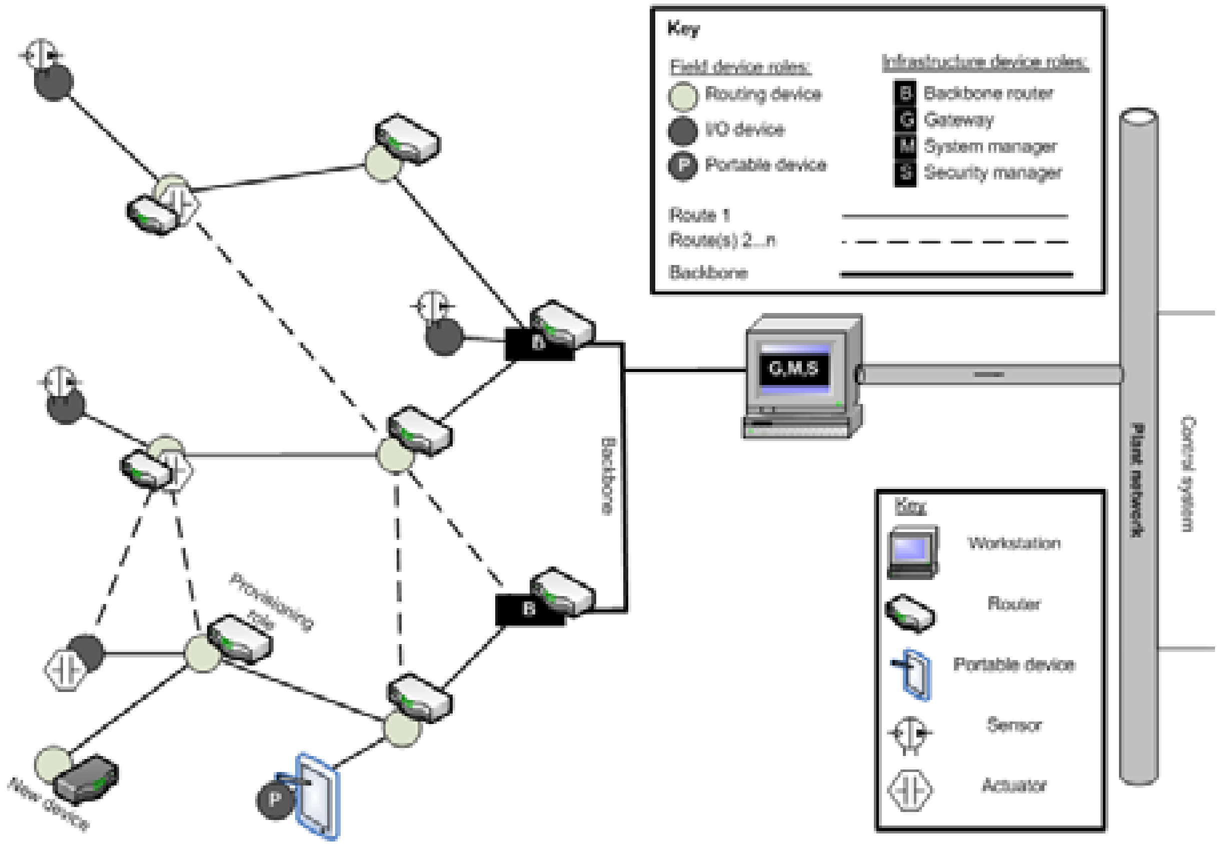


Figure 4 – Physical devices versus roles

Figure 4 shows a representative yet complete network compliant with this standard. Within this network are several types of devices, including sensors, actuators, routers, a handheld computer, and a workstation. As shown in Figure 4, each of these devices may assume different roles within the network. For example:

- The workstation has assumed the roles of gateway, system manager, and security manager. These roles are described in 5.2.6.10, 5.2.6.11, and 5.2.6.12, respectively.
- Two devices have assumed the role of backbone routers (described in 5.2.6.9), while seven other devices have assumed the role of routers (described in 5.2.6.7).
- Three sensors, one actuator and a portable computer have assumed the singular role of an I/O device (5.2.6.6).
- The router at the lower left of Figure 4 has assumed a provisioning role, as described in 5.2.6.8, and will provision the new device being introduced.
- Two actuator devices have assumed both the router and I/O roles.

NOTE 1 Although Figure 4 shows the use of a backbone network, the functionality of the backbone network is not specified within this standard.

NOTE 2 The physical devices and roles shown in Figure 4 are intended only as examples.

5.2.6.2 Field medium

5.2.6.3 General

This standard defines one specific field medium, Type A. A field medium type defines the protocol for the PhL and lower DL (i.e, the MAC). Future revisions of this standard may support multiple field media types.

5.2.6.4 Type A

The Type A field medium consists of the PhL and DL as specified by Clause 8 and Clause 9 of this standard.

Devices implementing the Type A field medium and the DL shall implement configuration settings for Radio Silence. The Radio Silence configuration is used to restrain the radio from transmitting during inappropriate times, such as when transmission is unsafe or when regulations prohibits radio transmissions. The Radio Silence configuration settings are defined in 9.1.15.4.

5.2.6.5 Role definitions

5.2.6.6 Input/output

A device with the I/O role shall provide (source) data to or use (consume) data from other devices (and may both provide and use data) and shall have at least one user application process (UAP) object. A device with only an I/O role is a device that has the minimum characteristics required to participate in a network compliant with this standard. The I/O role provides no mechanism for the forwarding of messages or routing for any other device. This enables the construction of devices with the least complexity and the potential for low energy consumption, since they need not expend energy routing other devices' messages, nor are they required to accept and provision new devices wishing to join the network.

NOTE A data source supplies data. An actuator would be an example of a consumer of data (i.e. sink), whereas a sensor would supply data (i.e. source).

Devices that implement the I/O role shall implement the Type A field medium.

5.2.6.7 Router

A device with the router role shall have routing capability, shall act as a proxy, and shall have clock propagation capability. These devices can provide range extension for a network and path redundancy and may provide different levels of QoS on a message-by-message basis. The system manager may disable the routing capabilities of the router role to optimize system performance requirements such as message latency or battery consumption.

Devices that implement the router role shall implement the Type A field medium.

5.2.6.8 Provisioning

A device with the provisioning role (provisioning device) shall be able to provision a device set to factory defaults and shall implement the device provisioning object (DPO; see Clause 13). The provisioning device inserts the required configuration data into a device to allow a device to join a specific network. Devices implementing the PhL shall be capable of being provisioned using the defined physical interface. This capability can be disabled (see Clause 13).

Devices that implement the provisioning role shall implement the Type A field medium.

5.2.6.9 Backbone router

A device with the backbone router role shall have routing capability via the backbone, and shall act as a proxy using the backbone. Backbone routers enable external networks to carry native protocol by encapsulating the PDUs for transport. This allows a network described by this standard to use other networks, including longer-range or higher-performance networks.

While the media and protocol suites of backbone networks are not defined in this standard, it is believed that many instantiations of the backbone router will be with internet protocol (IP) networks. Many of these backbone networks may conform to IPv4 as opposed to the newer

IPv6. Clause 10 describes how a WISN NPDU received by a BBR at the BBR's WISN DLE interface is converted into a fully compliant IPv6 NPDU. If the BBR's backbone interface implements IPv6, then the NPDU may simply be routed using standard IPv6. If the BBR's backbone interface implements IPv4, then the BBR shall support the use of IETF RFC 2529 to route the NPDU across the IPv4 backbone.

Devices implementing the backbone router role shall implement the Type A field medium in addition to the BBR's backbone network interface.

5.2.6.10 Gateway

A device with the gateway role implements a high-side interface. An example of an internal GIAP supporting such a high-side interface is given in Annex U. The gateway communicates over the WISN by native access and/or tunneling. Such a device shall have a UAP. The gateway role provides an interface between the WISN and the plant network, or directly to an end application on a plant network. More generally, a gateway marks the transition between communications compliant with this standard and other communications and acts as a protocol translator between an AE described by this standard and other AEs. There can be multiple gateways in a system.

5.2.6.11 System manager

A device implementing the system manager role shall implement the SMAP (6.3.2) and shall set the time source tree.

The system manager is a specialized function that governs the network, devices, and communications. The system manager performs policy-based control of the network runtime configuration, monitors and reports on communication configuration, performance, and operational status, and provides time-related services.

When two devices need to communicate, they do so using a contract. A contract is an agreement between the system manager and a device in the network that involves the allocation of network resources by the system manager to support a particular communication need of this device. This contract is made between the applications in both devices and the system manager. The system manager will assign a contract ID to the contract, and the application within the device will use the contract for communications. An application may only request the creation, modification, or termination to a contract. It is the sole responsibility of the system manager to create, maintain, modify, and terminate the contract.

For more information on system management, see Clause 6.

5.2.6.12 Security manager

The system security management function, or security manager, is a specialized function that works in conjunction with the system manager and, potentially, external security systems to enable secure system operation. The security manager is logically separable from the system manager, and in some use cases will be resident on a separate device and in a separate location. Every system compliant with this standard shall have a security manager. For more information on the security manager, see Clause 7.

NOTE The communication protocol used between the system manager and the security manager is not defined by this standard because such components are usually supplied by a vendor as a matched pair/set.

For more information on the security management functionality, see 7.7.

5.2.6.13 System time source

A device implementing the system time source role shall implement the master time source for the system. A sense of time is an important aspect of this standard; it is used to manage device operation. The system time source provides a sense of time for the entire system. This is described in more detail in 6.3.10.1.

Devices implementing the system time source role shall implement any of the I/O, router, backbone router, system manager, or gateway roles.

5.2.7 Device addressing

Each device that implements the Type A field medium shall be assigned a DL16Address for the D-subnet, which is used for local addressing. Each device shall have an EUI64Address that is unique. See Clause 9 for further information.

Each device shall also have an IPv6Address that is assigned by the system manager as described in 6.3.5. The system manager may choose to assign the IPv6Address as a logical address to maintain ALE linkage in the event of a device replacement. The IPv6Address may be used by the application to reach a particular device within a system after the join process is complete. See Clause 10 for further information.

5.2.8 Device phases

5.2.8.1 General

A device may go through several phases during its operational lifetime. Within each of these phases are multiple states. A notional representation of the phases of the life of a device is shown in Figure 5. See Figure 136 and Figure 137 for normative detail.

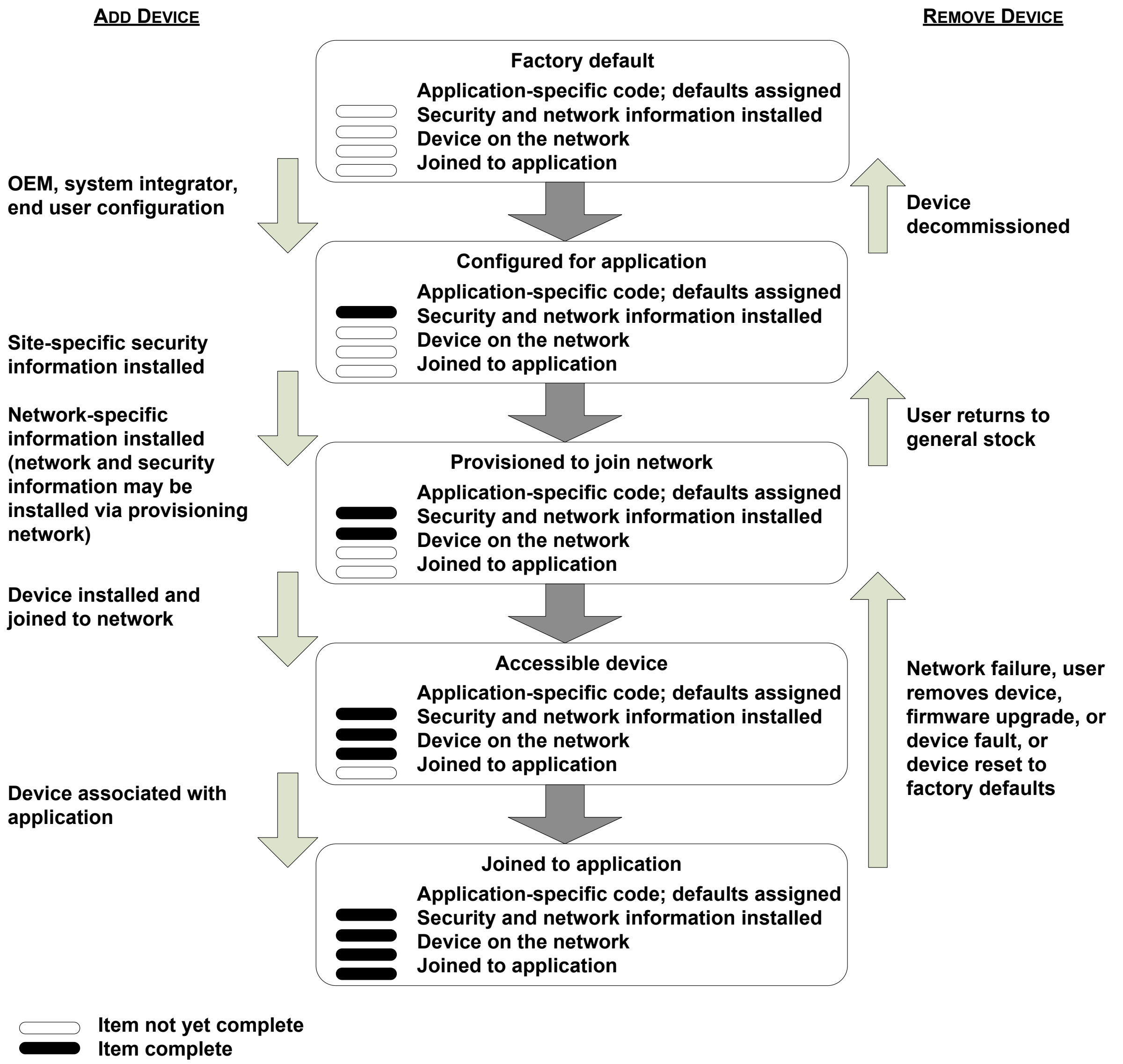


Figure 5 – Notional representation of device phases

A device can pass through these phases several times as it is commissioned and used, then decommissioned and re-commissioned for a different application. After joining the network, devices shall be able to report their status so that applications know whether a device is accessible and whether it is joined to an application.

5.2.8.2 Factory default

A device is considered non-configured if it has not been configured or commissioned with any application- or network-specific information. A non-configured device may come from a manufacturer or may enter a non-configured state as a result of decommissioning.

5.2.8.3 Configured for application

A device is considered configured for an application when it has received its own application-specific programming and when all appropriate defaults have been applied. A device configured for application may come from a manufacturer or may be supplied by a systems integrator or other value added reseller, already provisioned for the intended application. Over-the-air application program updates can occur, but are handled by the device ALE.

5.2.8.4 Provisioned to join the network

A device is provisioned to join the network when it has obtained the appropriate security credentials and network-specific information. A device will usually enter this phase when it has been prepared for installation in an automation application. Often the device will not communicate directly with the security manager; instead, the system manager serves as a relay for all communication with the security manager.

5.2.8.5 Accessible device

A device is considered accessible when it has joined the network and has been authenticated by the system manager. An accessible device can communicate with the system manager.

5.2.8.6 Joined to application

In this phase, an application object on the device can send or receive information to or from the desired application objects on peer devices. See 7.4 for additional detail on the join process.

NOTE Application objects in any two devices on the network are able to communicate with one another. Refer to Clause 12 for more detail.

5.2.9 Device energy sources

This standard does not restrict the types of energy sources a device may use. The standard allows for energy efficient device behavior that accommodates device operation for long periods of time (e.g., five to ten years) using suitable batteries.

The types of energy sources may be grouped into five categories:

- mains;
- limited battery (e.g., button cell);
- moderate battery (e.g., lead acid);
- rechargeable battery;
- environmental or energy-scavenging.

Devices implementing the roles of I/O or router may be expected to use any category of energy source. The roles of security manager, system manager, gateway, and backbone router are usually performance intensive; therefore devices implementing these roles are recommended to have more capable energy sources such as the mains or moderate battery categories.

The energy source status of devices is critical to proper system management. All devices shall provide energy supply information to the system manager. This information may be used in making routing decisions. See Clause 9 for further details.

5.3 Networks

5.3.1 General

The focus of networks is device-to-device communication. There are numerous aspects of the networks' ability to communicate. These aspects include the atomic (i.e., minimal or irreducible) network, network topologies, device relationships within a network, protocol suite structure, and the concept of shared time.

5.3.2 Minimal network

A minimal network is a network with the minimum amount of devices implementing the minimum number of roles. Although a minimum system could be constructed with just a system manager and a security manager, a more practical minimum system would include the roles of system manager, security manager, provisioning, system time source, and I/O. The system manager and security manager are two separate roles and may reside in the same device or may be split between two physical devices. A single physical device may assume multiple roles. Therefore, a minimal network shall consist of two devices communicating with each other, where one device implements the roles of system manager and security manager; the roles of provisioning, system time source, and I/O are implemented by either of the devices.

A small representative network of four field devices and one infrastructure device is shown in

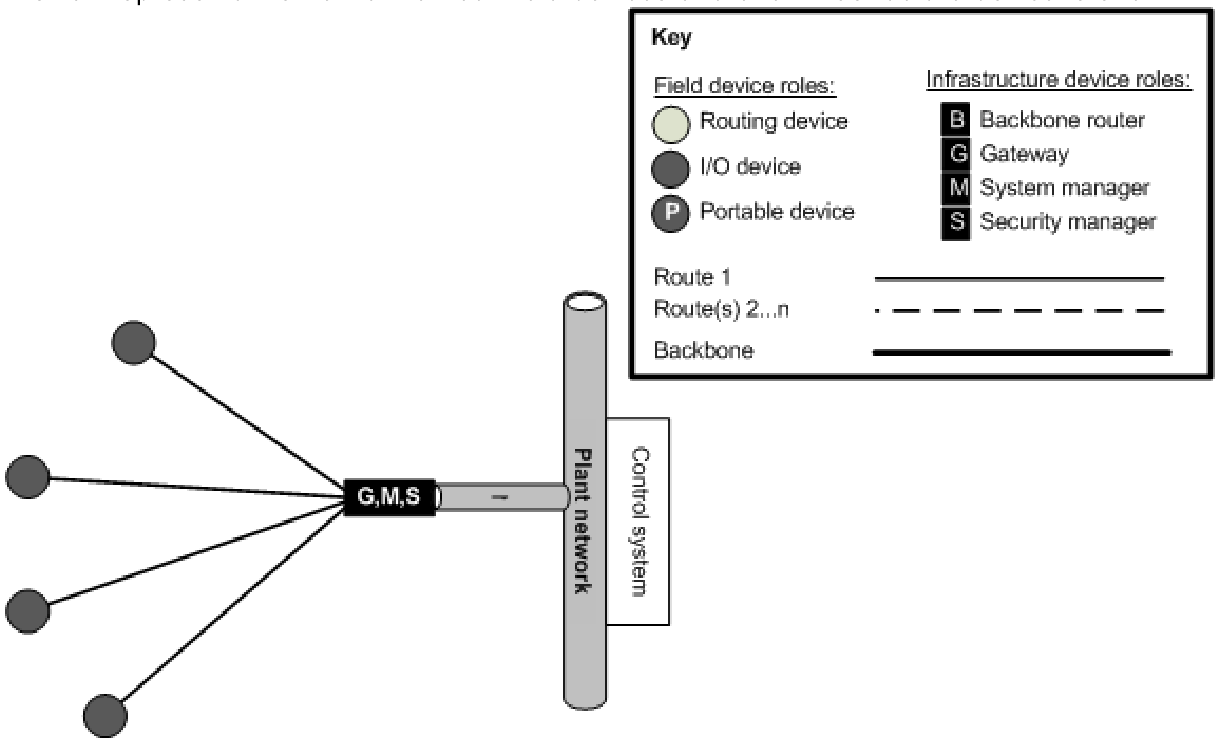


Figure 6. Although such a network is atypical, it represents a small compliant system. In this network, a single physical device has assumed the roles of gateway, system manager and security manager.

5.3.3 Basic network topologies supported

5.3.3.1 General

The figures in 5.3.3 provide several informative examples and illustrate the flexibility of the system architecture. The set of examples is not intended to be exhaustive. These examples are presented here only to provide a better understanding of the system elements.

5.3.3.2 Star topology

This standard supports a simple star topology, as shown in Figure 6.

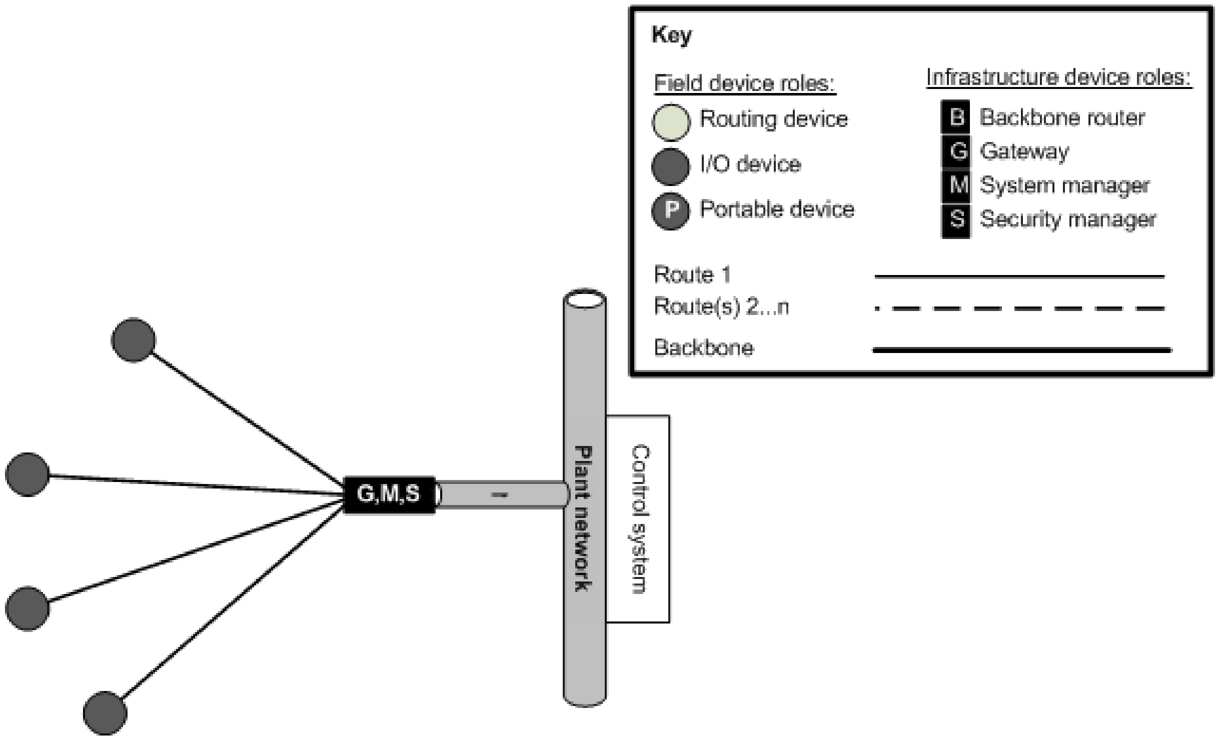


Figure 6 – Simple star topology

This system configuration can yield the lowest possible latency across the physical layer. It is architecturally very simple, but is limited to the range of a single hop.

Within the figures in 5.3.3, each box labeled G,M,S represents a collection of three separate roles combined into one physical device in the relatively simple networks depicted:

- a gateway;
- a system manager; and
- a security manager.

5.3.3.3 Hub-and-spoke topology

Expanding the network using the backbone routers allows the user to construct a hub-and-spoke network, as shown in Figure 7, wherein devices are clustered around each backbone router, providing access to the high-speed backbone.

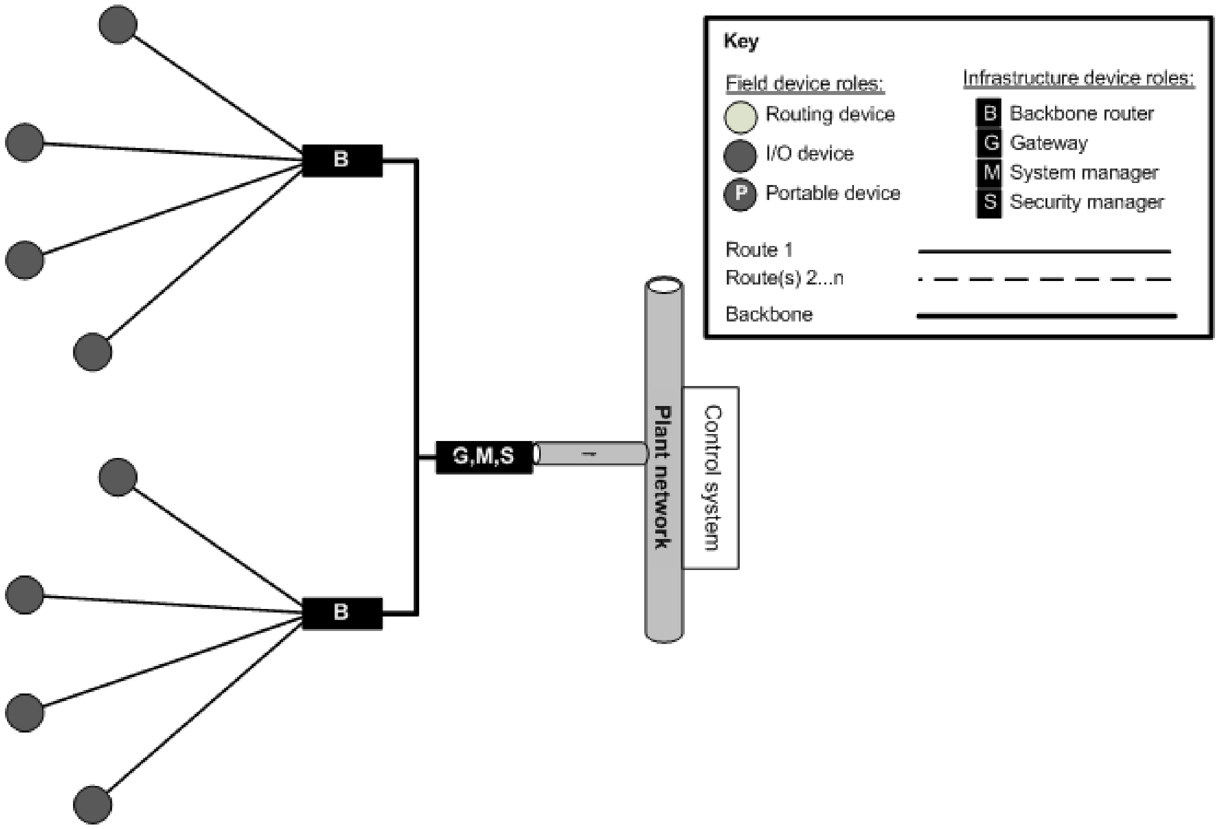


Figure 7 – Simple hub-and-spoke topology

In this case, latency is slightly degraded from the simple star topology, but overall throughput can increase, and in larger systems, average latency can decrease because of the multiple data pipes available (one through each backbone router). Although the network can expand further away from the gateway, it is nonetheless limited to single-hop range around a backbone router.

5.3.3.4 Mesh topology

This standard supports mesh networking topologies, as shown in Figure 8.

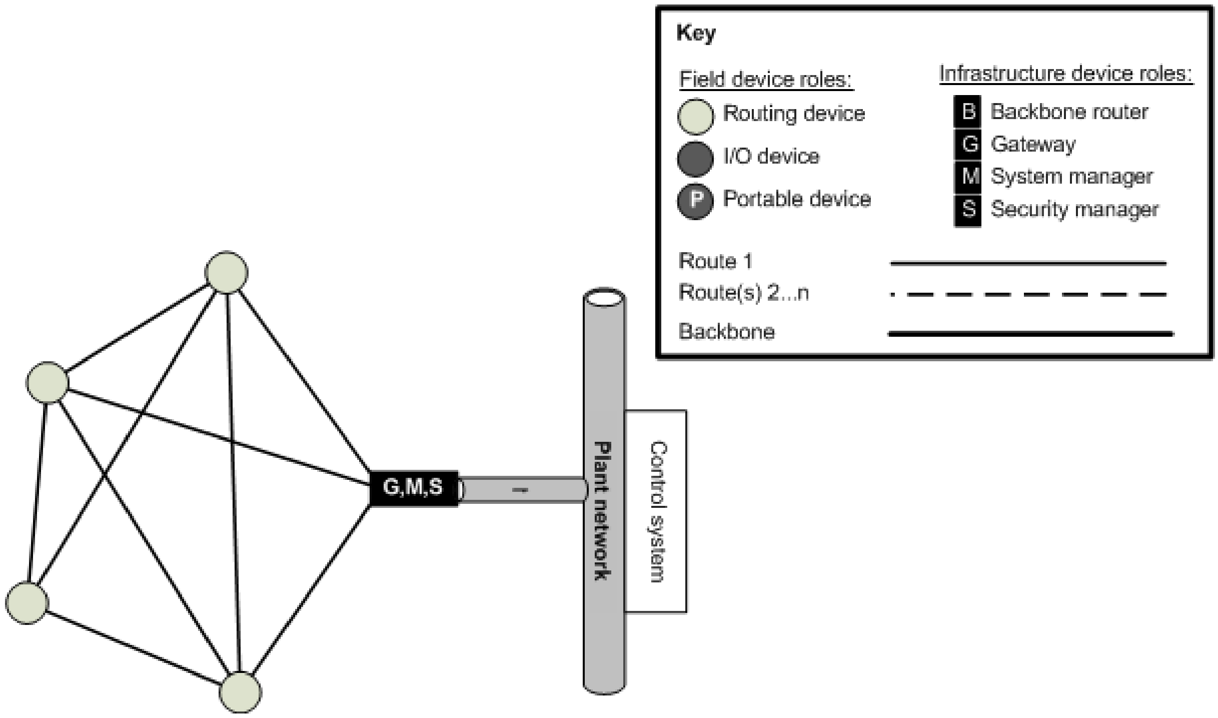


Figure 8 – Mesh topology

In some cases, the number of routes a device can support may be limited. Range is extended as multiple hops are supported. Latency is larger, but can be minimized by proper scheduling of transmissions. Throughput is degraded as device resources are used in repeating messages. Reliability may be improved through the use of path diversity.

For more information on mesh topology, see 9.1.14.

5.3.3.5 Star-mesh topology

The star topology combined with the mesh topology is shown in Figure 9.

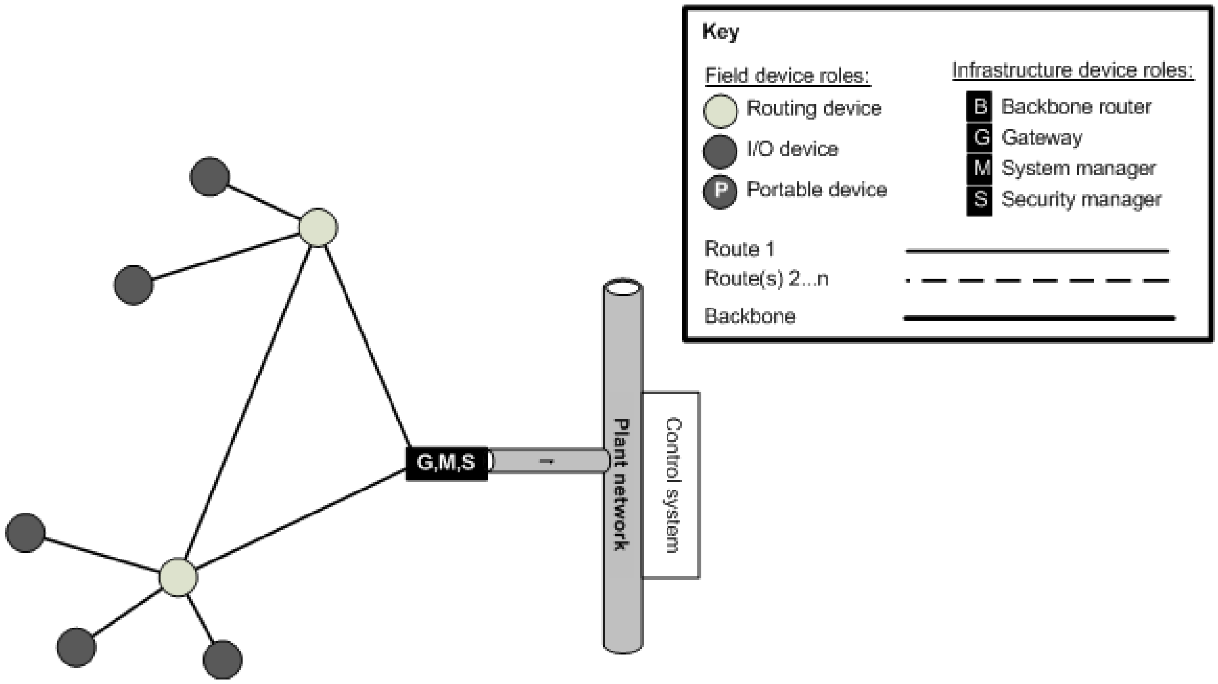


Figure 9 – Simple star-mesh topology

This configuration has the advantage of limiting the number of hops in a network. It does not have the added reliability that full mesh networking can provide.

5.3.3.6 Combinations of topologies

This standard allows for the combination of any of the previously mentioned topologies, so that a configuration can be constructed that best satisfies the needs of the application. For example, monitoring systems that span large physical areas within a plant may use the star-mesh topology or a combination of hub-and-spoke and star-mesh topologies, whereas certain control applications where latency is critical may benefit from a pure star or hub-and-spoke topology. The flexibility of the system allows for all of these topologies to operate in harmony, in any combination.

5.3.4 Network configurations

5.3.4.1 General

The D-subnet in this standard comprises one or more groups of wireless devices, with a shared system manager and (when applicable) a shared backbone. While a D-subnet stops at the backbone router (see 5.5.6), network routing may extend into the backbone and plant network. A complete network includes all related D-subnets, as well as other devices connected via the backbone, such as a gateway, system manager, or security manager. Figure 10 and Figure 11 illustrate the distinction between a D-subnet and a network.

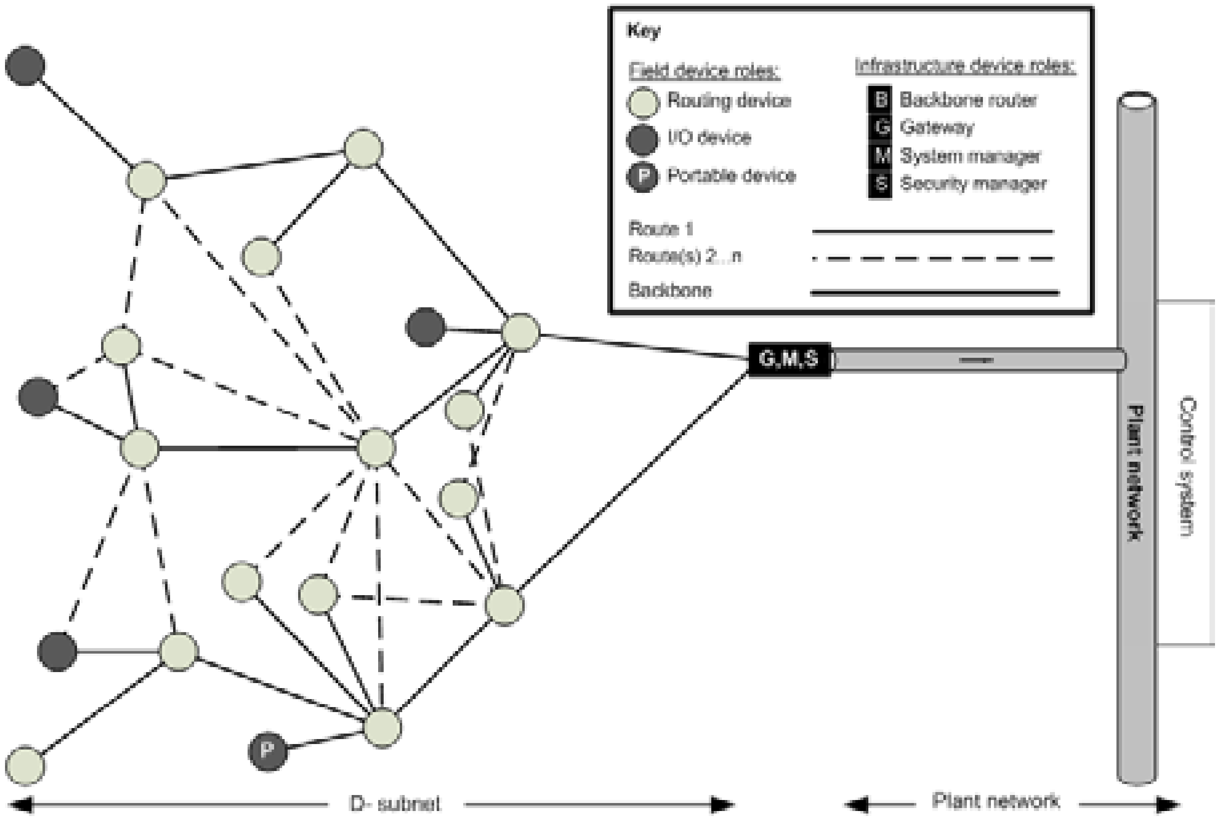


Figure 10 – Example where network and D-subnet overlap

Figure 10 illustrates a simple network comprised of a collection of wireless devices called a D-subnet and additional devices that manage the D-subnet and connect it to other networks. In Figure 10, the network and the D-subnet are the same.

The D-subnet is comprised of both routing and I/O devices. The solid lines between devices designate the first route established between devices, while dotted lines designate the second route, the third route, and so on. Messages may be routed using any one of the known routes.

In Figure 11, the D-subnet includes a collection of field devices up to the backbone routers (boxes labeled B). Backbone routers use connections to a network backbone to reduce the number of hops that messages would otherwise require; this can improve reliability, reduce latency, and extend the coverage of the network.

The network in Figure 11 includes the D-subnet, as well as the backbone and a gateway, system manager, and security manager, which are co-located on the backbone.

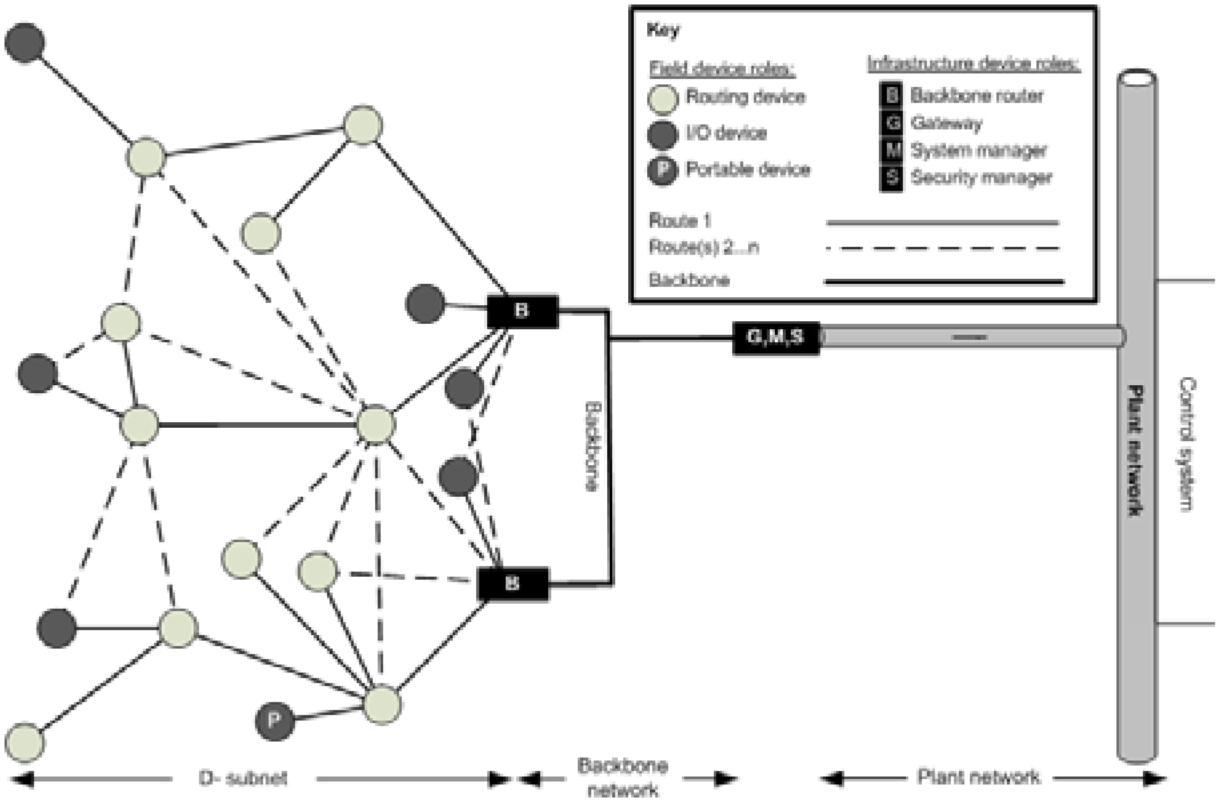


Figure 11 – Example where network and D-subnet differ

5.3.4.2 Multiple gateways – redundancy and additional functions

Figure 12 illustrates a different physical configuration with three gateway devices. One of the gateway devices also implements the system manager and security manager functions.

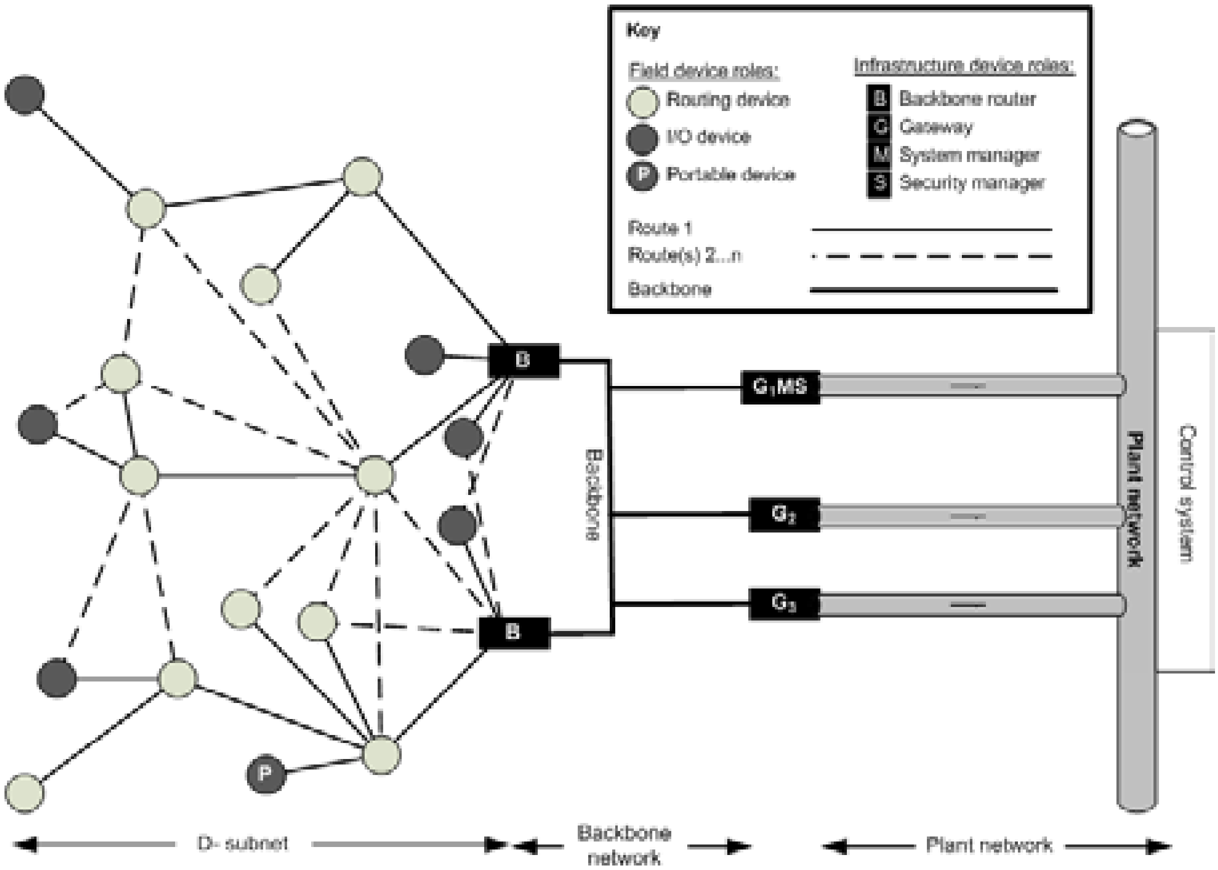


Figure 12 – Network with multiple gateways

The gateway devices may be identical (i.e., mirrored, for redundancy) or unique, for example, with each gateway implementing a software application to handle communications between a particular class of device and a control system attached to the plant network.

5.3.4.3 Multiple gateways - designating a gateway as a backup

NOTE This standard does not define the functionality of a backup gateway nor the mechanisms for synchronization of backup gateways.

Figure 13 is similar to Figure 10, but with a second G,M,S device (gateway, system manager, and security manager).

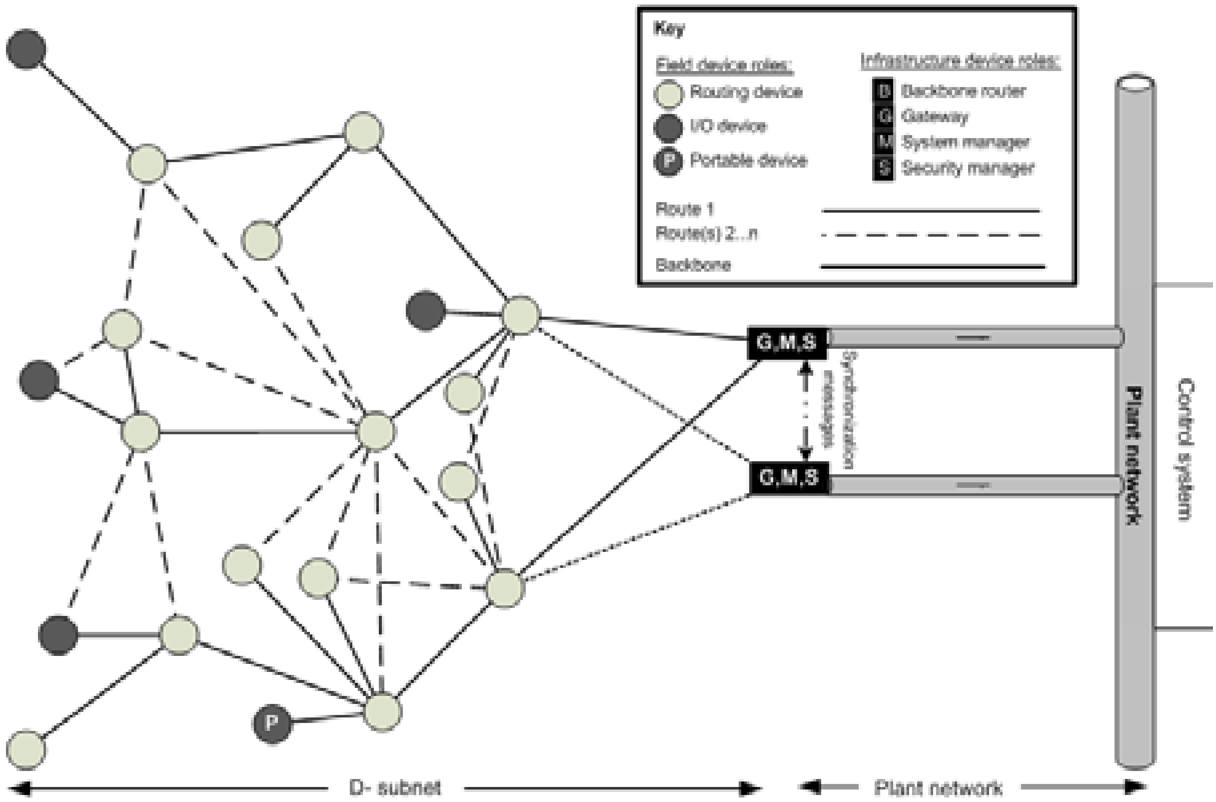


Figure 13 – Basic network with backup gateway

The two G,M,S devices offer identical functionality and may coordinate their operation via synchronization messages exchanged through a backchannel mechanism not specified by this standard. A single G,M,S device may be responsible for all gateway, system manager, and security manager functions, with a second G,M,S device acting as an active standby that remains idle until it is needed. Alternatively, the two G,M,S devices may divide the workload between them until one fails.

5.3.4.4 Adding backbone routers

To the basic network shown in Figure 13, Figure 14 adds backbone routers (boxes labeled B), which facilitate expansion of networks compliant with this standard, in terms of both the number of devices and the area the network occupies.

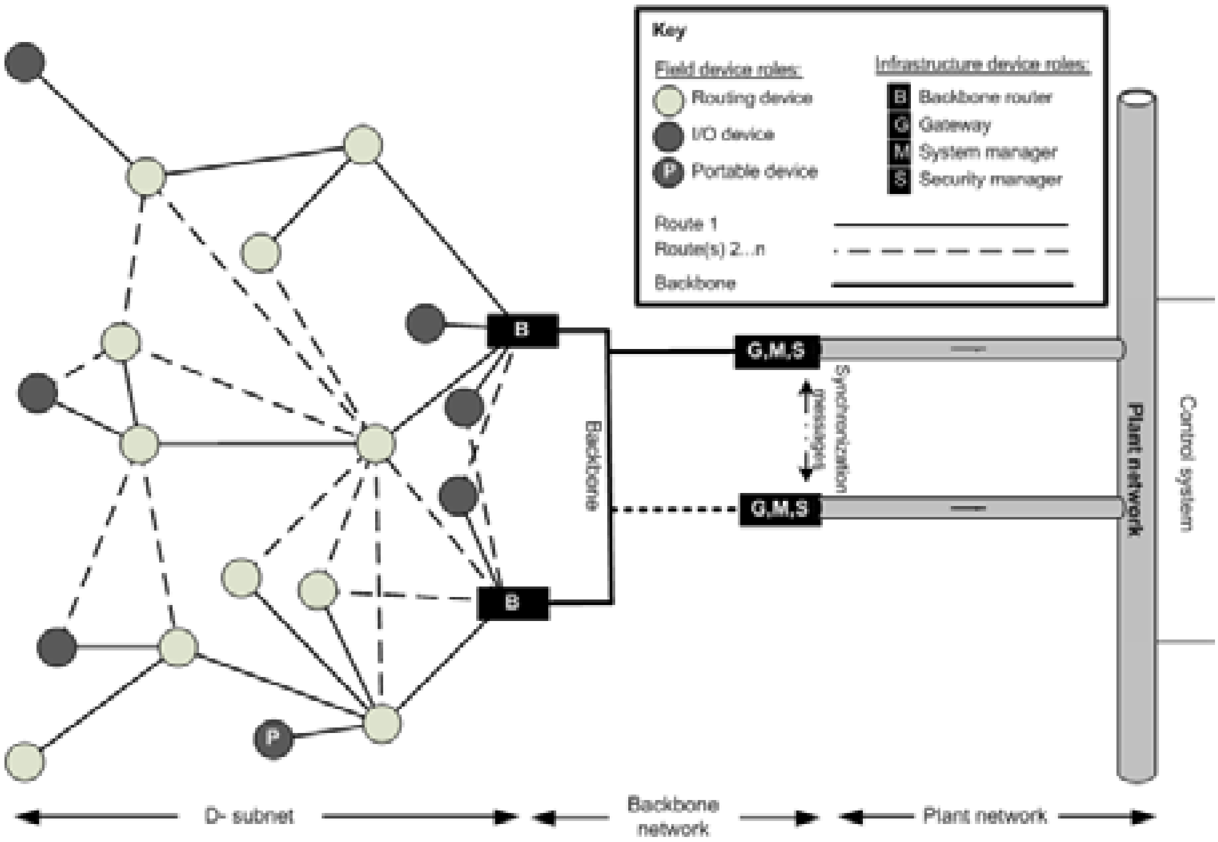


Figure 14 – Network with backbone

5.3.5 Gateway, system manager, and security manager

As shown in Figure 15, the functional roles fulfilled by the G,M,S device in Figure 10, Figure 11, Figure 12, and Figure 14 may be split into multiple physically separated devices, so that the gateway G, system manager M, and security manager S each operate on a separate device.

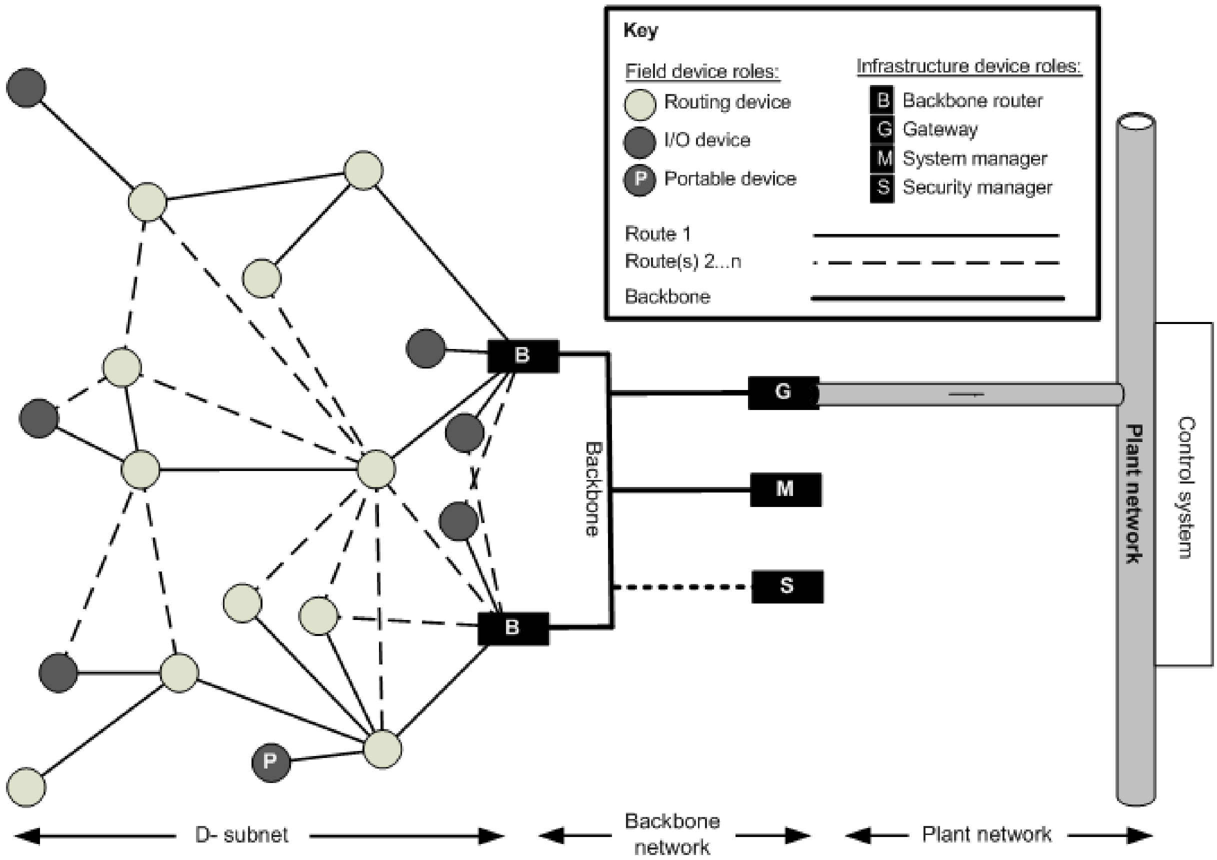


Figure 15 – Network with backbone – device roles

The physically separated gateway, system manager, and security manager shown in Figure 15 can be implemented only in networks with a network backbone.

5.4 Protocol suite structure

The protocol layers for a device conforming to this standard are described in terms of the OSI Basic Reference Model, which is adapted as shown in Figure 16. All roles and device types compliant with this standard can be derived from this model by extension or restriction of common elements depicted in Figure 16.

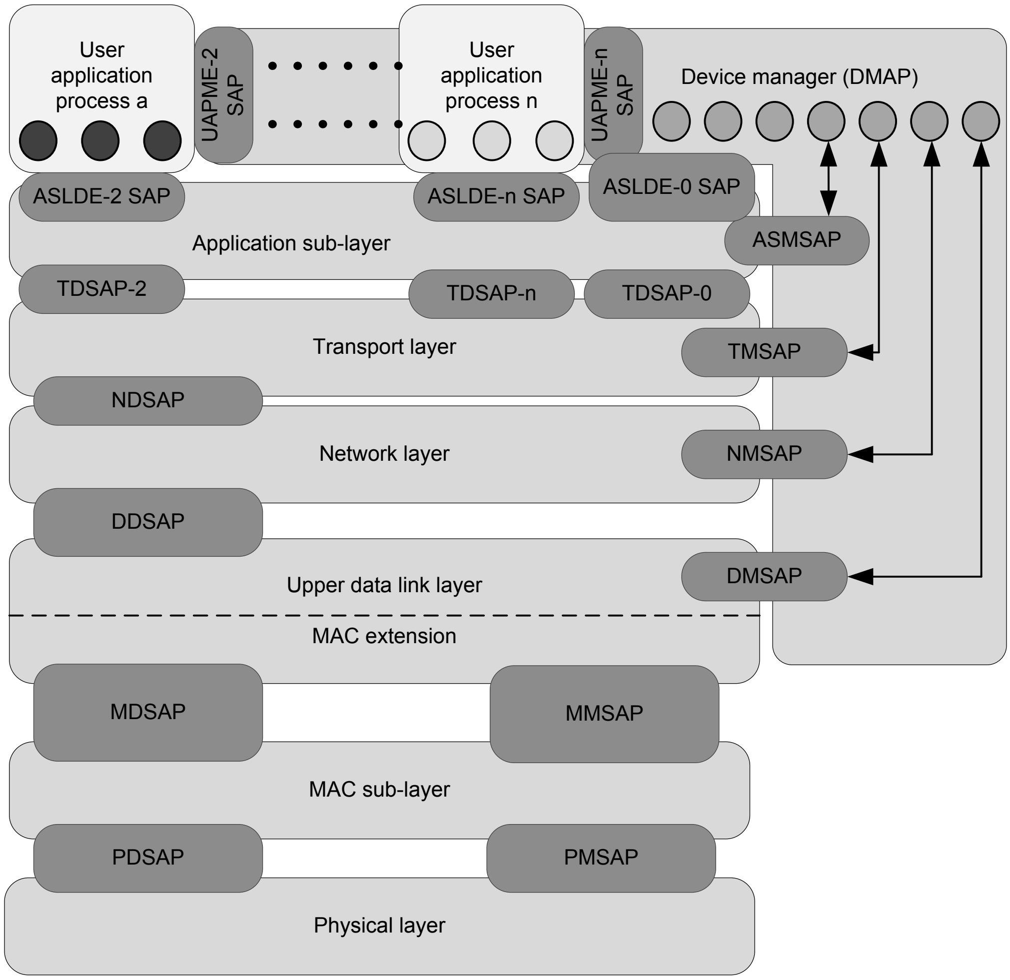


Figure 16 – Reference model used by this standard

As shown in Figure 16, each layer provides a service access point (SAP). The services of a layer are defined as the functions and capabilities of that layer that are exposed through the SAP to the surrounding layers. In general two types of SAPs are defined: data SAPs, which are used for operational data transfer, and management SAPs, which are used for layer management. The services provided by a layer are defined by the data flowing through the data SAPs and, in some cases, the states that a layer provides and the state transitions that are driven by the interaction across those SAP. The device manager is the entity within each device that performs the management function; in most cases it is accessed via a layer management SAP. The device manager has a dedicated path to several of the lower protocol layers within a device, to provide direct real-time control over the operation of those layers as well as direct access to diagnostics and status information.

All devices compliant with this standard are considered managed devices. All devices shall provide the functionality of each management SAP used by the DMAP for every protocol layer that they implement, as shown in Figure 16.

Since compliance can be assessed only at external interfaces, including the content of data structures conveyed at those interfaces, all notional descriptions of how specific functionality could be implemented is necessarily only informative, not normative.

5.5 Data flow

5.5.1 General

The descriptions in 5.5 are intended to provide examples of how data may flow through the system. The set of examples is not intended to be exhaustive.

5.5.2 Native communications

A device communicates over the network using only ASL defined services as defined in Clause 12; the payloads are classified as either native or non-native. Native payloads are defined in Clause 12; non-native payloads are not defined within this standard.

5.5.3 Basic data flow

Figure 17 illustrates the steady-state data flow for a basic network compliant with this standard, such as the one shown in Figure 10.

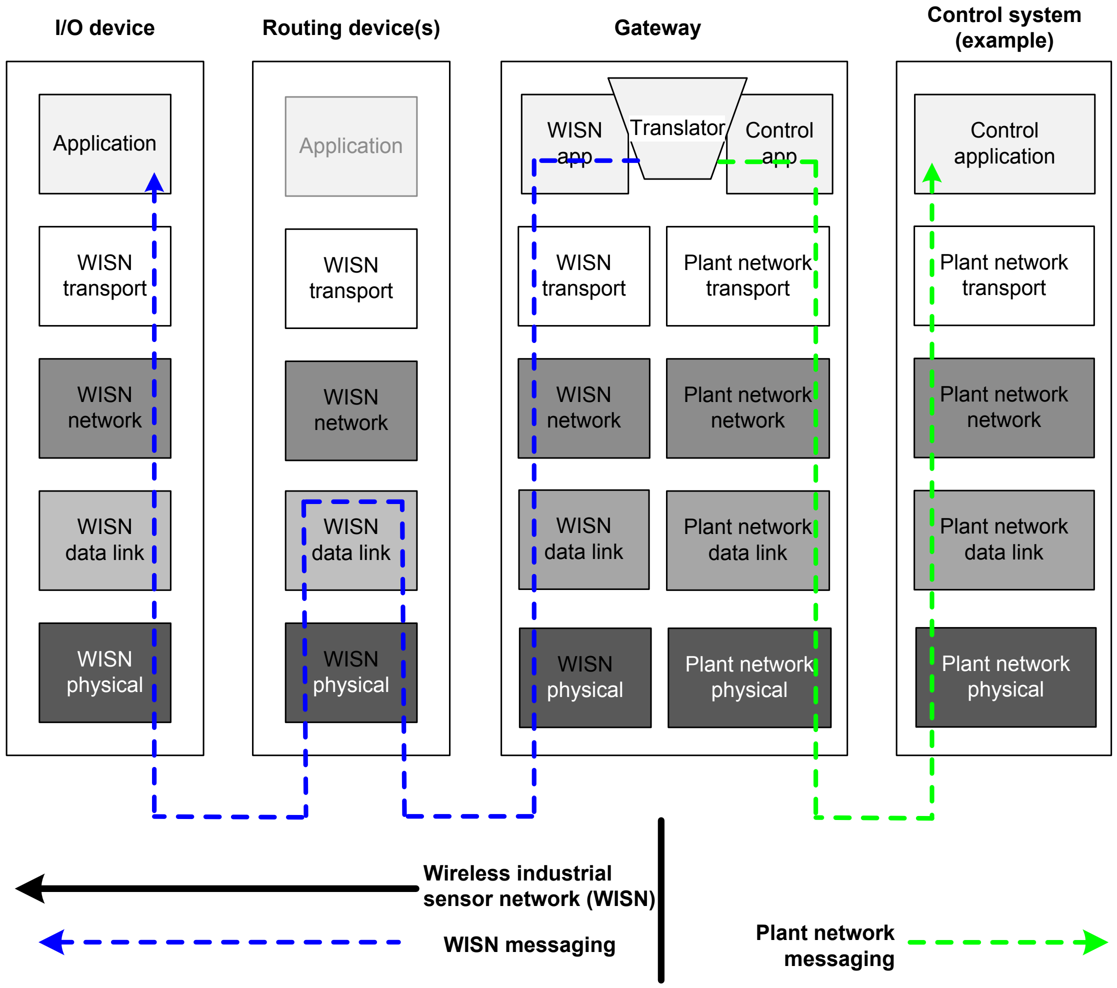


Figure 17 – Basic data flow

The I/O device is a sensor or actuator device within the D-subnet that contains physical, data-link, network, and transport layers as defined by this standard and runs an application that handles the sensor or actuator function.

The router routes messages on behalf of the I/O device. Routing within the D-subnet is performed entirely within the DL, and not within the NL (see Clause 9). In a real-world network, there will be one router for each additional hop between the device and the WISN-connected gateway or backbone router.

The gateway translates messages between the D-subnet and the plant network. The application running on the gateway consists of a component that communicates with the ALE of the I/O device, plus a component that communicates with an ALE within the control system, plus any components that facilitate translation between the two, such as a cache.

5.5.4 Data flow between I/O devices

Figure 18 illustrates the data flow for communication between I/O devices within a D-subnet. Routing within the D-subnet is performed entirely within the DL, and not within the NL.

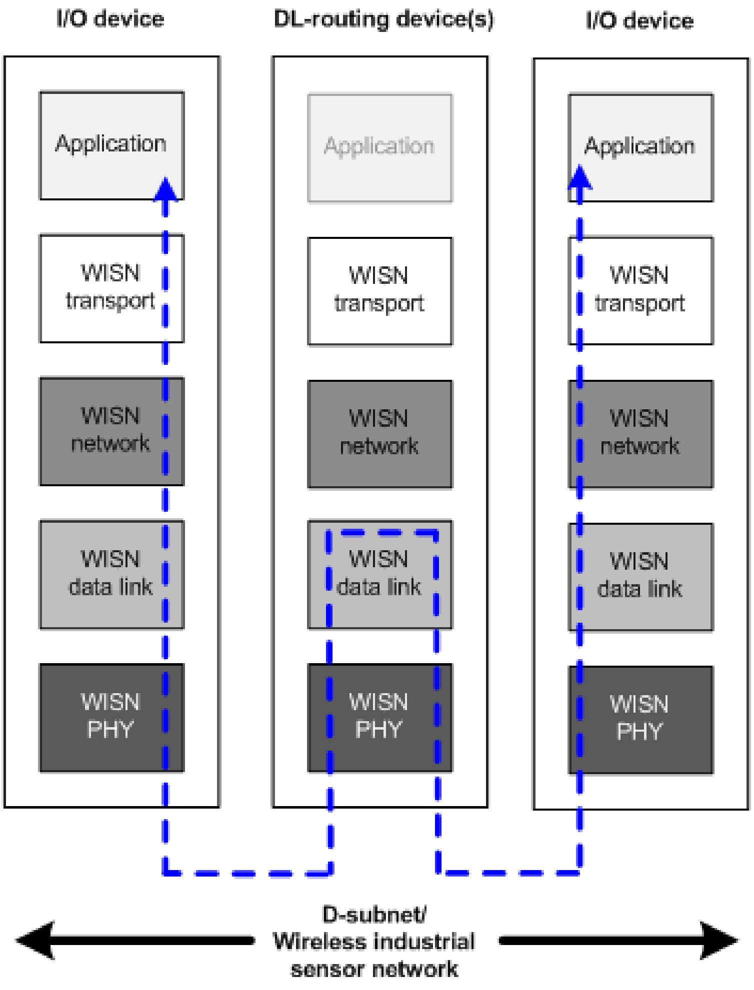


Figure 18 – Data flow between I/O devices

5.5.5 Data flow with legacy I/O device

Figure 19 illustrates a legacy I/O device that is integrated into a D-subnet via a legacy device adapter. An adapter is a subset of the gateway role and is a device that converts the protocol in the legacy device to that of a network compliant with this standard.

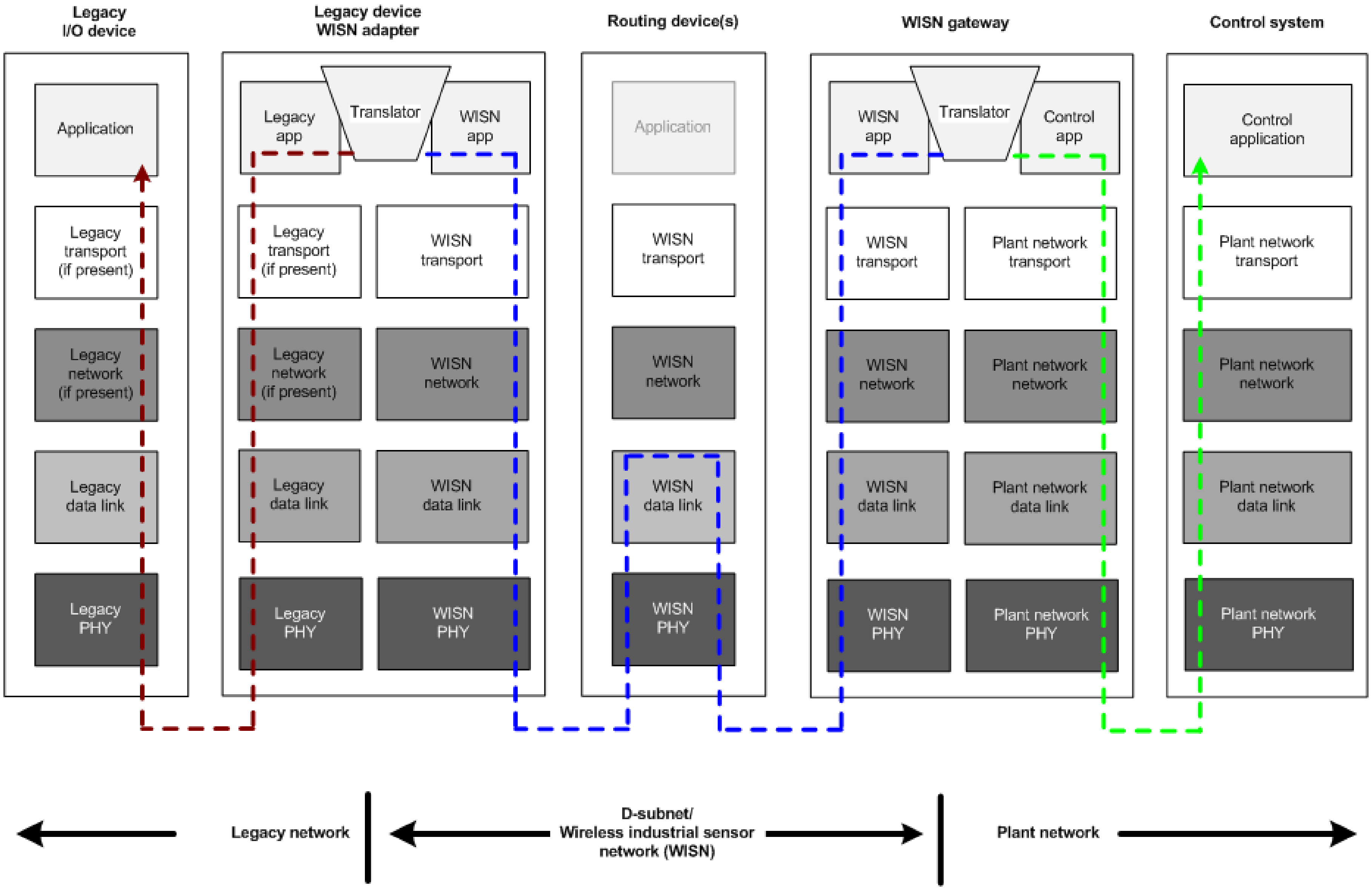


Figure 19 – Data flow with legacy I/O device

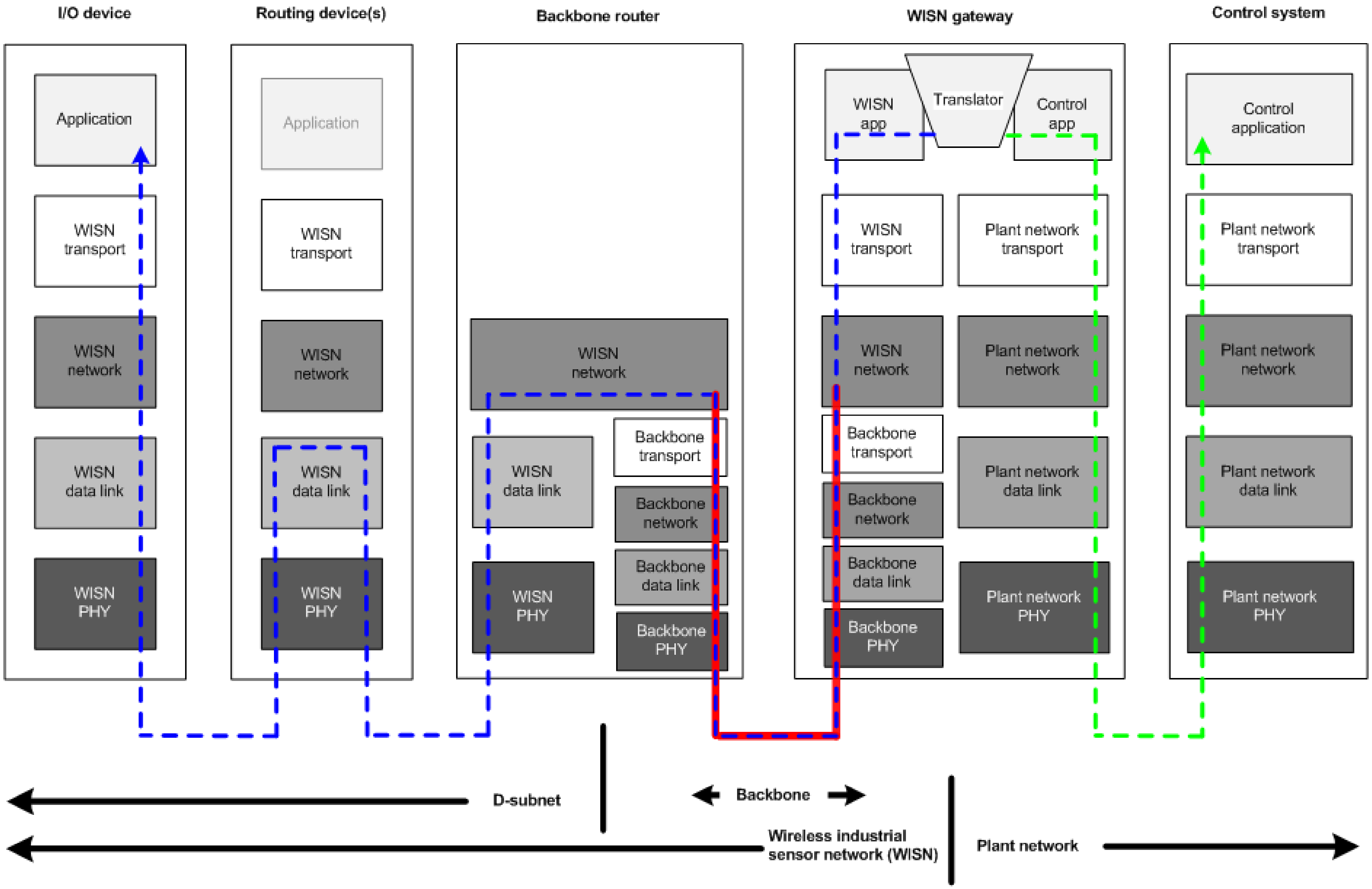


Figure 20 – Data flow with backbone-resident device

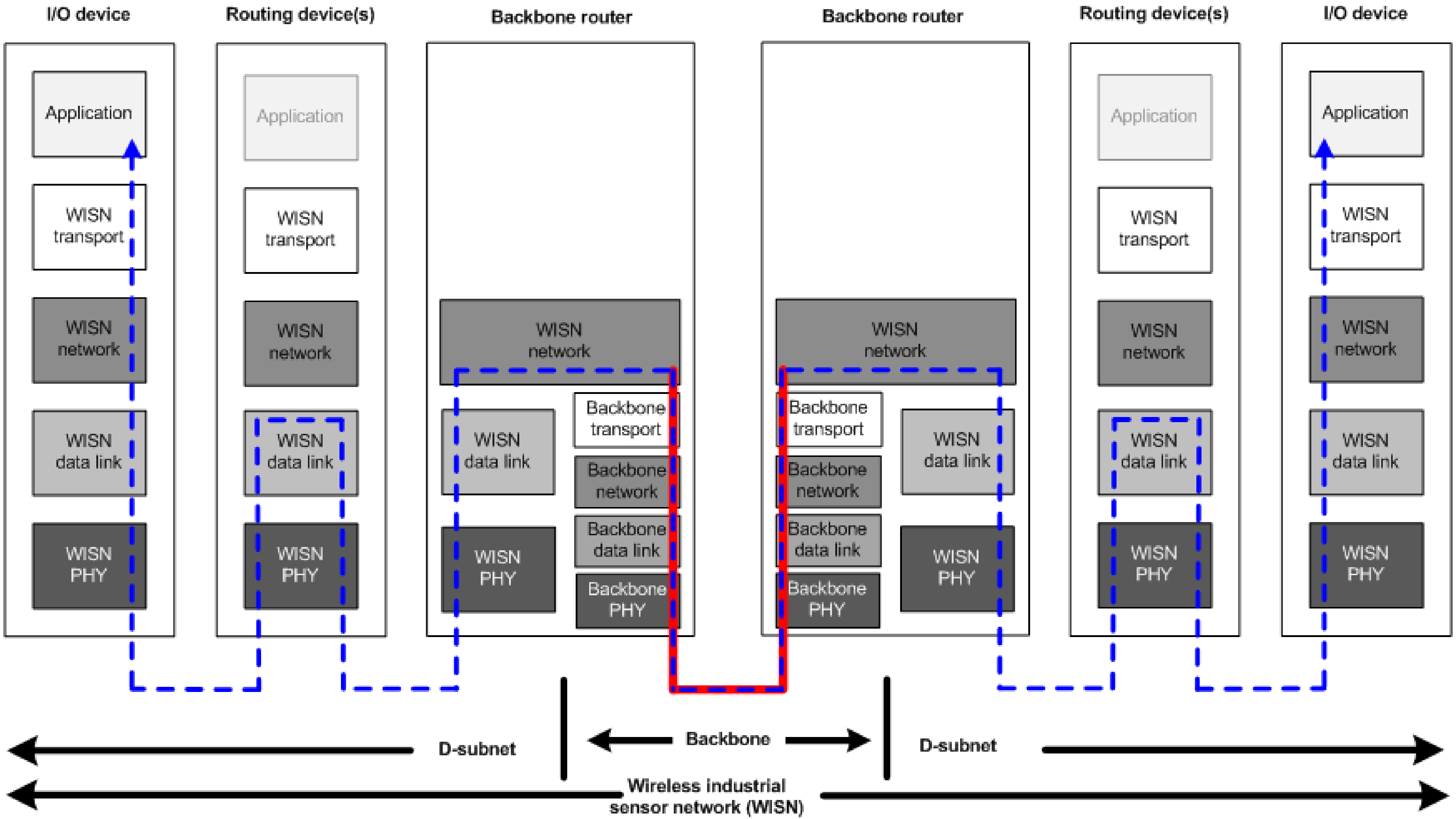


Figure 21 – Data flow between I/O devices via backbone subnet

5.5.6 Data flow with backbone

Figure 20 introduces a backbone router into the data flow.

The backbone router encapsulates NPDUs and relays them through backbone physical, data-link, network, and transport layers. The gateway uses the same backbone layers to recover the NPDUs. While it is not shown in Figure 20, the gateway may include both a PhLE and a DLE as defined by this standard, enabling the gateway to handle messages directly from a D-subnet, in addition to messages relayed through a backbone interface.

5.5.7 Data flow between I/O devices via backbone

Figure 21 illustrates how a backbone network handles standard-compliant message transfer when an I/O device communicates directly with another I/O device in a different D-subnet.

5.5.8 Data flow to a standard-aware control system or device

A standard-aware control system is a control system that understands messaging defined by this standard and does not need a gateway to perform protocol translation. Figure 22 illustrates data flow to a standard-aware control system.

NOTE Generic protocol translation is addressed in Annex O.

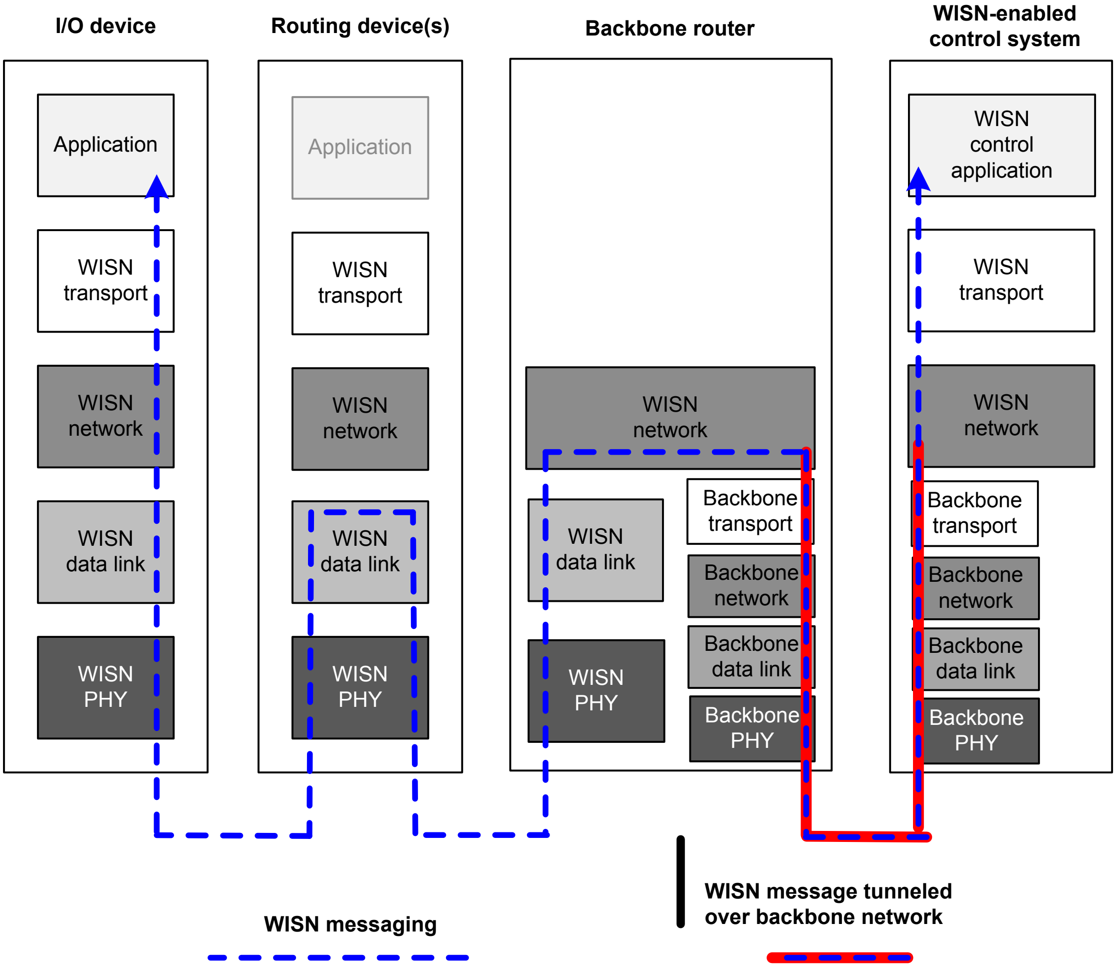


Figure 22 – Data flow to standard-aware control system

In general, for a device to be standard-aware, it needs only to support the application interface defined by this standard and to implement the application, transport, and network layers defined by this standard. This makes it possible for two standard-aware devices to communicate via a plant network without using or requiring any sublayers.

5.6 Time reference

5.6.1 General

This standard's time is based on international atomic time (TAI) as the time reference; see 6.3.10. This standard's time is reported as elapsed seconds since the TAI instant of 00:00 on 1 January 1958 (i.e., 1958/01/01 00:00).

It is not possible or even desirable for every network to track an atomic clock precisely. Rather, every network shall have a sense of time that is:

- monotonically increasing at a rate that closely matches real time;
- not to exceed an error of more than 1 s relative to the system time source; and
- delivered to various layers in field devices in consistent TAI units.

There are communication modes as defined in Clause 9 that require better than 1 s clock accuracy relative to the system time source.

For protocol operation, sequence of event reporting, and other purposes, time usually needs to be divided into increments of less than one second. For example, increments may be represented as:

WISN clock ticks: Two octets to mark time in increments of 2^{-15} s (32 768 Hz, or ~30,52 μs per tick).

Microsecond precision: Three octets to mark time in increments of 2^{-20} s (~0,95 μs per increment).

Nanosecond precision: Four octets to mark time in increments of 2^{-30} s (~0,93 ns per increment).

Devices needing to convert TAI time to hh:mm:ss format, such as on a user display, may account for a coordinated universal time (UTC) accumulated-leap-second adjustment. This adjustment is available to field devices from the system manager. If the UTC adjustment is used by a field device, it should refresh the adjustment at the start of each month.

NOTE A list of such UTC adjustments is maintained at ftp://maia.usno.navy.mil/ser7/tai-utc.dat .

Simultaneous UTC update requests by many devices may cause a storm of activity in the DL. This should be considered in the DMAP design; its avoidance is not covered by the current DL specification.

All devices in a network share the TAI time reference with varying degrees of accuracy. Each device within a network shall maintain time accurately to within 1 s.

The system manager directs devices on the system to a device implementing the role of system time source. In most cases, this device will also be filling the system manager role. However, the time-source responsibility can be redirected to any device with a more capable source of time.

The gateway shall be responsible for converting between nominal network TAI time and an external non-TAI time reference if one is being used.

For more information on the requirements for the time source, see 6.3.10.

5.6.2 Time synchronization

To propagate host time, a gateway may periodically synchronize the time sense in an attached D-subnet to an external time source by requesting time changes via DLMOs.

The WISN provides time synchronization for applications so that, at the device level, they may use time to coordinate activities or to time-stamp data, an activity that could improve energy use and enhance reliability. System time shall be available from at least one device (a system time source) on each WISN. See Clause 9.

5.7 Firmware upgrades

The overall system, and each device on the WISN, shall provide the capability of upgrading device firmware that implements this standard via the wireless network (see 6.3.6). The system shall support a common mechanism, such as a time-based trigger, to inform all devices to switch concurrently to the new firmware; this mechanism may be used to minimize the number of devices that are left stranded with an incompatible network protocol suite. The security mechanisms built into this standard are used during a firmware upgrade.

Each version of the protocol shall support previous versions to the extent necessary to support upgrading firmware via the wireless network.

5.8 Wireless backbones and other infrastructures

Devices compliant with this standard are managed devices. All devices compliant with this standard shall implement the device management interfaces at each layer, but they may implement only the functionality of their required functional layers.

The system supports both wired and wireless backbone networks through the use of backbone routers. The operation of backbone networks is not addressed by this standard.

More information on backbone networks and their implied characteristics can be found in Annex E.

6 System management role

6.1 General

6.1.1 Overview

The system management role supports network management of the network as a whole, as well as device management of the devices operating within the network. Network management includes management of the various communications resources across the network and across all protocol layers of the architecture. Device management supports localized management of the communications resources, and potentially other resources, of a device.

The management functions described by this standard support:

- joining the network and leaving the network;
- reporting of faults that occur within the network;
- communication configuration;
- configuration of clock distribution and the setting of system time;
- device monitoring;
- performance monitoring and optimization;
- security configuration and monitoring.

6.1.2 Components and architecture

The primary components of the management service include a device management application process (DMAP) that resides on every device compliant with this standard, as well as a system management application process (SMAP) that shall reside on a device that implements the system manager role. Roles are described in 5.2.6.5. The DMAP is a special type of user application process (UAP) that is dedicated to managing the device and its communications services, described in 12.4.3 and 12.4.4. The DMAP and the system manager shall be capable of communicating with each other over the network using the standard-defined application sublayer services, and shall together provide a means to access management information remotely and to manage the system and its devices. System management is accomplished via inter-device messaging, while device management is accomplished by local intra-device communications.

The management architecture of this standard is shown in Figure 23.

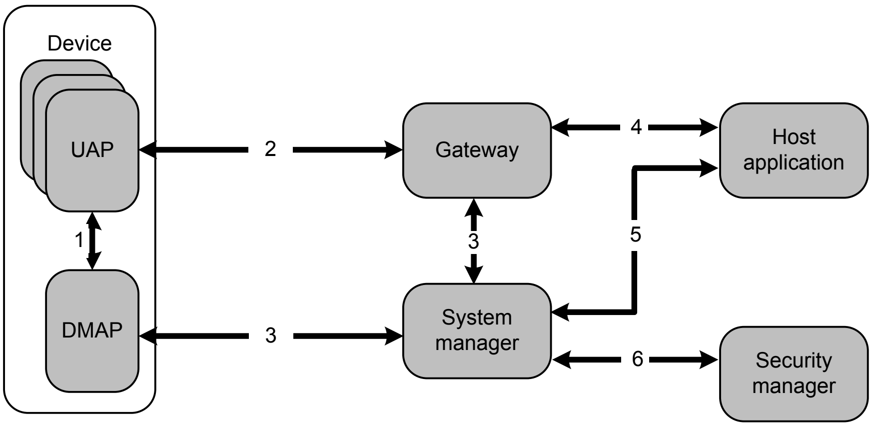


Figure 23 – Management architecture

Devices compliant with this standard shall be managed through two distinct classes of application processes, UAPs and the DMAP. The UAPs are configured and monitored by host applications, such as automated management systems, or by host proxy applications in the gateways. The DMAP in the device shall be managed by the system manager.

Figure 23 shows the management model relationships of this standard. For the paths illustrated in Figure 23, this standard provides a normative description of the communication protocols for paths 2 and 3. Communication protocols for paths 1, 4, 5 and 6 are informative examples of implementations in this standard.

This standard defines the system management communication protocols that shall be used to control and monitor the DMAPs in the network and the relevant communication paths. In this case these communications travel on path 3 in Figure 23.

The system manager includes communication paths outside of the standard-compliant network that allow other devices to interact with it. In Figure 23, path 5 shows a connection between the system manager and the host application. This path enables the host application to retrieve network status and request network services. The system manager also communicates with the security manager over path 6 to configure security in the network and to report status.

The user applications on devices compliant with this standard communicate with gateways and host applications using the standard protocols shown over path 2 in Figure 23. This is described in Clause 12 and Annex U.

The plant-based host application communicates with the gateway using plant protocols that travel over path 4. The system manager does not communicate directly with the UAPs. There is an intra-device communication path that enables the DMAP and UAP processes to interact via the intra-device communication path 1 in Figure 23 between the system manager and a gateway. This is performed over a virtual interface 1 in Figure 23, UAPME-SAP, using the UAP management object (UAPMO) which is described in 12.15.2.2.

6.1.3 Management functions

Every network that is compliant with this standard shall include at least one system management role and one security management role. These roles shall be accessible to all standard complaint devices on this network.

System management is a specialized role that governs the network, the operation of devices on the network, and network communications. The functions defined within this role are performed by the system manager, providing policy-based control of the runtime communication configuration. The system manager monitors and reports on communication configuration, performance, fault conditions, and operational status. This is described in 6.3.7. The system manager also provides time-related services. Some system management functions may be completely automated, while others may be human-assisted.

The system manager supports configuration of the standard-compliant network, including attributes of the protocol suite from DL to AL for system management applications. It manages the establishment, modification and termination of contracts that are used by devices compliant with this standard to communicate with each other. The functions of the system manager do not include the control, configuration, and monitoring of the UAPs on the device. These management functions are controlled by host applications on plant networks or in handheld maintenance tools.

Security management of the system is a specialized function that is realized in one entity and that works in conjunction with the system management function to enable secure system operation. This function is performed by the security manager. Some system security management functions may be completely automated, while others may be human-assisted.

Every device compliant with this standard shall contain a DMAP. The DMAP includes a local device security management function. The DMAP cooperates with the system manager and the security manager to enable the usage of system resources by the device and the secure management of the resources of a device. For example, the DMAP may ask to join the network, ask for communication bandwidth, request a communication configuration, and report its health. The system manager and the security manager authorize the device to join the network, allocate communication bandwidth, configure the device, and collect health reports. These health reports are stored in the system manager and are used to make communication configuration decisions.

In order to compartmentalize security functions, the management architecture defined by this standard supports separable system management and security management functions at both the system and device levels. Thus, the security manager is logically separable from the system manager. More details about security manager are provided in Clause 7.

NOTE The system management and security management functions often are included within a single physical entity.

6.2 DMAP

6.2.1 General

The DMAP is a special type of application process dedicated to managing the standard-compliant device and its communications services. A DMAP resides on every device compliant with this standard.

6.2.2 Architecture of device management

As shown in Figure 16, the protocol suite structure of a device includes the networking protocol layers and the UAPs.

The DMAP is shown in relation to the other protocol suite components on the right side of the protocol suite structure, including arrows depicting access to the management SAPs for several of the protocol layers. The components within the DMAP are modeled as objects, known as management objects, which have features that are accessible over the network. The DMAP, like all application processes, is able to use the application sublayer to communicate. The DMAP shall use the application-sublayer SAP ASLDE-0 SAP for normal data communications. This application sublayer SAP shall correspond to TL SAP TDSAP-0 which shall correspond to port number 0xF0B0. The application sublayer provides communication services to enable the objects within the DMAP to interact with the system manager over the network. These communication services are described in 12.17.

6.2.3 Definition of management objects

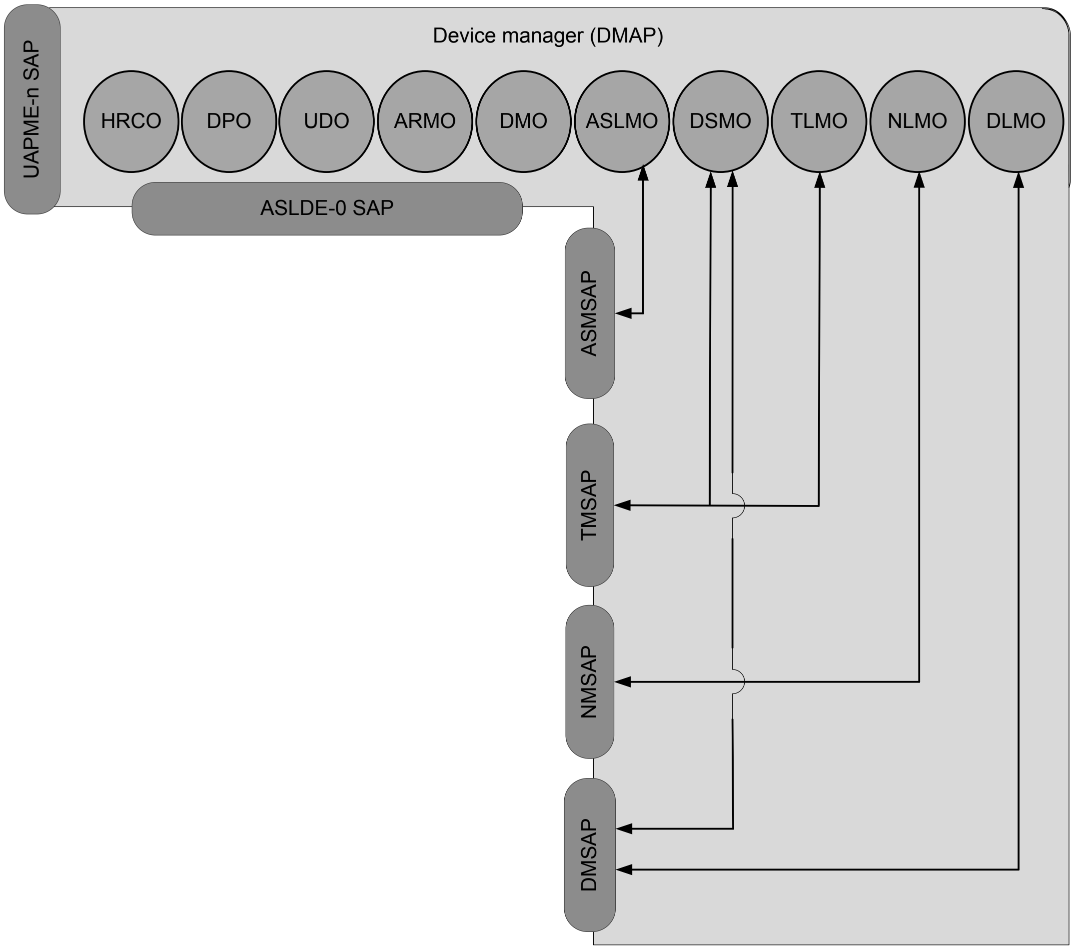
The objects defined in the DMAP follow the specification that is used to define UAP objects. The templates for defining object types, object attributes, object methods and object alerts are specified in Annex I.

The management objects are extensible by device manufacturers and network protocol suite/device developers. This is described in 12.5. Attribute and method identification space is set aside for manufacturer-defined device-specific objects. The system manager shall not be required to implement support for proprietary extensions for that device to interoperate and perform its primary function.

6.2.4 Management objects in DMAP

The DMAP shall contain a number of management objects that support device management operations. These objects shall collectively perform two types of device management functions. First, these objects shall manage the device locally by manipulating attributes and invoking methods on layer management SAPs. Second, the management objects shall be accessible remotely using the ASL services such that a system manager may manipulate attributes and invoke methods on the device management objects or capture alerts from the objects. These objects are conceptual in that there are no object-oriented implementation requirements in the device, except that the externally visible behavior in terms of over-the-air ASL messaging shall be consistent with the model of object communications having the specified attributes, methods, and alerts.

As shown in Figure 24, the DMAP shall include a set of layer management objects, a device management object, a device security management object, an alert reporting management object, an upload/download object, and other management objects.



2786

2787 **Figure 24 – DMAP**

2788 The standard management objects defined in this standard are given in Table 1.

Table 1 – Standard management object types in DMAP

Standard object type name	Standard object type identifier	Standard object identifier	Object description
Device management object (DMO)	127	1	This object facilitates the management of the general device-wide functions of the device; see 6.2.7.1
Alert reporting management object (ARMO)	126	2	This object facilitates the management of the alert reporting functions of the device; see 6.2.7.2
Device security management object (DSMO)	125	3	This object facilitates the management of the security functions of the device; see 6.2.7.5
DL management object (DLMO)	124	4	This object facilitates the management of a device DLE; see 6.2.8.2.2
NL management object (NLMO)	123	5	This object facilitates the management of a device NLE; see 6.2.8.2.2.5
TL management object (TLMO)	122	6	This object facilitates the management of a device TLE; see 6.2.8.2.2.6
Application sublayer management object (ASLMO)	121	7	This object facilitates the management of the device ALE; see 6.2.8.2.2.8
Upload/download object (UDO)	3	8	This object facilitates the management of the upload/download functions of the device; see 6.2.7.3
Device provisioning object (DPO)	120	9	This object facilitates the provisioning of the device before it joins a D-subnet; see 6.2.7.6
Health reports concentrator object (HRCO)	128	10	This object facilitates the periodic publication of device health reports to the system manager; see 6.2.7.7
Reserved for future editions of this standard	119..114	—	—

6.2.5 Communications services provided to device management objects

The application level services provided to the DMAP objects are the same as those provided by the application sublayer to UAP objects. These services include client/server (C/S), publish/subscribe (publish/subscribe), source/sink (source/sink), and alert reporting (AR). Details of these services, provided by the application sublayer, are given in 12.17.

As shown in Figure 25, TDSAP-0, which corresponds to port number 0xF0B0, shall be used for accessing the management objects in the DMAP, which in turn access the layer management attributes through the layer management SAP.

Access to the device management objects is protected by the TL security mechanisms described in 11.3.

Access to the DMAP objects is restricted to the SMAP with the following exceptions:

- The ARMO can also be accessed by alert masters that receive alerts originating in the device. See 6.2.7.2.3.
- A joining device is allowed to access the device management object methods used during the join process. See 6.3.9.2.2.

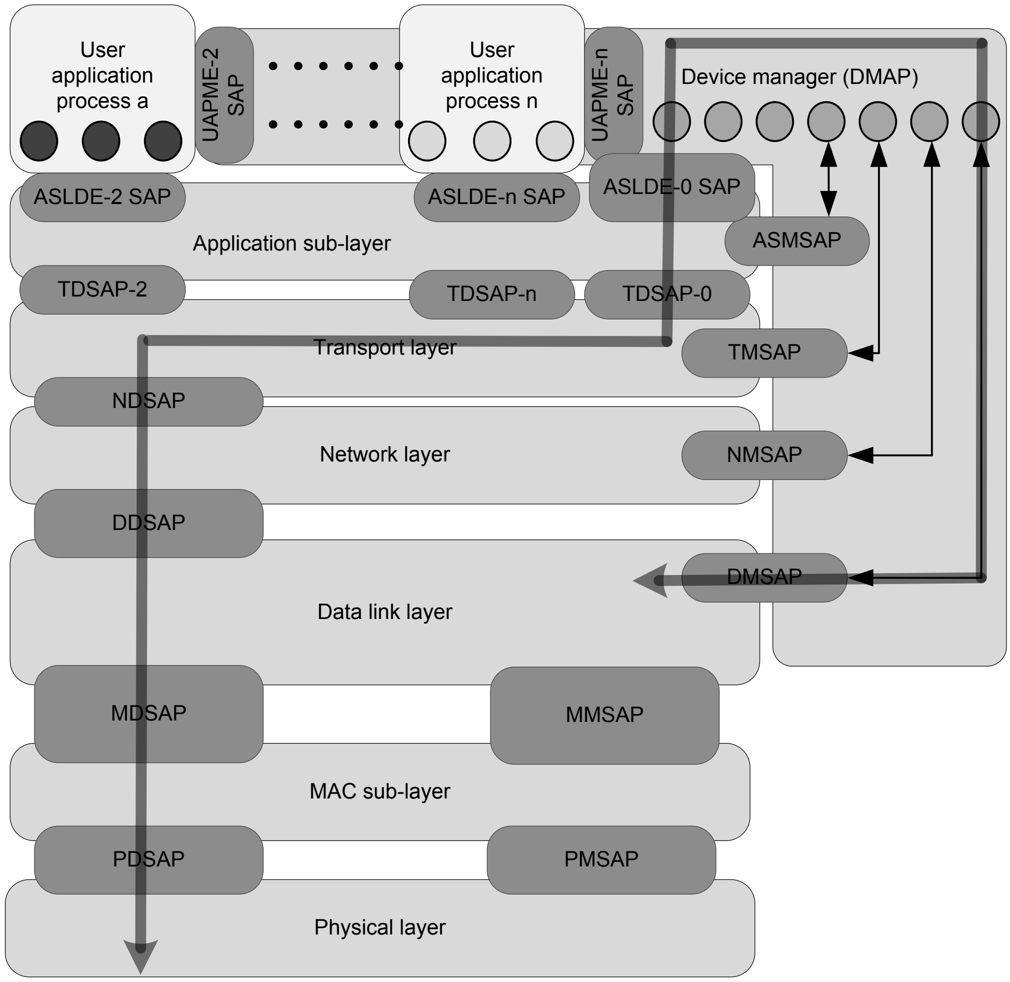


Figure 25 – Example of management SAP flow through standard protocol suite

Client/server interactions (including reading and writing of attributes, executing of methods, joining, and requesting and providing contracts) are the primary tools used for system management. In addition, the DMAP may use the alert reporting services of the ASL to report to the system manager when certain management-related conditions are detected. Designers of management objects may use alerts at various priority levels to help accomplish system and device management functions.

6.2.6 Attributes of management objects

6.2.6.1 General

The layer management SAPs shown in Figure 25 provide access to the management information in the various layers of the protocol suite.

This information is represented by attributes defined in the management objects of the DMAP, which can be monitored and operated on by the system manager. Details of the management objects are given in 6.2.3. The attributes in the layer management objects are used to configure the protocol layers and to monitor their status. The template for describing the attributes in all management objects is provided in Annex I, for use in proprietary extensions and future editions of this standard.

Attributes shall have a data type that is either a standard-defined scalar type or a standard-defined data structure. More details about attributes are given in 12.6.2.

A structured attribute is a special type of attribute that has a data type consisting of an array of standard-defined data structures. The array model is used to permit object access through indexing, where the index is the key attribute for access to the object.

Management information that needs to be visualized as a collection of one or more tables is modeled as structured attributes defined in the management objects.

Attributes defined in management objects can be accessed using the standard ASL-provided read or write services. Such operations enable configuration of each layer and monitoring of its status. They can be used to retrieve, set / modify, and reset the values of attributes. Operations on attributes are described in Annex J.

6.2.6.2 Structured attribute index field

Since structured attributes are described as arrays of data structures, one or more index fields for such arrays need to be indicated in the definition of each such structured attribute. This is done by including an * (asterisk) after the element name(s) in the table describing the data structure. The template for defining a data structure is given in Annex I.

6.2.6.3 Metadata of structured attribute

Structured attributes represent information tables. To provide external access to the number of objects in, and capacity of, any such table, additional meta-attributes that contain such information are defined for management objects. Such attributes represent the metadata of the corresponding structured attributes.

The standard data type for a metadata attribute is given in Table 2.

Table 2 – Metadata_attribute data structure

Standard data type name: Metadata_attribute		
Standard data type code: 406		
Element name	**Element identifier**	**Element type**
Count (number of indexed rows currently in the attribute)	1	Type: Unsigned16 Classification: Static Accessibility: Read only
Capacity (number of rows that the attribute can hold)	2	Type: Unsigned16 Classification: Static Accessibility: Read only

6.2.7 Definitions of management objects in DMAP

6.2.7.1 Device management object

As shown in Figure 24, the DMAP includes a set of management objects. The device management object (DMO) in the DMAP shall provide access to attributes having device-wide scope. Attributes of the DMO shall include the primary EUI64Address of the DLE, a vendor ID, a serial number, identification of the current revision of the communications software, and the device’s power source class. More details about DMO are provided in 6.2.8.1.

6.2.7.2 Alert reporting management object

6.2.7.2.1 General

The alert reporting management object (ARMO) is used to manage all the alert reports of the device. **Alert** is the term used to describe the action of reporting an event condition or an alarm condition. **Event** is the term for a transient (i.e., stateless) condition, used to report when something happened. **Alarm** is the term used for a condition that maintains state until the condition clears, which is reported on change of state. Alerts, including events and alarms, are envisioned to be of high utility for managing a network compliant with this standard.

There shall be at most one ARMO per device. Both alarms and events shall be reported through the ARMO. When an alert is triggered, it indicates a significant situation that needs to be reported. The ARMO shall encapsulate the report, handle timeouts and retries, and throttle alert reporting from the device.

The ARMO functions as an alert proxy for the objects present in the device. All alerts generated by any object present in a device shall be sent only by the ARMO which is a management object that is part of the DMAP. The Alert data APDU shall indicate in its APDU header that the originator of the communication is the ARMO object and the DMAP of the device reporting the alert. The object and UAP that originated the actual alert APDU shall be identified in the content of the Alert report rather than in the APDU headers.

Each alert shall be acknowledged by the device receiving the alert. Each alert acknowledgment shall be addressed to the ARMO of the device that originated the alert. Alerts are reported promptly and time-stamped accurately using queued alert reporting. Queued alert reporting involves the alert detecting device reporting the condition using an source/sink communication flow and receiving an ACK/NAK DPDU in return.

NOTE The intent of specifying the ARMO in this standard is to separate alert detection from the management of reporting the alert condition. Unlike some wire-oriented legacy protocols, this standard consolidates alerts locally in order to minimize externalized messaging and energy consumption.

The alert model used in this standard is described in 12.8.

The interfaces between the ARMO and all other objects, both in UAPs and in the DMAP, are device internal and are not specified in this standard.

6.2.7.2.2 Alert types

Alert classes, alert directions, and alert priorities are defined in 12.11. The alert category indicates whether the alert is a device diagnostic alert, a communication diagnostic alert, a security alert, or a process alert. The alert type provides additional information regarding the alert, specific to the alert category and specific to the application object generating the alert.

Table 3 provides the alert types for the alert categories of communication diagnostic alert category. Table 4 provides the alert types for the security alert category. Table 5 provides the alert types for the device diagnostic alert category. Table 6 provides the alert types for the process alert category.

Table 3 – Alert types for communication diagnostic category

Alert type	Alert category: Communication diagnostic					
	ARMO	ASLMO	DLMO	NLMO	TLMO	DMO
0	Alarm_Recovery_Start; see Table 8	MalformedAPDUCommunicationAlert; see 12.19.5	DL_Connectivity; see 9.6.1	NL Dropped PDU; see 10.4.3	IllegalUseOfPort; see 11.6.2.5.4	Device_Power_Status_Check; see 6.2.8.1.2
1	Alarm_Recovery_End; see Table 8	—	Neighbor Discovery; see 9.6.2	—	TPDUonUnregisteredPort; see 11.6.2.5.4	Device_Restart; see 6.2.8.1.2
2	—	—	—	—	TPDUoutOdSecurityPolicies; see 11.6.2.5.4	—

Table 4 – Alert types for security alert category

Alert type	Alert category: Security		
	ARMO	DSMO	DPO
0	Alarm_Recovery_Start; see Table 8	Security_MPDU_Fail_Rate_Exceeded; see 7.11.4	Not_On_Whitelist_Alert; see Table 374
1	Alarm_Recovery_End; see Table 8	Security_TPDU_Fail_Rate_Exceeded; see 7.11.4	Inadequate_Join_Capability_Alert; see Table 374
2	—	Security_Key_Update_Fail_Rate_Exceeded; see 7.11.4	—

Table 5 – Alert types for device diagnostic alert category

Alert type	Alert category: Device diagnostic
	ARMO
0	Alarm_Recovery_Start; see Table 8
1	Alarm_Recovery_End; see Table 8

Table 6 – Alert types for process alert category

Alert type	Alert category: Process				
	ARMO	AI	AO	BI	BO
0	Alarm_Recovery_Start; see Table 8	See 12.19.7	See 12.19.7	See 12.19.7	See 12.19.7
1	Alarm_Recovery_End; see Table 8	—	—	—	—

6.2.7.2.3 Alert master

Alerts shall be sent to alert-receiving objects. Alert-receiving objects are defined in 12.15.2.3. Each alert category may have a different alert-receiving object residing in a different device. Devices that receive these alerts are known as alert masters.

DMAP access is often restricted to the SMAP present in the system manager. In an exception to this general principle, alert masters are allowed to access the ARMO object present in the DMAP. DMAP access by alert masters shall be limited to the ARMO, unless the alert master uses the DMAP-SMAP session established when the device joined the network. The alert masters to which the device is configured to send alerts are listed in Table 7.

6.2.7.2.4 Alert queue

Alerts belonging to each category are assumed to be placed into an internal queue provided per-category in the device. Both types of alerts, events (stateless) and alarms (stateful) will be placed in the same queue, filtered by category. The queue is necessary to provide a guaranteed delivery of alerts to the alert master. Every alert to be reported to the alert master is placed into this reporting queue.

The size of the queue should be big enough to accommodate all events as well as all possible alarm conditions simultaneously in order to support alarm recovery without losing any alarms.

Although placed in the same queue, events and alarms will be prioritized differently. The device shall report an event with higher priority before an event with lower priority. For alarms, the queue is emptied sequentially; the oldest alarm is reported first. When the queue is full and a new alarm is submitted, the oldest alarm is dropped from the queue regardless of its reporting state.

6.2.7.2.5 Alert state models

The state tables and transitions for alarms and events are given in 12.9 and 12.10.

6.2.7.2.6 Alarm recovery

It is often useful to be able to recover all alarms currently active within a device. The need for alarm recovery arises whenever a connection to device is lost for a period of time or whenever an alert master commands an alarm recovery.

Alarm recovery consists of the following set of activities:

- The alert master commands an alarm recovery by using the Alarm_Recovery method of the ARMO. This method is described in Table 9.
- The ARMO sends a recovery start alert to the alert master, which indicates that the ARMO has received a command to recover alarms and that active alarms will follow.

  NOTE The process for re-sending these active alarms within a device is not specified.
- The ARMO sends a recovery end alert to the alert master.

The ARMO is responsible for generating alarm recovery start and end alerts and for coordinating the alarm recovery process with the application objects residing within the device.

6.2.7.2.7 Alert reporting management object attributes, alerts and methods

The attributes of the ARMO are defined in Table **7 – ARMO attributes**

Table 7 – ARMO attributes

Standard object type name: Alert reporting management object (ARMO)				
Standard object type identifier: 126				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Alert_Master_Device_ Diagnostics	1	Alert master for alerts that belong to the device diagnostics category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Valid range: See 12.16.3.5	Typically set to a gateway for the device's information, but can be changed to any other standard-compliant device with a valid IPv6Address (1)
Confirmation_Timeout_ Device_Diagnostics	2	Timeout waiting for acknowledgment of a device diagnostic alarm that was sent to the alert master	Type: Integer16 Classification: Static Accessibility: Read/write Default value: 10	Timeout independent of proximity to alert master. A value of $N > 0$ specifies a duration of N s, while $N < 0$ specifies a duration of $-1/N$ s. $N = 0$ is not permitted (2)
Alerts_Disable_Device_ Diagnostics	3	Command to disable / enable all device diagnostic alerts	Type: Boolean8 Classification: Static Accessibility: Read/write Default value: FALSE	FALSE = enable, TRUE = disable
Alert_Master_Comm_ Diagnostics	4	Alert master for alerts that belong to the communication diagnostics category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Valid range: See 12.16.3.5	Typically set to be the system manager for the device, but can be changed to any other standard-compliant device with a valid IPv6Address; the device shall set this to be its system manager after it joins the network; see 6.3.7.2
Confirmation_Timeout_ Comm_Diagnostics	5	Timeout waiting for acknowledgment of a communication diagnostic alarm that was sent to the alert master	Same as attribute 2	Same as attribute 2
Alerts_Disable_Comm_ Diagnostics	6	Command to disable / enable all communication diagnostic alerts	Type: Boolean8 Classification: Static Accessibility: Read/write Default value: FALSE	FALSE = enable, TRUE = disable

Table 7 (continued)

Standard object type name: Alert reporting management object (ARMO)				
Standard object type identifier: 126				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Alert_Master_Security	7	Alert master for alerts that belong to the security category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Valid range: See 12.16.3.5	Typically set to be the system/security manager for the device, but can be changed to any other standard-compliant device with a valid IPv6Address. The device shall set this to be its security manager after it joins the network; see 6.3.7.2
Confirmation_Timeout_Security	8	Timeout waiting for acknowledgment of a security alarm that was sent to the alert master	Same as attribute 2	Same as attribute 2
Alerts_Disable_Security	9	Command to disable / enable all security alerts	Type: Boolean8 Classification: Static Accessibility: Read/write Default value: FALSE	FALSE = enable, TRUE = disable
Alert_Master_Process	10	Alert master for alerts that belong to the process category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Valid range: See 12.16.3.5	Typically set to a gateway for the device's information, but can be changed to any other standard-compliant device with a valid IPv6Address [1)]
Confirmation_Timeout_Process	11	Timeout waiting for acknowledgment of a process alarm that was sent to the alert master	Same as attribute 2	Same as attribute 2
Alerts_Disable_Process	12	Command to disable / enable all process alerts	Type: Boolean8 Classification: Static Accessibility: Read/write Default value: FALSE	FALSE = enable, TRUE = disable

Table 7 (continued)

Standard object type name: Alert reporting management object (ARMO)				
Standard object type identifier: 126				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Comm_Diagnostics_Alarm_ Recovery_AlertDescriptor	13	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the comm. diagnostics category; these events can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 3] Valid range: See 12.16.3.7	—
Security_Alarm_Recovery_ AlertDescriptor	14	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the security category; these events can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 3] Valid range: See 12.16.3.7	—
Device_Diagnostics_Alarm_ Recovery_AlertDescriptor	15	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the device diagnostics category; these events can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 3] Valid range: See 12.16.3.7	—
Process_Alarm_Recovery_ AlertDescriptor	16	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the process category; these events can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 3] Valid range: See 12.16.3.7	—
Reserved for future editions of this standard	17..63	—	—	—
NOTE 1 This information is expected to be configured by the host application after the device joins the network. NOTE 2 All alarms require acknowledgement.				

The alerts of the ARMO are defined in Table 8.

Table 8 – ARMO alerts

Standard object type name(s): Alert reporting management object (ARMO)				
Standard object type identifier: 126				
Description of the alert: Alarm recovery begin and end events for alarms that belong to all categories				
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: urgent, high, med, low, journal)**	**Description of value included with alert**
0 = Event	1 = Comm. diagnostics	0 = Alarm_Recovery_Start	3 = Low	Generated by ARMO for the comm. diagnostics alert master indicating that the alarm recovery command has been received; all outstanding comm. diagnostic alarms are reported after this event is raised
0 = Event	1 = Comm. diagnostics	1 = Alarm_Recovery_End	3 = Low	Generated by ARMO for the comm. diagnostics alert master indicating that the alarm recovery process has ended
0 = Event	2 = Security	0 = Alarm_Recovery_Start	3 = Low	Generated by ARMO for the security alert master indicating that the alarm recovery command has been received; all outstanding security alarms are reported after this event is raised
0 = Event	2 = Security	1 = Alarm_Recovery_End	3 = Low	Generated by ARMO for the security alert master indicating that the alarm recovery process has ended
0 = Event	0 = Device diagnostics	0 = Alarm_Recovery_Start	3 = Low	Generated by ARMO for the device diagnostics alert master indicating that the alarm recovery command has been received; all outstanding device diagnostic alarms are reported after this event is raised
0 = Event	0 = Device diagnostics	1 = Alarm_Recovery_End	3 = Low	Generated by ARMO for the device diagnostics alert master indicating that the alarm recovery process has ended
0 = Event	3 = Process	0 = Alarm_Recovery_Start	3 = Low	Generated by ARMO for the process alert master indicating that the alarm recovery command has been received; all outstanding process alarms are reported after this event is raised
0 = Event	3 = Process	1 = Alarm_Recovery_End	3 = Low	Generated by ARMO for the process alert master indicating that the alarm recovery process has ended

The method of the ARMO used to recover alarms of the different categories shall be as defined in Table 9.

Table 9 – Alarm_Recovery method

Standard object type name(s): Alert reporting management object (ARMO)				
Standard object type identifier: 126				
Method name	**Method ID**	**Method description**		
Alarm_Recovery	1	Method to recover alarms that belong to the category mentioned in the input argument		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Alert_Category	Data type: Unsigned8	Named values: 0: device diagnostics 1: comm. diagnostics 2: security 3: process
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	—	—	—	—

6.2.7.3 Upload/download object

The attributes, methods and state machines of the UDO in the DMAP shall be as per the definition given in 12.15.2.4. The object identifier of the UDO in the DMAP shall be 8.

An upload/download object (UDO) is used for uploading or downloading large blocks of information to/from a device. The UDO may be used to support downloading a new version of communications firmware or data. The UDO maintains revision control information. The UDO is described in 12.15.2.4.

The firmware upgrade process used by the system manager for over-the-air firmware upgrades is described in 6.3.6. The methods and attributes of the upload/download object in the DMAP of the device can be used for sending firmware updates to the device.

The firmware upgrade process may include a cut-over mechanism that specifies a cut-over time (after the update is delivered), at which point devices begin using the new firmware. The CutoverTime attribute in the UDO shall be used to indicate this cut-over time. The cut-over time uses the shared sense of time configured by the system manager.

Support is provided for vendor-specific, device model-specific, and device instance-specific updates. Before cut-over, the upload/download object of the device may perform safety checks on a received update to assure that an update is appropriate for a specific device type. As all the communication takes place between application objects, such an update is protected by the end to end TL security mechanism. In addition, the firmware update may use security mechanisms to authenticate the update. As part of the firmware upgrade process, the upload/download object of the device may be provided with the appropriate standardized labeling, versioning, and security information for these verifications by the host update application. These verifications may be vendor-specific and are not specified by this standard.

NOTE The UDO in the DMAP to update firmware is described here. Details about general upload/download objects that are availble for use by general application processes are given in 12.15.2.4.

6.2.7.4 Layer management objects

The set of objects within the DMAP shall include objects representing access to each of the layer management SAPs. These objects include:

- The application sublayer management object (ASLMO), which provides access to the ASMSAP.
- The TL management object (TLMO), which provides access to the TMSAP.
- The NL management object (NLMO), which provides access to the NMSAP.
- The data-link management object (DLMO), which provides access to the DMSAP.

The services, defined by the layer SAP definitions, are reflected in the features of the management objects such that, effectively, the services made available at the management SAPs become remotely accessible using secure standard communications mechanisms. Generically, the management SAPs provide for reading and writing attributes, invoking methods, and reporting events. The various layer specifications specify the exact features that are available on each of these SAPs. In effect, these specifications define the layer management objects. See 6.2.8.2 for more details on the layer management objects.

6.2.7.5 Device security management object

The DMAP shall include a device security management object (DSMO) that provides appropriately limited access to device security management functions. The DSMO manages security key material and cryptographic operations. The details of this object are provided in 7.11.

6.2.7.6 Device provisioning object

The DMAP shall include a device provisioning object (DPO) that is accessed during the provisioning process of the device. More details about the attributes, methods and alerts of the DPO are provided in 13.9.

6.2.7.7 Health reports concentrator object

The DMAP shall include a health reports concentrator object (HRCO) that can be configured by the system manager to enable periodic publication of device health reports. The device health reports may consist of periodic publication of one or more attributes from the management objects in the DMAP.

The attributes of the HRCO are as per the definition of the concentrator object given in 12.15.2.5. The object identifier of the HRCO in the DMAP shall be 10. The CommunicationEndPoint and Array of ObjectAttributeIndexAndSize attributes of the HRCO are used by the system manager to set up periodic publications of health reports from the device. The system manager may choose to include any attribute from any management object in such health reports which are used for system performance monitoring. System performance monitoring is described in 6.3.7.

6.2.8 Functions of device management and layer management

6.2.8.1 Device management functions

6.2.8.1.1 General

Device management capabilities are provided primarily via access to DMO attributes and invocation of methods. The DMO contains critical attributes of device-wide scope that shall be available in all devices. Product implementers may extend the list of attributes beyond the required attributes described below.

Some attributes available via the DMO may also be available as an attribute of a particular layer management object. In that case, change in the value of any such attribute shall be reflected in the corresponding attribute.

The DMO shall provide system time information to other management objects; the DMO may obtain this system time information by interacting with the DL of the device and / or the system manager or from another source, such as a GPS receiver within the device. Time-keeping by the DL is described in the 9.1.9. The role of the system manager in maintaining time across the network is described in 6.3.10.

The establishment, modification and termination of contracts for a device shall be managed by its DMO. Contracts are described in 6.3.11.2.

The attributes of the DMO are defined in Table 10.

Table 10 – DMO attributes

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
EUI64	1	64-bit unique identifier of device	Type: EUI64Address Classification: Constant Accessibility: Read only Default value: 0x0000 0000 0000 0001	This shall be a global unique EUI64Address. This attribute is a duplicate of corresponding attributes in DLMO and NLMO
DL16Address	2	16-bit identifier for device, unique in its D-subnet	Type: DL16Address Classification: Static Accessibility: Read/write Default value: 0 Valid range: 0: address unassigned; 1..0x7FFF: unicast address	Address unique in D-subnet of device; assigned by system manager. This attribute is a duplicate of the corresponding attribute in the DLMO and NLMO. Configured by the system manager during the device join process.
IPv6Address	3	IPv6Address assigned by system manager	Type: IPv6Address Classification: Static Accessibility: Read/write Default value: 0 Valid range: 0: address unassigned; other with higher-order bit reset: unicast address.	Network address unique in network of device and used by application to identify devices across the network. This attribute is a duplicate of corresponding attributes in DLMO and NLMO. Configured by the system manager during the device join process

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Device_Role_Capability	4	Role(s) that the device is capable of playing in the network; roles are defined in 5.2.6.2	Type: BitArray16 Classification: Constant Accessibility: Read only	This attribute shall be sent to the system manager during the device join process; see 6.3.9.2. See 5.2.6.2 for acceptable roles and their descriptions. Named indices: 0: I/O; 1: router; 2: backbone router; 3: gateway; 4: system manager; 5: security manager; 6: system time source; 7: provisioning device; 8..15: reserved (shall be 0)
Assigned_Device_Role	5	Role(s) of the device as assigned by the system manager; roles are defined in 5.2.6.2	Type: BitArray16 Classification: Static Accessibility: Read/write	This attribute shall be written by the system manager during the device join process; see 6.3.9.2. Refer to 5.2.6.2 for acceptable roles and their descriptions. The assigned role for a device shall not exceed its capabilities as specified in attribute 4. The bit array indices are identical to those of attribute 4.
Vendor_ID	6	Human-readable identification of device vendor	Type: VisibleString16 Classification: Constant Accessibility: Read only	Assigned by vendor during device manufacturing
Model_ID	7	Human-readable identification of device model	Type: VisibleString16 Classification: Constant Accessibility: Read only	Assigned by vendor during device manufacturing
Tag_Name	8	Tag name of device	Type: VisibleString16 Classification: Static Accessibility: Read/write	Assigned by user
Serial_Number	9	Serial number of device	Type: VisibleString16 Classification: Constant Accessibility: Read only	Assigned by vendor during device manufacturing

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Power_Supply_Status	10	Status information of power supply of device	Type: Unsigned8 Classification: Dynamic Accessibility: Read/write	Named values: 0: line powered; 1: battery powered, greater than 75% remaining capacity; 2: battery powered, between 25% and 75% remaining capacity; 3: battery powered, less than 25% remaining capacity
Device_Power_Status_ Check_AlertDescriptor	11	Used to change the priority of Device_Power_Status_Check alert (described in Table 11); this alert can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 8]	—
DMAP_State	12	Status of DMAP	Type: Unsigned8 Classification: Dynamic Accessibility: Read only Default value: 1: active	DMAP state diagram is same as UAP state diagram given in 12.15.2.2.3. Named values: 0: inactive; 1: active; 2: failed

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Join_Command	13	Command informing device to join the system, restart itself and re-join, or reset to factory defaults	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0: none	The use of this attribute is described in 6.3.9. The value 0: none shall not be indicated in a write request. Only the provisioning device is expected to be able to issue the Join_Command with value 1 = join / start, since the device has not yet joined the network and so is not accessible by any other device. WarmRestart shall preserve static and constant attributes data including contracts and T-keys. RestartAsProvisioned corresponds to the provisioned state of the device in which the device only retains the information that is received during its provisioning step. Reset to factory defaults corresponds to the unconfigured device phase in Figure 5. Named values: 0: none; 1: join and start; 2: warm restart; 3: restart as provisioned; 4: reset to factory defaults
Static_Revision_Level	14	Revision level of the static data associated with all management objects	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0: none	Revision level is incremented each time a static attribute value in any management object is changed; value rolls over when limit is reached; value resets whenever the device is reset to factory defaults (Join_Command value of 4: reset to factory defaults).

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Restart_Count	15	Number of times device restarted	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	Device restart can be due to battery replacement, warm restart command, firmware download, link failure; value rolls over if max value is reached; value resets to 0 when device is reset to factory defaults.
Uptime	16	Low accuracy counter for counting seconds since last device restart	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Units in seconds; reset to 0 if device restarts
Device_Memory_Total	17	Total memory of device expressed in octets	Type: Unsigned32 Classification: Constant Accessibility: Read only	Units in octets
Device_Memory_Used	18	Memory currently used in device expressed in octets	Type: Unsigned32 Classification: Dynamic Accessibility: Read only	Units in octets
TAI_Time	19	Current TAI time	Type: TAINetworkTime Classification: Dynamic Accessibility: Read only	Value is obtained either from DL (if device is not system time source) or from backbone / external source (if device is system time source or is on the backbone and does not have a DL).
Comm_SW_Major_Version	20	Major version of communications software currently being used in the device	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: 0	8-bit communications software major version number, assigned by this standard, equals 0.
Comm_SW_Minor_Version	21	Minor version of communications software currently being used in the device	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: 1	8-bit communications software minor version number assigned by this standard, equals 1.
Software_Revision_ Information	22	Revision information about communications software for particular major and minor version numbers	Type: VisibleString16 Classification: Constant Accessibility: Read only	Revision information assigned by vendor
System_Manager_ IPv6Address	23	Network address of system manager	Type: IPv6Address Classification: Static Accessibility: Read/write Default value: 0	This information shall be provided to device either during provisioning process or during join process

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
System_Manager_ EUI64Address	24	EUI64Address of system manager	Type: EUI64Address Classification: Static Accessibility: Read/write Default value: 0	This information shall be provided to device either during provisioning process or during join process
System_Manager_ DL16Address	25	DL16Address of system manager in D-subnet of device	Type: DL16Address Classification: Static Accessibility: Read/write Default value: 0	This attribute shall be configured by the system manager during the device join process.
Contracts_Table	26	Table that includes information about all existing contracts of the device	Type: Array of Contract_Data Classification: Static Accessibility: Read/write	Updated when a corresponding contract gets established, modified, renewed or terminated; see 6.3.11.2 for more details about contracts and about data type Contract_Data. A new entry in the Contracts_Table shall be created each time a contract response associated with the successful creation of a new contract is received from the system manager. For additional details see 6.3.11
Contract_Request_ Timeout	27	Timeout for DMO before the contract request can be retried	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 30 s	System manager sets this timeout value after the device joins the network. Unit: s
Max_ClientServer_Retries	28	The maximum number of client request retries DMAP shall send in order to have a successful client/server communication	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 3 Valid range: 0..8	The number of retries sent for a particular message may vary by message based on application process determination of the importance of the message. For example, some messages may not be retried at all, and others may be retried the maximum number of times
Max_Retry_Timeout_ Interval	29	The maximum timeout interval for a client request before it is sent again	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 30 s	System manager sets this timeout value after the device joins the network. Unit: s

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
DMAP_Objects_Count	30	Number of management objects in DMAP including this DMO	Type: Unsigned8 Classification: Static Accessibility: Read only Default value: 1 Valid range: > 0	Total count of the management objects such as DLMO, NLMO, etc in the DMAP of this device; all application processes in the device shall include an attribute with such information
DMAP_Objects_List	31	List of all the management objects in the DMAP	Type: Array of ObjectIDandType Classification: Static Accessibility: Read only	List to identify all the management objects that are available in the DMAP; all application processes in the device shall include an attribute with such information. See 12.16.3.10 for details about this data type
Metadata_Contracts_Table	32	Metadata (count and capacity) of the Contracts_Table attribute	Type: Metadata_attribute	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for the table; see 6.2.6.3 for details about this data type
Non_Volatile_Memory_ Capability	33	Indicates if device is capable of maintaining all DMAP information that falls under the Static classification in non-volatile memory over a power-cycle or not	Type: Boolean8 Classification: Constant Accessibility: Read only	See 6.3.9.4.2 for more information
Warm_Restart_Attempts_ Timeout	34	The timeout after which a device that is trying to re-join the network through a warmRestart converts to a restartAsProvisioned command	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 60	Units in minutes; see 6.3.9.4.2 for more information

Table 10 (continued)

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Device_Restart_ AlertDescriptor	35	Used to change the priority of Device_Restart alert (described in Table 11); this alert can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 8]	—
Proxy_Join_Request_Rate	36	Used to control the maximum rate at which a proxy router will accommodate join requests	Type: Integer8 Classification: Static Accessibility: Read/write Default value: 6 Valid range: -4..127	Minimum required interval between join requests that the proxy router is permitted to accept. A value of $N > 0$ specifies a period of N s, while $N < 0$ specifies a period of $-1/N$ s. $N = 0$ disables the attribute. This parameter is used to reduce the impact of denial of service (DoS) attacks.
Reserved for future editions of this standard	37..63	—	—	—

6.2.8.1.2 Device management object alerts

The DMO of the device shall send an alert to indicate a change in its power status. The DMO of the device shall send an alert whenever it goes through a device restart. Device restart is described in 6.3.9.4.2.

The alerts of the DMO are defined in Table 11.

Table 11 – DMO alerts

Standard object type name(s): Device management object (DMO)					
Standard object type identifier: 127					
Description of the alert: Communication diagnostic alerts to indicate that the device power supply status has changed and to indicate that the device has restarted					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: urgent, high, med, low, journal)**	**Value data type**	**Description of value included with alert**
0 = Event	1 = Comm. diagnostics	0 = Device_Power_Status_Check	8 = Medium	Type: Unsigned8	The current value of the Power_Supply_Status attribute in Table 10 is included in this alert
0 = Event	1 = Comm. diagnostics	1 = Device_Restart	8 = Medium	Type: N/A	Only a device that has DMO attribute Non_Volatile_Memory_Capability = 1 can send this alert to indicate that it has gone through a warm restart

6.2.8.1.3 Device management object methods

The methods of the DMO are described in 6.3.9.2.2, 6.3.11.2.10.5, and 6.3.11.2.11.3.

6.2.8.2 Layer management

6.2.8.2.1 General

Each communication layer within the protocol suite has a self-contained layer management functionality. Each layer management function provides a management SAP. Device management of layers is accomplished via access to the management SAPs on each of the layers, as shown in Figure 25. Management of the layers within a device may be done locally by a functionality that resides within the DMO, since the DMO has access to the management SAPs. In addition, as discussed in 6.2.4, layer management may be accomplished remotely by a system manager.

The formal definition of each of the layer management objects is included below. The definition of the layer management objects within the DMAP corresponds directly with the layer management SAP definition. However, there may be a need to restrict remote access to specific features of a given layer management SAP. Thus, some attributes or methods, while accessible to the local DMO, should not be remotely accessible. Such restrictions, whenever necessary, are specified in the layer specifications. The operations described in Annex J can be used to access attributes in these layer management objects.

6.2.8.2.2 DL management object

6.2.8.2.2.1 General

In the architecture defined by this standard, all DL, MAC, and PhL layer management services are provided via a unified data-link management service access point (DMSAP). There are no directly accessible management SAPs for the physical layer and MAC layer, because those layers sometimes require management actions to be time-synchronous with data flow. Thus te DLMO provides the attributes of those layers that are accessible remotely.

6.2.8.2.2.2 Physical layer management

Physical layer management entities are manipulated indirectly via the DMSAP. DL attributes relate to a subset of those defined in IEEE 802.15.4, as described in 9.1.5.

6.2.8.2.2.3 Media access control sublayer management

MAC sublayer management entities are manipulated indirectly via the DMSAP. DL attributes relate to a subset of those defined in IEEE 802.15.4, as described in 9.1.5.

6.2.8.2.2.4 DL management

DL management attributes and methods are available via the DMSAP. Attributes, methods, and alerts of the DLMO are defined in 9.4 and 9.6.

6.2.8.2.2.5 NL management

NL management attributes and methods are available via the NMSAP. Attributes, methods and alerts of the NLMO are defined in 10.4.

6.2.8.2.2.6 TL management

TL management attributes and methods are available via the TMSAP. Attributes, methods, and alerts of the TLMO are defined in 11.6.

6.2.8.2.2.7 Security management

Device security management attributes and methods are available via the DMSAP and TMSAP. Attributes, methods and alerts of the DSMO are defined in 7.11.

6.2.8.2.2.8 Application sublayer management

Application sublayer management attributes and methods are available via the ASMSAP. Attributes, methods, and alerts of the ASLMO are defined in 12.19.

6.3 System manager

6.3.1 General

The functions of the system manager include security management, address allocation, software updating, system performance monitoring, device management, system time services, and communication configuration including contract services, and redundancy management.

The system manager shall use ASL services to remotely access management objects in the DMAPs of devices compliant with this standard.

System manager is a role and is not tied into a specific fixed physical address.

6.3.2 System management architecture

Conceptually, the system manager can be viewed as an application process running on any device in the network. Such a device shall be capable of supporting the system manager role. The SMAP is accessible only on such a device. The SMAP shall use the application sublayer SAP ASLDE-1 SAP for communicating with the devices. This application sublayer SAP shall correspond to TL SAP TDSAP-1 which shall correspond to port number 0xF0B1.

Figure 26 shows the system manager that resides in a field device compliant with this standard.

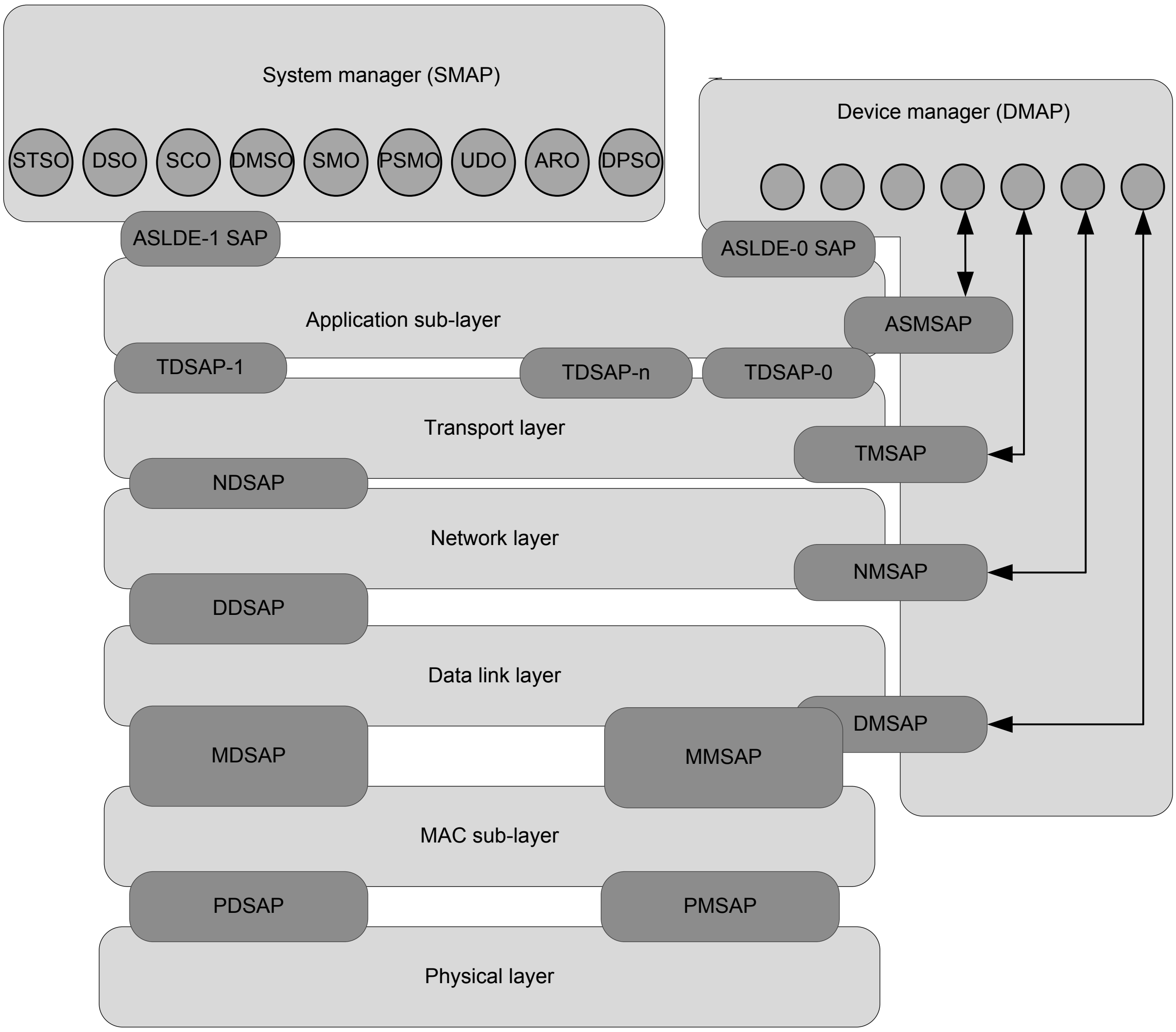


Figure 26 – System manager architecture concept

As shown in Figure 26, TDSAP-1 shall be used to access the management objects in the SMAP. The definition of these system management objects is necessary to provide remote access to these functions for the devices in the network that are compliant with this standard.

6.3.3 Standard system management object types

Table 12 includes a list of system management object types that are specified in this standard.

Table 12 – System management object types

Standard object type name	Standard object type ID	Standard object identifier	Object description
System time service object (STSO)	100	1	This object facilitates the management of system-wide time information; see 6.3.10
Directory service object (DSO)	101	2	This object facilitates the management of addresses for all existing devices in the network; see 6.3.5
System communication configuration object (SCO)	102	3	This object facilitates the communication configuration of the system including contract establishment, modification and termination; see 6.3.11
Device management service object (DMSO)	103	4	This object facilitates device joining, device leaving, and device communication configuration; see 6.3.9
System monitoring object (SMO)	104	5	This object facilitates the monitoring of system performance; see 6.3.7
Proxy security management object (PSMO)	105	6	This object acts as a proxy for the security manager; see 6.3.4
Upload/download object (UDO)	3	7	This object facilitates downloading firmware/data to devices and uploading data from devices; see 6.3.6
Alert-receiving object (ARO)	2	8	This object receives all the alerts destined for the system manager; see 6.3.7
Device provisioning object (DPO)	106	9	This object facilitates device provisioning; see 6.3.8
Reserved for future editions of this standard	107..113	—	—

Devices that require system management services communicate with the appropriate objects given above.

6.3.4 Security management

The system manager interfaces with the security manager to generate keys and authenticate devices. The security manager is functionally separated from the system manager so that the security policies can be common across the networks of the administrator and other types of networks. Placing the security manager functionally behind the system manager also hides from the devices the various protocols, such as Kerberos, that may be used by a security management function. More details are provided in Clause 7.

The interface between the system manager and the security manager is not specified in this standard. Conceptually, the system manager can be viewed as including a proxy security management object (PSMO). This PSMO forwards all security related messages between the security manager and the devices in the network that are compliant with this standard. This PSMO can be used by the security manager to access information from other system management objects, such as current TAI time, if necessary. The security manager does not have a valid address as defined by this standard; thus, devices that wish to communicate with the security manager can only do so by communicating with the PSMO.

The attributes, methods and alerts of the PSMO are defined in Clause 7.

6.3.5 Addresses and address allocation

6.3.5.1 General

The system manager is responsible for assigning addresses to devices when they join the network.

6.3.5.2 Address types

Every device compliant with this standard shall have one identifier and two addresses:

- Each device compliant with this standard shall have an EUI64Address identifier that is presumed to be globally unique (vendors are expected to ensure global uniqueness of these identifiers). Failover mechanisms for the gateway, system manager and security manager roles are usually provided through redundancy. Redundancy for the purposes of failover may involve EUI64Address identifier duplication for the redundant entities. Such EUI64Address identifier duplication is outside the scope of this standard.
- Each device compliant with this standard shall be assigned one IPv6Address by the system manager when it joins the network; this IPv6Address shall be unique across the network.
- Each device compliant with this standard that is accessible through a D-subnet shall have a D-subnet-unique DL16Address for its IPv6Address. This DL16Address shall be assigned by the system manager. The scope of any DL16Address is limited to a particular D-subnet.

  The ranges and uses of 16-bit D-addresses are:

  – 0x0000: reserved by this standard to indicate that a DL16Address has not been assigned to the device;

  – 0x0001..0x7FFF: reserved by IETF RFC 4944 for 6LoWPAN unicast device addressing;

  – 0x8000..0xBFFF: reserved by IETF RFC 4944 for 6LoWPAN multicast;

  – 0xC000..0xCFFF: reserved by this standard for graph IDs;

  – 0xD000..0xFFFD: reserved;

  – 0xFFFE: reserved by IEEE 802.15.4:2012 for the local PAN coordinator;

  – 0xFFFF: reserved by IEEE 802.15.4 for local broadcast.

In this standard, DL16Addressing is always used within a D-subnet, with the exception that EUI64Address identifiers are used in a limited way during the join process until a joining device has been received a subnet-local DL16Address from the system manager. The join process is described in 7.4.

6.3.5.3 Address allocation

The system manager shall allocate the IPv6Address, as well as the D-subnet-unique DL16Address, to a device when it joins the network. This is described in 7.4.

When a source device that belongs to a particular D-subnet communicates over the backbone with a destination device that belongs to a different D-subnet, the system manager shall assign local D-subnet-unique DL16Addresses to both devices in each other’s D-subnets. Such local DL16Addresses for remote devices (i.e., devices residing in another D-subnet) may be established by the system manager upon a contract request. (Contracts are described in 6.3.11.2.)

When the source sends a message to the destination, the DL16Address of the destination shall be used at the NL and DL to construct the NPDU and DPDU. These layers can use the directory look-up service provided by the system manager to obtain the DL16Address of the destination, if not already known. This service is described in 6.3.5.4.

The backbone routers shall do the address translations between the DL16Address (per D-subnet) and IPv6Address for a given device. Note that the DL16Address is used only within the DL. Once a message reaches the backbone, the full IPv6Address shall be used. Informative examples for such scenarios are given in 10.2.7.

This standard does not specify any mechanisms for how the system manager allocates the IPv6Addresses and DL16Addresses. 10.2.7 contains examples for Ethernet based routing and fieldbus based routing. The Ethernet-based routing example describes the use of IPv6-based IPv6Addresses. The fieldbus routing example describes the use of non-IPv6-based IPv6Addresses.

Devices shall only have one valid IPv6Address. Multi-homing devices that require multiple IPv6Addresses are not covered in this standard.

Addressed entities on the backbone, such as system managers and gateways, shall also be assigned DL16Addresses for use within a D-subnet. Thus, the addresses most used within a D-subnet will be DL16Addresses.

6.3.5.4 Directory service

The directory service object (DSO) in the system manager provides the necessary attributes for looking up the address translation between the EUI64Address, the IPv6Address, and the DL16Address(es) of a given device. D-subnet IDs are also maintained by the DSO, but the allocation of these D-subnet IDs is not specified in this standard.

The attributes of the DSO are defined in Table 13.

Table 13 – DSO attributes

Standard object type name: Directory service object (DSO)				
Standard object type identifier: 101				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Address_Translation_Table	1	Address translation table containing EUI64Address, IPv6Address, D-subnet ID(s) and DL16Address(es) of all devices in the network	Type: Array of Address_Translation_Row Classification: Dynamic Accessibility: Read only	Structured attribute used to look-up address translations; See Table 14
Reserved for future editions of this standard	2..63	—	—	—

The data structure Address_Translation_Row is defined in Table 14.

Table 14 – Address_Translation_Row data structure

Standard data type name: Address_Translation_Row		
Standard data type code: 402		
Element name	**Element identifier**	**Element type**
EUI64Address	1	Globally unique EUI64Address of device; Type: EUI64Address Classification: Static Accessibility: Read only
IPv6Address	2	IPv6Address of device assigned by system manager; Type: IPv6Address Classification: Static Accessibility: Read only
DL_Subnet_ID	3	D-subnet in which or from which this device is reachable; a device may be reachable from multiple D-subnets in which case this element corresponds to one such D-subnet; Type: Unsigned16 Classification: Static Accessibility: Read only
DL16Address	4	DL16Address of device in the D-subnet indicated by the DL_Subnet_ID element given above; Type: Unsigned16 Classification: Static Accessibility: Read only

Address translation look-up service is provided by the Read_Address_Row method defined in Table 15.

Table 15 – Read_Address_Row method

<table>
<tr><th colspan="5">Standard object type name: Directory Service object (DSO)</th></tr>
<tr><th colspan="5">Standard object type identifier: 101</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td>Read_Address_Row</td><td>1</td><td colspan="3">Method to use the address translation look-up service for reading the values of other addresses / identifier of a device given an index (one of its address / identifier)</td></tr>
<tr><td rowspan="3"></td><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Attribute_ID</td><td>Data type: Unsigned8</td><td>Value = 1 (Address_Translation_Table attribute of DSO)</td></tr>
</table>

Standard object type name: Directory Service object (DSO)				
Standard object type identifier: 101				
	2	Index_Info	Data type: Unsigned8	Named indices: 0: only EUI64Address provided; 1: only IPv6Address is provided; 2: EUI64Address and D-subnet ID are provided; 3: IPv6Address and D-subnet ID are provided; 4: D-subnet ID and DL16Address are provided. See Table 16
	3	Index_EUI64	Data type: EUI64Address	Value: EUI64Address of device for which address look-up is needed
	4	Index_128_Bit_Address	Data type: IPv6Address	Value: IPv6Address of device for which address look-up is needed
	5	Index_DL_Subnet_ID	Data type: Unsigned16	Value: D-subnet ID of device for which address look-up is needed
	6	Index_DL_address_16_Bit	Data type: DL16Address	Value: DL16Address of device for which address look-up is needed
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Value_Type	Data type: Unsigned8	Indicates the type of information being provided; Named values: 0: single row if D-subnet ID value provided as input argument; 1: all rows if D-subnet ID value not provided as input argument (i.e., all rows for given EUI64Address or IPv6Address are returned). See Table 17
	2	Value_Size	Data type: Unsigned8	Number of rows being returned
	3	Data_Value_1	Data type: Address_Translation_Row	EUI64Address, IPv6Address, D-subnet ID, DL16Address of device
	2+n	Data_Value_n	Data type: Address_Translation_Row	EUI64Address, IPv6Address, D-subnet ID, DL16Address of device

Some of the input arguments are not applicable if the Index_Info argument has certain values and so shall not be included in the request. The usage of input arguments for the Read_Address_Row method is described in Table 16.

Table 16 – Input argument usage for Read_Address_Row method

Input argument	Not applicable for Index_Info value
Attribute_ID	—
Index_Info	—
Index_EUI64	1, 3, 4
Index_128_Bit_Address	0, 2, 4
Index_DL_Subnet_ID	0, 1
Index_DL_Address_16_Bit	0, 1, 2, 3

Some of the output arguments are not applicable if the Value_Type argument has certain values and so shall not be included in the response. The usage of output arguments for the Read_Address_Row method is described in Table 17.

Table 17 – Output argument usage for Read_Address_Row method

Output argument	Not applicable for Value_Type value
Value_Type	—
Value_Size	0
Data_Value_1	—
Data_Value_n	0

High-side interfaces on the DSO to delete addresses or modify addresses are not specified in this standard.

6.3.5.5 Multicast DL16Address management

DL16Addresses of the form 0x 100x xxxx xxxx xxxx shall be reserved for multicast, following the convention set by IETF RFC 4944.

Multicast DL16Address management is not specified in this standard. Additional attributes for the directory service object may be specified by vendors that support multicast DL16Address management.

6.3.6 Firmware upgrade

The system manager provides support for over-the-air firmware upgrades to devices. The system manager supports communication protocol suite firmware updates.

The system manager shall provide an interface for accepting the firmware upgrades that need to be sent to any device in the network. The system manager shall use the UDO in the DMAP of the device for sending this update to the device. Communication protocol suite firmware updates shall be performed only through the system manager UDO. This UDO is described in 12.15.2.4.

As the system manager maintains information about all the devices in the network, the host update application can obtain information about the devices that are in the network from the system manager. The host update application may use this information to determine which devices need such firmware upgrades. The gateway may also be used to send firmware upgrades to the device. If these upgrades are communication protocol suite upgrades, they must be sent through the system manager UDO. This is described in U.3.2.

The firmware upgrade process shall assure that network operations are maintained across updates. This process may include a cut-over mechanism that specifies a cut-over time (after the update is delivered), at which point devices shall begin using the new firmware. The cut-over time shall use the shared sense of time configured by the system manager. If the system manager is sending the firmware upgrade to the device, it may send the cut-over time along with the download, or it may send the cut-over after the download is complete.

Since firmware upgrades may be vendor-specific, the updates are opaque to the system manager providing the update service. The system manager accepts updates via unspecified protocols (e.g., a user interface tool with a DVD reader) and provides updating to selected devices at specified times based on user or other input. Details of how the host update application communicates with the system manager, such as what devices should be updated, what their vendor IDs are, when the devices should be updated, and in what order they should be updated are not specified by this standard, but such functions are expected to be supported by the system manager. The system manager may schedule updates to the devices in such a manner that network downtime is either avoided or minimized. This schedule may depend on network topology.

Multicasting of firmware upgrades is not specified by this standard.

The UDO in the system manager shall be used if the firmware of the system manager itself needs to be upgraded. The attributes, methods and state machines of the UDO in the system manager are as per the definition provided in 12.15.2.4. The object identifier for the UDO in the SMAP shall be 7.

6.3.7 System performance monitoring

6.3.7.1 General

System performance monitoring is done by the system manager in order to collect information that can be used to take necessary actions for optimizing system performance and for reacting to changes in the radio environment and device status. Such actions are done through system communication configuration which is described in 6.3.11.

System performance monitoring is accomplished via polling of device attributes or by configuring devices to generate alerts that provide event-driven information.

System performance monitoring using periodic publication of health reports from the devices is supported through the use of the HRCO in the DMAP of each device. HRCO is described in 6.2.7.7 and can be configured by the system manager to periodically report the values of one or more attributes in the management objects of the device. Before the system manager configures the HRCO of any particular device to publish health reports, it needs to create a unique dispersion object in the SMAP to act as the subscriber to the data that will be published by the HRCO of this particular device. Information about this unique dispersion object shall be conveyed to the HRCO of this particular device by configuring the CommunicationEndPoint attribute in the HRCO. The dispersion object is described in 12.15.2.6.

Information about the capabilities of a new device is provided to the system manager during its join process. This is discussed in 6.3.9.

Devices may be configured by the system manager to generate alerts to provide event-driven information, for example, when a link stops working or when the battery of a field device has less than 25% remaining capacity. This is described in 6.3.7.2.

While the device implementing the system manager may have an interface that allows plant operations and maintenance personnel to observe and control the performance of the network and devices, this interface is neither mandatory nor is it specified by this standard.

The UDO in the DMAP of a device may be used for downloading large blocks of device performance data from the device to the system manager. Such data may be vendor-specific, device model-specific, or device instance-specific. The UDO in the system manager may be used to upload such data for further analysis. Such data collection and analysis is not specified by this standard.

6.3.7.2 System management alerts

The system manager contains an alert-receiving object (ARO) that receives communication diagnostic alerts and security alerts. Such alerts are described in 6.2.7.2. These alerts may be used by the system manager to monitor system performance and take suitable action when necessary. The object identifier for the ARO in the SMAP shall be 8.

After a device joins the network, it shall set the Alert_Master_Comm_Diagnostics and Alert_Master_Security attributes of the ARMO to point to the system manager.

The attributes of the ARO in the system manager are as per the definition given in 12.15.2.3. The default value for the ARO.Categories attribute shall be 0110 0000.

The state diagram describing the handling of alert reports by the ARO is given in 12.15.2.3.

6.3.7.3 System monitoring object

The attributes of the SMO are given in Table 18.

Table 18 – Attributes of SMO in system manager

Standard object type name: System monitoring object (SMO)				
Standard object type identifier: 104				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved for future editions of this standard	1..63	—	—	—

6.3.7.4 System monitoring configuration

System performance monitoring may have its own dynamic configuration such that monitoring may be increased when necessary. For example, the list of active alerts may be adjusted depending on the current state of the network. For network diagnostics, if a failure mode is suspected, activating specific alerts may provide evidence of that failure. Such configuration of the system manager is not specified by this standard.

6.3.8 Device provisioning service

Before a device joins the network, it requires the appropriate security credentials and network-specific information. These are provided to the device during the provisioning process. The provisioning process, the role of the system manager in this process, and the definition of the device provisioning object (DPO) are described in Clause 13.

6.3.9 Device management services

6.3.9.1 General

A device compliant with this standard may go through several phases in its operational lifetime. These phases are described in 5.2.8. The management of a device through some of these phases is performed by the system manager. Specifically, the system manager plays a role in the joining and leaving processes as well as the communication configuration of a device.

6.3.9.2 Join process

6.3.9.2.1 General

A new device shall obtain the necessary provisioning information from the provisioning device. This is described in Clause 13. The Join_Command attribute in the DMO of a device shall be used to command the device to join the network. Only the provisioning device can set the Join_Command attribute to 1, hence explicitly triggering the join process of the device, given that the device has no connectivity with other entities in the network. The Join_Command attribute shall be set to 1 by the provisioning device only following a successful provisioning of the device. Advertising routers shall proxy join requests at the rate indicated by the DMO attribute Proxy_Join_Request_Rate (attribute 36). This rate ensures that the network is protected from denial of service attacks attempted via the proxy routers. For a description of Proxy_Join_Request_Rate, see Table 10.

The system manager controls the process of new devices joining the network. Non-joined devices that implement a DLE as per this standard listen for advertisement messages from local routers whose advertisement functions are configured by the system manager. Such an advertising router shall assist during the join process of a new device by acting as a proxy for the system manager. This advertising router forwards the join request from the new device to the system manager and forwards the join response from the system manager to the new device. A join request from the new device shall be processed by the system manager. The system manager shall generate a join response after communicating with the security manager. Advertising routers shall proxy join requests at the rate indicated by the DMO attribute Proxy_Join_Request_Rate (attribute 36). This rate protects the network from denial of service attacks attempted by the proxy routers. For a description of this attribute, see Table 10.

The join request from a new device shall include non-security information such as the EUI64Address and capabilities of the device as well as security information of the device. The join response from the system manager shall include non-security information such as the assigned IPv6Address, assigned DL16Address and contract information of the device as well as security information such as T-key. The security information in the join request and join response is described in 7.4. Contracts are described in 6.3.11.2.

More details about the join process are described in 7.4.

6.3.9.2.2 Device management object methods for advertising router

The new device shall use the Proxy_System_Manager_Join method and Proxy_System_Manager_Contract method defined for the DMO of the advertising router to send its non-security information that is part of the join request and to get its non-security information that is part of the join response. The non-security information that is part of the join request is split into the network join request and the contract request. The non-security information that is part of the join response is split into the network join response and the contract response. Contracts are described in 6.3.11.2.

The Proxy_System_Manager_Join method is defined in Table 19. The Proxy_System_Manager_Contract method is defined in Table 20.

The new device shall use the methods defined in 7.4 for the DMO of the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response. The use of all these methods by the new device is described in 7.4.

Access to the DMAP of the device is restricted to the SMAP present in the system manager. The joining shall be only allowed to access the Proxy_System_Manager_Join and Proxy_System_Manager_Contract DMO methods during the join process.

Table 19 – Proxy_System_Manager_Join method

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Method name	**Method ID**	**Method description**		
Proxy_System_Manager_Join	3	Method to use advertising router as proxy system manager to send network join request of a new device and get network join response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	EUI64	EUI64Address	DMO attribute EUI64; see Table 10
	2	DL_Subnet_ID	Unsigned16	D-subnet that the new device is trying to join, which is also the D-subnet of the advertising router. Data value: 0: device is not part of any D-subnet
	3	Device_Role_ Capability	Unsigned16	DMO attribute Device_Role_Capability; see Table 10
	4	Size_of_Tag_Name	Type: Unsigned8	Size in octets of the Tag_Name
	5	Tag_Name	Type: VisibleString SIZE(0..16)	DMO attribute Tag_Name; see Table 10
	6	Comm_SW_Major _Version	Type: Unsigned8	DMO attribute Comm_SW_Major_Version; see Table 10
	7	Comm_SW_Minor _Version	Type: Unsigned8	DMO attribute Comm_SW_Minor_Version; see Table 10
	8	Size_of_Software_ Revision_Information	Type: Unsigned8	Size in octets of the Software_ Revision_Information
	9	Software_Revision_ Information	Type: VisibleString SIZE(0..16)	DMO attribute Software_Revision_Information; see Table 10
	10	DeviceCapability	Type: OctetString	DLMO attribute DeviceCapability; see Table 141

Standard object type name: Device management object (DMO)				
Standard object type identifier: 127				
Method name	**Method ID**	**Method description**		
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Assigned_Network_ Address_128_Bit	Type: IPv6Address	This value is written to DMO attribute Network_Address_128_Bit; see Table 10
	2	Assigned_DL_ Address_16_Bit	Type: DL16Address	This value is written to DMO attribute DL_Address_16_Bit; see Table 10
	3	Assigned_Device_Rol e	Type: BitArray16	This value is written to DMO attribute Assigned_Device_Role; see Table 10
	4	System_Manager_ Network_Address_ 128_Bit	Type: IPv6Address	This value is written to DMO attribute System_Manager_128_Bit_Address; see Table 10
	5	System_Manager_DL _ Address_16_Bit	Type: DL16Address	This value is written to DMO attribute System_Manager_DL_Address_16_ Bit; see Table 10
	6	System_Manager_ EUI64	Type: EUI64Address	This value is written to DMO attribute System_Manager_EUI64Address; see Table 10
	7	MIC	Type: OctetString4	This value is used for protecting argument 1 through 6 with Join key. This MIC value is generated by the Security Manager. The Advertisement router shall not overwrite this value. See 7.4.4.3.2
	8	Assigned_Max_TSDU _Size	Type: Unsigned16	Indicates the maximum TSDU supported in octets which can be converted by the source into max APDU size by taking into account the TL, security, AL headers and TMIC sizes

Table 20 – Proxy_System_Manager_Contract method

<table>
<tr><th colspan="5">Standard object type name: DMO (Device management object)</th></tr>
<tr><th colspan="5">Standard object type identifier: 127</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="9">Proxy_System_ Manager_Contract</td><td>4</td><td colspan="3">Method to use advertising router as proxy system manager to send contract request of a new device and get contract response. Contracts are described in 6.3.11.2</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>EUI64</td><td>Type: EUI64Address</td><td>DMO attribute EUI64; see Table 10</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Contract_Response</td><td>Type: New_Device_Contract_Response (see Table 31</td><td>Contract response to support future communication from new device to system manager; contracts are described in 6.3.11.2</td></tr>
<tr><td>2</td><td>MIC</td><td>Type: OctetString4</td><td>This value is used for protecting argument1 with join key. This MIC value is generated in Security Manager. Advertisement router shall not overwrite this value</td></tr>
</table>

6.3.9.2.3 Capabilities of new device

Information about the capabilities of a new device with respect to the device role shall be provided to the system manager during the join process of the device. This information is described in Table 19.

6.3.9.3 Device configuration

A device is configured after joining a network. Device configuration includes obtaining communication resources to support the communication needs of the device and configuring the protocol stack of the device to use these resources to communication. During this configuration, the system manager may take into account the capabilities of a device. A device may be reconfigured as the network changes or as the applications on the device need to change their services.

Device configuration is usually performed during the establishment of contracts. This is described in 6.3.11.2. Attributes and methods defined for the management objects of the DMAP shall be used by the system manager to configure the device.

The system manager does not configure the UAPs on the device. This is done by host applications on plant networks or by handheld maintenance tools.

6.3.9.4 Leave process

6.3.9.4.1 General

The system manager controls the process of a previously joined device leaving the network. This leave process may be initiated by the device when it intends to leave the network, or it may be initiated by the system manager.

The leave process includes two scenarios: device restart and device reset to factory defaults.

Device restart occurs when either the device itself or the system manager cause the device to restart. Some examples that lead to this scenario are battery replacement or rebooting to apply a new firmware image. A device is reset to its factory default settings if the device is being returned to its factory default state. Some examples that lead to this scenario are the device being returned to general stock for future deployment or the device being moved to a different network.

6.3.9.4.2 Device restart

A device restart process may be initiated by the device itself or by the system manager. There are two types of restarts: warmRestart and restartAsProvisioned. In both cases, the devices will immediately initiate the join process following the restart event. An explicit writing of the Join_Command DMO attribute to 1 is not needed following a warmRestart or a restartAsProvisioned event.

The Join_Command attribute in the DMO of a device shall be used to command the device to perform either a warmRestart or restartAsProvisioned.

A device that receives the warmRestart command shall reboot itself. If the Non_Volatile_Memory_Capability attribute in the DMO is 1, the device shall retain the values of all constant and static attributes in all application objects present in the DMAP as well as in the UAPs of the device. All other attributes are reset to their default values. If the Non_Volatile_Memory_Capability attribute in the DMO is 0, the device shall retain all the information that was provided to it during the provisioning step before it first joined the network as well as all the constant and static information present in the UAPs. All other attributes are reset to their default values and the device goes back to its provisioned state.

A device that receives the restartAsProvisioned command shall reset all constant and static attributes in all application objects present in the DMAP regardless of the Non_Volatile_Memory_Capability attribute setting present in the DMO except the DPO. A device that receives the restartAsProvisioned command shall reboot itself while retaining all the information that was provided to it during the provisioning step before it first joined the network. This information is described in Clause 13. This information is usually necessary for the device to rejoin the network without having to go through the provisioning step once again. The device shall also retain all the constant and static information present in the UAPs.

Table 21 collects and presents the effects of the different join commands on various attribute sets.

Table 21 – Effect of different join commands on attribute sets

Join Command Type	DMAP attributes (except DPO)	UAP Attributes	DPO Attributes
WarmRestart (Join_Command =2) when Non_Volatile_Memory_Capability = 1	KEEP	KEEP	KEEP
WarmRestart (Join_Command =2) when Non_Volatile_Memory_Capability = 0	CLEAR	KEEP	KEEP
RestartAsProvisioned (Join_Command = 3)	CLEAR	KEEP	KEEP
Reset to factory defaults (Join_Command = 4)	CLEAR	CLEAR	CLEAR

A firmware download may also result in a device restart. Information necessary for the device to use the new firmware and join the network may be stored in the device. The device usually goes through a restartAsProvisioned cycle in such cases.

6.3.9.4.3 Device reset to factory defaults

A device reset process may be initiated by the device itself or by the system manager.

The Join_Command attribute in the DMO of a device shall be used to command the device to perform a reset. A reset command forces the device to reset to factory defaults, and all attributes are reset to their default values. The device is expected to return to the factory default state which is described in Clause 13.

6.3.9.4.4 Device replacement

If an old device (i.e., joined device) is replaced by a new device (i.e., non-joined device) the system manager is expected to provide the old IPv6Address to this replacement device. The host application or the user is expected to inform the system manager about this replacement through communication path 5 in Figure 23. The system manager may choose to configure other attributes in the DMAP of the replacement device to match those in the old device. Configuration of the UAPs in the replacement device is expected to be done by host applications on plant networks or by handheld maintenance tools.

6.3.9.5 Device management service object

The device management service object (DMSO) in the system manager shall handle the non-security information in the join request from the new device that is forwarded by the advertising router and shall generate the non-security information in the join response.

The attributes of the DMSO are given in Table 22.

Table 22 – Attributes of DMSO in system manager

Standard object type name: Device management service object (DMSO)				
Standard object type identifier: 103				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved for future editions of this standard	1..63	—	—	—

A new device communicates with an advertising router which acts as a proxy for the system manager and forwards all the join messages between the new device and the system manager. The join process is described in 7.4. The methods used for sending join request and join response messages between the new device and the advertising router are given in 6.3.9.2.2. The advertising router shall use the System_Manager_Join method and the

System_Manager_Contract method defined in the DMSO for sending the network join request and the contract request and for receiving the network join response and the contract response associated with the join process of this new device.

The source object of the System_Manager_Join and System_Manager_Contract methods is the DMO of the proxy advertising router that communicates with the system manager on behalf of the new device.

The System_Manager_Join method is defined in Table 23. The System_Manager_Contract method is defined in Table 24.

Table 23 – System_Manager_Join method

Standard object type name: Device management service object (DMSO)				
Standard object type identifier: 103				
Method name	**Method ID**	**Method description**		
System_Manager_Join	1	Method to send network join request of a new device to system manager and get network join response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	EUI64	Type: EUI64Address	DMO attribute EUI64; see Table 10
	2	DL_Subnet_ID	Type: Unsigned16	D-subnet that the new device is trying to join; this is also the D-subnet of the advertising router; Named values: 0: device is not part of any D-subnet
	3	Device_Role_Capability	Type: Unsigned16	DMO attribute Device_Role_Capability; see Table 10
	4	Size_of_Tag_Name	Type: Unsigned8	Size in octets of the Tag_Name
	5	Tag_Name	Type: VisibleString SIZE(0..16)	DMO attribute Tag_Name; see Table 10
	6	Comm_SW_Major_Version	Type: Unsigned8	DMO attribute Comm_ SW_Major_Version; see Table 10
	7	Comm_SW_Minor_Version	Type: Unsigned8	DMO attribute Comm_ SW_Minor_Version; see Table 10
	8	Size_of_Software_ Revision_Information	Type: Unsigned8	Size in octets of the Software_Revision_ Information
	9	Software_Revision_Information	Type: VisibleString SIZE(0..16)	DMO attribute Software_Revision_ Information; see Table 10
	10	DeviceCapability	Type: OctetString	DLMO attribute DeviceCapability; see Table 141

Standard object type name: Device management service object (DMSO)					
Standard object type identifier: 103					
Method name	Method ID	Method description			
	Output arguments				
	Argument number	Argument name	Argument type (data type and size)	Argument description	
	1	Assigned_Network_ Address_128_Bit	Type: IPv6Address	This value is written to DMO attribute Network_Address_ 128_Bit; see Table 10	
	2	Assigned_DL_Address_16_ Bit	Type: DL16Address	This value is written to DMO attribute DL_Address_16_Bit; see Table 10	
	3	Assigned_Device_Role	Type: BitArray16	This value is written to DMO attribute Assigned_Device_Role; see Table 10	
	4	System_Manager_ Network_Address_128_Bit	Type: IPv6Address	This value is written to DMO attribute System_Manager_ 128_Bit_Address; see Table 10	
	5	System_Manager_ DL_Address_16_Bit	Type: DL16Address	This value is written to DMO attribute System_Manager_ DL_Address_16_Bit; see Table 10	
	6	System_Manager_EUI64	Type: EUI64Address	This value is written to DMO attribute System_Manager_EUI64Address; see Table 10	
	7	MIC	Type: OctetString4	This value is used for protecting argument 1 through 6. This MIC value is generated by the Security Manager. See 7.4.4.3.2	
	8	Assigned_Max_TSDU_Size	Type: Unsigned16	Indicates the maximum TSDU supported in octets which can be converted by the source into max APDU size by taking into account the TL, security, AL headers and TMIC sizes	

Table 24 – System_Manager_Contract method

Standard object type name: Device management service object (DMSO)				
Standard object type identifier: 103				
Method name	**Method ID**	**Method description**		
System_Manager_Contract	2	Method to send contract request of a new device to system manager and get contract response; Contracts are described in 6.3.11.2		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	EUI64	Type: EUI64Address	EUI64Address of the new device trying to join the network
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Contract_Response	Type: New_Device_Contract_Response (see Table 31)	Contract response to support future communication from new device to system manager; contracts are described in 6.3.11.2
	2	MIC	Type: OctetString4	This value is used for protecting argument 1 through 6 with the join key. This MIC value is generated in Security Manager. See 7.4.4.3.2

When the DMSO generates the System_Manager_Join and System_Manager_Contract responses, it first sends these responses to the PSMO which in turn sends them to the security manager. The security manager shall protect these responses with a MIC field using the join key and send them back to the PSMO which in turn hands them back to the DMSO. The DMSO shall then send the responses back to the advertising router. This interaction with the PSMO is described in 7.4.

6.3.10 System time services

6.3.10.1 General

The time in this standard is based on international atomic time (TAI) as the time reference. The time in this standard is reported as elapsed seconds since 1958/01/01 00:00:00.

The system supports time synchronization so that, at the device level, applications may use time to coordinate activities or time-stamp information, improving energy use and reliability. System time shall be available from at least one device (system time source) on the network.

A TAI time source is not required for operation of a network compliant with this standard. Alternative time sources are converted to TAI units. The time base used by the network shall be within ±1 second of actual TAI time. The gateway shall convert nominal network TAI time to the local system time reference if it is available.

The system manager shall configure at least one system time source in each D-subnet of the network. The system manager itself may be the system time source. This is described in

6.3.10.3. The system manager shall also configure the distribution topology for the dissemination of time in the D-subnet and for the synchronization of device clocks. The system manager configures each device in the D-subnet with the clock parent(s) that the device shall use for synchronizing its clock. The DL in each device is responsible for measuring time and keeping the clocks synchronized. This is described in 9.1.9.

Backbone routers are expected to use either proprietary or standardized techniques for maintaining time synchronization. These techniques are not specified by this standard.

Devices needing to convert TAI time to hh:mm:ss format, such as on a user display, may account for a coordinated universal time (UTC) accumulated leap second adjustment.[6] The system manager shall provide this UTC adjustment to these devices. If the device needs this UTC adjustment information from the system manager, it should refresh it infrequently but periodically, such as at the start of each month or any other arbitrary clock boundary.

All devices in a network compliant with this standard share the TAI time reference with variable degrees of accuracy. To support sequence of events or other timing related operations of the application processes, all routing devices within a network compliant with this standard should be accurate to within ±10 ms. The exact clock accuracy requirement for each routing device is described in 9.1.9.2.2.

The system manager is responsible for coordinating the time across different D-subnets by selecting the appropriate system time sources in each D-subnet. This coordination is not specified by this standard.

To support sequence of events or other timing related operations of the application processes, backbone routers also should be accurate to within ±10 ms. Neither conversion of the time units used by the backbone routers nor adjustment to align with the TAI time being used by the devices compliant with this standard are specified by this standard.

If the system manager is part of the D-subnet, then the DL in the system manager is responsible for measuring time and keeping the device clock synchronized. If the system manager is connected to the backbone, it is expected to maintain clock synchronization with the rest of the network and maintain time information in TAI units. In this case, the techniques for doing so are not specified by this standard.

Protocol layers that require the current TAI time of the device may obtain it from the DMO in the DMAP.

6.3.10.2 Device clock accuracy capabilities

The system manager needs to know the clock accuracy of each device. For example, it needs to know whether a device is capable of maintaining ±1 ms accuracy for 30 s without a clock update. Such information about a device shall be provided to the system manager by the DLMO. This is described in Table 147.

6.3.10.3 System time source selection

The device implementing the system manager role may also implement the system time source role in a network, or it may delegate the system time source role to any device(s) in the network capable of playing this role as indicated by the Device_Role_Capability attribute in the DMO of the device. The system manager shall use the Assigned_Device_Role attribute in the DMO of the device to configure it as a system time source. The system manager may select the system time source based on the clock accuracy capabilities of the device.

---

6 A list of such adjustments is maintained at ftp://maia.usno.navy.mil/ser7/tai-utc.dat .

The system time source is the ultimate source of the time sense in a D-subnet. The system time source within a D-subnet shall be accurate to within ±1 s of actual TAI time, and shall monotonically increase at a rate that tracks TAI time with a maximum error of 1×10^{-6}, i.e., the rate of increase of time shall be relatively precise, even if the time source itself is relatively inaccurate.

If multiple system time sources exist within a D-subnet, they should track each other within 0,1 ms. If 0,1 ms synchronization among D-subnet system time sources cannot be arranged, the system manager shall dictate when a device switches from one system time source to the other. The dlmo.ClockStale attribute described in 9.1.9.2.3 and 9.4.2.14 shall be used to inform the device when to switch over to the other system time source. Time propagation paths are described in 6.3.10.4. If devices are to switch system time sources, the time propagation paths can be re-arranged by the system manager as appropriate.

The system manager shall ensure that each D-subnet has at least one system time source. Some examples of system time sources include:

- In an outdoor application, the system manager may designate a few devices with global positioning system (GPS) capabilities as system time sources. Time may be propagated through the D-subnets from these sources.
- In a large network with an Ethernet backbone, the backbone itself may provide a time service that is synchronized to within 0,1 ms to a shared time reference for devices that are on the backbone. The system manager may designate the backbone routers as system time sources, and time may be propagated from these backbone routers to devices in the D-subnets.
- The system manager may periodically synchronize to a remote time source via a long-distance wired or wireless connection. This time source provides ±1 s accuracy. The system manager may then act as the system time source in the network.

If multiple system time sources exist in a D-subnet, the system manager may assign one of the system time sources as the default and the others as back-ups. The techniques for such assignment are not specified by this standard.

Clock corrections within a system time source are usually applied at a rate that can ensure that the correction does not exceed 0,5 ms in a given 30 s period. Discontinuous clock corrections are supported, with devices on a D-subnet being instructed to adjust their clocks at a specific time. This is described in 9.1.9.3.6.

6.3.10.4 Time distribution topology

In addition to system time sources, the system manager also configures clock recipients and clock repeaters in a D-subnet.

All devices in a D-subnet, except for system time sources, are configured as clock recipients, i.e., they receive periodic clock updates from one or more clock sources in their immediate neighborhoods. A clock source may be a system time source or a clock repeater.

Clock repeaters are clock recipients that also act as clock sources to certain neighbors. Clock repeaters propagate time through a D-subnet. The clock accuracy requirement for a clock repeater is described in 9.1.9.2.2.

Clock source/recipient relationships, and thus time distribution topologies, are defined by the system manager. Time propagation paths in these topologies usually match routing graphs, but this is not required. Circular time propagation paths are not allowed. Clock propagation may be arranged so that clock repeaters provide updates to their recipients soon after they themselves receive their updates.

The system manager uses the DLMO of a device to configure its clock source(s). The time of a clock recipient may be updated during each interaction with a designated clock source. The selection of clock sources and the timing of clock updates are arranged by the system manager. These clock updates are described in 9.1.9.2.

6.3.10.5 Monitoring of time synchronization accuracy

System management may support mechanisms for gathering information about the accuracy of the distributed time sense, as well as for producing an alert when the sense of time between a pair of devices varies enough to cause problems within the system. The alerts described in 9.6 may be used for this purpose by the system manager.

Vendor-specified attributes in the DMO of a device may be used for gathering such information. Vendor-specified alerts from the DMO may be used for diagnosing problems related to clock synchronization and clock maintenance.

6.3.10.6 System time service object

The system manager contains the system time service object (STSO), which shall provide the UTC accumulated leap second adjustment to the devices in the network. Other vendor-specified attributes may be added to the STSO.

The attributes of the STSO in the system manager are given in Table 25.

NOTE For more information on this leap second adjustment, see https://en.wikipedia.org/wiki/Leap_second .

Table 25 – Attributes of STSO in system manager

Standard object type name: System time service object (STSO)				
Standard object type identifier: 100				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Current_UTC_Adjustment	1	The current value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Dynamic Accessibility: Read only Default value: 35	Devices that need to convert TAI time to hh:mm:ss format need this adjustment from the system manager; units in seconds; note that the adjustment can be negative; note that UTC and TAI are based on different start dates but this difference is not covered by this attribute; on 2012.06.30 23:59:60 the value changed from 34 s to 35 s. The mechanism used by the system manager to obtain this adjustment is not specified
Next_UTC_Adjustment_Time	2	The TAI time when the UTC adjustment value will change from the current one	Type: TAITimeRounded Classification: Dynamic Accessibility: Read only Default value: See description	If the system manager knows the next time this UTC adjustment value will change, the SM is expected to indicate this time in TAI units. If the system manager does not know this time, it is expected to indicate the current TAI time and as a result the value of the Next_UTC_Adjustment attribute shall be same as the value of the Current_UTC_Adjustment
Next_UTC_Adjustment	3	The next value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Dynamic Accessibility: Read only Default value: 35	The UTC adjustment that will go into effect at the time specified by the Next_UTC_Adjustment_Time attribute
Reserved for future editions of this standard	4..63	—	—	—
NOTE UTC and TAI and GPS-time are based on different start dates but this difference is not covered by this attribute. On 2009/01/01 the Current_UTC_Adjustment changed from 33 s to 34 s. GPS-time is always 19 s behind TAI. GPS time information includes the offset needed to convert to/from UTC.				

6.3.11 System communication configuration

6.3.11.1 General

The system manager provides control of the runtime system communication configuration. It supports configuration of the network, including attributes of the protocol suite from DL to AL.

System communication configuration includes the assignment of slots, templates, and graphs to the devices in the network. The system manager should take into account the capabilities of the device in the network while configuring such assignments. When necessary, the system should be reconfigured to recover from failure scenarios.

6.3.11.2 Contract services

6.3.11.2.1 Definition of contract

System communication configuration is achieved through the contract services provided by the system manager.

A contract refers to an agreement between the system manager and a device in the network that shall involve the allocation of network resources by the system manager to support a particular communication need of this device. This device is the source of the communication messages and the device it wants to communicate with is the destination.

A contract shall establish and support the communication path between devices in the network that are compliant with this standard to support the communication need of an application process. An application process that requires communication with an application process in another device shall request a contract. Such contract requests may originate from an application process in any one of the devices compliant with this standard, such as a field device, gateway, backbone router, or system manager.

As shown in Figure 27, contracts shall be established by the system manager. They shall also be maintained, modified and terminated by the system manager. The system manager shall interact with the affected devices in the network to perform each of these operations.

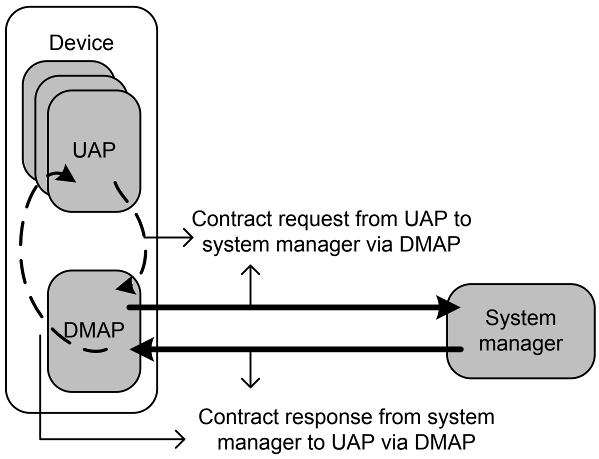


Figure 27 – UAP-system manager interaction during contract establishment

6.3.11.2.2 Directionality of contract

Contracts shall be unidirectional, i.e., a particular contract is limited to the communication from a source to a destination. For communication in the opposite direction, a separate contract shall be established.

For a two-way communication between two devices, each device shall obtain an independent contract from the system manager in order to send its messages to the other device. The peer application processes in these devices are expected to be configured such that they establish these contracts before commencing messaging in either direction. Such configurations are done either by the system manager if the application processes are the DMAPs or by host applications on plant networks or by handheld maintenance tools if the application processes are UAPs.

6.3.11.2.3 Definition of contract identifier

A contract ID (contract identifier) is a system manager-assigned identifier that shall be provided to the source after the necessary network resources have been allocated to provide the requested communication support.

The contract ID is relevant at the source, as it is used by the system manager to inform each protocol layer in the source how to treat service data units. The layers need this information before transmitting SDUs to the destination through the network. The contract requesting application process shall retrieve the assigned contract ID and shall use it to send protocol data units down the protocol suite. Protocol suite configurations at each layer for treating such upper layer PDUs shall be referenced to the contract ID. More details are provided in 6.3.11.2.9.

Contract IDs are unique only with respect to the source, i.e., the system manager may assign the same contract ID numerical value to two independent devices to support their independent contract requests. The combination of a source IPv6Address and its contract ID shall be unique across the network.

The contract ID is also relevant at the backbone router that supports communication intended for a destination in the D-subnet supported by that backbone router. This is because the DL in the backbone router needs to determine how to send the NPDU through the D-subnet to its destination. Configuration of the DL in the backbone router for treating such NPDUs shall be referenced to the combination of the source IPv6Address and its contract ID. More details are provided in 6.3.11.2.9.2.

The contract ID is not relevant at any other intermediate device along the path between the source and the destination.

While contract IDs are 2-octet values, the system manager shall restrict the assignment of contract IDs that fall within the range of 1..255 to contracts involving one or more field devices, since such devices usually have tighter memory constraints than other devices, thus enabling such field devices to store contract IDs using only one octet. Contract ID 0 is reserved to mean noContract.

6.3.11.2.4 Architecture supporting contract related messaging

6.3.11.2.4.1 General

The DMO in each device and the SCO in the system manager shall work together to provide contract related services such as contract establishment, contract maintenance and modification, and contract termination to each device.

6.3.11.2.4.2 Handling contract-related services within device

The DMO in each device shall be responsible for requesting, maintaining, modifying, and terminating each and every contract assigned to that device.

Any application process that requires a contract needs to send the request to the DMO which in turn shall send the request to the system manager. After the contract has been established, this application process shall not try to use network resources in excess of the ones allocated for this contract. After the contract has been established, this application process may later request for a modification or termination of this contract as appropriate.

The Contract_Table structured attribute of the DMO may be accessed directly or indirectly by the SCO present in the system manager.

The system manager indirectly accesses the Contract_Table structure attribute when it sends any contract related response to the device. The device shall update the Contract_Table structured attribute of the DMO every time a contract response is received from the system manager. A new entry in the Contracts_Table structured attribute of the DMO shall be created every time a contract response associated with the successful creation of a new contract is received from the system manager.

The system manager may directly read or write any element present in the Contract_Table structured attribute of the DMO once the contract entry exists in the device.

6.3.11.2.4.3 Handling contract related services in the network

The SCO in the system manager is responsible for establishing, maintaining, modifying, and terminating all contracts in the network. The SCO shall coordinate with the DMO of each device to perform these operations.

6.3.11.2.4.4 System communication configuration object

The attributes of the SCO are given in Table 26. The methods of the SCO are given in Table 27.

Table 26 – Attributes of SCO in system manager

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved for future editions of this standard	1..63	—	—	—

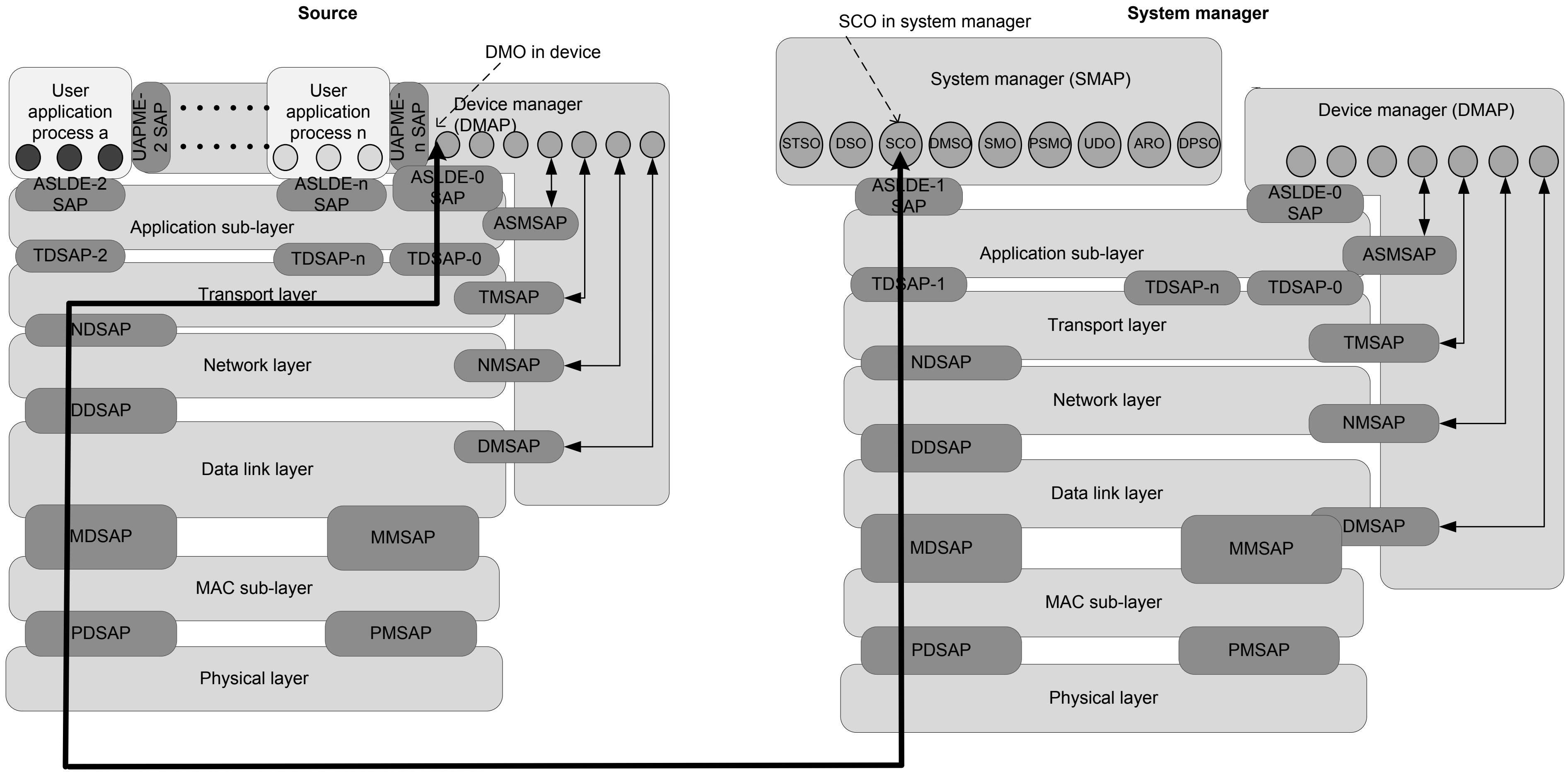


Figure 28 – Contract-related interaction between DMO and SCO

6.3.11.2.4.5 Contract-related messages

All contract-related messages between the SCO in the system manager and the DMO of the device shall be application level client/server messages (i.e., writes and reads on standard object attributes and executes on standard object methods). This is illustrated in Figure 28.

Contract-related messages include contract requests and contract responses; these are described in 6.3.11.2.5.4.

6.3.11.2.5 Contract establishment

6.3.11.2.5.1 General

Contracts shall be established by the system manager when it receives a contract request. An application process, which needs to communicate with a peer process across the network, issues a contract request to the DMAP within the device. The DMO in this requesting device shall send this contract request to the SCO in the system manager. Each contract request shall include arguments that are used by the SCO to determine the network resource allocation necessary to support this request. These arguments are discussed in 6.3.11.2.5.4.

If a device receives a service request but does not already have a contract needed in order to send the service response, it should request a contract. The device shall not send the service response until the contract response is received from the SCO of the system manager and all resources needed to support the contract are successfully configured.

The algorithms used by the SCO to determine the necessary allocation of network resources are not specified in this standard, as they are all internal to the system manager. Vendors are expected to implement algorithms in the system manager that can determine the necessary allocation of network resources for the contract requests sent by the devices being managed by that system manager.

Based on this determination, the SCO shall allocate the network resources by communicating with the necessary devices in the network and providing necessary protocol suite configurations to each one of them. This shall include the configuration of the destination and the source. Details are provided in 6.3.11.2.6. Contracts shall be established for both scheduled and unscheduled communication between applications.

As part of the configuration of the source, the SCO shall also provide the contract ID. When the source receives this configuration, it shall use this contract information to start transmitting the TSDUs that needed this communication support. After the contract has been established, the source shall not try to use network resources in excess of the ones allocated for this contract.

6.3.11.2.5.2 Relation between contracts and sessions

Sessions shall be established between the T-port in the source and the corresponding T-port in the destination. All communication between these ports shall be secured using a T-key that is issued by the security manager. Session establishment is described in 7.5. Contracts shall support the communication between peer application processes that reside on top of these T-ports in the protocol stack.

Multiple contracts may be established between these peer application processes to support different communication needs. As each application process in a device is associated with a T-port, all these contracts of these peer application processes shall use the same T-ports in the source and destination and so the same T-key shall be used for securing all the communication that occurs using these multiple contracts.

The T-key between the corresponding T-ports in the source and the destination needs to be established before a contract can be used to send messages through these T-ports. So, before the DMO sends the contract request to the SCO it is expected to check with the DSMO in the DMAP to see if a T-key exists between the corresponding T-ports in the source and the destination. If such a T-key does not exist, the DSMO is expected to send a T-key request to the security manager and obtain a new T-key. This T-key request is described in 7.6.3. If a T-key already exists between the corresponding T-ports, the DMO can send the contract request to the SCO immediately.

If an existing T-key of a particular T-port in the device is terminated for some reason by the security manager or by the device itself, any contract that uses this particular T-port will fail as the message will not get sent down the protocol stack in the device. In this case, TL is expected to send back a failure indication to the application process that generated the message. This application process in turn is expected to send a contract request to the DMO which checks with the DSMO for a T-key. The DSMO may send a new T-key request to the security manager if necessary.

6.3.11.2.5.3 Devices that can request contracts

Only a traffic source shall submit a contract request to the system manager. Other devices in the network are not allowed to submit a contract request on behalf of the source.

6.3.11.2.5.4 Contract request and response arguments

Contract request arguments are pieces of information that are provided by the contract requesting application process. They are based on the communication need that the requesting application process is interested in.

Applications should be designed such that they request only the least amount of communication support needed to satisfy their communication needs. For example, an application that needs to publish periodic messages every 1 minute should not request a 10 second period.

The contract response arguments shall include the contract ID and other information. This information is intended to provide the basic protocol suite configuration necessary at the source.

The arguments included in the contract request and response messages are described in Table 27.

Table 27 – SCO method for contract establishment, modification, or renewal

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
Method name	**Method ID**	**Method description**		
Contract_ Establishment_ Modification_ Renewal	1	Method to establish a new contract / modify an existing contract / renew an existing contract by sending request to system manager		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Contract_Request_ID	Unsigned8	A numerical value, uniquely assigned by the device sending the request to the system manager, to identify the request being made. Defaults to zero, and resets to zero. Increments with each use. This ID shall be repeated if exactly the same request is re-sent due to lack of response. Rolls over to zero
	2	Request_Type	Unsigned8	Type of contract request sent to the system manager. Named values: 0: new contract 1: contract modification 2: contract renewal Some of the input arguments below are not applicable based on this argument value; see Table 28 for details
	3	Contract_ID	Unsigned16	Existing contract ID that needs to be modified or renewed
	4	Communication_ Service_Type	Unsigned8	Type of communication service for which the contract is being requested. Named values: 0: periodic / scheduled 1: aperiodic / unscheduled Some of the input arguments below are not applicable based on this argument value; see Table 28 for details
	5	Source_SAP	Unsigned16	TDSAP in the source that will be using this contract, once it is assigned, to send application messages down the protocol stack
	6	Destination_Address	IPv6Address	The address of the device that the source wants to send application messages to; note that this information may be provided to the source during provisioning or during configuration of the application process
	7	Destination_SAP	Unsigned16	TDSAP in the destination that will be used to send these messages to the AL; note that this information may be provided to the source during provisioning or during configuration of the application process

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	8	Contract_Negotiability	BitString8	Determines if the system manager can change the requested contract to meet the network resources available and if the system manager can revoke this contract to make resources available to higher priority contracts. Named indices: 0: not revocable; 1: non-negotiable; 2..7: reserved Contract negotiability is described in 6.3.11.2.7.2
	9	Contract_Expiration_Time	Unsigned32	Determines how long the system manager should keep the contract before it is terminated; units in seconds
	10	Contract_Priority	Unsigned8	Requests a base priority for all messages sent using the contract. Named values: 0: best effort queued; 1: real time sequential; 2: real time buffer; 3: network control Contract priority is described in 6.3.11.2.7.3
	11	Payload_Size	Unsigned16	Indicates the maximum payload size in octets (represented as APDU size) that the source is interested in transmitted. Value range: 3..1 252
	12	Reliability_And_PublishDoNotAutoRetransmit	Unsigned8	PublishDoNotAutoRetransmit: Bit 0 indicates whether retransmission of old publish data is not supported because the prior buffer value is always overwritten with new publish data. Bit 0 is only applicable for periodic communication and has value 0 for aperiodic communication (see 12.12.2 for description). (Note 1) Reliability: Bits 1..7 indicate the supported reliability for delivering the transmitted APDUs to the destination. Bit 0: Unsigned1: Named values: 0: auto-retransmit; 1: do not auto-retransmit. Bits 1..7: Unsigned7: Named values: 0: low; 1: medium; 2: high

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	13	Requested_Period	Integer16	Used for periodic communication; to identify the desired publishing period in the contract request. Valid range: A value of N > 0 specifies a period of N s, while N < 0 specifies a period of -1/N s. N = 0 disables the communication
	14	Requested_Phase	Unsigned8	Used for periodic communication; to identify the desired phase (within the publishing period) of publications in the contract request. Valid range: 0..99; any other value indicates that device only cares about period and does not care about phase. See 6.3.11.2.7.4
	15	Requested_Deadline	Unsigned16	Used for periodic communication to identify the maximum end-to-end transport delay desired. Unit:10 ms
	16	Committed_Burst	Integer16	Used for aperiodic communication to identify the long term rate that needs to be supported for client/server or source/sink messages. Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid.
	17	Excess_Burst	Integer16	Used for aperiodic communication to identify the short term rate that needs to be supported for client/server or source/sink messages. Valid range: see input argument 16
	18	Max_Send_Window_Size	Unsigned8	Used for aperiodic communication; to identify the maximum number of client requests that may be simultaneously awaiting a response
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Contract_Request_ID	Unsigned8	The input argument Contract_Request_ID that was received in the corresponding contract request is used as this output argument

3779

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	2	Response_Code	Unsigned8	Indicates if the system manager was successful or not in supporting the contract request; indicates if the source can use the contract immediately or if it has to wait; also indicates if the requested communication is being supported as is or if the system manager negotiated the request down. Named values: 0: success with immediate effect; 1: success with delayed effect; 2: success with immediate effect but negotiated down; 3: success with delayed effect but negotiated down; 4: failure with no further guidance; 5: failure with retry guidance; 6: failure with retry and negotiation guidance Depending on the value of this argument, some of the output arguments below are not applicable. See Table 29 for details and 6.3.11.2.12 for failure scenarios
	3	Contract_ID	Unsigned16	A numeric value uniquely assigned by the system manager to the contract being established and sent to the source. Contract IDs are unique per device. Depending on the requested resources, multiple contract request IDs from a device may be mapped to a single contract ID. In the device, the contract ID is passed in the DSAP control field of each layer and is used to look up the contracted actions that shall be taken on the associated PDU as it goes down the protocol suite at each layer (value 0 reserved to mean no contract)
	4	Communication_Service_Type	Unsigned8	Type of communication service supported by this contract. Unsigned8: see input argument 4 Some of the output arguments below are not applicable based on this argument value; see Table 29 for details
	5	Contract_Activation_Time	TAINetwork Time	Start time for the source to start using the assigned contract
	6	Assigned_Contract_Expiration_Time	Unsigned32	Determines how long the system manager shall keep the contract before it is terminated. Units: s

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	7	Assigned_Contract_ Priority	Unsigned8	Establishes a base priority for all messages sent using the contract. see input argument 10 Contract priority is described in 6.3.11.2.7.3
	8	Assigned_Max_TSDU_ Size	Unsigned16	Indicates the maximum TSDU in octets which can be converted by the source into max APDU size supported by taking into account the TL, security, AL header and TMIC sizes. Valid range: 70..1 280. The system manager shall take into account the Max_NSDU_Size constant attribute reported by the NLMOs of the source and the destination (see Table 206) while determining the value of this argument. Fragmentation is done at the NL if the NPDU exceeds the max size of a DSDU. Fragmentation and reassembly is described in 10.2.5
	9	Assigned_Reliability_ And_PublishDoNot AutoRetransmit	Unsigned8	see input argument 12
	10	Assigned_Period	Integer16	see input argument 13
	11	Assigned_Phase	Unsigned8 Valid range: 0..99	Used for periodic communication; to identify the assigned phase (within the publishing period) of publications in the contract
	12	Assigned_Deadline	Unsigned16	Used for periodic communication; to identify the maximum end-to-end transport delay supported by the assigned contract. Unit: 10 ms
	13	Assigned_Committed_ Burst	Integer16	Used for aperiodic communication to identify the long term rate that is supported for client/server or source/sink messages. Valid range: A value of $N > 0$ specifies a mean rate of APDUs per second, while $N < 0$ specifies a mean rate of $-1/N$ APDUs per second. $N = 0$ is invalid.
	14	Assigned_Excess_Burst	Integer16	Used for aperiodic communication to identify the short term rate that is supported for client/server or source/sink messages. Valid range: A value of $N > 0$ specifies a mean rate of APDUs per second, while $N < 0$ specifies a mean rate of $-1/N$ APDUs per second. $N = 0$ is invalid.

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)				
Standard object type identifier: 102				
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	15	Assigned_Max_Send_ Window_Size	Unsigned8	Used for aperiodic communication; to identify the allowed maximum number of client requests that can simultaneously await a response
	16	Retry_Backoff_Time	Unsigned16	Used in the case of response code = failure with retry guidance or failure with retry and negotiation guidance; indicates the amount of time the source should back off before resending the contract request; units in seconds; failure scenarios are described in 6.3.11.2.12
	17	Negotiation_Guidance	BitString8	Used in the case of response code = failure with retry and negotiation guidance; indicates the Contract_Negotiability value supportable by system manager. index assignments: see input argument 8 Failure scenarios are described in 6.3.11.2.12
	18	Supportable_Contract_ Priority	Unsigned8	Indicates the base priority supportable by system manager for all messages sent using the contract. Unsigned8: see input argument 10
	19	Supportable_max_ TSDU_Size	Unsigned16 Valid range: 70..1 280	Indicates the maximum NSDU supportable by the system manager; units in octets.
	20	Supportable_Reliability_ And_PublishDoNot AutoRetransmit	Unsigned8	See input argument 12
	21	Supportable_Period	Integer16	Used for periodic communication; to identify the supportable publishing period by the system manager. Valid range: see input argument 13
	22	Supportable_Phase	Unsigned8 Valid range: 0..99	Used for periodic communication; to identify the phase (within the publishing period) of publications supportable by the system manager.
	23	Supportable_Deadline	Unsigned16	Used for periodic communication to identify the maximum end-to-end transport delay supportable by the system manager. Unit:10 ms
	24	Supportable_Committed _Burst	Integer16	Used for aperiodic communication to identify the long term rate that can be supported for client/server or source/sink messages. Valid range: see input argument 16

Table 27 *(continued)*

Standard object type name: System communication configuration object (SCO)					
Standard object type identifier: 102					
	Output arguments				
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**	
	25	Supportable_Excess_ Burst	Integer16	Used for aperiodic communication to identify the short term rate that can be supported for client/server or source/sink messages. Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid.	
	26	Supportable_Max_Send _Window_Size	Unsigned8	Used for aperiodic communication; to identify the supportable maximum number of client requests that can simultaneously await a response	
NOTE 1 The coding of this attribute is the inverse of the related attribute 7 of Table 265.					

Table 27 also contains input and output arguments that shall be used for contract modification and contract renewal. Contract modification and contract renewal are discussed in 6.3.11.2.11.

Table 27 also contains output arguments that shall be used for failure scenarios when the system manager is not able to support the contract request. These failure scenarios are discussed in 6.3.11.2.12.

Some of the input arguments in Table 27 are not applicable when the Request_Type and/or Communication_Service_Type arguments are given certain values and so shall not be included in the request. This information is provided in Table 28.

Table 28 – Input argument usage for SCO method for contract establishment, modification, or renewal

Input argument	Not applicable for	
	Request_Type value	Communication_Service_Type value
Contract_Request_ID	—	—
Request_Type	—	—
Contract_ID	0	—
Communication_Service_Type	—	—
Source_SAP	—	—
Destination_Address	—	—
Destination_SAP	—	—
Contract_Negotiability	—	—
Contract_Expiration_Time	—	—
Contract_Priority	—	—
Payload_Size	—	—
Reliability_And_PublishDoNotAutoRetransmit	—	—
Requested_Period	—	1
Requested_Phase	—	1
Requested_Deadline	—	1
Committed_Burst	—	0
Excess_Burst	—	0
Max_Send_Window_Size	—	0

Some of the output arguments in Table 27 are not applicable when the Response_Code and/or Communication_Service_Type arguments are given certain values and so shall not be included in the response. This information is provided in Table 29.

Table 29 – Output argument usage for SCO method for contract establishment, modification, or renewal

Output argument	Not applicable for	
	Response_Code value	Communication_Service_Type value
Contract_Request_ID	—	—
Response_Code	—	—
Contract_ID	4, 5, 6	—
Communication_Service_Type	4, 5	—
Contract_Activation_Time	0, 2, 4, 5, 6	—
Assigned_Contract_Expiration_Time	4, 5, 6	—
Assigned_Contract_Priority	4, 5, 6	—
Assigned_Max_TSDU_Size	4, 5, 6	—
Assigned_Reliability_And_PublishDoNotAutoRetransmit	4, 5, 6	—
Assigned_Period	4, 5, 6	1
Assigned_Phase	4, 5, 6	1
Assigned_Deadline	4, 5, 6	1
Assigned_Committed_Burst	4, 5, 6	0
Assigned_Excess_Burst	4, 5, 6	0
Assigned_Max_Send_Window_Size	4, 5, 6	0
Retry_Backoff_Time	0, 1, 2, 3, 4	—
Negotiation_Guidance	0, 1, 2, 3, 4, 5	—
Supportable_Contract_Priority	0, 1, 2, 3, 4, 5	—
Supportable_max_TSDU_Size	0, 1, 2, 3, 4, 5	—
Supportable_Reliability_And_PublishDoNotAutoRetransmit	0, 1, 2, 3, 4, 5	—
Supportable_Period	0, 1, 2, 3, 4, 5	1
Supportable_Phase	0, 1, 2, 3, 4, 5	1
Supportable_Deadline	0, 1, 2, 3, 4, 5	1
Supportable_Committed_Burst	0, 1, 2, 3, 4, 5	0
Supportable_Excess_Burst	0, 1, 2, 3, 4, 5	0
Supportable_Max_Send_Window_Size	0, 1, 2, 3, 4, 5	0

6.3.11.2.6 Protocol suite configuration

6.3.11.2.6.1 General

As part of contract establishment, the SCO shall configure the necessary devices in the network by providing necessary protocol suite configurations to each one of them. This shall include the configuration of the destination and the source, as illustrated in Figure 29.

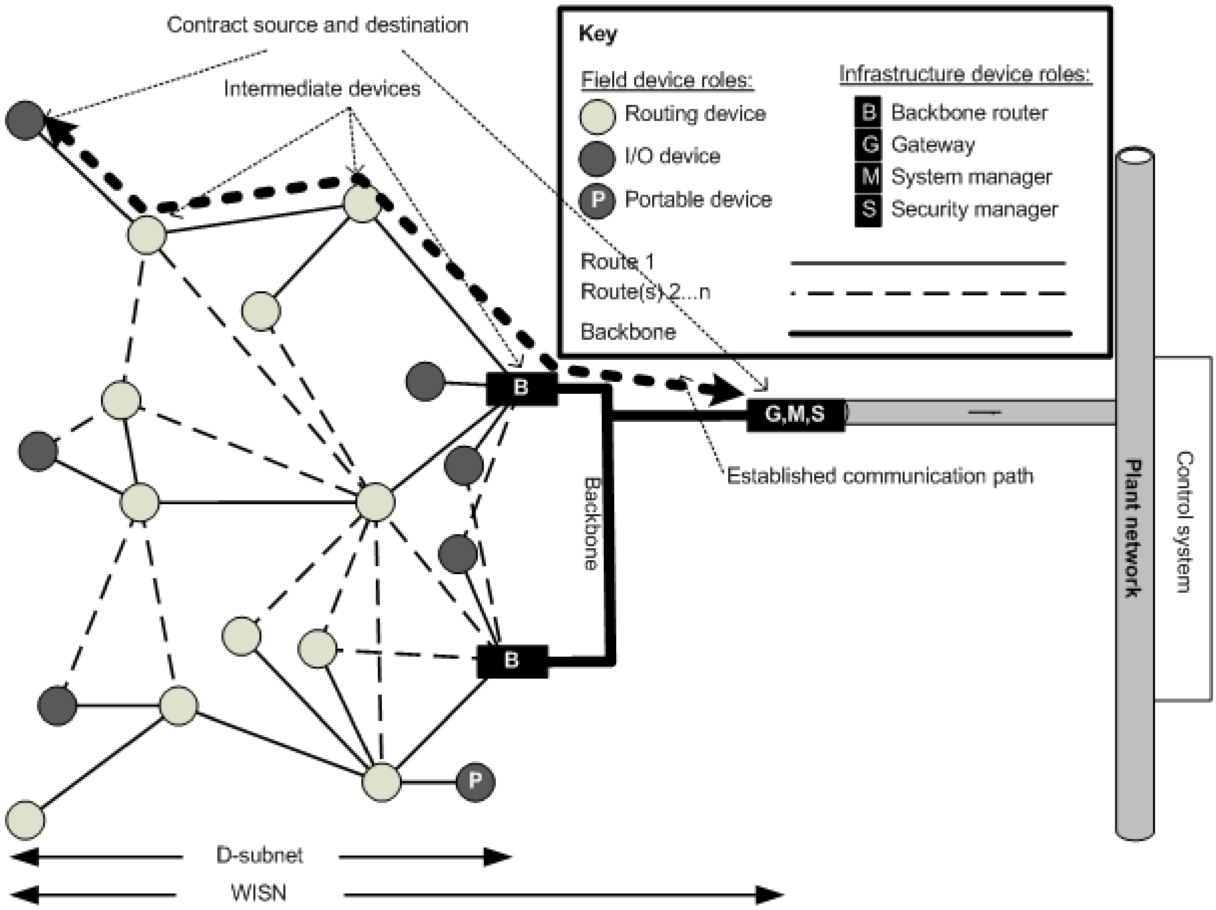


Figure 29 – Contract source, destination, and intermediate devices

Intermediate devices in the network that support the communication path being established between the source and the destination shall be configured by the SCO. Such intermediate devices along the path may include both field routers and backbone routers.

6.3.11.2.6.2 Configuration of intermediate field routers

Configuration of intermediate field routers shall be limited to the DLE in each field router, as the message from the source to the destination traverses only through the DLE of each field router along the path.

Attributes and methods defined for the DLMO of the field routers shall be used by the system manager to configure the intermediate field routers.

6.3.11.2.6.3 Configuration of intermediate backbone routers

Configuration of intermediate backbone routers shall be limited to the NL and, in some cases, the DLE in each backbone router, as the message from the source to the destination traverses through the DLE in the case of backbone routers that belong to the corresponding source and destination D-subnets, and the NLE of each backbone routers along the path.

Attributes and methods defined for the DLMO and NLMO of the backbone routers shall be used by the system manager to configure the intermediate backbone routers.

6.3.11.2.6.4 Configuration of destination

Configuration of destination shall include the configuration of all the protocol layers. The attributes and methods defined for the DLMO, NLMO, and TLMO shall be used by the system manager to configure the destination.

6.3.11.2.6.5 Configuration of source

The output arguments described in Table 27 are used at various layers of the source to determine the treatment of PDUs belonging to this contract.

The attributes and methods defined for the DLMO, NLMO, and TLMO shall be used by the system manager to configure the source.

A contract response shall be sent to the source either after all necessary network resources have been configured or after the system manager determines the time it would take to configure all necessary network resources. Depending on the situation, the system manager shall indicate if the assigned contract can be used with immediate effect or with delayed effect. The message sequence diagram in Figure 30 illustrates the case of immediate effect.

After the contract has been established, the source shall not try to use network resources in excess of the ones allocated for this contract.

6.3.11.2.6.6 Contract information in device management object

The DMO in the source shall maintain a list of all assigned contracts using the Contracts_Table attribute. This attribute shall be based on the data structure Contract_Data. When a new contract gets established, a new row shall be added to this Contracts_Table attribute with the relevant contract information. When an existing contract gets modified or terminated, the corresponding row shall be modified or deleted in this Contracts_Table attribute.

The SCO can also modify the parameters of the Contract_Table attributes by accessing them directly without exchanging entire Contract_Data structures.

The elements of the data structure Contract_Data are defined in Table 30.

Table 30 – Contract_Data data structure

Standard data type name: Contract_Data		
Standard data type code: 401		
Element name	**Element identifier**	**Element type**
Contract_ID*	1	Type: Unsigned16 Classification: Static Accessibility: Read/write Named values: 0: no contract; This element is the same as output argument Contract_ID in Table 27 * This element is used as the index field for methods described in Table 33 and Table 34.
Contract_Status	2	Type: Unsigned8 Classification: Static Accessibility: Read only Named values: 0: success with immediate effect; 1: success with delayed effect; 2: success with immediate effect but negotiated down; 3: success with delayed effect but negotiated down. This element is related to the output argument Response_Code in Table 27
Communication_Service_Type	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Named values: 0: periodic / scheduled communication; 1: aperiodic / unscheduled communication. This element is the same as output argument Communication_Service_Type in Table 27
Contract_Activation_Time	4	Type: TAINetworkTime Classification: Static Accessibility: Read/write This element is the same as output argument Contract_Activation_Time in Table 27
Source_SAP	5	Type: Unsigned16 Classification: Static Accessibility: Read/write This element is the same as input argument Source_SAP in Table 27
Destination_Address	6	Type: IPv6Address Classification: Static Accessibility: Read/write This element is the same as input argument Destination_Address in Table 27

Standard data type name: Contract_Data		
Standard data type code: 401		
Element name	**Element identifier**	**Element type**
Destination_SAP	7	Type: Unsigned16 Classification: Static Accessibility: Read/write This element is the same as input argument Destination_SAP in Table 27
Assigned_Contract_Expiration_Time	8	Type: Unsigned32 Classification: Static Accessibility: Read/write Unit: 1 s This element is the same as output argument Assigned_Contract_Expiration_Time in Table 27
Assigned_Contract_Priority	9	Type: Unsigned8 Classification: Static Accessibility: Read/write Named values: 0: best effort queued; 1: real time sequential; 2: real time buffer; 3: network control. This element is the same as output argument Assigned_Contract_Priority in Table 27
Assigned_Max_TSDU_Size	10	Type: Unsigned16 Classification: Static Accessibility: Read/write Valid range: 70..1 280 This element is the same as output argument Assigned_Max_TSDU_Size in Table 27
Assigned_Reliability_And_ PublishDoNotAutoRetransmit	11	Type: Unsigned8 Classification: Static Accessibility: Read/write Valid range: Bit 0: 0 -- i.e., always auto-retransmit (Note 1) Bits 1..7:: Named values: 0: low; 1: medium; 2: high. This element is the same as output argument Assigned_Reliability_And_PublishDoNotAutoRetransmit in Table 27
Assigned_Period	12	Type: Integer16 Classification: Static Accessibility: Read/write Valid range: A value of $N > 0$ specifies a period of N s, while $N < 0$ specifies a period of $-1/N$ s. $N = 0$ is invalid. This element is the same as output argument Assigned_Period in Table 27

Standard data type name: Contract_Data		
Standard data type code: 401		
Element name	**Element identifier**	**Element type**
Assigned_Phase	13	Type: Unsigned8 Classification: Static Accessibility: Read/write Valid range: 0..99 This element is the same as output argument Assigned_Phase in Table 27
Assigned_Deadline	14	Type: Unsigned16 Classification: Static Accessibility: Read/write Unit:10 ms This element is the same as output argument Assigned_Deadline in Table 27
Assigned_Committed_Burst	15	Type: Integer16 Classification: Static Accessibility: Read/write Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid. This element is the same as output argument Assigned_Committed_Burst in Table 27
Assigned_Excess_Burst	16	Type: Integer16 Classification: Static Accessibility: Read/write Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid. This element is the same as output argument Assigned_Excess_Burst in Table 27
Assigned_Max_Send_Window_Size	17	Type: Unsigned8 Classification: Static Accessibility: Read/write This element is the same as output argument Assigned_Max_Send_Window_Size in Table 27
NOTE 1 The coding of this attribute is the inverse of the related attribute 7 of Table 265.		

6.3.11.2.6.7 Configuration of new device

The process for a new device to join is described in 7.4. As part of the join process for a new device, a contract between the new device and the system manager shall be established.

The new device shall use the Proxy_System_Manager_Contract method defined for the DMO of the advertising router to send this contract request, which is then forwarded to the system manager, and to get the contract response from the system manager via the advertising router. The Proxy_System_Manager_Contract method is defined in Table 20. The advertising router shall use the System_Manager_Contract method defined in the DMSO for forwarding this contract request and for receiving the contract response associated with the join process of this new device. The System_Manager_Contract method is defined in Table 24. The DMSO works with the SCO to generate this contract response. When the new device gets this contract response, a new row shall be added to the Contracts_Table attribute in the DMO of the new device with the relevant contract information.

The output arguments in both these methods shall be based on the data structure New_Device_Contract_Response. The elements of the data structure New_Device_Contract_Response are defined in Table 31.

Table 31 – New_Device_Contract_Response data structure

Standard data type name: New_Device_Contract_Response		
Standard data type code: 405		
Element name	**Element identifier**	**Element type**
Contract_ID	1	Type: Unsigned16 Classification: Static Accessibility: Read/write Named values: 0: no contract This element is related to the output argument Contract_ID in Table 27
Assigned_Max_TSDU_Size	2	Type: Unsigned16 Classification: Static Accessibility: Read/write Valid range: 70..1 280 This element is related to the output argument Assigned_Max_TSDU_Size in Table 27
Assigned_Committed_Burst	3	Type: Integer16 Classification: Static Accessibility: Read/write Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid. This element is related to the output argument Assigned_Committed_Burst in Table 27
Assigned_Excess_Burst	4	Type: Integer16 Classification: Static Accessibility: Read/write Valid range: A value of N > 0 specifies a mean rate of APDUs per second, while N < 0 specifies a mean rate of -1/N APDUs per second. N = 0 is invalid. This element is related to the output argument Assigned_Excess_Burst in Table 27
Assigned_Max_Send_Window_Size	5	Type: Unsigned8 Classification: Static Accessibility: Read/write This is related to the output argument Assigned_Max_Send_Window_Size in Table 27
NL_Header_Include_Contract_Flag	6	Type: Boolean1 Classification: Static Accessibility: Read/write This is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device

Standard data type name: New_Device_Contract_Response		
Standard data type code: 405		
Element name	**Element identifier**	**Element type**
NL_Next_Hop	7	Type: IPv6Address Classification: Static Accessibility: Read/write This is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device
NL_NWK_HopLimit	8	Type: Unsigned8 Classification: Static Accessibility: Read/write This is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device
NL_Outgoing_Interface	9	Type: Unsigned8 Classification: Static Accessibility: Read/write Named values: 0: DL; 1: backbone This is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device

The new device shall use the DL information provided in the advertisement DPDU to support this contract. After the device joins the network, the system manager shall access the relevant DLMO attributes in the device to modify this DL information as appropriate. More information about this DL information and the DLMO attributes is given in 9.1.14.

If the new device is not allowed by the security manager to join the network, the security manager should inform the system manager to free up this contract and the associated network resources. When so notified, the system manager shall free up the contract and the associated network resources of such a device that is not allowed to join the network.

6.3.11.2.7 Quality of service

6.3.11.2.7.1 General

The contract assigned by the system manager to a requesting application process also indicates the quality of service (QoS) for the provided communication service. The contract establishment shall be used to reach this QoS agreement between the requesting application process and the system manager.

An application process that wants to communicate with its peer may indicate the QoS desired for this communication in its contract request. The input arguments described in Table 27 may be used for this purpose.

The input arguments Contract_Priority and Payload_Size may be used in contract requests pertaining to both periodic and aperiodic communication services. Input arguments Requested_Period, Requested_Phase and Requested_Deadline are relevant for periodic communications. Input arguments Committed_Burst, Excess_Burst and Max_Send_Window_Size are relevant for aperiodic communications. The input argument Reliability_And_PublishDoNotAutoRetransmit contains information about desired reliability which is relevant for both periodic and aperiodic communications. It also indicates if the application process wants to retransmit old periodic communication data if new data is not available.

In the contract response, the system manager shall indicate the QoS level provided for the assigned communication service. The output arguments described in Table 27 corresponding to the above mentioned input arguments shall be used for this purpose.

6.3.11.2.7.2 Contract negotiability

A source that is sending a contract request shall also indicate whether the requested communication service and the QoS are negotiable, i.e., whether the system manager can assign a contract that provides a different communication service and QoS than the ones requested if it cannot support the request as is, and whether the system manager can revoke the contract if necessary. The input argument Contract_Negotiability shall be used for this purpose.

Table 27 contains arguments that are necessary for contract negotiation between the source and the system manager. If the system manager is unable to support a contract request, it may choose to provide contract negotiation guidance. Such guidance shall be provided using the output arguments in Table 27 that start with the word supportable, e.g., Supportable_Contract_Priority.

If the system manager is unable to support the contract request at the time it was received but expects to be able to support such a request in the future, it may indicate this by using the output argument Retry_Backoff_Time.

6.3.11.2.7.3 Contract priorities and message priorities

Two priority levels shall be supported in the system, contract priority and message priority.

Contract priority shall establish a base priority for all messages sent using that contract. Four contract priorities shall be supported using 2 bits:

- Network control = 3. Network control may be used for critical management of the network by the system manager.
- Real time buffer = 2. Real time buffer may be used for periodic communications in which the message buffer is overwritten whenever a newer message is generated.
- Real time sequential = 1. Real time sequential may be used for applications such as voice or video that need sequential delivery of messages.
- Best effort queued = 0. Best effort queued may be used for client/server communications.

Message priority shall establish priority within a contract. Two message priorities shall be supported using 1 bit, low = 0 and high = 1. Another 1 bit is reserved for future releases of this standard and shall be set to 0.

Contract priority shall be specified by the application, during contract establishment time, in its contract request. It may be used by the system manager to establish preferred routes for high priority contracts and for load balancing the network. The system manager shall convey the assigned contract priority to the source in the contract response.

Message priority shall be supplied by the application for every message sent down the protocol suite. In the source, the message priority shall flow down the protocol suite. The contract priority shall be added at the NL. Contract priority shall have precedence over message priority.

Combined contract/message priority shall be used to resolve contention for scarce resources when these messages are forwarded through the network. DL shall use this information to drive queuing decisions when forwarding messages on the D-subnet. It shall be included only in the DL header. When a message is sent on a backbone, priority shall be included in the network headers. The NL shall use priority to drive queuing decisions on a backbone.

6.3.11.2.7.4 Arguments related to phase

The input argument Requested_Phase shall be used by the application process requesting the contract to request a phase which is the time offset from the beginning of a period. This time offset is expressed as a percentage of the time within a period. All periods shall be calculated such that their start times are synchronous with the beginning of TAI time. Applications may use the Requested_Phase to achieve time-synchronized, distributed loop execution with minimum latency and bounded jitter. The exact timing of the phase as it relates to the DL is specified by the link number, which is described in 9.4.3.7.

6.3.11.2.8 Contract establishment message sequence diagram

Figure 30 shows an example of a message sequence chart for the establishment of a contract. This example does not involve any timeouts and the source device accepts the contract established by the system manager even if this contract provides a different communication service than the one requested.

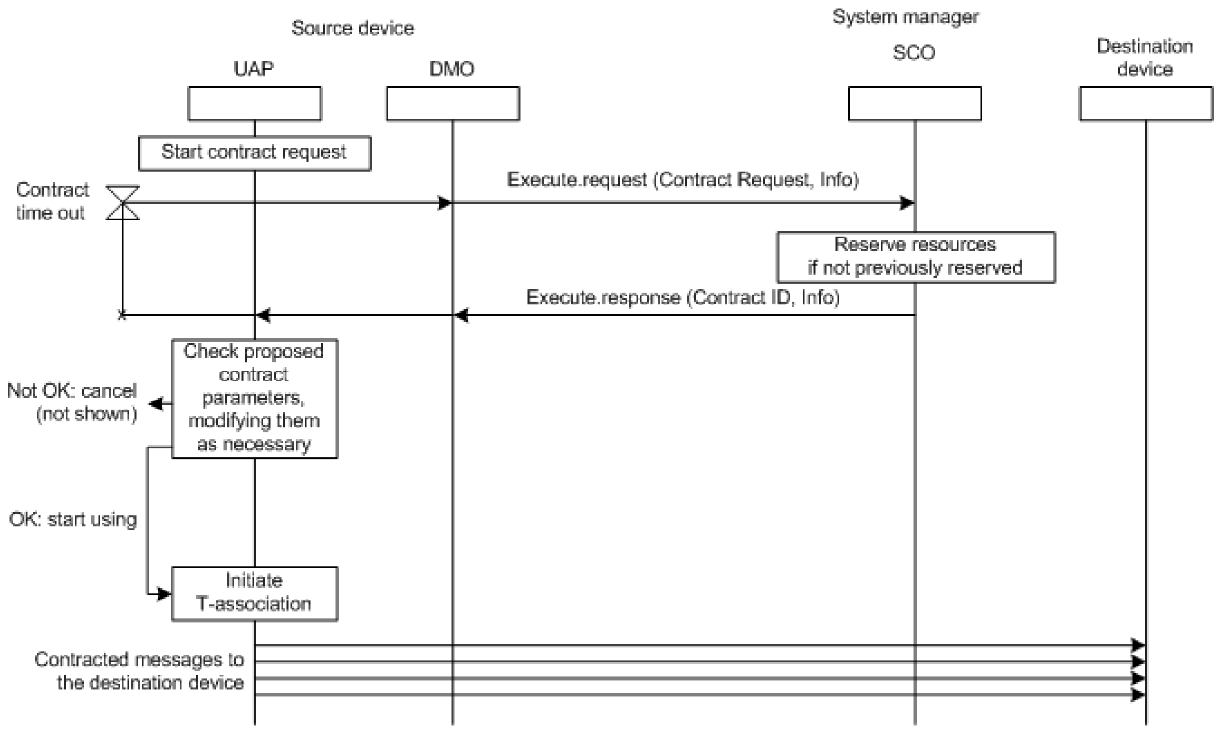


Figure 30 – Contract establishment example

6.3.11.2.9 Use of contract identifier

6.3.11.2.9.1 General

The contract ID shall be provided by the system manager to the source. The contract requesting application process shall retrieve the assigned contract ID and shall use it to send protocol data units down the protocol suite. As described previously, each layer of the source is configured for treating such upper layer PDUs that are accompanied by the contract ID, which is passed along as a DSAP control parameter.

Figure 31 illustrates how the contract ID shall be used as the data unit flows down the protocol suite of the source.

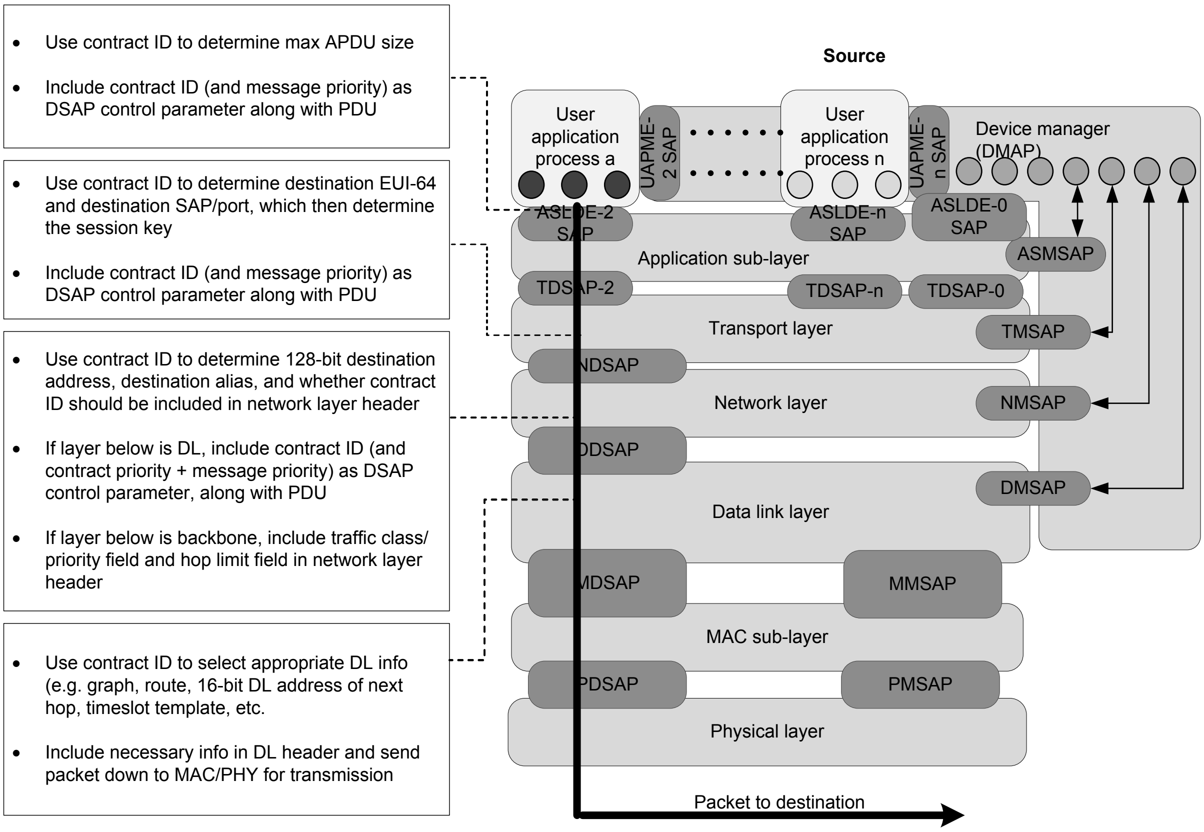


Figure 31 – Contract ID usage in source

6.3.11.2.9.2 Use of contract identifier in intermediate backbone routers

Inclusion of the contract ID in the network header of the NPDU by the source shall be configured by the system manager. If the communication path from the source to the destination goes through the backbone, then the system manager shall inform the source to include the contract ID in its network header. More details are provided in 10.5.3.

6.3.11.2.9.3 Relation between contracts and alerts

Access to the DMAP is restricted to the SMAP that resides in the system manager. In contradiction to this general principle, alert masters are allowed to access the ARMO object present in the DMAP. DMAP access by alert masters shall be limited to the ARMO, unless the alert master uses the DMAP-SMAP session established when the device joined the network. The ARMO in the DMAP shall transmit alerts that belong to the different alert categories to the respective alert masters which are described in 6.2.7.2. If these alert masters are different devices with their own unique IPv6Addresses, the ARMO shall have a separate contract with each one to communicate the alerts. The ARMO in the device requests for these contracts from the system manager through the DMO in the device.

6.3.11.2.10 Contract termination, deactivation and reactivation

6.3.11.2.10.1 General

Contracts may be terminated when the communication need that established the contract has been satisfied. Contracts may also be terminated when either the source or the destination are no longer available.

When there is a contract termination, the SCO shall inform the DMO of the source, if the source is still available. The DMO in turn informs the application process that was using this contract.

When there is a contract termination, the SCO may also free up the network resources that were allocated for supporting the contract. In addition, security information, including T-keys between the source and the destination, may also be deleted by the security manager based on interactions with the system manager.

A contract may also be deactivated if the communication need is expected to be suspended for a period of time. The contract can be reactivated when the communication need resumes.

6.3.11.2.10.2 Contract termination when a device leaves the network or is no longer available

When the system manager determines that a device is no longer part of the network, it shall terminate all the contracts associated with that device and free up the network resources that were allocated for supporting those contracts. The system manager may use information from other devices in the neighborhood of this device to decide that this device is no longer part of the network. The system manager may read the dlmo.Neighbor attribute (described in 9.4.3.4) of these neighboring devices to make this decision.

When a device that has DMO attribute Non_Volatile_Memory_Capability = 1 loses network connectivity / power cycles or goes through a warm restart for any reason, it shall maintain all necessary information related to contracts as described in 6.3.9.4.2. So, this device can resume normal operation as soon as it re-establishes time synchronization with the network. The device is expected to re-establish time synchronization by listening for advertisements or by soliciting advertisements.

If the system manager terminated all the contracts of this device while the device was not part of the network, the device is expected to be unsuccessful in resuming normal operation and so is expected to execute a restartAsProvisioned cycle. This device shall retain all the information that was provided to it during the provisioning step before it first joined the network as well as all the constant and static information present in the UAPs.

When a device that has DMO attribute Non_Volatile_Memory_Capability = 0 loses network connectivity, cycles power, or undergoes a restartAsProvisioned cycle for any reason, it is expected to repeat the join process by using the information that was provided to it during the provisioning step before it first joined the network. This device shall also retain all the constant and static information present in the UAPs.

The DMO of a device that is resetting to the factory default state or is undergoing a restartAsProvisioned cycle shall terminate all its contracts by using the method defined in Table 27 before resetting or restarting.

6.3.11.2.10.3 Contract termination when the T-key is terminated

The system manager may terminate a contract of a particular device if it is informed by the security manager that a corresponding T-key of that device has been terminated. Any contract of the device that uses the T-port corresponding to this particular T-key may be terminated.

6.3.11.2.10.4 Devices that can terminate, deactivate and reactivate contracts

Only the source or the system manager shall have the ability to terminate an existing contract of the source.

Only the source shall have the ability to deactivate and reactivate an existing contract of the source.

6.3.11.2.10.5 Contract termination, deactivation, and reactivation request and response arguments

If the source decides to terminate a contract, it shall send a contract termination request to the SCO. The SCO shall then send the response back to the source informing it that the contract has been terminated. The request shall be an Execute.Request to the SCO with the contract ID as one of the input arguments, and the response shall be an Execute.Response with status as the output argument. This is described in Table 32.

If the source decides to deactivate / reactivate a contract, it shall send a contract deactivation / reactivation request to the SCO. The SCO shall then send the response back to the source informing it that the contract has been deactivated / reactivated. The request shall be an Execute.Request to the SCO with the contract ID as one of the input arguments, and the response shall be an Execute.Response with status as the output argument. This is described in Table 32.

Table 32 – SCO method for contract termination, deactivation and reactivation

<table>
<tr><th colspan="5">Standard object type name:System communication configuration object (SCO)</th></tr>
<tr><th colspan="5">Standard object type identifier: 102</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="9">Contract_Termination _Deactivation_Reactivation</td><td>2</td><td colspan="3">Method to terminate, deactivate or reactivate a contract</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Contract ID</td><td>Unsigned16</td><td>ID of contract being terminated, deactivated or reactivated</td></tr>
<tr><td>2</td><td>Operation</td><td>Unsigned8</td><td>Named values:
0: contract termination;
1: contract deactivation;
2: contract reactivation.</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Error</td><td>Unsigned8</td><td>Named values:
0: success;
>0: failure</td></tr>
</table>

If the system manager decides to terminate a contract, it shall send a contract termination command with the contract ID to the DMO of the source. The DMO shall then return a response with the status.

The DMO method to notify an application that an existing contract has been terminated is described in Table 33.

Table 33 – DMO method to notify of contract termination

Standard object type name: Device management object (DMO)		
Standard object type identifier: 127		
Method name	**Method ID**	**Method description**
Contract_Terminated	1	Method to notify an application of the termination of an existing contract, as found in the Contracts_Table attribute in Table 10. This method uses the Delete_Row method template defined in Table J.5 with the following arguments: Attribute_ID: 26 (Contracts_Table) Index_1: 1 (Contract_ID)

6.3.11.2.10.6 Protocol suite configuration

When the SCO terminates a contract, in addition to informing the source about the termination, it may also free up the network resources that were allocated in the source, destination, and intermediate devices. Procedures similar to those used for protocol suite configuration during contract establishment (see 6.3.11.2.6) may be used by the SCO to free up these network resources.

The SCO informs the security manager through the PSMO about the contract termination. The security manager may decide to delete the T-key that has been assigned for the communication between the source and the destination. In this case, the security manager shall send T-key delete messages to the source and the destination through the PSMO.

6.3.11.2.10.7 Contract termination message sequence diagram

Figure 32 shows the message sequence chart for termination of a contract initiated by the system manager.

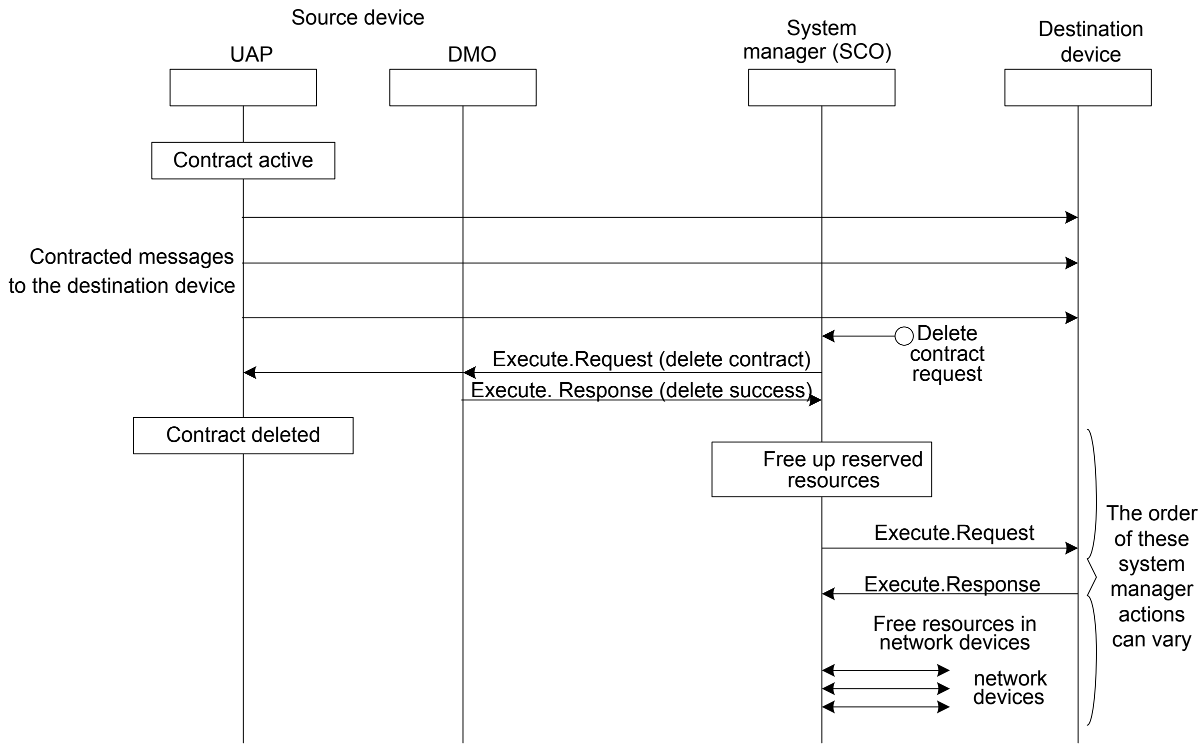


Figure 32 – Contract termination

6.3.11.2.11 Contract maintenance and modification

6.3.11.2.11.1 General

The SCO needs to maintain established contracts by ensuring that the allocated network resources are available under normal conditions. If the allocated network resources become unavailable, the SCO may choose to allocate alternate network resources in order to continue to maintain the established contract.

A contract may be modified if the communication need of the corresponding application (supported by that contract) changes. A contract may also be modified if the system manager decides to change the network resources allocated for the contract.

Contract modifications fall into two categories:

- modifications resulting in a reduction of the allocated network resources, and
- modifications resulting in a change or increase of allocated network resources.

For application-initiated contract modifications, these two categories follow slightly different steps.

Contract modifications that result in a reduction of the allocated network resources may go into immediate effect, i.e., the source may start using the protocol suite configuration of the modified contract as soon as it receives the response along with this configuration information from the SCO if this response indicates so in the Response_Code output argument.

Contract modifications that result in an increase or change of the allocated network resources shall not go into immediate effect, i.e., the source shall not start using the protocol suite configuration of the modified contract as soon as it receives the response along with this configuration information from the SCO. This is because the SCO still needs to increase or change the allocation of network resources. The response from the SCO shall include an Activation_Time output argument that indicates to the source when it can start using the new protocol suite configuration. This results in a delayed effect.

6.3.11.2.11.2 Devices that can modify contracts

Only the source or the system manager shall have the ability to modify an existing contract of this source.

6.3.11.2.11.3 Contract modification request and response arguments

If the source decides to modify a contract, it shall send a contract modification request to the SCO. The SCO shall then send a response back to the source informing it that the contract has been modified. The request shall be an Execute.Request to the SCO, and the response shall be an Execute.Response message. The input and output arguments are provided in Table 27. The SCO may also communicate with relevant devices to allocate or de-allocate the necessary network resources.

If the system manager decides to modify a contract, it shall send a contract modification command to the DMO of the source by using the Modify_Contract method. The DMO shall then send a response back with the status. The SCO may also communicate with relevant devices to allocate or de-allocate the necessary network resources.

The DMO method to notify an application that an existing contract has been modified is described in Table 34.

Table 34 – DMO method to notify of contract modification

Standard object type name: Device management object (DMO)		
Standard object type identifier: 127		
Method name	**Method ID**	**Method description**
Contract_Modified	2	Method to notify an application of the modification of an existing contract, as found in the Contracts_Table attribute in Table 10. This method uses the Write_Row method template defined in Table J.3 with the following arguments: Attribute_ID: 26 (Contracts_Table) Index_1: 1 (Contract_ID)

6.3.11.2.11.4 Contract renewal

Contract renewal is equivalent to a simple contract modification, with only the Contract_Expiration_Time input argument being updated and all other input arguments being the same as those in the original contract request.

6.3.11.2.11.5 Protocol suite configuration

As part of contract modification, the SCO shall configure / re-configure the necessary devices in the network by providing necessary protocol suite configurations to each of them. This shall include the re-configuration of the destination and the source. Procedures similar to the ones used for protocol suite configuration during contract establishment (see 6.3.11.2.6) may be used by the SCO for this purpose.

6.3.11.2.11.6 Contract modification message sequence diagram

Figure 33 shows the message sequence chart for modifying a contract with immediate effect.

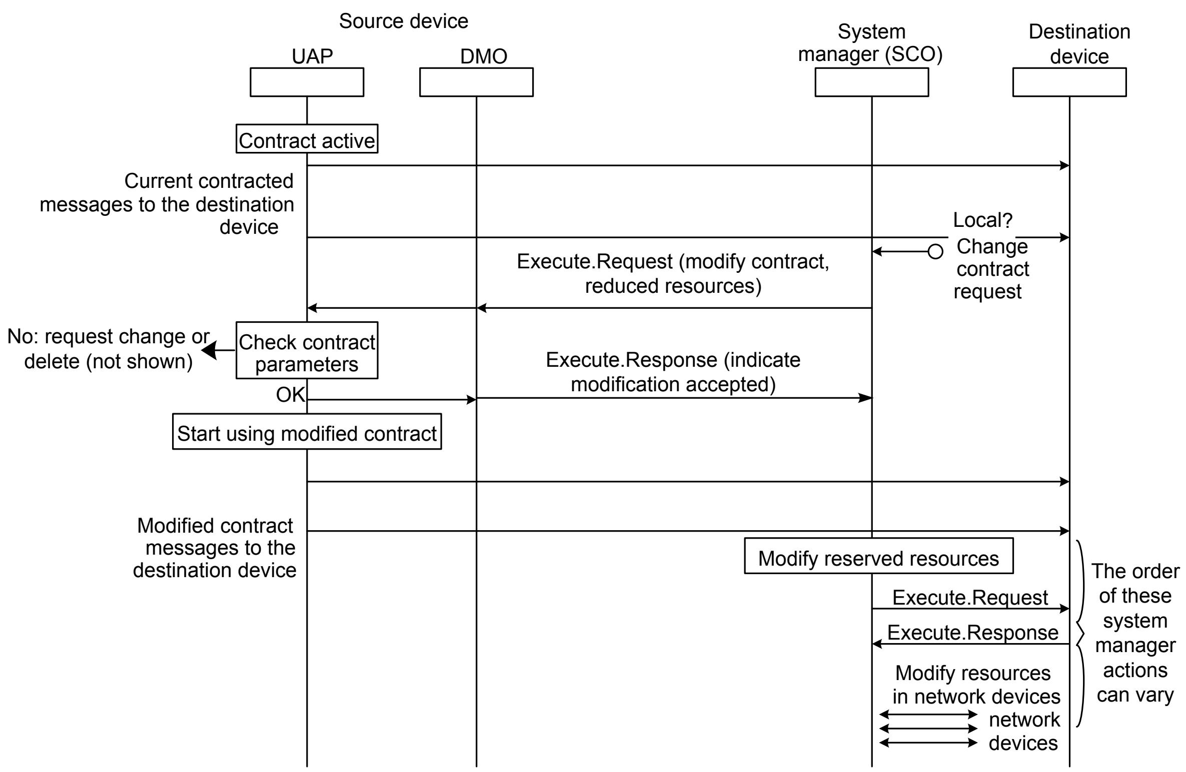


Figure 33 – Contract modification with immediate effect

6.3.11.2.11.7 Contract modification and T-key updates

T-key updates are not treated as contract modifications. Such key updates shall be sent from the security manager, through the proxy security management object (PSMO) in the system manager, to the relevant devices that have the corresponding session.

6.3.11.2.12 Contract failure scenarios

Table 27 contains output arguments for failure scenarios in which the system manager is not able to support the contract request. Such failures may occur if the requested communication service cannot be supported at all, if it cannot be supported due to a temporary condition, or if it cannot be supported unless the request is resent by the source with arguments negotiated down. In such cases, the system manager may choose to include output arguments in the response that provide some guidance to the source. These include Retry_Backoff_Time and Negotiation_Guidance.

6.3.12 Redundancy management

Although this standard incorporates features that provide for both simplex and fully-redundant wireless connection from field devices to a backbone plant network, the management of that redundancy is not specified in this standard.

The system manager is expected to be capable of configuring path redundancy in the D-subnet through the field routers. Field devices, including field routers, can be configured to communicate with redundant backbone routers.

Device-level redundancy that requires synchronization between the redundant devices to maintain state information is allowed, but is not specified in this standard.

6.3.13 System management protocols

Management-related communication between devices compliant with this standard and the system manager shall be accomplished via standard application sublayer messaging, as described in 12.12.

6.3.14 Management policies and policy administration

Management policies and policy administration are not specified by this standard. A default policy may be established to make all device information available to the system manager (with appropriate security). Overview information may be made available outside the network (e.g., the network is operating within nominal limits).

6.3.15 Operational interaction with plant operations or maintenance personnel

While the device implementing the system manager may have an interface that allows plant operations and maintenance personnel to observe and control the performance of the network and devices, this interface is neither mandatory nor is it specified by this standard.

7 Security

7.1 General

Clause 7 describes the security component functionality, its interface with the DLE and the TLE, and the protection of data in transit. It also describes the security manager role.

The primary focus of Clause 7 is to provide transmission security and related security aspects including the join process, session establishment, key updates, and associated policies. This standard does not address other types of security, such as security of data-at-rest or physical device security.

The specific messages that are protected are single-hop (hop-by-hop) DPDUs, end-to-end transport TPDUs, and security management data structures when conveyed in APDUs. A steady-state data flow using DPDUs and TPDUs that may be protected is outlined in Figure 34. The TLE endpoints of a T-security association are defined by the endpoint devices as well as the end application.

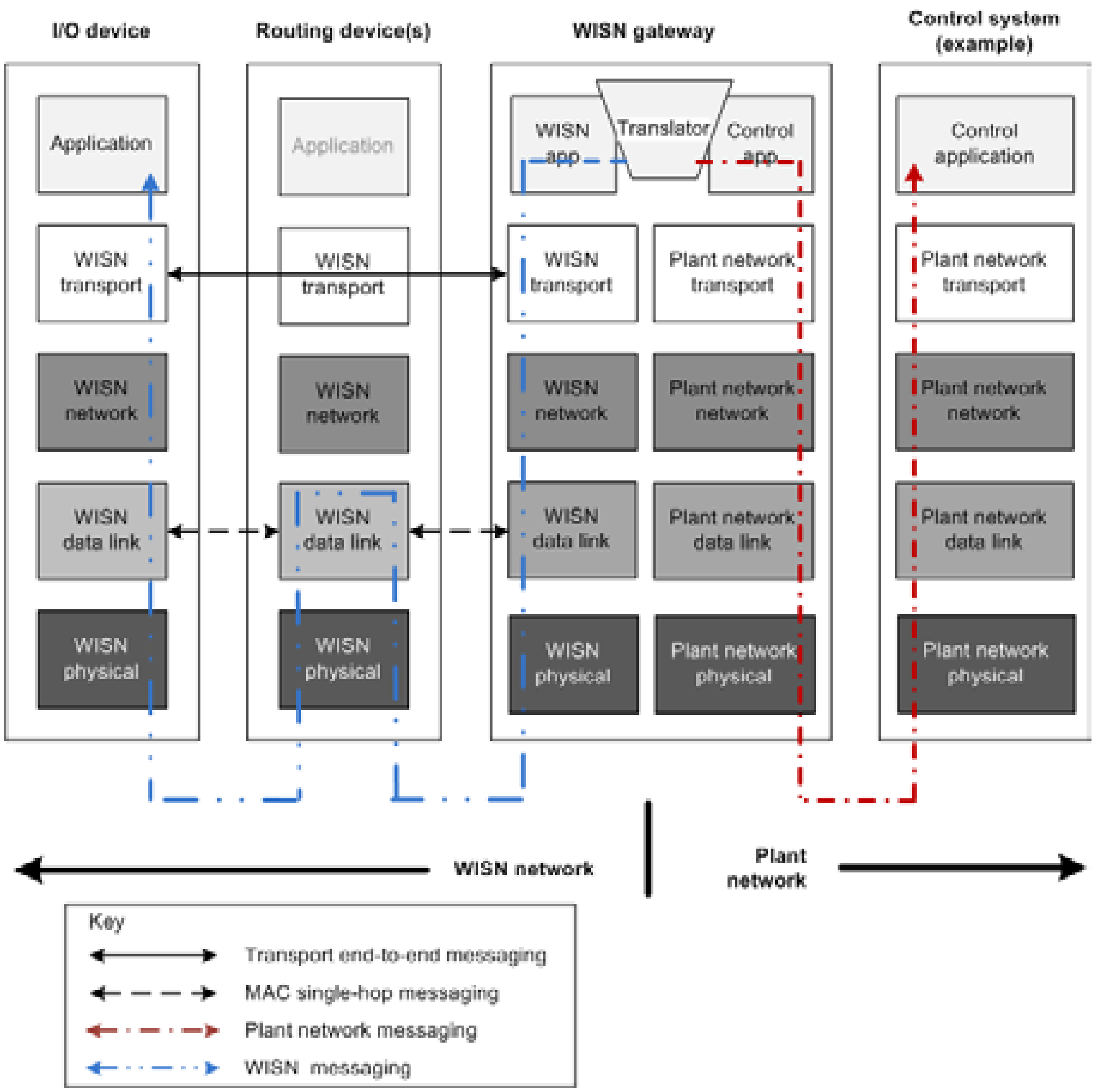


Figure 34 – Examples of DPDU and TPDU scope

7.2 Security services

7.2.1 Overview

The security services in this standard are selected by policy. The policy is distributed with each cryptographic material, permitting focused policy application. Since a single key is used at a time at the DL, except for a brief period of key switchover, the entire sub-network is subject to the same policies at the DL. The security manager controls the policies for all the cryptographic materials it generates.

Devices with appropriate credentials participate in secured communications with other such devices through the use of a shared-secret symmetric key that is used to authenticate and, by security configuration, to encrypt their messages to each other.

NOTE 1 Although authentication involves use of an encryption primitive, it does not result in confidentiality of the message contents; a separate encryption process (pass) is required for message content confidentiality.

The security services are applied at the bottom of the communication protocol stack, hop-by-hop at the DL, and at the top of the communication protocol stack, end-to-end at the TL. Security management services are also used by the AL for the join process, key distribution and session management. When secret keys are used, DL security defends against attackers which are outside the system and do not share system secrets, while TL security defends against attackers which may be on the network path between the source and the destination.

In both cases, a symmetric data key (also known as a T-key), shared among intended communicants, is used to add a cryptographically-hard, keyed, message integrity check (MIC) to the PDU and, when so specified, to provide confidentiality (via encryption) of the PDU's payload. Attackers that do not share the key cannot modify the message without a very high probability of detection and cannot decrypt any encrypted payload.

The security operation is based on a shared sense of time that usually is aligned with TAI time (see 5.6). The sending DLE and TLE authenticate to their receiving peers using the nominal TAI time of DPDU transmission and the approximate time of TPDU creation.

When three or more devices share a common secret key, source authentication is no longer guaranteed within that group because of the shared symmetric key. In this case, intra-group source authentication requires complex mechanisms; thus, authentication of the specific sending node (within the multicast group) is not addressed.

The primary security components of the provided services include:

- authorization of secure communications relationships between entities;
- message authenticity, ensuring that messages originate from an authorized member of a communications relationship and that they have not been modified while in transit between originator and receiver by an entity outside of the relationship;
- assurance that delivery timing and message reordering does not exceed anticipated bounds;
- data confidentiality that conceals the contents (other than size) among message payloads; and
- protection against malicious replay attack.

Various combinations of these services are provided to both a DLE and a TLE. Additionally, various cryptographic services are available for use by the DSMO for the join process, session establishment and key update.

NOTE 2 Protection against compromise of the cryptographic boundaries inside the hardware of devices compliant with this standard is beyond the scope of this standard. Other publications, including ISO/IEC 15408 and ISO/IEC 19790 (similar to the [US] NIST FIPS 140 series), address those issues. Compliance decisions are left to those who evaluate devices.

7.2.2 Keys

7.2.2.1 General

Symmetric keys are used for data encryption and authentication; see 7.3.2.5, 7.3.2.6, 7.3.3.8, and 7.3.3.9. Asymmetric keys can be used for the join process, see 7.4. Each key is limited in time and can be updated. Figure 35 shows the types of keys specified by this standard and their associated lifetimes, including an asymmetric-key security certificate (should one exist).

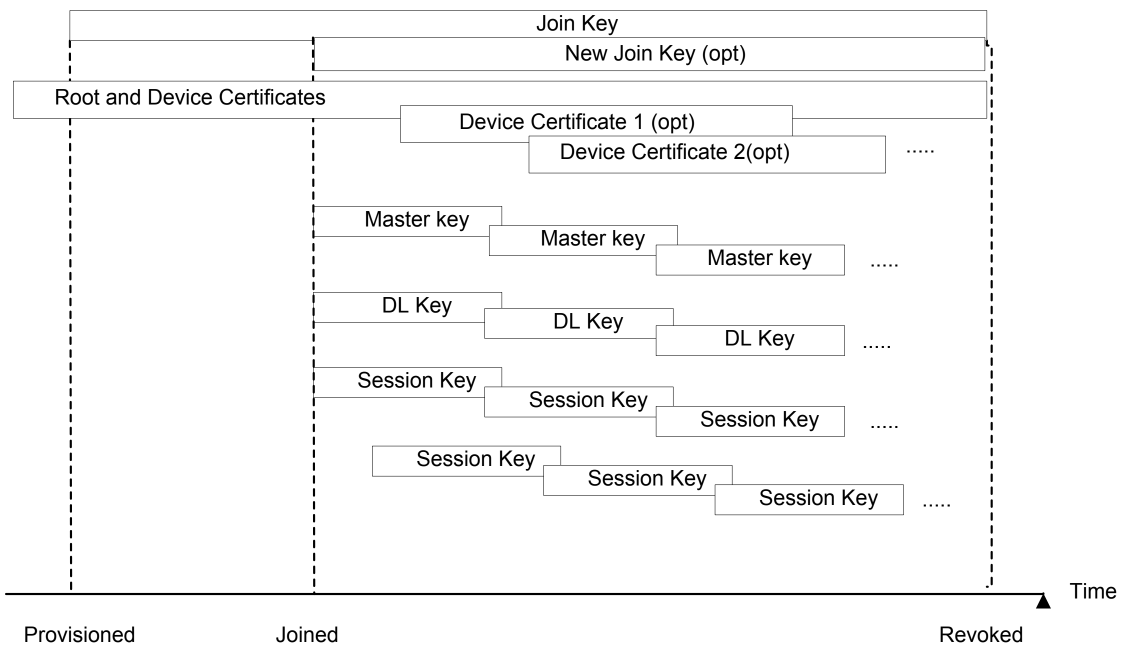


Figure 35 – Keys and associated lifetimes

7.2.2.2 Symmetric keys

All WISN symmetric keys shall be 128-bit values. The symmetric keys used include:

- Global key: a well-known key that cannot be used to guarantee any security properties and which never expires.
- K_open: a global key used as the join key in the provisioning step described in 13.3. The actual value for this key is 0x004F 0050 0045 004E 0000 0000 0000 0000, which is the representation of the null-terminated 16-octet Unicode string “OPEN(null)(null)(null)(null)”. The crypto key identifier for this key is 1.
- K_global: a global key used as the join key in the provisioning phase, and as the D-key in the joining phase. Use of this key in the provisioning phase is described in 13.3. The actual value for this key is 0x0049 0053 0041 0020 0031 0030 0030 0000, which is the representation of the null-terminated 16-octet Unicode string “ISA(space)100(null)“. The crypto key identifier for this key is 0.
- Join key (K_join): a key received at the conclusion of the provisioning step, is used to join a network for which the device was provisioned. The default value of the K_join key is the same as the default value of K_global.
- Master key: a key derived at the conclusion of a key agreement scheme, which is used as a KEK for communication between the security manager and the device, as well as a basis for deriving other keys. This key expires and needs to be updated periodically.
- D-key: a key used to encrypt/decrypt and/or authenticate DPDUs. This key expires and needs to be updated periodically.
- T-key: a key used to encrypt/decrypt and/or authenticate TPDUs. That key expires and needs to be periodically updated.

7.2.2.3 Asymmetric keys and certificates

Support of asymmeric cryptography is a device construction option.

All WISN asymmetric keys shall have a cryptographic strength of at least 128 –bits. The asymmetric keys used include:

- CA_root: The public key of the certificate authority that signed the device's asymmetric-key certificate. This key is commonly referred to as a root key; it is used in verifying the true identity of the device communicating the certificate, as well as some related keying information.
- Cert-A: The asymmetric-key certificate of device A, used to evidence the true identity of the device, as well as related keying information. It is used during execution of an authenticated asymmetric-key key establishment protocol.

The description of the asymmetric-key cryptographic material is provided in H.3.

7.2.2.4 Key lifetime

7.2.2.4.1 General

Symmetric keys are limited by a lifetime and should be invalidated after the lifetime expires. To maintain security of ongoing communications, the current keys are updated. In this specification, the key lifetimes (and related information) are defined as follows:

ValidNotBefore: TAI time at which a key will be enabled;

ValidNotAfter: TAI time after which a key will become invalid;

SoftExpirationTime: TAI time when a device should prepare for updating a key;

HardLifeSpan: Relative duration from *ValidNotBefore* to *ValidNotAfter*;

KeyExchangeMargin: Minimum time required to complete a key update cycle.

NOTE 1 Since the above are used herein as variables in formulae they use the typeface for variables.

The relationship of the above lifetime definitions is illustrated in Figure 36. The key update mechanism using those time definitions is described in 7.6.

The special value 0xFFFF FFFF is used to designate keys that never expire, which is used for Global keys specified in 7.2.2.2. Thus any compution of the expiration time of a key shall increment a result value of 0xFFFF FFFF to 0x0000 0000. Similarly, any logic that determines whether a key has expired because the key's expiration time is in the near past shall determine that expiry has not occurred when that value for that expiration time is 0xFFFF FFFF.

NOTE 2 DL, TL and KEKs / master keys are safer if they do expire, since keys that do not expire increase the system's vulnerability to prolonged observation and attack.

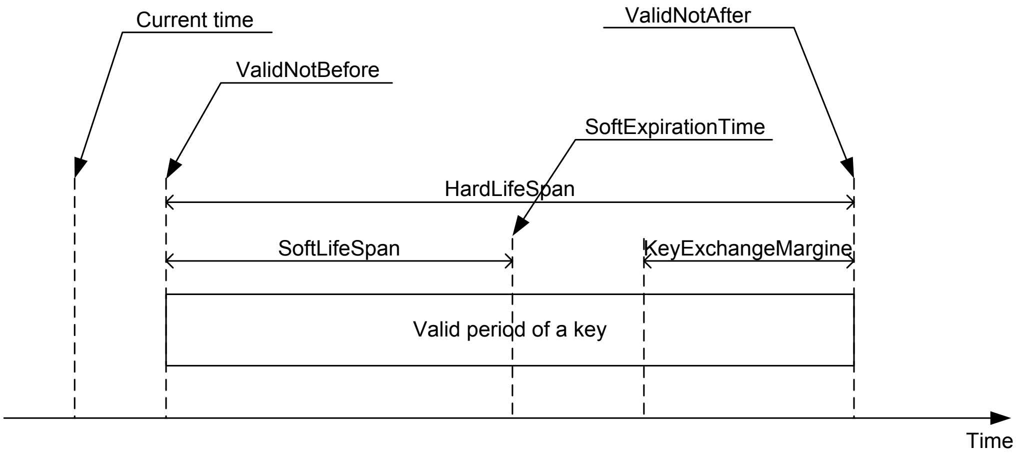


Figure 36 – Key lifetimes

NOTE 3 A key used after its hard lifetime can make communications vulnerable to replay attacks.

Asymmetric-key certificates should have a lifetime (*ValidNotBefore* and *ValidNotAfter*), as defined in 7.4.6.2.1.1.

KeyExchangeMargin can be used as a trigger for invoking the PSMO.Key_Update_Request() method to keep the continuous secure session. It is recommended that *KeyExchangeMargin* is set to “5 times DSMO.pduMaxAge” seconds, consisting of:

- 2 × DSMO.pduMaxAge seconds for a Security_New_Session() method round-trip communication;
- 2 × DSMO.pduMaxAge seconds for a New_Key() method round-trip communication; and
- another DSMO.pduMaxAge seconds for processing time.

7.2.2.4.2 Key lifetime expiration

7.2.2.4.2.1 SoftExpirationTime

When the *SoftExpirationTime* is past, the device owning the key prepares to get a new key from the security manager. The device may call the PSMO.Security_New_Session() method on the system manager to explicitly request a key, or it can wait to have its DSMO.New_Key() method called by the security manager. If the device wants to be certain about updating a key, it should call the PSMO.Security_New_Key() on the system manager method explicitly.

It is not necessary for the device to start the key update process immediately after *SoftExpirationTime*. The key update can be accomplished at any time up to *ValidNotAfter*. To keep the current secure session with a peer, a request for a new key can be issued at some point between *SoftExpirationTime* and *ValidNotAfter*. The device should call the PSMO.Security_New_Key() method before (*HardExpirationTime* – *KeyExchangeMargin*).

7.2.2.4.2.2 ValidNotAfter

The key shall not be used in active communication after *ValidNotAfter* and should be zeroized by all devices using the key. However, the key can be archived in a secure manner, depending on system key archiving policy.

7.3 PDU security

7.3.1 General

7.3.1.1 Security level

The security level specifies the method to be applied to certain PDUs. The security level consists of a combination of the MIC size (0 bits, 32 bits, 64 bits or 128 bits) and whether the associated PDU payload is to be encrypted or not. Table 35 shows the security levels used in this specification and their corresponding security attributes.

Table 35 – Security levels

Security level value	Security attributes	Where usable
0	none	TPDU
1	MIC-32	Data DPDU, ACK/NAK DPDU, TPDU
2	MIC-64	Data DPDU, TPDU
3	MIC-128	TPDU
4	ENC-only	never
5	ENC-MIC-32	TPDU, Data DPDU
6	ENC-MIC-64	TPDU, Data DPDUL
7	ENC-MIC-128	TPDU

NOTES

PhPDU size constraints and loss rates dictate the ACK/NAK DPDU restriction to MIC-32 and the Data DPDU restriction to MIC-32, ENC-MIC-32, MIC-64, or ENC-MIC-64.

ACK/NAK DPDUs do not contain a payload field to which the ENC operation could apply.

ENC-only is excluded because it is not possible to determine whether the eventual decryption is correct.

7.3.1.2 Security control field

The security control field is part of each DL and TL security header. Its value specifies the presence of the key identifier and the security level to be applied to the PDU. The SecurityControl field octets shall conform to IEEE 802.15.4:2011, 7.4.1.

Table 36 shows the structure of the security control field.

Table 36 – Structure of the security control field

Octet	Bits						
	7	6	5	4	3	2	1	0
1	Reserved			Crypto key identifier mode		Security level		

The CryptoKeyIdentifierMode field encodes the size of the CryptoKeyIdentifier field that immediately follows the SecurityControl field in the PDU. If the key identifier mode is set to 0, the following CryptoKeyIdentifier field is elided.

The security level field shall consist of 3 bits as defined in IEEE 802.15.4:2011, Table 58, and summarized in Table 35 of this standard. The security level 0x04, corresponding to encryption only, shall never be used for a TPDU, or the first DPDU of a D-transaction, of this standard. The security level of 0x00, corresponding to no protection shall never be used for a DPDU in this standard.

NOTE ENC, encryption-only, does not provide any protection against an active attacker, because such an attacker is able to arbitrarily complement selected bits of any PDU in transit. Without a cryptographically-difficult-to-forge integrity field, there is no secure method for the recipient to detect such a change, and thus any active attacker can easily fabricate a malicious PDU.

7.3.2 DPDU security

7.3.2.1 General

The degree to which a device is permitted to participate in a D-subnet shall be determined by system policy applied to credentials supplied by the device. Devices without credentials shall be permitted full, limited, or no participation beyond join attempts, as determined by system policy for such devices.

All DPDUs include security fields and a cryptographically-strong DMIC. The details of the cryptographic building blocks are in Annex H. In non-secure mode, the key distributed might have traveled over an insecure channel. When a properly secured secret D-key is used, the following security services are always provided:

a) DPDU source-set authentication;

b) DPDU integrity; and

c) proof that the DPDU was received at the intended time, providing rejection of DPDUs
   - that were not sourced by a device within the network that shares an appropriate data key, or
   - that were not received within an acceptable time window relative to their nominal time of formation or transmission, or
   - that were previously received.

NOTE 1 Authorization is implied by the fact that the sending device has knowledge of a shared symmetric data key. When the key is not a shared secret, the authorization extends to all possible devices through the use of the global key. When the key is a shared secret, an inference is available that the sending device obtained the shared-secret key from a security manager, and that it would have obtained the key only if the security manager's authorization database permitted the resulting protected communications relationship. Such permission usually is based on the device's role. See Device_Role_Capability (standard object type identifier 127, attribute identifier 4, in Table 10) for a definition of the roles and their respective bitmap.

NOTE 2 The detection of reception at an inappropriate time renders ineffective attacks on the MAC message stream that are based on DPDU delay, reordering, or replay, since the transmission duration of each DPDU is greater than the 1 ms window within which such reordering would be undetectable.

NOTE 3 This service uses the sender's time of transmission, the receiver's time of reception, and the fact that MAC transmission and reception are highly concurrent to ensure that any DPDU received at an unintended time, including DPDU replay or DPDU stream reordering, will be detected and the anachronistic DPDU(s) rejected.

The amount of redundancy (i.e., DMIC size) that is used to provide DPDU integrity is selected by policy associated with the relevant data key.

The following additional DL security service is selectable by policy associated with the relevant D-key:

d) DPDU payload confidentiality (i.e., encryption).

This confidentiality service shall not be offered with the K_open and K_global keys specified in 7.2.2.2, because use of these keys with their well-known constant values renders confidentiality impossible.

7.3.2.2 DPDU structure

The structure of a DPDU is described in 9.3.1 and outlined in Figure 88 and Figure 37 in this standard, with the DSDU possibly encrypted and the MHR, DHR and DSDU protected by the DMIC.

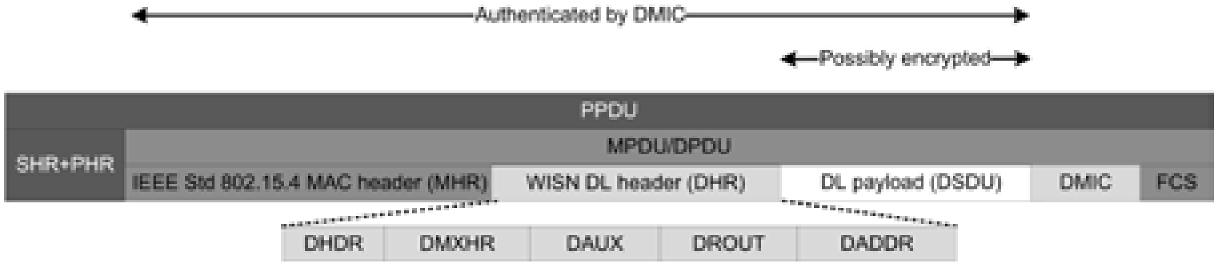


Figure 37 – DPDU structure

The complete DPDU from the start of the MHR to the end of the DSDU shall be protected by the DMIC. Information relevant to the DSC is the DL's MAC extension header (DMXHR) as outlined in Table 112, the 8-bit DPDU sequence number as outlined in Table 110, and the PhPDU's channel number (in the range 0..15).

7.3.2.3 DPDU headers

7.3.2.3.1 IEEE 802.15.4:2011 MAC header

DPDU security is provided by the protocol stack defined in this standard, above the IEEE 802.15.4 MAC sublayer. The MAC header is defined in 9.3.3.2.

7.3.2.3.2 DL MAC extension header

The DMXHR outlined in Table 112 shall contain 2 fields used by the security layer. The first field shall contain the security control field as outlined in 7.3.1.2. The second field shall contain the Crypto Key Identifier as specified in IEEE 802.15.4:2011, 7.4.3. In the DMXHR, the Crypto Key Identifier shall never be elided with the Crypto Key Identifier Mode = 0.

The default value of the security level for the DL shall be set to 1 (MIC-32), corresponding to authentication only with a DMIC size of 32 bits.

For the DPDU processing steps, the following constraints shall be observed:

- DMIC sizes of 0 bits and 128 bits are prohibited, therefore prohibiting DPDU security levels of 0 (none), 3 (MIC-128), 4 (ENC-only) and 7 (ENC-MIC-128);

  NOTE 1 MIC-64 provides adequate protection for Data DPDUs, given their small maximal size. That size constraint makes MIC-128 problematic, whereas the error rate of the underlying PhPDUs dictates that some MIC be used for additional DPDU integrity. A MIC also provides statistical protection against spoofing by an attacker that does not know the relevant symmetric enryption key.

  NOTE 2 ENC-only is not useful because it is not possible on receipt to determine that the DPDU is received unchange.

- ACK/NAK DPDUs shall use only 32-bit DMICs, regardless of the security level of the Data DPDU of a D-transaction.

  NOTE 3 MIC-32 provides adequate protection for ACK/NAK DPDUs, given their minimal size and regulatory constraints on the duration of short control signaling (SCS), for which they qualify. ACK/NAK DPDUs carry no payload to which the DL's ENC (encryption) capabilities could be applied.

7.3.2.4 Interface between the DLE and DSC

7.3.2.4.1 General

Figure 38 summarizes the relationship between the DLE and DSC for DPDU transactions. This flow covers the normal case where a DPDU is transmitted and acknowledged, and no errors occur. For more detail, see the documentation for the corresponding DSAPs in 7.3.2.4.2, 7.3.2.4.3, 7.3.2.4.4, 7.3.2.4.5, 7.3.2.4.6, 7.3.2.4.7, 7.3.2.4.8, and 7.3.2.4.9.

All interfaces between the DLE and DSC are internal interfaces within the DLE, and thus are unobservable. Therefore they are not subject to standardization.

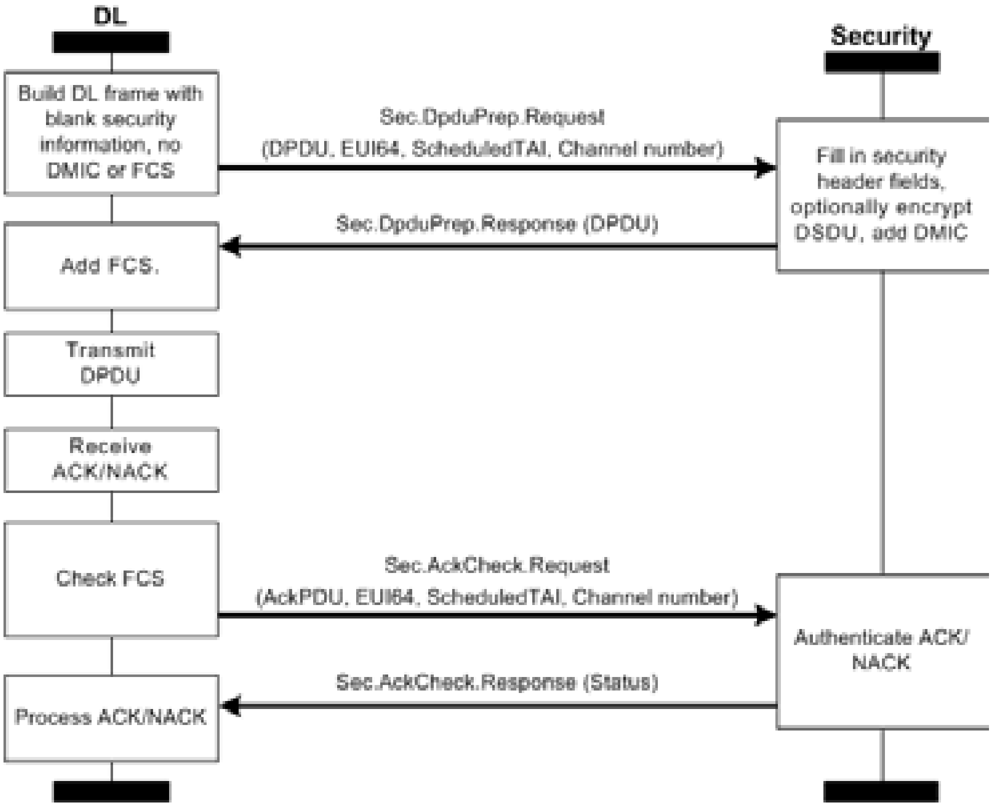


Figure 38 – DLE and DLS processing for a D-transaction initiator

The DLE assembles the DPDU to be protected. By documentation convention, security fields in the DPDU's header are populated by the DSC.

Certain DPDU security information is provided by the DLE to the DSC:

- the scheduled TAI time of the timeslot, used in the nonce to detect delayed and replayed DPDUs;

NOTE 1 The scheduled TAI time passed to the DSC is the scheduled TAI start time of the timeslot to which the D-transaction is assigned. The DSC truncates this time to 2^{-10} s (approximately 1 ms).

NOTE 2 There is one scenario under this standard where a single device might initiate multiple transmissions that all have the same scheduled timeslot start time. In that case a device (usually a backbone router) operates on multiple channels simultaneously, using synchronized timeslot templates in such a way that it can use either a single shared antenna or multiple closely-spaced antennas, phased in such a way that transmissions on one or more channels do not disrupt reception on other channels. While such operation is not explicitly described by this standard it is also intentionally not prohibited. Support for such concurrent operation gives rise to the following two nonce components that are included in the DPDU's nonce construction.

- the channel number of each DPDU, used in the nonce to detect DPDUs constructed for use in one channel that are replayed within the same timeslot in another channel;

NOTE 3 The channel number uses the channel-numbering convention of this standard, where the numbers 0..15 correspond to IEEE 802.15.4 channels 11..26 respectively.

- the one-octet sequence number found in the MAC header, used in the nonce to differentiate between the Data DPDU of a D-transaction and any ACK/NAK DPDUs that might be generated in timeslots with the same scheduled TAI start time;

NOTE 4 The low-order bits of the MHR sequence number octet encode the DPDU's zero-origin position in the D-transaction: 0 for the Data DPDU, 1 for the first ACK/NAK DPDU, 2 for a second ACK/NAK DPDU, etc.

- the DSC needs the EUI64Address of the destination device in order to process its ACK/NAK DPDU;

NOTE 5 When known to the DLE, this D-address is retrieved directly from the dlmo.Neighbor table.

- when a unicast destination's EUI64Address is not known to the DLE, the EUI64Address-requested indicator in the DHDR frame control octet (Table 111) shall be set (to 1), which triggers the destination to return its EUI64Address in the ACK/NAK DPDU.

NOTE 6 The DSC uses the DSDU size to encrypt only the DSDU and not the DPDU header, whereas the DMIC protects the entire DPDU. This detail is not shown in Figure 38 or Figure 39.

The DLE retains a copy of the outgoing DMIC, to be used subsequently to unambiguously connect the reply ACK/NAK DPDUs to the Data DPDU of the D-transaction. The DLE then appends an IEEE 802.15.4 FCS to the DPDU and transmits it without undue delay.

When the DLE receives an ACK/NAK DPDU, it requests the DSC to authenticate the DPDU. Certain DPDU security information is provided by the DLE to the DSC:

- Each ACK/NAK DPDU shall echo the D-transaction's intial DPDU's DMIC as a virtual field (see Table 117) in the computation of its D-MIC. The full ACK/NAK DPDU, including this virtual field, is reconstructed by the DLE before it is checked by the DSC.
- The EUI64Address of the ACK/NAK DPDU's originator is either looked up or provided within the ACK/NAK DPDU itself.
- The scheduled TAI time of the start of the D-transaction's timeslot, which is usually the same as the TAI start-of-timeslot time used by the D-transaction initiator. However, when slow-channel-hopping is used, the ACK/NAK DPDU may include a timeslot offset (see 9.3.4), in which case the nonce formed to check the ACK/NAK DPDU shall use the scheduled TAI start time of the timeslot referenced by the timeslot offset; that is, the scheduled timeslot of the acknowledging DLE.
- The channel number for sending the ACK/NAK DPDU is provided to the DSC.
- The MHR sequence number is provided to the DSC in the same manner that it is provided for the Data DPDU of the D-transaction.

Figure 39 illustrates the relationship between a DLE and its DSC for D-transactions in which the DLE is a recipient of or respondent to the D-transaction's Data DPDU.

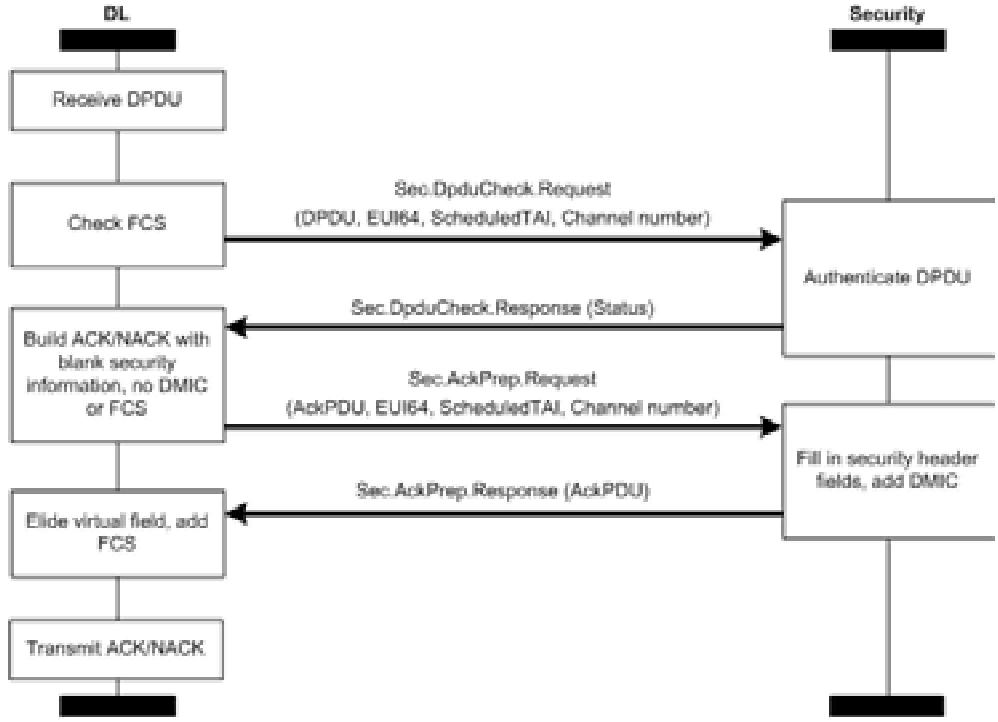


Figure 39 – Received DPDUs – DLE and DSC

When receiving a DPDU, the DLE sends a request to the DSE to authenticate the DPDU and, if necessary, decrypt the DPDU payload. The scheduled TAI time and the channel number are included with this request. The EUI64Address of the DPDU's source needs to be known by the DLE a priori, except in the case of a join request where it is carried in the DPDU header as a source address. The DSC normally responds with a positive authentication.

The DLE constructs the ACK/NAK DPDU. The ACK/NAK DPDU shall use the same scheduled TAI time as the received Data DPDU of the D-transaction, except when a slow-channel-hopping-offset correction is provided in the ACK/NAK DPDU as discussed above. The ACK/NAK DPDU shall also echo the DMIC of the initial received DPDU of the D-transaction as a virtual field, as shown in Table 117. The DSC then secures the ACK/NAK DPDU, including the DPDU's DMIC as a virtual field. The DLE elides the virtual field, appends an IEEE 802.15.4 FCS, and transmits the ACK/NAK DPDU.

When a DLE's local time sense is corrected by an ACK DPDU, such that its time is reset to an earlier timeslot, there shall be a forced pause in service, equal to the magnitude of the timeslot correction plus at least one timeslot.

7.3.2.4.2 Sec.DpduPrep.Request

7.3.2.4.2.1 General

Sec.DpduPrep.Request instructs the DSC to protect a DL protocol data unit as appropriate.

7.3.2.4.2.2 Semantics of the service primitive

The semantics of Sec.DpduPrep.Request are as follows:

```
Sec.DpduPrep.Request        (
        DPDU,
        EUI64,
        ScheduledTAI,
```

ChannelNumber,

AckHandle)

Table 37 specifies the elements for the Sec.DpduPrep.Request.

Table 37 – Sec.DpduPrep.Request elements

Element name	Element identifier	Element scalar type
DPDU (the DPDU to be transmitted)	1	Type: OctetString
EUI64 (the EUI64Address of the sending device)	2	Type: EUI64Address
ScheduledTAI (32-bits of the start time of the slot truncated to a 2^{-10} s resolution)	3	Type: Unsigned32
ChannelNumber (the channel number used in the transmitted DPDU)	4	Type: Unsigned8 Valid range: 0..15
AckHandle (abstraction that connects each invocation of Sec.DpduPrep.Request with the subsequent callback by Sec.DpduPrep.Response)	5	Type: Abstract

The DSC provides the DLE with the appropriate security control (octet 1) and the Crypto Key Identifier (octet 2) obtained from the KeyDescriptor for the current D-key, to be used in the DPDU's DMXHR subheader, the format of which is described in 9.3.3.4. See 7.3.2.5 on selecting the proper D-key.

The DSC populates the DMIC field as specified by the policy of the selected D-key.

7.3.2.4.2.3 Appropriate usage

The DLE invokes the Sec.DpduPrep.Request primitive to add security protection to a DPDU before it is transmitted.

7.3.2.4.2.4 Effect on receipt

On receipt of the Sec.DpduPrep.Request primitive, the DSC starts the appropriate DPDU processing steps to protect the DPDU as dictated by policy.

7.3.2.4.3 Sec.DpduPrep.Response

7.3.2.4.3.1 General

Sec.DpduPrep.Response reports the result of a Sec.DpduPrep.Request.

7.3.2.4.3.2 Semantics

The semantics of Sec.DpduPrep.Response are as follows:

Sec.DpduPrep.Response (

DPDU,

Status,

AckHandle)

Table 38 specifies the elements for Sec.DpduPrep.Response.

Table 38 – Sec.DpduPrep.Response elements

Element name	Element identifier	Element scalar type
DPDU	1	Type: OctetString
Status (the result of a Sec.DpduPrep.Request primitive)	2	Type: Unsigned Named values: 0: success; >0: failure
AckHandle (abstraction that connects each invocation of Sec.DpduPrep.Request with the subsequent callback by Sec.DpduPrep.Response)	3	Type: Abstract

7.3.2.4.3.3 When generated

The DSC generates Sec.DpduPrep.Response in response to a Sec.DpduPrep.Request. The Sec.DpduPrep.Response returns a status value that indicates either SUCCESS and the unsecured DPDU or the appropriate error code.

7.3.2.4.3.4 Appropriate usage

On receipt of Sec.DpduPrep.Response, the DL is notified of the result of request to protect an outgoing DPDU.

7.3.2.4.4 Sec.DAckCheck.Request

7.3.2.4.4.1 General

Sec.DAckCheck.Request instructs the DSC to verify an incoming ACK/NAK DPDU.

7.3.2.4.4.2 Semantics of the service primitive

The semantics of Sec.DAckCheck.Request are as follows:

```
Sec.DAckCheck.Request     (
        AckPDU,
        EUI64,
        ScheduledTAI,
        ChannelNumber,
        AckHandle)
```

Table 39 specifies the elements for the Sec.DAckCheck.Request.

Table 39 – Sec.DAckCheck.Request elements

Element name	Element identifier	Element scalar type
AckPDU (the AckPDU to be verified)	1	Type: OctetString
EUI64 (the EUI64Address of the acknowledging device)	2	Type: EUI64Address
ScheduledTAI (32-bits of the start time of the slot truncated to a 2^{-10} s resolution)	3	Type: Unsigned32
ChannelNumber (the channel number used to receive the incoming ACK/NAK DPDU)	4	Type: Unsigned Valid range: 0..15
AckHandle (abstraction that connects each invocation of Sec.DAckCheck.Request with the subsequent callback by Sec.DAckCheck.Response)	5	Type: Abstract

The DSC verifies that the DHR of the ACK/NAK DPDU has employed the DMIC mode (see Table 118) specified by the current D-key policy. The D-key used in authenticating the ACK/NAK DPDU is the same as that used for the Data DPDU of the D-transaction.

The DSC verifies the DMIC field as dictated by the DPDU processing steps and current policies.

7.3.2.4.4.3 Appropriate usage

The DLE invokes the Sec.DAckCheck.Request primitive to verify an ACK/NAK DPDU after its reception.

7.3.2.4.4.4 Effect on receipt

On receipt of the Sec.DAckCheck.Request primitive, the DSC performs the appropriate DPDU processing steps to verify the received ACK/NAK DPDU as specified in 7.3.2.6.

7.3.2.4.5 Sec.DAckCheck.Response

7.3.2.4.5.1 General

Sec.DAckCheck.Response reports the result of a Sec.DAckCheck.Request.

7.3.2.4.5.2 Semantics

The semantics of Sec.DAckCheck.Response are as follows:

```
Sec.DInitialCheck.Response (
        AckPDU,
        Status,
        AckHandle)
```

Table 40 specifies the elements for Sec.DAckCheck.Response.

Table 40 – Sec.DAckCheck.Response elements

Element name	Element identifier	Element scalar type
AckPDU	1	Type: OctetString
Status (the result of a Sec.DAckPrep.Request primitive)	2	Type: Unsigned Named values: 0: success; >0: failure
AckHandle (abstraction that connects each invocation of Sec.DAckCheck.Request with the subsequent callback by Sec.DAckCheck.Response)	3	Type: Abstract

7.3.2.4.5.3 When generated

The DSC generates Sec.DAckCheck.Response in response to a Sec.DAckCheck.Request. The Sec.DAckCheck.Response returns a status value that indicates either SUCCESS or the appropriate error code.

7.3.2.4.5.4 Appropriate usage

On receipt of Sec.DAckCheck.Response, the DL is notified of the result of verifying and possibly decrypting an incoming DPDU.

7.3.2.4.6 Sec.DInitialCheck.Request

7.3.2.4.6.1 General

Sec.DInitialCheck.Request instructs the DSC to verify and possibly decrypt an incoming DL protocol data unit as appropriate.

7.3.2.4.6.2 Semantics of the service primitive

The semantics of Sec.DInitialCheck.Request are as follows:

```
Sec.DInitialCheck.Request (
    DPDU,
    EUI64,
    ScheduledTAI,
    ChannelNumber,
    AckHandle)
```

Table 41 specifies the elements for the Sec.DInitialCheck.Request.

Table 41 – Sec.DInitialCheck.Request elements

Element name	Element identifier	Element scalar type
DPDU (the DPDU to be verified and possibly decrypted)	1	Type: OctetString
EUI64 (the EUI64Address of the sending device)	2	Type: EUI64Address
ScheduledTAI (32-bits of the start time of the slot truncated to a 2^{-10} s resolution)	3	Type: Unsigned32
ChannelNumber (the channel number used to receive the incoming DPDU)	4	Type: Unsigned Valid range: 0..15
AckHandle (abstraction that connects each invocation of Sec.DInitialCheck.Request with the subsequent callback by Sec.DInitialCheck.Response)	5	Type: Abstract

The DSC verifies that the DMXHR of the DPDU has the appropriate security control (octet 1) by comparing it to the current policy. The Crypto Key Identifier (octet 2) is used to retrieve the correct key material. See 7.3.2.6.

The DSC verifies the DMIC field as dictated by the current policies.

7.3.2.4.6.3 Appropriate usage

The DL invokes the Sec.DInitialCheck.Request primitive to verify and possibly decrypt a DPDU before it is transmitted.

7.3.2.4.6.4 Effect on receipt

On receipt of the Sec.DInitialCheck.Request primitive, the DSC starts the appropriate PDU processing steps to verify the incoming DPDU as dictated by the incoming PDU processing steps in 7.3.2.6.

7.3.2.4.7 Sec.DInitialCheck.Response

7.3.2.4.7.1 General

Sec.DInitialCheck.Response reports the result of a Sec.DInitialCheck.Request.

7.3.2.4.7.2 Semantics

The semantics of Sec.DInitialCheck.Response are as follows:

```
Sec.DInitialCheck.Response (
        DPDU,
        Status,
        AckHandle)
```

Table 42 specifies the elements for Sec.DInitialCheck.Response.

Table 42 – Sec.DInitialCheck.Response elements

Element name	Element identifier	Element scalar type
DPDU	1	Type: OctetString
Status (the result of a Sec.DpduPrep.Request primitive)	2	Type: Unsigned Named values: 0: success; >0: failure
AckHandle (abstraction that connects each invocation of Sec.DInitialCheck.Requestwith the subsequent callback by Sec.DInitialCheck.Response)	3	Type: Abstract

7.3.2.4.7.3 When generated

The DSC generates Sec.DInitialCheck.Response in response to a Sec.DInitialCheck.Request. The Sec.DInitialCheck.Response returns a status value that indicates either SUCCESS or the appropriate error code.

7.3.2.4.7.4 Appropriate usage

On receipt of Sec.DInitialCheck.Response, the DL is notified of the result of verifying and possibly decrypting an incoming DPDU.

7.3.2.4.8 Sec.DAckPrep.Request

7.3.2.4.8.1 General

Sec.DAckPrep.Request instructs the DSC to protect an ACK/NAK DPDU as appropriate.

7.3.2.4.8.2 Semantics of the service primitive

The semantics of Sec.DAckPrep.Request are as follows:

```
Sec.DAckPrep.Request        (
        AckPDU,
        EUI64,
        ScheduledTAI,
        ChannelNumber,
        AckHandle)
```

Table 43 specifies the elements for the Sec.DAckPrep.Request.

Table 43 – Sec.DAckPrep.Request elements

Element name	Element identifier	Element scalar type
AckPDU (includes the virtual header)	1	Type: OctetString
EUI64 (the EUI64Address of the acknowledging device)	2	Type: EUI64Address
ScheduledTAI (32-bits of the start time of the slot truncated to a 2^{-10} s resolution)	3	Type: Unsigned32
ChannelNumber (the channel number used to transmit the ACK/NAK DPDU)	4	Type: Unsigned8 Valid range: 0..15
AckHandle (abstraction that connects each invocation of Sec.DAckPrep.Request with the subsequent callback by Sec.DAckPrep.Response)	5	Type: Abstract

The DSC populates the ACK/NAK DPDU with the appropriate security control (octet 1) as described in Table 118. In the case where multiple D-keys are currently valid, the key used to authenticate the ACK/NAK DPDU is the same one used as the corresponding DPDU for this ACK/NAK DPDU.

The DSC populates the DMIC field as dictated by the current policies. Note that the DMIC field in an ACK/NAK DPDU is always 32 bits.

7.3.2.4.8.3 Appropriate usage

The DL invokes the Sec.DAckPrep.Request primitive to protect an ACK/NAK DPDU before it is transmitted.

7.3.2.4.8.4 Effect on receipt

On receipt of the Sec.DAckPrep.Request primitive, the DSC starts the appropriate PDU processing steps to protect the ACK/NAK DPDU as dictated by policy. Note that the ACK/NAK DPDU is only authenticated and never encrypted.

7.3.2.4.9 Sec.DAckPrep.Response

7.3.2.4.9.1 General

Sec.DAckPrep.Response reports the result of a Sec.DAckPrep.Request.

7.3.2.4.9.2 Semantics

The semantics of Sec.DAckPrep.Response are as follows:

```
Sec.DAckPrep.Response     (
        AckPDU,
        Status,
        AckHandle)
```

Table 44 specifies the elements for Sec.DAckPrep.Response.

Table 44 – Sec.DAckPrep.Response elements

Element name	Element identifier	Element scalar type
AckPDU	1	Type: OctetString
Status (the result of a Sec.DAckPrep.Request primitive)	2	Type: Unsigned Named values: 0: success; >0: failure
AckHandle (abstraction that connects each invocation of Sec.DAckPrep.Request with the subsequent callback by Sec.DAckPrep.Response)	3	Type: Abstract

7.3.2.4.9.3 When generated

The DSC generates Sec.DAckPrep.Response in response to a Sec.DAckPrep.Request. Sec.DAckPrep.Response returns a status value that indicates either SUCCESS or the appropriate error code.

7.3.2.4.9.4 Appropriate usage

On receipt of Sec.DAckPrep.Response, the DL is notified of the result of request to verify an incoming AckPDU.

7.3.2.4.10 Nonce construction for DPDUs

This standard uses a different DPDU nonce construction from that of IEEE 802.15.4:2011. A 13-octet nonce is required for the CCM* engine. The nonce shall be constructed as the concatenation from first (leftmost) to last (rightmost) octets of data fields as shown in Table 45, wherein:

- the EUI64Address shall be used as an array of 8 octets (in MSB convention) in the same manner as the source address of the CCM* nonce in IEEE 802.15.4:2011, 7.3.2;
- the TAI time shall be the least significant 32 bits of the TAI time in units of 2^{-10} s as described in Table 46;
- the last octet shall be constructed as follows:
  - Bit 7 shall be zero, thereby reserving the value 0xFF for the transport layer (see Table 57).
  - Bits 6..3 (4 bits) shall indicate the radio channel of transmission, in a range of 0..15, corresponding to IEEE 802.15.4 channel numbers 11..26, in the same order.
  - Bits 2..0 shall be copied from the corresponding low-order 3 bits of the MHR’s sequence number.

Table 45 – Structure of the WISN DPDU nonce

Octet	Bits							
	7	6	5	4	3	2	1	0
1	EUI64Address of DPDU originator							
…								
8								
9	Least significant 32 bits of TAI time of nominal slot start (in units of 2^{-10} s)							
…								
12								
13	Reserved = 0	Channel number (0..15)				Low-order 3 bits of MHR sequence number		

The TAI time used shall be a 32-bit truncated fixed-point fractional representation of TAI time at a granularity of 2^{-10} s and a span of 2^{22} s. With this representation, there will be over 48,5 days before the same value of TAI time recurs. Thus, the maximum lifetime of a D-key shall be 48,5 days before a new D-key needs to be deployed. The TAI time for this operation shall be that maintained by the DLE.

NOTE 1 It is important that the value of the 32-bit representation of TAI time does not recur within the lifetime of any relevant secret symmetric key, to avoid a potential nonce collision resulting in an identical keystream.

The representation in the D-nonce of this truncated 32-bit TAI time, specified to 2^{-10} s, is described in Table 46.

Table 46 – Structure of the 32-bit truncated TAI time used in the D-nonce

Octet	Bits							
	7	6	5	4	3	2	1	0
1	Truncated TAI time (bits with weight $2^{21}..2^{14}$ s)							
2	Truncated TAI time (bits with weight $2^{13}..2^{6}$ s)							
3	Truncated TAI time (bits with weight $2^{5}..2^{-2}$ s)							
4	Truncated TAI time (bits with weight $2^{-3}..2^{-10}$ s)							

The lower order 3 bits of the MHR sequence number, together with the channel number, are used to construct the last octet of a D-nonce. The sending DLE shall ensure that the MHR sequence number bits used in the D-nonce are unique among all those it generates within the same 2^{-10} s interval for the same channel and same D-key (see 9.3.3.2 and 9.3.4). The value of 0xFF shall not be used for the MHR. Because this D-nonce has at most eight distinct values for a given channel and 2^{-10} s interval, a DLE shall not transmit more than eight DPDUs per 2^{-10} s on the same channel using the same D-key.

NOTE 2 Inclusion of the channel number in the D-nonce provides support for devices that operate concurrently on multiple channels.

NOTE 3 The construction of the MHR sequence number is described in 9.3.3.2.

7.3.2.5 Processing of a DPDU to be transmitted

The inputs to the DPDU security procedure are:

- the DPDU to be secured;
- the EUI64Address of the source DLE;
- the nominal TAI start time of the timeslot being used for the D-transaction;

- the MHR sequence number octet; and
- the channel number (0..15) to be used for the D-transaction.

The outputs from this procedure are:

- the status of the procedure; and
- if this status is success, the secured DPDU.

The security procedure for DPDUs that are being constructed for transmission consists of the following steps:

a) The procedure shall obtain the KeyDescriptor from Table 93 meeting the following selection criteria:
   1) The entries with KeyUsage = '0x00' (i.e., D-key). In the initial case, where a joining DLE does not have any KeyDescriptor, the joining device creates a KeyDescriptor with K_global. The KeyDescriptor shall include at least the following parameters:
      - Crypto Key Identifier = 0;
      - Security Level = 0x01 (MIC-32);
      - KeyUsage = 0x00 (group key for PDU processing);
      - Key lifetime = never-expires (0xFFFF FFFF).
   2) Of those entries, the entries valid for the current period, satisfying the inequality

      ValidNotBefore < currentTime < ValidNotAfter

      shall be selected. If none are available, the procedure shall return with a status of UNAVAILABLE_KEY.
   3) Of those entries, if two or more keys are valid for the current time, and the procedure was called from DAckPrep.Request or an DAckCheck.Request, the procedure shall select the key used to authenticate the Data DPDU of the D-transaction.

      Otherwise, if two or more keys are valid for the current time, the procedure shall select the key with the larger ValidNotAfter value.
   4) Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the larger ValidNotBefore.
   5) Of those entries, if two or more keys have the same SoftExpirationTime, the procedure shall select the key with the highest Crypto Key Identifier.
b) The procedure shall retrieve the policy from the selected KeyDescriptor.
c) The procedure shall determine whether the DPDU to be secured satisfies the constraint on the maximum size of DPDUs, as follows:
   1) The procedure shall set the size M, in octets, of the DMIC authentication field from the security level.
   2) The Crypto Key Identifier Mode field in the DMXHR shall have the value 1. If the DMXHR includes the slow-channel-hopping timeslot offset field, the size of DMXHR is 3 octets; otherwise it is 2 octets.
   3) The procedure shall determine the resulting data expansion as (DMXHR_size + M).
   4) The procedure shall check whether the size of the DPDU to be secured, including data expansion, is less than or equal to the maximum DPDU size. If this check fails, the procedure shall return a status of DPDU_TOO_LONG.
d) The procedure shall use the scheduled TAI time of the start of the timeslot, as described in Table 46. If there is a potential for the device to send multiple DPDUs with the same TAI time value, then the procedure shall select a value to be conveyed in the DPDU's MHR header that is different from all other such values originated by the device at this particular value of TAI time. A procedure for determining the sequence number from whch MHR is derived is defined in 9.3.3.2.

e) The procedure shall insert the DMXHR into the DPDU outlined in Table 112, with fields set as follows:

   1) The security level subfield of the security control field shall be set to the security level 001 by default.

   2) The Crypto Key Identifier Mode subfield of the security control field shall be set to the Crypto Key Identifier Mode parameter 01 by default.

f) The procedure shall set the Crypto Key Identifier octet in the DMXHR. See Table 112.

g) The procedure shall insert the MHR sequence number in the Data DPDU MHR. See Table 110.

h) The procedure shall use the EUI64Address of the transmitting device, the 32 least significant bits of TAI time in 2^{-10} s, the low-order 3 bits of the MHR sequence number, and the channel number to build the nonce as outlined in Table 45.

i) The procedure shall use the nonce, the key material, the header, the payload and the CCM* mode of operation as described in IEEE 802.15.4:2011, 7.3.4, to secure the DPDU:

   1) If the SecurityLevel parameter specifies the use of encryption (see IEEE 802.15.4:2011, Table 58), the encryption operation shall be applied only to the DPDU's payload field. The corresponding payload field is passed to the CCM* transformation process described in IEEE 802.15.4:2011, 7.3.4, as the unsecured payload. The resulting encrypted payload shall be substituted for the original payload.

   2) The remaining fields in the DPDU, up to but not including the payload field, shall be passed to the CCM* transformation process described in IEEE 802.15.4:2011, 7.3.4, as the non-payload field.

   3) The ordering and exact manner of performing the encryption and integrity operations and the placement of the resulting encrypted data or integrity code within the DPDU payload field shall be as defined in IEEE 802.15.4:2011, 7.3.4.

j) The procedure shall return the secured DPDU and a status of SUCCESS.

7.3.2.6 Processing of received DPDUs

The inputs to the security procedure for received DPDUs are the DPDU to be unsecured, the channel number on which the DPDU was received, and the nominal TAI time of the start of the time slot in which the DPDU was received. The outputs from this procedure are the unsecured DPDU, the security level, the Crypto Key Identifier Mode, the Crypto Key Identifier, and the status of the procedure. All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure. It is assumed that the KeyDescriptors with a single, unique device or a number of devices will have been established by the DSMO.

The security procedure on DPDU reception consists of the following steps:

a) The procedure shall set the security level and the Crypto Key Identifier Mode to the corresponding subfields of the security control field of the DMXHR of the incoming DPDU, and the Crypto Key Identifier to the corresponding subfields of the Crypto Key Identifier field of the DMXHR of the DPDU to be unsecured.

b) The procedure shall obtain the KeyDescriptor from Table 93 meeting the following selection criteria:

   1) The entries with KeyUsage = '0x00' (D-key). In the initial case, where a joining device does not have any KeyDescriptors, the joining device creates a temporary KeyDescriptor with K_global. The KeyDescriptor shall include at least following parameters:

      - CryptoKeyIdentifier = 0;
      - Security Level = 0x01 (MIC-32);
      - KeyUsage = 0x00 (group key for PDU processing);
      - Key lifetime = never-expires (0xFFFF FFFF).

      NOTE The usage of the KeyDescriptor for K_global is described in 9.1.10.

2) Of those entries, the entry with the CryptoKeyIdentifier matching the Crypto Key Identifier of the incoming PDU shall be selected.

3) If that procedure fails, the procedure shall return with a status of UNAVAILABLE_KEY.

c) The procedure shall determine whether the security level of the incoming DPDU conforms to the security level policy by comparing the SecurityLevel of the matching KeyDescriptor obtained from step b) above. If there is a mismatch, the procedure shall return with a status of IMPROPER_SECURITY_LEVEL.

d) If the lifetime in the KeyDescriptor is finite (> 0x0000), the procedure shall verify that the 8-bit MHR sequence number has not been received previously for the same value of the source EUI64Address, the same 32-bit fixed-point fractional representation of TAI time, and the same key. If this check fails, the procedure shall return with a status of DUPLICATE_DPDU.

e) The procedure shall then use the EUI64Address of the sender, the scheduled TAI time, and low-order 3 bits of the MHR sequence number, and the channel number to generate the nonce as outlined in Table 45. Additionally, the procedure shall verify that the 8-bit MHR sequence number is not 0xFF. If the 8-bit MHR sequence number is 0xFF, the procedure shall return with a status of INVALID_SEQUENCE_NUMBER.

f) The procedure shall use the nonce, the crypto key from the KeyDescriptor obtained in step 2, the actual headers (the non-payload fields), the payload and the MIC of the incoming DPDU and the CCM* mode of operation as described in operations (see IEEE 802.15.4:2011, 7.3.5) to authenticate and, when specified, decrypt the DPDU:

1) If the security level specifies the use of encryption (see IEEE 802.15.4:2011, Table 58), the decryption operation shall be applied only to the actual DPDU payload field (see IEEE 802.15.4:2011, 5.2.2.2.2). The corresponding payload field shall be passed to the CCM* inverse transformation process described in IEEE 802.15.4:2011, 7.3.5 as the secure payload.

2) The remaining fields in the DPDU shall be passed to the CCM* inverse transformation process described in IEEE 802.15.4:2011, 7.3.5 as the non-payload fields (see IEEE 802.15.4:2011, Table 57).

3) The ordering and exact manner of performing the decryption and integrity checking operations and the placement of the resulting decrypted data within the DPDU payload field shall be as defined in IEEE 802.15.4:2011, 7.3.5.

g) If the CCM* inverse transformation process fails, the procedure shall set the unsecured DPDU to be the DPDU to be unsecured and return a status of SECURITY_ERROR.

h) If the lifetime in the KeyDescriptor is one that expires, the procedure shall insert the nonce value (includes MHR sequence number, channel number, source EUI64Address, and scheduled TAI time) in the NonceCache field of the corresponding KeyDescriptor, to enable replay protection.

i) The procedure shall return with the unsecured DPDU, the security level, the Crypto Key Identifier Mode, the Crypto Key Identifier, and a status of SUCCESS.

7.3.2.7 Detection and discard of duplicated or replayed protocol data units

See 7.3.2.6, d).

7.3.3 TL security functionality

7.3.3.1 General

The interaction of the DSC and the TL is outlined. The TL processing steps were written to reuse the commonalities between the DL and the TL. However, since the DL and the TL exist at different network abstraction layers with different requirements and assumptions, there are significant differences between the DL and TL processing steps.

Security services at the TL are selected by policy associated with the relevant transport data key, obtained as part of a new session request or a key update and based on transport policy maintained by the security manager associated with any or all of:

- the sending device;
- the requesting UAP; and/or
- the transport association, as defined by its endpoints.

The following transport security service shall always be provided with an active key:

- Authorized communication with TPDU authentication, integrity, and conveyance of the nominal time of TPDU creation, providing rejection of outdated TPDUs:
  - that were not sourced by a device within the network that shares an appropriate data key; or
  - that were severely outdated, i.e., not received at a time within DSMO.pduMaxAge seconds of the TPDU’s nominal time of creation.
- Confidentiality of the application-layer payload within the TPDU.

NOTE 1 Sixteen bits of time information are transmitted with each TPDU.

NOTE 2 This service uses the originator’s nominal time of transmission as authenticated at the receiver to cause rejection of TPDUs that are delayed excessively and to provide detection of duplicated TPDUs within that time window.

The confidentiality service shall not be employed with keys that are not shared secrets, because this would render true confidentiality impossible, and because this aspect of the policy associated with such keys is constant.

The MIC should be validated within the DSMO.pduMaxAge period. If the check fails, the MIC validation can be repeated by decrementing a time window to recover the creation time of the PDU.

7.3.3.2 TPDU structure

7.3.3.2.1 General

The structure of a TPDU is described in 11.5 and outlined in Figure 109 and in Figure 40 in this standard, with the TSDU possibly encrypted and the contents of the UDP header, security header, and TSDU protected by the TMIC.

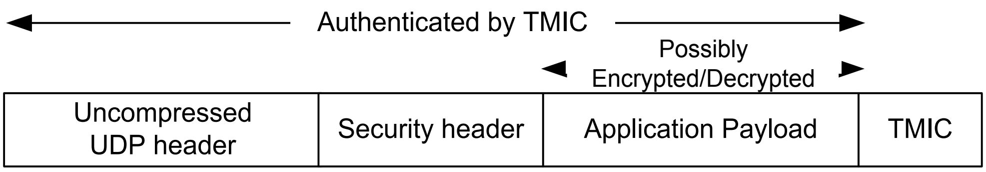


Figure 40 – TPDU structure and protected coverage

The complete TPDU from the start of the UDP header to the end of the Application Payload shall be protected by the TMIC. Each parameter in the UDP header is protected by using the Transport Security Component (TSC) pseudo-header for the TMIC calculation. The TSC pseudo-header is described in 7.3.3.2.2.

NOTE TSC is described in 11.2. See also the brief discussion of the use of pseudo-headers in 4.5.2.1.

7.3.3.2.2 TPDU Protection

The TMIC is used to protect the information in the UDP header, the TL security header and the TSDU. It also protects the NL source and destination IPv6 addresses by using an extended form of the UDP pseudo-header for IPv6. The UDP pseudo-header for IPv6 is described in 11.4.2 and RFC 2460, 8.1. The UDP payload size and the (virtual) checksum in the UDP header are not used for the TMIC calculation.

NOTE Checksum and UDP payload size do not appear in the pseudo-header since the checksum is elided (i.e., not present in the TPDU) when the TMIC is present, and the UDP payload size is determined from the NSDU size in the UDP pseudo-header for IPv6.

The parameters for the TMIC are shown in Figure 41.

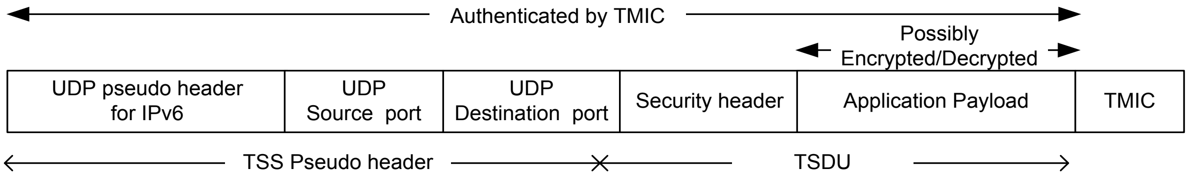


Figure 41 – TMIC parameters

The TSC constructs the TMIC parameters as outlined in Figure 41, with the received TPDU, the nominal TAI time at which the TPDU was created, the KeyDescriptor and the contract information provided by the TL. The TSC can then use the parameters for the appropriate security operation on the TPDU.

The IPv6 header and the UDP source and destination ports are passed from the TL to the TSC. The combination of those parameters is an extended UDP pseudo-header that is called the TSC pseudo-header in this standard. The structure of the TSC pseudo-header is shown in Table 47. The appropriate usage is described in 7.3.3.5.4, 7.3.3.5.5, and 7.3.3.5.8.

Table 47 – TSC pseudo-header structure

Element name	Element identifier	Element scalar type
Source IPv6Address	1	Type: IPv6Address Description: Uncompressed IPv6 address of the TPDU initiator
Destination IPv6Address	2	Type: IPv6Address Description: Uncompressed IPv6Address of the intended TPDU recipient
NSDU size	3	Type: Unsigned16 Description: NDSU size in octet
Reserved	4	Type: Unsigned8 Description: Reserved field. Currently filled with 0
Next header	5	Type: Unsigned8 Valid range: 17 (UDP) Description: Next header value in the IPv6 header. This value should be only 17
UDP source port	6	Type: Unsigned16 Description: The source UDP port number of the TPDU initiator.
UDP destination port	7	Type: Unsigned16 Description: The destination UDP port number of the intended TPDU recipient

7.3.3.3 Interface with the TL for a TPDU being formed for transmission

The TL interaction with the security layer for a TPDU being formed for transmission is summarized in Figure 42. When the TSC receives the source address, source port, destination address, destination port, and payload size, it performs a lookup in the KeyDescriptor table to see whether security is enabled for that particular session.

If the session’s security level is 0: none, a header size of 3 (octets) and a TMIC size of 0 octets shall be returned.

NOTE When the security level is zero, the standard, trivially-forged UDP checksum is used to detect errors that occur during TPDU conveyance.

Otherwise the TSC shall return the appropriate Crypto Key Identifier and TMIC sizes. All sessions at the TL are unicast; therefore, the Crypto Key Identifier size shall be either 0 or 1 depending on the number of valid keys available at that time for that security association.

The TL will then call the TSC with the header and the payload. Depending on the security policy for that particular session, the payload may be encrypted, and a TMIC may be generated. The resulting header and payload will be returned to the TL for transmission.

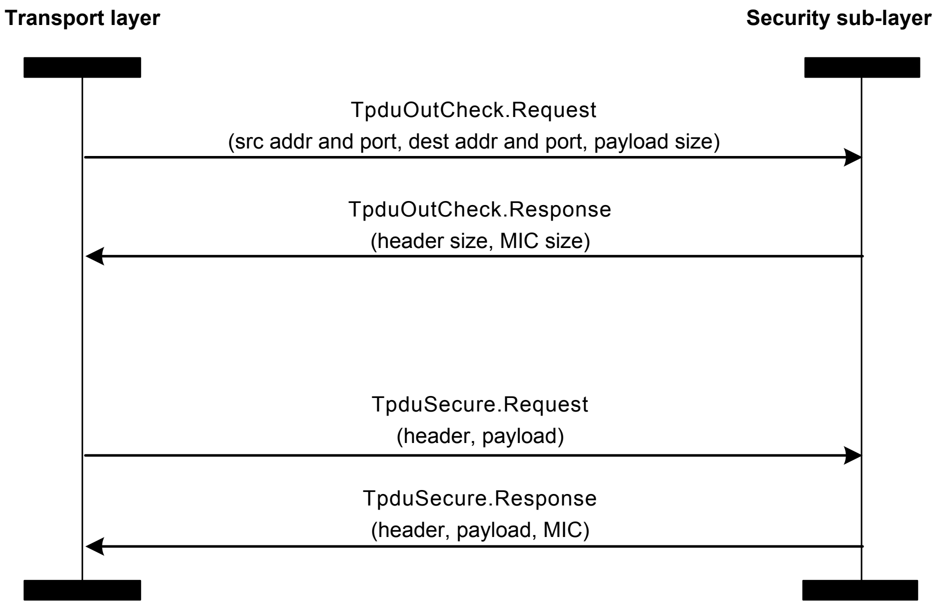


Figure 42 – TL and TSC interaction, outgoing TPDU

7.3.3.4 Processing overview for received TPDUs

The TL interaction with the security layer for a received TPDU is summarized in Figure 43. When the TSC receives the source address, source port, destination address, and destination port, it performs a lookup in the MIB to see whether security is enabled for that particular session and returns SEC_CHECK_REQUIRED or SEC_CHECK_NOT_REQUIRED.

The TL shall then call the TSC with the header, the payload, and the MIC. Depending on the security policy for that particular session, the payload may be decrypted, and a MIC may be verified. If the security check fails, a status of FAILURE will be returned, with the payload returned untouched. If the operation succeeds, the resulting payload and the recovered time of TPDU encoding will be returned to the TL.

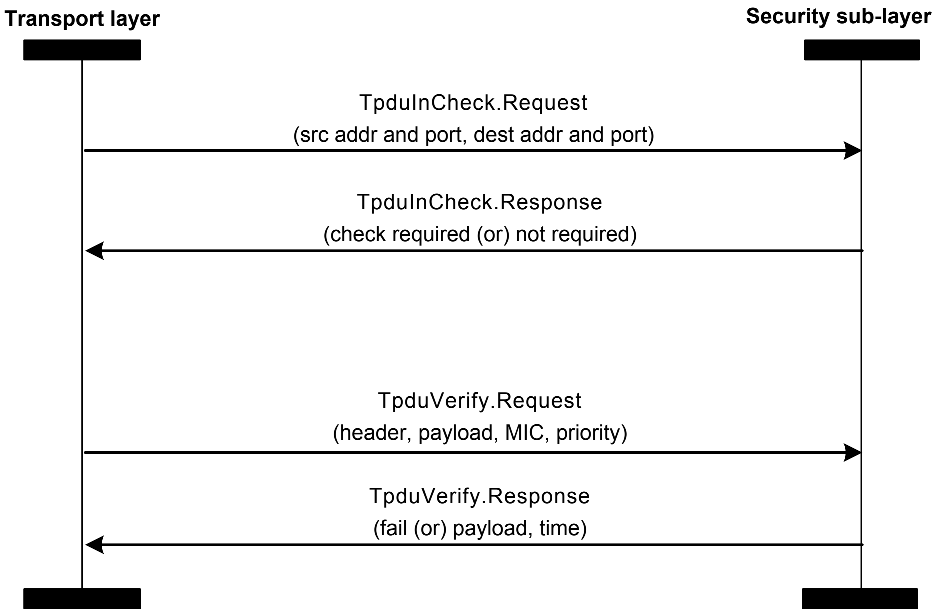


Figure 43 – TL and TSC interaction, incoming TPDU

7.3.3.5 TL interface to the TSC

7.3.3.5.1 General

The relationship between the TL and the TSC is outlined in 7.3.3.2 for TL interface for an outgoing TPDU and 7.3.3.4 for TL interface for an incoming TPDU.

7.3.3.5.2 Sec.TpduOutCheck.Request

7.3.3.5.2.1 General

Sec.TpduOutCheck.Request is a check from the TL to the TSC to obtain the size of the security fields (if any) required in the outgoing TPDU.

7.3.3.5.2.2 Semantics of the service primitive

The semantics of Sec.TpduOutCheck.Request are as follows:

```
Sec.TpduOutCheck.Request (
    Source_Address,
    Source_Port,
    Destination_Address,
    Destination_Port,
    Payload_Size)
```

Table 48 specifies the elements for the Sec.TpduOutCheck.Request.

Table 48 – Sec.TpduOutCheck.Request elements

Element name	Element identifier	Element scalar type
Source_Address	1	Type: IPv6Address Valid range: all with high-order bit reset
Source_Port	2	Type: Integer
Destination_Address	3	Type: IPv6Address
Destination_Port	4	Type: Integer
Payload_Size	5	Type: Integer Valid range: 0..Assigned_Max_TSDU_Size; see 11.4.3.3

The TSC shall use the Source_Address, Source_Port, Destination_Address, and Destination_Port to retrieve the appropriate policy (if any) for this security association.

7.3.3.5.2.3 Appropriate usage

The TL invokes the Sec.TpduOutCheck.Request primitive to protect a TPDU before it is transmitted.

7.3.3.5.2.4 Effect on receipt

On receipt of the Sec.TpduOutCheck.Request primitive, the TSC determines if the TPDU needs to be protected and returns the corresponding header and TMIC sizes.

7.3.3.5.3 Sec.TpduOutCheck.Response

7.3.3.5.3.1 General

Sec.TpduOutCheck.Response reports the result of a Sec.TpduOutCheck.Request.

7.3.3.5.3.2 Semantics

The semantics of Sec.TpduOutCheck.Response are as follows:

```
Sec. TpduOutCheck.Response      (
        Sec_Header_Size,
        TMIC_Size)
```

Table 49 specifies the elements for Sec.TpduOutCheck.Response.

Table 49 – Sec.TpduOutCheck.Response elements

Element name	Element identifier	Element scalar type
Sec_Header_Size (the additional header size required by the TSC, in full octets)	1	Type: Integer Valid range: 0..Assigned_Max_TSDU_Size; see Table 30
TMIC_Size (the size of the Transport Integrity Code, in full octets)	2	Type: Integer Valid values: 0, 4, 8 or 16

7.3.3.5.3.3 When generated

The TSC generates Sec.TpduOutCheck.Response in response to a Sec.TpduOutCheck.Request. The Sec.TpduOutCheck.Response returns the additional sizes required to support the security layer functionality. A security association is a secure TL association based on:

- source address;
- source port;
- destination address;
- destination port.

7.3.3.5.3.4 Appropriate usage

On receipt of Sec.TpduOutCheck.Response, the TL is notified of the need to apply a security operation on the TPDU, along with the additional octets required to support that operation.

7.3.3.5.4 Sec.TpduSecure.Request

7.3.3.5.4.1 General

Sec.TpduSecure.Request instructs the TSC to carry out the appropriate steps to secure an outgoing TPDU. The information about the security association is contained in the pseudo-header passed to the TSC. See Figure 108 in 11.4.2.

7.3.3.5.4.2 Semantics of the service primitive

The semantics of Sec.TpduSecure.Request are as follows:

```
Sec. TpduSecure.Request    (
        TSC_Pseudo_Header,
        TSC_Pseudo_Header_Size,
        TSDU,
        TSDU_Size)
```

Table 50 specifies the elements for the Sec.TpduSecure.Request.

Table 50 – Sec.TpduSecure.Request elements

Element name	Element identifier	Element scalar type
TSC_Pseudo_Header	1	Type: OctetStringN
TSC_Pseudo_Header_Size	2	Type: Integer Valid range: 0..127
TSDU	3	Type: OctetStringN
TSDU_Size	4	Type: Integer Valid range: 0..Assigned_Max_TSDU_Size; see Table 30

The TSC shall obtain the source address, source port, destination address, and destination port from the TSC pseudo-header. That information is used to retrieve the appropriate keying material and policies for this security association.

The TSC includes the Priority information in the TPDU header, provides data confidentiality for the TPDU, and generates the TMIC field as dictated by the PDU processing steps and current policies.

The TSC populates the TL security header as specified in the TPDU processing steps and policies.

7.3.3.5.4.3 Appropriate usage

The TL invokes the Sec.TpduSecure.Request primitive to protect an outgoing TPDU after the TL receives the number of additional octets required in the transport header and the TMIC.

7.3.3.5.4.4 Effect on receipt

On receipt of the Sec.TpduSecure.Request primitive, the TSC starts the appropriate PDU processing steps to protect the outgoing TPDU as dictated by the outgoing TPDU processing steps in 7.3.3.8.

7.3.3.5.5 Sec.TpduSecure.Response

7.3.3.5.5.1 General

Sec.TpduSecure.Response reports the result of a Sec.TpduSecure.Request.

7.3.3.5.5.2 Semantics

The semantics of Sec.TpduSecure.Response are as follows:

```
Sec. TpduSecure.Response (
        TSC_Pseudo_Header,
        TSC_Pseudo_Header_Size,
        TSDU,
        TSDU_Size,
        TMIC,
        TMIC_Size,
        Status)
```

Table 51 specifies the elements for Sec.TpduSecure.Response.

Table 51 – Sec. TpduSecure.Response elements

Element name	Element identifier	Element scalar type
TSC_Pseudo_Header	1	Type: OctetStringN
TSC_Pseudo_Header_Size	2	Type: Unsigned7 Valid range: 0..127
TSDU	3	Type: OctetStringN
TSDU_Size	4	Type: Unsigned Valid range: 0..Assigned_Max_TSDU_Size; see Table 30
TMIC	5	Type: OctetStringN
TMIC_Size	6	Type: Integer Valid values: 0, 4, 8, 16
Status	7	Type: Unsigned Named values: 0: success; >0: failure

7.3.3.5.5.3 When generated

The TSC generates Sec.TpduSecure.Response in response to a Sec.TpduSecure.Request. The Sec.TpduSecure.Response returns a populated security transport header, a possibly encrypted TSDU and a TMIC along with the appropriate sizes. Finally the Sec.TpduSecure.Response returns a status value that indicates either SUCCESS or the appropriate error code.

7.3.3.5.5.4 Appropriate usage

On receipt of Sec.TpduSecure.Response, the TL is notified of the result of protecting an outgoing TPDU.

7.3.3.5.6 Sec.TpduInCheck.Request

7.3.3.5.6.1 General

Sec.TpduInCheck.Request instructs the TSC to verify and possibly decrypt an incoming TL protocol data unit as appropriate.

7.3.3.5.6.2 Semantics of the service primitive

The semantics of Sec. TpduInCheck.Request are as follows:

```
Sec.TpduInCheck.Request (
    Source_Address,
    Source_Port,
    Destination_Address,
    Destination_Port,
    Payload_Size)
```

Table 52 specifies the elements for the Sec.TpduInCheck.Request.

Table 52 – Sec.TpduInCheck.Request elements

Element name	Element identifier	Element scalar type
Source_Address	1	Type: IPv6Address
Source_Port	2	Type: Unsigned16
Destination_Address	3	Type: IPv6Address
Destination_Port	4	Type: Unsigned16
Payload_Size	5	Type: Unsigned Valid range: 0..Assigned_Max_TSDU_Size

The TSC uses the Source_Address, Source_Port, Destination_Address, and Destination_Port to retrieve the appropriate policy (if any) for this security association.

7.3.3.5.6.3 Appropriate usage

The TL invokes the Sec.TpduInCheck.Request primitive to check if a secure verification and possibly a decryption of a TPDU beforehand are required.

7.3.3.5.6.4 Effect on receipt

On receipt of the Sec.TpduInCheck.Request primitive, the TSC determines if the TPDU needs to be verified and, potentially, decrypted and returns a status of success or failure.

7.3.3.5.7 Sec. TpduInCheck.Response

7.3.3.5.7.1 General

Sec.TpduInCheck.Response reports the result of a Sec.TpduInCheck.Request.

7.3.3.5.7.2 Semantics

The semantics of Sec.TpduInCheck.Response are as follows:

```
Sec.TpduInCheck.Response (
        Status)
```

Table 53 specifies the elements for Sec.TpduInCheck.Response.

Table 53 – Sec.TpduInCheck.Response elements

Element name	Element identifier	Element scalar type
Status (the result of a Sec.TpduInCheck.Request primitive)	1	Type: Unsigned Named values: 0: success; >0: failure

7.3.3.5.7.3 When generated

The TSC generates Sec.TpduInCheck.Response in response to a Sec.TpduInCheck.Request. The Sec.TpduInCheck.Response returns a status value that indicates either TRUE or FALSE depending on the policies on the current security association.

7.3.3.5.7.4 Appropriate usage

On receipt of Sec.TpduInCheck.Response, the TL is notified of the need to call the Sec.TpduVerify.Request to verify and possibly decrypt the incoming TPDU.

7.3.3.5.8 Sec.TpduVerify.Request

7.3.3.5.8.1 General

Sec.TpduVerify.Request instructs the TSC to verify and, where so configured, decrypt an incoming TPDU.

7.3.3.5.8.2 Semantics of the service primitive

The semantics of Sec.TpduVerify.Request are as follows:

```
Sec.TpduVerify.Request      (
        TSC_Pseudo_Header,
        TSC_Pseudo_Header_Size,
        TSDU,
        TSDU_Size,
```

TMIC,

TMIC_Size)

Table 54 specifies the elements for the Sec.TpduVerify.Request.

Table 54 – Sec.TpduVerify.Request elements

Element name	Element identifier	Element scalar type
TSC_Pseudo_Header	1	Type: OctetString
TSC_Pseudo_Header_Size	2	Type: Unsigned Valid range: 0..127
TSDU	3	Type: OctetString
TSDU_Size	4	Type: Unsigned Valid range: 0..Assigned_Max_TSDU_Size; see Table 30
TMIC	5	Type: OctetString
TMIC_Size	6	Type: Integer Valid values: 0, 4, 8 or 16
Priority	7	Type: Unsigned4

The TSC verifies that the TL Security Header of the TPDU has the appropriate security control (octet 1) by comparing it to the current policy. The Crypto Key Identifier (octet 2), if present, is used to retrieve the correct key material. See 7.3.3.9.

The TSC verifies the TMIC field as dictated by the current policies.

The time conveyed in the TL security header is used in the nonce construction for the authentication and, where configured, decryption of the received TPDU.

The priority is provided to the TSC in order to allow efficient implementation of replay protection.

7.3.3.5.8.3 Appropriate usage

The TL invokes the Sec.TpduVerify.Request primitive to verify and possibly decrypt a TPDU before it is transmitted.

7.3.3.5.8.4 Effect on receipt

On receipt of the Sec.TpduVerify.Request primitive, the TSC starts the appropriate TPDU processing steps to verify and, where configured, decrypt the received TPDU as dictated by the processing steps for received TPDUs in 7.3.3.9.

7.3.3.5.9 Sec.TpduVerify.Response

7.3.3.5.9.1 General

Sec.TpduVerify.Response reports the result of a Sec.TpduVerify.Request.

7.3.3.5.9.2 Semantics

The semantics of Sec.TpduVerify.Response are as follows:

Sec.TpduVerify.Response (

TSDU,
TSDU_Size,
Time_Of_TPDU_Creation,
Status)

Table 55 specifies the elements for Sec.TpduVerify.Response.

Table 55 – Sec.TpduVerify.Response elements

Element name	Element identifier	Element scalar type
TSDU (after any required decryption)	1	Type: OctetString
TSDU_Size	2	Type: Unsigned Valid range: 0..Assigned_Max_TSDU_Size; see 11.4.3.3
Time_Of_TPDU_Creation (32-bit fixed-point fractional representation of TAI time, modulo 2^{22} s, used in the nonce)	3	Type: Unsigned32
Status	4	Type: Unsigned Named values: 0: success; >0: failure

7.3.3.5.9.3 When generated

The TSC generates Sec.TpduVerify.Response in response to a Sec.TpduVerify.Request. The Sec.TpduVerify.Response returns a status value that indicates either SUCCESS or the appropriate error code.

7.3.3.5.9.4 Appropriate usage

On receipt of Sec.TpduVerify.Response, the TL is notified of the result of verifying and possibly decrypting the incoming TPDU.

7.3.3.6 TPDU security header structure

The TPDU security header structure is as described in Table 56.

Table 56 – Structure of TL security header

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Security_Control							
2 (opt)	Crypto_Key_Identifier							
3	Nominal_Time							
4								

The TL security header shall be added all TPDU to use header compression in NL. In case of no KeyDescriptor corresponding to certain TPDU, the TPDU shall be treated as no security (security level = NONE).

NOTE 1 If the TPDU has no TMIC, the UDP checksum is used for error detection.

Fields include:

- Security_Control: As defined in 7.3.1.2.
- Crypto_Key_Identifier: Specifies the current Crypto Key Identifier used to protect this TPDU.
- Nominal_Time: The time portion shall be 16 bits of TAI time, expressed modulo 2^6 s in units of 2^{-10} s, presented in MSB order.

NOTE 2 Fixing the time granularity to 2^{-10} s gives the TL the ability to transmit 1 023 TPDUs per second. With the maximum payload size of a TPDU, this is adequate for the throughput specified by 6LoWPAN. A varying granularity for TPDU time, suitable for supporting higher-rate automation processes, is a possible area of future standardization.

7.3.3.7 Nonce construction for TPDUs

This standard uses a different (but related) TPDU nonce construction than that of its DPDUs. A 13-octet nonce is required for the CCM* engine. The nonce shall be constructed as the concatenation from first (leftmost) to last (rightmost) octets of data fields as shown in Table 57, wherein:

- the EUI64Address shall be used as an array of 8 octets and the truncated TAI time;
- the nominal TAI time of the TPDU creation shall be set at a granularity of 2^{-10} s, and shall be no more than 1 s earlier than the actual local time of start of TPDU creation, and no more than 1 s later than the actual local time of end of TPDU creation. Each outgoing TPDU from a given source EUI64Address that uses a given key shall be created with a unique value for the 32-bit truncated nominal TAI time of TPDU creation. This encoding restricts the maximum data rate of the TL to 1 024 TPDUs per second, which shall not be exceeded. The structure of the 32-bit truncated nominal TAI time shall be as described in Table 58.

Table 57 – Structure of the TPDU nonce

Octet	Bits							
	7	6	5	4	3	2	1	0
1	EUI64Address							
…								
8								
9	Truncated nominal TAI time of TPDU creation							
…								
12								
13	0xFF							

The 32-bit truncated nominal representation of TAI time used in the T-nonce is described in Table 58.

Table 58 – Structure of 32-bit truncated nominal TAI time used in the T-nonce

Octet	Bits							
	7	6	5	4	3	2	1	0
1	Nominal TAI time (bits with weight $2^{21}..2^{14}$ s)							
2	Nominal TAI time (bits with weight $2^{13}..2^{6}$ s)							
3	Nominal TAI time (bits with weight $2^{5}..2^{-2}$ s)							
4	Nominal TAI time (bits with weight $2^{-3}..2^{-10}$ s)							

At a 2^{-10} s granularity, there will be 48,5 days before this 32-bit time representation repeats, thus providing at most 48,5 days before a new key needs to be deployed to avoid a potential nonce collision with resultant keystream reuse.

NOTE This representation is chosen because the sender and intended receivers are presumed to share approximately the same time sense and same nominal start time for any MAC transaction timeslot.

7.3.3.8 Processing for TPDUs to be transmitted

The inputs to the security procedure for TPDUs to be transmitted are:

- the TPDU to be secured;
- the EUI64Address of the source device;
- the nominal TAI time;
- the source and destination IPv6Addresses; and
- the source and destination port.

The outputs from this procedure are:

- the status of the procedure; and
- if this status is SUCCESS, the secured TPDU.

The security procedure for TPDUs being constructed for transmission consists of the following steps:

a) The procedure shall obtain the KeyDescriptor from Table 93 meeting the following selection criteria:
   - The entries with Type = 10 (TL). If none are available, the procedure shall return with a status of UNAVAILABLE_KEY.
   - Of those entries, the entries with the KeyLookupData matching the SourceAddress||SourcePort||DestinationAddress||DestinationPort (see Table 94) for this TPDU. If no KeyDescriptor is available and the following two conditions are both true, the procedure shall treat the TPDU as a no-security (security level = NONE) TPDU. Otherwise, the procedure shall return with a status of UNAVAILABLE_KEY.
   - Condition1: Join state of the receiving device is Provisioned or Joining (see Table 79).
   - Condition2: Source and destination ports are both for the DMAP (i.e., 0xF0B0).

   Those conditions need to be satisfied to transmit a join TPDUs that has a security level of NONE.
   - Of those entries, the entries valid for the current period, satisfying the inequality ValidNotBefore < current time < ValidNotAfter shall be selected. If none are available, the procedure shall return with a status of UNAVAILABLE_KEY.
   - Of those entries, if two or more keys are valid for the current time, the procedure shall select the key with the longest ValidNotAfter value.
   - Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the smallest ValidNotBefore.
   - Of those entries, if two or more keys have the same ValidNotBefore, the procedure shall select the key with the highest Crypto Key Identifier.

   If the procedure fails, the procedure shall handle the TPDU as no security (security level = NONE).
b) The procedure shall retrieve the policy from selected KeyDescriptor.
c) The procedure shall determine whether the TPDU to be secured satisfies the constraint on the maximum size of TPDUs, as follows:
   - The procedure shall set the size M, in octets, of the TMIC authentication field from the security level.

- The size of the Key Index field in the TL security header shall be 1 octet, if more than 1 key is valid for the current security association and 0 otherwise.
- The procedure shall determine the data expansion as Crypto Key Identifier size + M.
- The procedure shall check whether the size of the TPDU to be secured, including data expansion, is less than or equal to the Assigned_Max_TSDU_Size (see Table 30). If this check fails, the procedure shall return a status of TPDU_TOO_LONG.

d) The procedure shall build the security control octet of the TL security header. If the security level matches more than one KeyDescriptor from the current Key Descriptor, the Crypto Key Identifier shall be used with the Crypto Key Identifier Mode = 0x01; otherwise the procedure shall set the Crypto Key Identifier Mode = 0x00.

e) The procedure shall set the Crypto Key Identifier = Crypto Key Identifier from the current Key Descriptor (if present) in the TL security header. See Table 56.

f) The procedure shall build the nominal TAI time in TAITimeRounded format as outlined in Table 58.

g) The procedure shall set the Nominal_Time octets in the outgoing TL security header as the last 16 bits of the nominal TAI time in TAITimeRounded format. See octets 3 and 4 in Table 58.

h) If no Key Descriptor was found, then go to step j); otherwise, the procedure shall use the EUI64Address, the nominal TAI time in TAITimeRounded format and the 8-bit value 0xFF to build the nonce as outlined in Table 57.

i) The procedure shall use the nonce, the key material, the TPDU header, the TPDU payload and the CCM* mode of operation as described in IEEE 802.15.4:2011, 7.3.4, to secure the TPDU:

- If the SecurityLevel parameter specifies the use of encryption (see IEEE 802.15.4:2011, Table 58), the encryption operation shall be applied only to the TPDU's payload field. The corresponding payload field is passed to the CCM* transformation process described in IEEE 802.15.4:2011, 7.3.4, as the unsecured payload. The resulting encrypted payload shall be substituted for the original payload.
- The remaining fields in the TPDU, up to but not including the payload field, plus any required virtual fields, shall be passed to the CCM* transformation process described in IEEE 802.15.4:2011, 7.3.4, as the non-payload field.
- The ordering and exact manner of performing the encryption and integrity operations and the placement of the resulting encrypted data or integrity code within the TPDU payload field shall be as defined in IEEE 802.15.4:2011, 7.3.4.

j) The procedure shall return the secured TPDU and a status of SUCCESS.

7.3.3.9 Processing for received TPDUs

The input to the security procedure for received TPDUs is the TPDU to be unsecured, which contains the source and destination IPv6Addresses and the source and destination ports. The outputs from this procedure are the unsecured TPDU, the security level, the Crypto Key Identifier Mode, the key source, the key index, and the status of the procedure. All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure. Each receiver of TPDUs maintains a cache of authenticated nonce values of recently received TPDUs.

The security procedure on TPDU reception consists of the following steps:

a) The procedure shall obtain the security level and the Crypto Key Identifier Mode from the corresponding subfields of the security control field and the key index from the corresponding subfields of the Crypto Key Identifier (if present) of the security header of the incoming TPDU.

b) The procedure shall reconstruct the inferred originator's nominal TAI time of TPDU formation (see Note 2).

c) The procedure shall compare the time in step b) and the receiver's current TAI time. If the time in step b) is more than 2 s ahead of the receiver's current TAI time, or more than N seconds behind the receiver's current TAI time (where N is a policy-determined parameter whose default value is 62 s) the security process returns FAILURE_TPDU_DID_NOT_AUTHENTICATE.

d) The procedure shall obtain the KeyDescriptor from Table 93 meeting the following selection criteria:

- The entries with Type = 10 (TL).
- Of those entries, the entries with the KeyLookupData matching the SourceAddress||SourcePort||DestinationAddress||DestinationPort (see Table 94) for this TPDU. If no KeyDescriptor is available and the following two conditions are all true, the procedure shall treat the TPDU as no security (security level = NONE) TPDU. Otherwise, the procedure shall return with a status of UNAVAILABLE_KEY.
- Condition1: Join state of the receiving device is Provisioned or Joining (see Table 79).
- Condition2: Source and destination ports are for both DMAP's (i.e. 0xF0B0).

NOTE 1 This information is used when processing a received join TPDU that has a security level of NONE.

- Of those entries, the entries valid for the current period, satisfying the inequality ValidNotBefore < current time < ValidNotAfter shall be selected. If none is available, the procedure shall return with a status of UNAVAILABLE_KEY.
- Of those entries, if two or more keys are valid for the current time, the procedure shall select the key with the longest ValidNotAfter value.
- Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the smallest ValidNotBefore.
- Of those entries, if two or more keys have the same ValidNotBefore, the procedure shall select the key with the highest Crypto Key Identifier.
- If the procedure fails, the procedure shall return with a status of UNAVAILABLE_KEY.

e) The procedure shall determine whether the security level of the incoming TPDU conforms to the security level policy by comparing the SecurityLevel of the matching Key Descriptor obtained from step b) above. If there is a mismatch, the procedure shall return with a status of IMPROPER_SECURITY_LEVEL.

f) The procedure shall then use the EUI64Address of the originator, the nominal TAI time of TPDU formation from step b), the received low-order 16 bits of nominal TAI time (see Table 58) to generate the nonce outlined in Table 57.

g) The procedure shall use the nonce, the key from the Key Descriptor obtained in step d, the headers (the non-payload fields), the payload and the MIC of the incoming TPDU and the CCM* mode of operation as described in operations (see IEEE 802.15.4:2011, 7.3.5) to authenticate and, where configured for the transport association, decrypt the TPDU:

- If the security level specifies the use of encryption (see IEEE 802.15.4:2011, Table 58), the decryption operation shall be applied only to the actual TPDU payload field (see IEEE 802.15.4:2011, 5.2.2.2.2). The corresponding payload field shall be passed to the CCM* inverse transformation process described in IEEE 802.15.4:2011, 7.3.5 as the secure payload.
- The remaining fields in the TPDU, plus any required virtual fields, shall be passed to the CCM* inverse transformation process described in IEEE 802.15.4:2011, 7.3.5 as the non-payload fields (see IEEE 802.15.4:2011, Table 57).
- The ordering and exact manner of performing the decryption and integrity checking operations and the placement of the resulting decrypted data within the TPDU payload field shall be as defined in IEEE 802.15.4:2011, 7.3.5.

h) If the CCM* inverse transformation process fails, then the procedure may decrement the nominal TAI time by 64 s and repeat the above process during DSMO.pduMaxAge period. Otherwise, the procedure shall set the TPDU to be unsecured and return a status of SECURITY_ERROR.

i) The procedure shall look up the nonce of the just authenticated TPDU in the cache, with the following possible outcomes:

- The time of the nonce is older than the oldest time in the cache and the cache does not have space for an additional older entry, so the security process returns FAILURE_OVERAGE_TPDU.
- The nonce is already in the cache, so the security process returns FAILURE_DUPLICATE_TPDU.
- Any encrypted payload of the TPDU is decrypted, and the security process returns SUCCESS.

j) The procedure shall insert the nonce value in the cache, if necessary bumping from the cache the cache entry with the oldest inferred nominal TAI time of TPDU formation and return with the unsecured TPDU, the security level, the Crypto Key Identifier Mode, the key source, the key index (if present) and a status of SUCCESS.

NOTE 2 The originator's nominal TAI time of TPDU formation is inferred initially from the receiver's current TAI time and the fractional nominal TAI time of TPDU creation specified in the TPDU such that it satisfies the relationship
((current-receiver-time + 2 s) ≥ originator's-nominal-time ≥ (current-receiver-time - DSMO.pduMaxAge)).
The time duration of 2 s is intended to cover ±1 s boundary conditions.

It is permitted for the cache to be segmented into separate caches for each sending EUI64Address. It is further permitted for the cache to be segmented by the reported network-layer QoS, so that a cache that holds only a few nonces of low-priority TPDUs need not also hold dozens to hundreds of nonces for overtaking higher-priority TPDUs. It is further permitted for the cache size to be adaptive, so that repeated occurrences of the first outcome in step g) above cause the cache to grow, with appropriate reduction in cache size if and when the excess cache capacity has not been used for an extended period of time.

7.3.3.10 Detection and discard of duplicated or replayed TPDUs

See 7.3.3.9, i).

7.4 Join process

7.4.1 General

The join process describes the steps by which a new device is admitted into a standard-compliant network and obtains all the relevant information to be able to communicate with other devices as well as the system manager and security manager.

NOTE This description assumes that the joining device has a DL-protocol stack conforming to some edition of this standard. However, since this procedure is an AL protocol, it is also usable for devices that do not have a DL stack simply by omitting the DL steps.

7.4.2 Prerequisites

The join process follows the provisioning step, during which cryptographic information and non-cryptographic configuration parameters may be provided to the new device. A new device shall obtain such necessary provisioning information from the provisioning device. This is described in Clause 13. The Join_Command attribute in the DMO of a device shall be used to command the device to join the network.

A joining device shall join the target network using one of the following security approaches:

- symmetric keys;
- asymmetric keys;
- no-security.

The no-security approach does not use a secret key for transfer of join keys. Instead, it uses one of the predefined, well-known keys K_global or K_open, as specified in 7.2.2.2. In this case the MIC functions as a strong CRC, which offers no security assurances but has a very high probability of detection of errors not due to deliberate attack. In this case end-to-end secure sessions (T-associations) are not permitted.

A device implementing the symmetric-key join approach shall have both a symmetric join key and the EUI64Address of a security manager that shares that join key.

A device implementing the asymmetric-key join approach shall have a certificate signed by a certificate authority trusted by the target network.

A device implementing the no-security join approach shall have the well-known, published, non-secret symmetric key common to all standard-compliant networks, K_global or K_open, as specified in 7.2.2.2.

7.4.3 Desired device end state and properties

At the conclusion of the join process, the system shall have the following state:

- the new device and the security manager securely share a symmetric long-term master key;
- if a WISN DLE is present in the device, the new device has the required cryptographic material for that DLE to exchange DPDUs with its direct neighbors;
- if a WISN DLE is present in the device, the new device has the required non-cryptographic material and resources for that DLE to exchange DPDUs with at least one of its direct neighbors; and
- the new device shall have a contract with the system manager.

NOTE 1 A contract with the system manager includes a T-key shared between the system manager and a TLE of the new device.

When using either the symmetric-key or asymmetric-key approach, the join process provides the following security assurances:

- protection against replay attacks on join APDUs:
  - cryptographic assurance to the new device that the security manager is alive;
  - cryptographic assurance to the security manager that the new device is alive;
- authenticity:
  - cryptographic assurance that the join request comes from a device that has valid trust material;
  - cryptographic assurance that the join APDUs have not been altered;
- confidentiality:
  - cryptographic protection for the keys in the join reply, such that an eavesdropper cannot recover the transported keys.

NOTE 2 A challenge / response protocol is used for the secure join process to eliminate any need to rely, at the time of the join process, on a mutually trusted source of TAI time.

7.4.4 Join process steps common for symmetric-key and asymmetric-key approaches

7.4.4.1 General

When using the secure symmetric-key or asymmetric-key approach for the join process, the device goes through the following general steps to complete the join process.

The system manager controls the process of a new device joining the network. A non-joined device that implements a DLE conforming to this standard listens for advertisement DPDUs from local routers, whose advertisement functions are configured by the system manager.

Advertisements can be found by using active scanning, passive scanning, or a combination of both. Active scanning involves solicitation DPDUs sent by the joining device to request the transmission of advertisement DPDUs. Detailed information for active/passive scanning is found in 9.1.13.

Such an advertising router shall assist during the join process by acting as a proxy for a system manager, relative to the new device. As a proxy, this advertising router forwards the join request from the new device to a system manager and forwards the join response from that system manager to the new device. Upon receipt of a join request from a new device, a system manager processes the request, authenticates the acceptability of the request through communication with a security manager, and generates a join response in reply.

7.4.4.2 Construction of join process PDUs

The join request from a new device consists of concatenated PDUs that separate the security information, exchanged between the device and a security manager, from the non-security information, exchanged between the device and a system manager. The join response consists of similar concatenated PDUs that separate the security information from the non-security information.

The non-security information exchanged during the join process is described in 6.3.9.2. That exchange uses the method defined in 6.3.9.2.2 for the advertising router's DMO and methods defined in 6.3.9.5 for the system manager's DMSO.

The security information exchanged during the join process is dependent on the join approach used, as described in 7.4.5 for the symmetric-key approach and in 7.4.6 for the asymmetric-key approach.

In order for the new device to construct its join process PDUs and send them to the advertising router, it needs to know the EUI64Address of the advertising router. The process followed by the new device to obtain this EUI64Address is described in 9.1.14. The join process request PDUs, from the new device to the advertising router, and the join process response PDUs, from the advertising router to the new device, shall be constructed as follows, using this EUI64Address of the advertising router and the EUI64Address of the new device:

- If there is a WISN DLE present in the joining device, the DL header for these join process PDUs is constructed as described in 9.3.
- The NL header for these join process PDUs is constructed as described in 10.5.3.
- The TL header for these join process PDUs is constructed as described in 11.5. For calculation of the UDP checksum, the UDP pseudo-header for IPv6 uses the EUI64Address of the new device and the EUI64Address of the advertising router, represented as link-local IPv6Addresses of the respective devices, as the IPv6Addresses of the PDUs' source and destination.
- The DMAPs of the joining device and of the advertising router use these link-local IPv6Addresses when conveying join-request-related PDUs to their TLEs.
- At the TL, the Sec.TpduOutCheck.Response shall return with a value of 3 for the security header size and 0 for the TMIC size, indicating a TL security header with security level = 0 (NONE). Due to a (usual) lack of shared secret keys, TL security protection is not generally available for the TPDU exchange of the join request PDUs from the new device and the the join response PDUs from the advertising router.
- If there is a DLE present, the DPDU security header has a security level = 1 (MIC-32), thus using a 32-bit DMIC for join DPDUs, which shall be constructed as described in 7.3.2.5 using the D-key K_global (Crypto Key Identifier = 0). Because this key is well-known, it provides no protection against deliberate attack. Thus this 32-bit DMIC is used only to detect uninentional errors in join request and response DPDUs. The advertising D-router shall use its existing contract with the system manager to forward the join process PDUs on behalf of the device that is making the join request.

NOTE A new device that is trying to join the network does not have any contracts assigned by the system manager. Thus the communication between the new device and the advertising router is not based on any contract.

The PSMO.Security_Sym_Confirm() request and response messages should be protected by the D-key and T-key information that is distributed in the PSMO.Proxy_Security_Sym_Join() response message.

7.4.4.3 Protection of join process messages

7.4.4.3.1 General

As the new device does not have the necessary D-subnet key and a TL level T-key with the advertising router, all join process messages, other than confirm messages, between the new device and the advertising router shall use the K_global at the DL level to construct a 32-bit DMIC. At the TL level, the UDP checksum shall be used for these messages.

The security information in the join request, as well as all the information coming back from the system manager and the security manager in the join response messages, is protected using the join key. This is described in 7.4.5 for the symmetric-key approach and in 7.4.6 for the asymmetric-key approach.

7.4.4.3.2 Protection against join PDU replay attacks

To protect against a join PDU replay attack, it is recommended that the security manager check for duplicate challenges with a valid MIC from the new device. If no challenge duplicates are detected, the security manager stores the challenge value for further duplication checking. In the event of a duplicate detection, the security manager discards the PDU before processing the join PDU.

7.4.4.3.3 Protecting non-security message in the join process

7.4.4.3.3.1 General

This section describes the configuration of the security settings during the join process. The non-security related network information is configured with the DMSO.System_Manager_Join() and DMSO.System_Manager_Contract() methods. The details of the methods are specified in 6.3.9.2. Such non-security messages are protected with cryptographic operations at the AL. The non-security messages are generated in the system manager and passed to the security manager which then adds the MIC using the join key. The protected messages are transmitted to the joining device via the DMSO in the system manager. At the joining device, the MIC in the received response is validated with the same operation.

At the joining device, the MIC in the received response is validated with same operation.

7.4.4.3.3.2 MIC generation for System_Manager_Join response

The DMSO.System_Manager_Join() method is defined in 6.3.9.5. The MIC field is the most significant 4 octets in MACTag generated with the following operation.

MACTag = HMAC-MMO$_{K_join}$[Output Argument number1 .. number6 in Table 23 || EUI64Address$_{join_device}$ || Challenge$_{join_device}$].

7.4.4.3.3.3 MIC generation for System_Manager_Contract response

The DMSO.System_Manager_Contract method is defined in 6.3.9.5. The MIC field is the most significant 4 octets in MACTag generated with the following operation.

MACTag = HMAC-MMO$_{K_join}$[Output Argument number1 in Table 24 || EUI64Address$_{join_device}$ || Challenge$_{join_device}$]

7.4.4.3.3.1 Confirmation

After the DMO.Proxy_System_Manager_Join().Response and DMO.Proxy_System_Manager_Contract().Response are received, the join device sends a message to inform that the correct network information has been received by the system manager.

In the symmetric-key join process, the confirmation process is integrated in the PSMO.Security_Confirm() method specified in Table 61.

In the asymmetric-key join process, the confirmation process is accomplished with the PSMO.Network_Information_Confirmation() method specified in Table 74.

7.4.4.4 Join timers

In the join process, two timers are defined. Upon expiration of either of those timers, any information (e.g., state and received parameter) cached for particular join process shall be removed or re-initialized.

JT_1: Time duration managed in the joining device, from the time of transmission of the security join request, to the time of correct validation of the confirmation response generated by the security manager. If the joining device is not a backbone device, the actual value for JT_1 shall be set at the DauxJoinTimeout value that is distributed within the DL advertisement (see Table 127). Otherwise, the actual value for JT_1 shall be set to 60 s for the backbone device.

JT_2: Time duration managed in the security manager, from the time of reception of the security join request to the time of correct validation of the security confirmation message generated by the joining device. The actual value of JT_2 is not specified in this standard.

NOTE JT_2 can be less than JT_1.

7.4.4.5 Join process of backbone device

A backbone device joins a target network by executing the join method in the system manager's DMO instead of the advertisement router's. Therefore, the backbone device does not need to discover an advertisement router; the DPO.Target_System_Manager_Address shall be set in the provisioning phase. The overview of the backbone device join process is illustrated in Figure 45 for the symmetric-key join process and in Figure 48 for the asymmetric-key join process.

7.4.4.6 TMIC size constraints for session between join node and system manager

At the end of the join process, the security manager assigns an initial security level for the session between the joining device and the system manager. That security level shall be 0, 1, 2, 5 or 6; it shall not be 3 (MIC-128) or 7 (ENC-MIC-128), and 4 (ENC-only) is always invalid.

7.4.5 Symmetric-key join process

7.4.5.1 General

Figure 44 illustrates the messaging involved in the symmetric-key join process by which a new device shall join an operating network in which it has not recently been a participant. The flow shows the normal case in which no errors or timeouts occur. The timeouts are specified in Table 92.

On the joining device, the symmetric-key join process shall be initiated with a DMO.Proxy_Security_Sym_Join().Request and finalized with a valid PSMO.Security_Confirm().Response. On the security manager, the symmetric-key join process shall be initiated with a valid message derived from

PSMO.Security_Sym_Join().Request and finalized with a valid message derived with PSMO.Security_Confirm().Request.

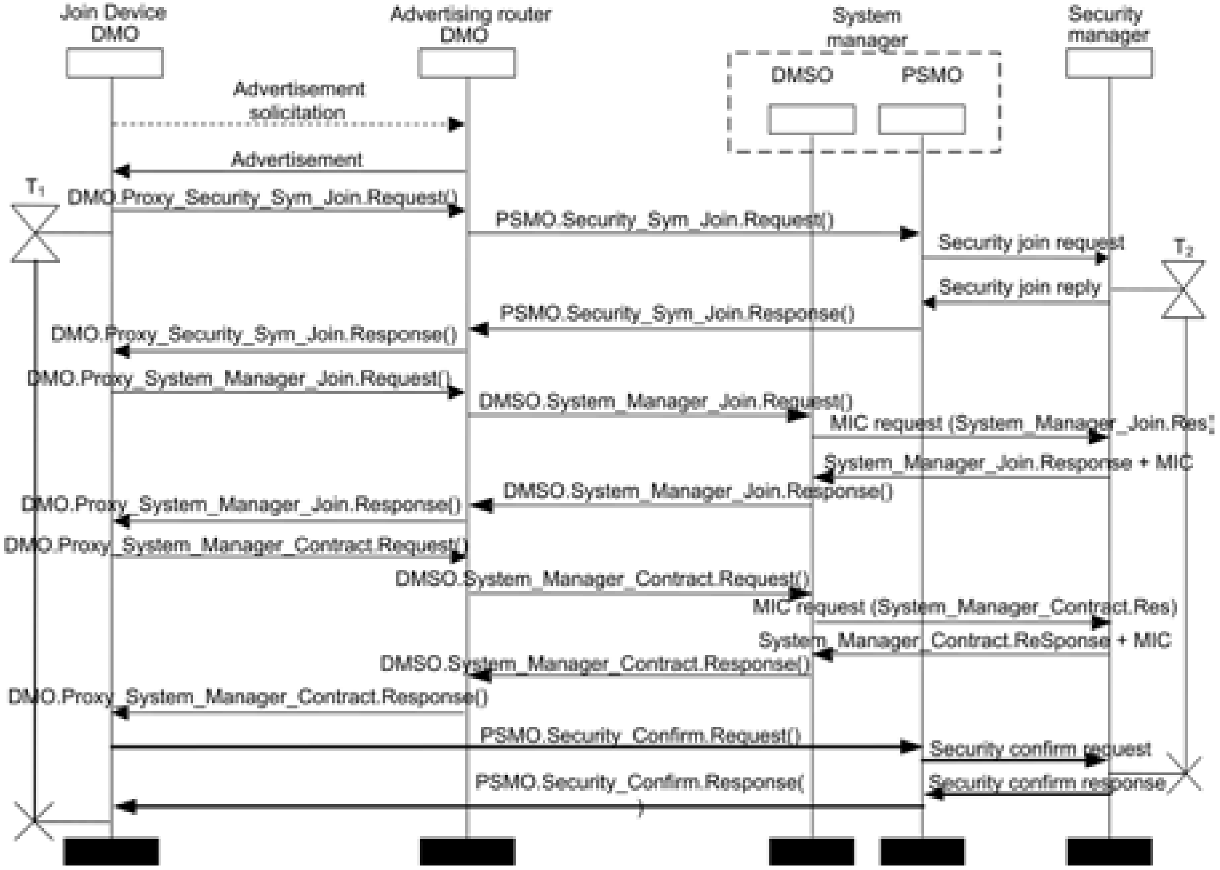


Figure 44 – Example: Overview of the symmetric-key join process

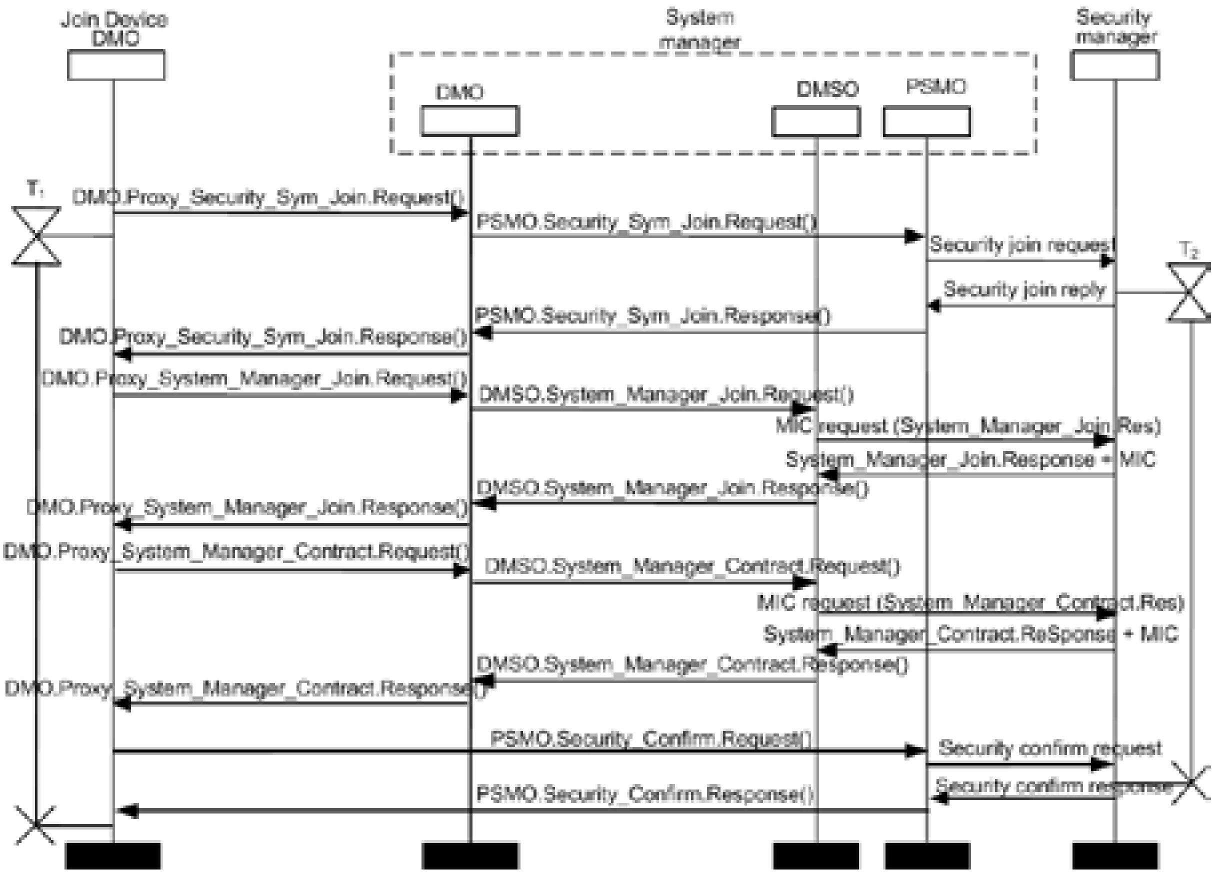


Figure 45 – Example: Overview of the symmetric-key join process of a backbone device

As shown in Figure 44, a new device shall use the methods defined for the advertising router's DMO to send and receive the join request and join response messages. The methods related to the non-security information are described in 6.3.9.2. The DMO methods related to the security information are described in 7.4.5.2. After transmission of the DMO.Proxy_Security_Sym_Join ().Request, the new device shall start to count join timer JT_1.

The advertising router shall use the methods defined for the system manager's DMSO to send and receive the non-security related join request and response messages. These DMSO methods are described in 6.3.9.5. Methods defined for the system manager's proxy security management object (PSMO) shall be used by the advertising router to send and receive the security related join request and response messages. The PSMO methods related to the security information are described in 7.4.5.2. As shown in Figure 44, the PSMO receives the security-related join request and forwards it to the security manager. The security manager may check a white or black list for the device and ask for human verification before deciding to admit or reject the joining device. If the result is positive, the security manager verifies the cryptographic information of the join request. If the checks fail, the system manager is instructed to revoke the resources allocated to the new device. If the test succeeds, the security manager does the following:

NOTE 1 The methodology of filtering the joining device in security manager is beyond the scope of this standard.

a) starts join timer JT_2;
b) generates a new master key for the new device;
c) creates a new secure session for the contract between the system manager and the new device;
d) retrieves the current D-key and Crypto Key Identifier for the new device's D-subnet;
e) generates a fresh, unique challenge for the new device;
f) cryptographically protects the aforementioned keys and forms a message integrity check code on the entire response; and

g) sends the security-related response, including the message integrity check code, back to the PSMO.

The PSMO sends this security-related response back to the advertising router which in turn forwards it to the new device.

The new device checks the cryptographic integrity of this security-related response APDU. If the test fails, the received APDU is discarded. If the test succeeds, then the security-related response is processed by the device, which cancels join timer JT_1.

The non-security related join response APDU that is generated by the system manager's DMSO is forwarded to the security manager in order to cryptographically protect the information in the APDU. Once the DMSO receives this protected APDU, it sends the protected APDU back to the advertising router which in turn forwards it to the new device. The new device checks the cryptographic integrity of this protected APDU before using the information in the APDU to complete the join process. The APDU includes information about the initial contract that the system manager established between the new device and the system manager. This contract is described in 6.3.11.2.6.7.

As part of the last step of the join process, the new device shall send back a security confirmation APDU to the security manager that contains the challenge from the security manager, authenticated by the new, shared master symmetric key. This security confirmation is sent to the PSMO which forwards it to the security manager. The PSMO method used for this is described in 7.4.5.2.

The security manager checks the confirmation message. If the test fails, the received confirmation response, the join state and the cached information for the new device shall be dropped. If the test succeeds, the security manager cancels its JT_2 timer and sends a confirmation response back to the new device.

If the new device receives a positive response to its confirmation request, it cancels its JT_1 timer.

The contract that was established between the new device and the system manager during the join process is used to support these messages.

NOTE 2 By sending the response of the challenge authenticated under the master key, the new device proves to the security manager that it was able to extract the master key, and therefore that it had the join key.

The ASL may concatenate ASDUs resulting from multiple method calls into a single TSDU, thus guaranteeing that if one is received, all are received. For example, to reduce the traffic overhead or the join time, the Proxy_Security_Sym_Join().Request and the Proxy_System_Manager_Contract().Request may be concatenated in the same TSDU sent to the advertising router's DMO.

7.4.5.2 Device management object and proxy service management object methods related to the symmetric-key join process

7.4.5.2.1 General

The new device shall use the Proxy_Security_Sym_Join method defined for the advertising router's DMO in the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response.

NOTE 1 To mitigate flooding by join messages, the system manager limits wireless resources (e.g., timeslots) assigned in advertising routers for receiving joining messages. The resources are described in 9.3.5.2.4.2.

Table 59 describes the Proxy_Security_Sym_Join method. The source object for invoking the DMO.Proxy_Security_Sym_Join().Request shall be the DMO in the Joining device's DMAP.

Table 59 – Proxy_Security_Sym_Join method

Standard object type name: device management object (DMO)				
Standard object type identifier: 127				
Method name	**Method ID**	**Method description**		
Proxy_Security_Sym_Join	5	Method to use advertising router as proxy to send security join request and get security join response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_ Request	Security_Sym_Join_Request; see 7.4.5.2.2	Security join request based on symmetric keys from new device that needs to be forwarded to security manager
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_ Response	Security_Sym_Join_Respons e; see 7.4.5.2.3	Security join response based on symmetric keys from security manager that needs to be forwarded to new device; this is protected using the join key

The advertising router shall use the Security_Sym_Join method defined for the system manager's PSMO for sending the security information that is part of the join request on behalf of the new device and to get the security information that is part of the join response.

Table 60 describes the Security_Sym_Join method.

Table 60 – Security_Sym_Join method

<table>
<tr><th colspan="5">Standard object type name: PSMO (proxy security management object)</th></tr>
<tr><th colspan="5">Standard object type identifier: 105</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="7">Security_Sym_Join</td><td>1</td><td colspan="3">Method to use the PSMO in the system manager to send security join request and get a security join response</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Join_Request</td><td>Security_Sym_Join_Request; see 7.4.5.2.2</td><td>Security join request from new device to security manager</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Join_Response</td><td>Security_Sym_Join_Response; see 7.4.5.2.3</td><td>Security join response from security manager to new device that is protected using the join key</td></tr>
</table>

As part of the last step of the join process, the new device shall use the Security_Confirm method defined for the system manager's PSMO for sending a security confirmation to the security manager.

Table 61 describes the Security_Confirm method.

Table 61 – Security_Confirm method

<table>
<tr><th colspan="5">Standard object type name: PSMO (Proxy security management object)</th></tr>
<tr><th colspan="5">Standard object type identifier: 105</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="7">Security_Confirm</td><td>2</td><td colspan="3">Method used by new device to send security confirmation to the security manager through the PSMO</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Security_Sym_Confirm</td><td>Security_Sym_Confirm; see 7.4.5.2.4</td><td>Security confirmation from new device to security manager</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

NOTE 2 Although the Security_Confirm method does not have any output arguments, the Execution response message in the Application Sublayer is returned as a result of this method.

7.4.5.2.2 Symmetric-key join request

The Security_Sym_Join_Request data structure that is used to form the symmetric-key join request is defined in Table 62.

Table 62 – Security_Sym_Join_Request data structure

Standard data type name: Security_Sym_Join_Request		
Standard data type code: 410		
Element name	**Element identifier**	**Element type**
New_Device_EUI64	1	Type: EUI64Address Classification: Constant Accessibility: Read only
128_Bit_Challenge_From_New_Device	2	Type: SymmetricKey Classification: Static Accessibility: Read/write
Algorithm_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1
MIC	4	Type: Unsigned32 Classification: Static Accessibility: Read only

Fields include:

- New_Device_EUI64 is the EUI64Address of the joining device. This EUI64Address is used by the advertising router when forwarding the message to the system manager to identify this device uniquely, as there could be multiple new devices joining at the same time.
- 128_Bit_Challenge_from_new_device is a fresh unique challenge generated by the new device to verify that the security manager is alive.
- The algorithm identifier shall be used to specify the symmetric-key algorithm used in the target network. The value of 0x0 shall be reserved. A symmetric-key algorithm of 0x01 corresponding to AES_CCM* shall be the only symmetric algorithm and mode supported for the join process.

  NOTE Currently, only AES_CCM* is defined as a symmetric-key algorithm. However, this field is prepared for algorithms for future use or national regulation.

- The MIC-32 is computed over the elements 1 through 4, using the join key and the 13 most significant octets of the challenge as nonce.

7.4.5.2.3 Symmetric-key join response

The Security_Sym_Join_Response data structure that is used to form the symmetric-key join response is defined in Table 63.

Table 63 – Security_Sym_Join_Response data structure

Standard data type name: Security_Sym_Join_Response		
Standard data type code: 411		
Element name	**Element identifier**	**Element type**
128_Bit_Challenge_From_SecurityManager	1	Type: SymmetricKey Classification: Static Accessibility: Read/write
128_Bit_Response_To_New_Device_Hash_B	2	Type: SymmetricKey Classification: Static Accessibility: Read/write
Combined_Security_Level	3	Type: Unsigned8 (see Table 64) Classification: Static Accessibility: Read/write
Master_Key_HardLifeSpan	4	Type: Unsigned16 Classification: Static Accessibility: Read/write
DL_Key_HardLifeSpan	5	Type: Unsigned16 Classification: Static Accessibility: Read/write
Sys_Mgr_Session_Key_HardLifeSpan	6	Type: Unsigned16 Classification: Static Accessibility: Read/write
DL_Key_ID	7	Type: Unsigned8 Classification: Static Accessibility: Read/write
Encrypted_DL_Key	8	Type: SymmetricKey Classification: Static Accessibility: Read/write
Encrypted_Sys_Mgr_Session_Key	9	Type: SymmetricKey Classification: Static Accessibility: Read/write

This data structure consists of a plaintext section and an encrypted section. The plaintext section shall be composed of the header, the original challenge from the new device, and a new challenge from the security manager which is different from the challenge generated by the new device, and the key policies. The encrypted section shall be composed of the D-key and the T-key with the system manager.

- 128_Bit_Challenge_From_Security_Manager is a fresh unique challenge generated by the Security Manager to verify that the new device is alive.
- 128_Bit_Response_To_New_Device_Hash_B shall be calculated as:
  - Hash_B= HMAC-MMO$_{K_join}$[challenge_from_security_manager || challenge_from_new_device || EUI64Address of new_device || EUI64Address of security_manager || Message_Key_Transport]
  - Message_Key_Transport = Combined_Security_Level || Master_Key_HardLifeSpan || DL_Key_HardLifeSpan || Sys_Mgr_Session_Key_HardLifetime || DL_Key_ID || Encrypted D-key || Encrypted SysMan T-key

- The Master_Key_HardLifeSpan, DL_Key_HardLifeSpan and Sys_Mgr_Session_Key_HardLifeSpan shall be the HardLifeSpan, in units of hours. The Key Type='001' and Key Usage can be inferred implicitly from Table 89 and Table 90 by the element Identifier. A default granularity of 0x2='hours' shall be used for the policies in the join response message.
- The DL_Key_ID shall be the Crypto Key Identifier associated with the D-key sent in the join response. The Crypto Key Identifier of the master key and T-key shall be set implicitly (not transmitted but inferred) as 0x00.
- The 13 most significant octets of the challenge sent from security manager shall be used as the nonce to encrypt D-key and T-key. The D-key and T-key are encrypted in the same time (single operation of AES-CCM* encryption with MIC size = 0).
- Response to new device shall be the keyed hash defined as follows:
- The new master key shall be derived as:
  – K_master = HMAC-MMO$_{K_join}$[EUI64Addressnew_device || EUI64Address of security_manager || challenge_from_new_device || challenge_from_security_manager]
- The D-key and the T-key to support the contract with the system manager shall be encrypted using the new master key.

NOTE 1 By including the challenge from the new device and calculating a MIC over it, the security manager proves that it is a live device with knowledge of the join key.

NOTE 2 16 bits of validity period, with units of hours, give a range of over 7 years, which is adequate to express the current maximum key lifetime.

Table 64 – Structure of compressed security level field

Octet	Bits							
	7	6	5	4	3	2	1	0
1	DL Security Level			Sys Mgr Ses Security Level			Master key Sec level	

Fields include:

- DL_Security_Level: The security level applied to the D-key conveyed in the Security_Sym_Join response message. The format of this field shall be as specified in Table 35. Security level 0, None, and security level 4, ENC, shall not be used.
- Sys_Mgr_Ses_Security_Level: The security level applied to the T-key with the system manager conveyed in the Security_Sym_Join response message. The format of this field shall be as specified in Table 35. Security level 4, ENC shall not be used.
- Master_Key_Sec_Level: The MIC size applied to the master key generated in the join process. The format of this field is defined in Table 65. Since the encryption factor is different in each message protected by the master key, only the MIC size is specified in this field. The actual security level shall be selected from Table 65 with a combination of encryption conditions in each message.

  NOTE 3 For example, since the Security_New_Session_Request and the Security_New_Session_Response data structure don't have any elements to be encrypted, the Security_Level in the Security_Control field is set to MIC-*n* with the MIC size specified in this structure. While the Security_Key_and_Policies data structure has elements to be encrypted, the Security_Level in the Security_Control field is set to ENC-MIC-*n* with MIC size specified in this structure.

Table 65 – Master key security level

Security level identifier	Master_Key_Sec_Level	Security attributes
0	0	Reserved
1	1	MIC-32
2	2	MIC-64
3	3	MIC-128

NOTE 4 Since a MIC is always used to protect the join process PDUs with the master key, the security level identifier 0 is Reserved.

- ValidNotBefore = TAI time of APDU reception;

  NOTE 5 ValidNotBefore can be inferred as the reconstructed time used in the authentication of the PDU, and is not included due to space restrictions in the response PDU.

  The time that the APDU is received shall not be more than DSMO.pduMaxAge seconds after it was created. That gives an acceptable start time.

- HardLifeSpan: The key validity duration in hours. A value of 0x0000 shall prohibit the key from expiring.

  If HardLifeSpan is zero (i.e., effectively infinite), the inferred key lifetime shall be:

  – ValidNotAfter = 0xFFFF FFFF, which is interpreted by key expiration logic as a key that never expires;

  – SoftExpirationTime = ValidNotAfter .

  If HardLifeSpan is non-zero (i.e., finite), the inferred key lifetime shall be:

  – ValidNotAfter = ValidNotBefore + (HardLifeSpan x 3600);

  – SoftExpirationTime = ValidNotBefore + (SoftLifeSpanRatio x HardLifeSpan x 3600).

  A SoftLifeSpanRatio of 50 % shall be used as a default for keys sent with a Key_HardLifeSpan field.

7.4.5.2.4 Symmetric-key security confirmation

The Security_Sym_Confirm data structure that is used to form the symmetric-key security confirmation is defined in Table 66. The source object for invoking PSMO.Security_Sym_Confirm().Request shall be DMO in joining device's DMAP.

Table 66 – Security_Sym_Confirm data structure

Standard data type name: Security_Sym_Confirm		
Standard data type code: 412		
Element name	**Element identifier**	**Element type**
128_Bit_Response_To_Security_Manager	1	Type: SymmetricKey Classification: Static Accessibility: Read/write

128_Bit_Response_To_Security_Manager shall be calculated as:

HMAC-MMO$_{K_join}$[challenge_from_new_device || challenge_from_security_manager || EUI64Address of new_device || EUI64Address of security_manager || MIC_1 || MIC_2]

where

MIC_1 is the 32-bit MIC value in System_Manager_Join response and MIC_2 is the 32-bit MIC value in System_Manager_Contract response.

NOTE 1 The join confirmation tells the security manager that the device was able to recover the master key using the join key, thus providing proof that the device, which knows the join key, is alive.

NOTE 2 The construction of the hash for the challenge-response protocol was modeled after the protocol outlined in The Handbook of Applied Cryptography, 10.17 (see Bibliography).

7.4.6 Asymmetric-key join process

7.4.6.1 Overview

The asymmetric-key join process, like the symmetric-key join process specifies the sequence of steps by which, and the conditions under which, a device may become part of the network and gain access to information required to communicate within the network, both with immediate neighboring devices and with particular infrastructure devices, such as devices assuming the role of system manager or security manager of the network. As such, this entails the sub-processes described below. Note that distribution of the keying material and resource allocation steps are identical for asymmetric-key-based and symmetric-key-based join processes. The table of roles and their respective bitmap assignment are defined in Annex B.

The enrollment process includes:

- Network membership enrollment. A device and a security manager engage in a mutual entity authentication protocol based on asymmetric-key techniques. This protocol provides evidence regarding the true device identity of both the joining device and the security manager, based on authentic asymmetric keys. In addition, admission may be based on non-cryptographic acceptability criteria (e.g., via a membership test of the device via an access control list). If the device has been positively authenticated and is authorized to join the network, it may be admitted to the network. The entity authentication protocol also results in the establishment of a shared key between the joining device and the security manager, thereby facilitating ongoing secure and authentic communications between these devices.
- Distribution of keying material. A security manager allocates keying material to a newly admitted device, so as to facilitate subsequent communications and continuous authentication of the device to other members of the network as a legitimate network device. The keying material may include D-keys, which are used to evidence network membership amongst devices in the network, and T-keys, which are used to secure and authenticate ongoing communications between a newly admitted device and a system manager.

The join process assumes that devices have been endowed with sufficient information to allow proper device authentication. A joining device may have been endowed with non-security related information as well.

The asymmetric-key key agreement scheme is specified in 7.4.6.2, the key distribution scheme is specified in detail in 7.4.6.3, the resource allocation scheme is specified in detail in 6.3.9, and the asymmetric-key based join protocol is specified in 7.4.6.3.5. The integers, octets and entities used in the asymmetric-key-based join protocol are defined in Annex F. The asymmetric-key cryptographic building blocks are defined in Annex H.

7.4.6.2 Asymmetric-key key agreement scheme

7.4.6.2.1 Overview

7.4.6.2.1.1 General

Network membership enrollment is based on the execution of the asymmetric-key key agreement scheme specified in H.4.2 and involves device authentication based on implicit certificates, as specified in H.5.1. Both schemes involve asymmetric-key techniques using elliptic curves.

7.4.6.2.1.2 Format of implicit certificate

The implicit certificate is a proof of identity and is used in the asymmetric-key join process. It can convey an arbitrary data structure; however, to support interworkability among devices, the format of the implicit certificate used in this standard is defined in Table 67.

Table 67 – Implicit certificate format

Element name	Element identifier	Element type
PublicKey_reconstruction_data	1	Type: OctetString37
Subject	2	Type: EUI64Address
Issuer	3	Type: EUI64Address
Usage_serial_number	4	Type: Usage_Serial structure (see Table 68)
ValidNotBefore	5	Type: TAITimeRounded
ValidNotAfter	6	Type: TAITimeRounded

- PublicKey_reconstruction_data: Parameter for generating a public key using the CA's public key.
- Subject: EUI64Address of a device whose public/private key is associated with PublicKey_Reconstruction_Data
- Issuer: EUI64Address of a device that has generated this certificate.
- Usage_Serial: Indicating a certification usage and serial number.
- ValidNotBefore: Absolute TAI time (in second) when this certificate becomes valid.
- ValidNotAfter: Absolute TAI time (in second) when this certificate becomes invalid.

Table 68 – Usage_serial_number structure

Octet	Bits							
	7	6	5	4	3	2	1	0
1	Reserved	Issuable	Serial_number					

- Reserved: Reserved field should be 0.
- Issuable: If this field is 0, the key pair corresponding to this certificate shall not be used to sign another certificate. Otherwise, the key pair may be used to sign another certificate.
- Serial: Serial number of this certificate managed by the issuer.

7.4.6.2.2 Description of the scheme

Figure 46 illustrates the messaging involved in the asymmetric-key key agreement scheme used with this standard.

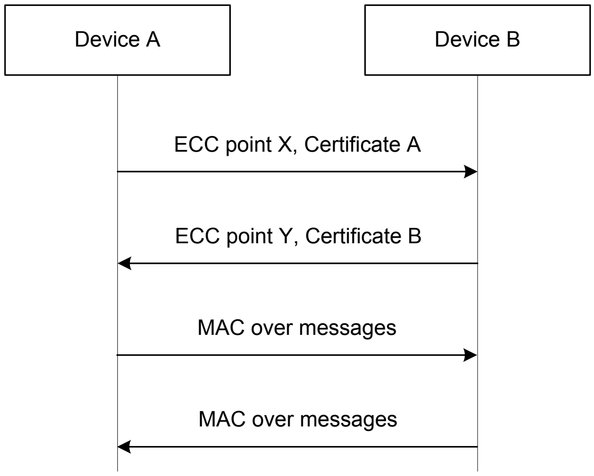


Figure 46 – Asymmetric-key-authenticated key agreement scheme

In the context of the join protocol, the key agreement scheme involves messaging between a joining device and a security manager, whereby the joining device initiates the protocol and whereby the security manager acts as the so-called responder. Thus, in terms of Figure 46, the joining device assumes the role of device A and the security manager assumes the role of device B.

The protocol includes the following sequential components:

a) Key contributions. Each party randomly generates a short-term (ephemeral) public key pair and communicates the ephemeral public key (but not the private key) to the other party. In addition, each party communicates the certificate of its long-term (static) public key to the other party.

b) Key establishment. Each party computes the shared key based on the static and ephemeral elliptic curve points it received from the other party, and also based on the static and ephemeral private keys it generated itself. Due to the properties of elliptic curves, either party arrives at the same shared key.

c) Key authentication. Each party verifies the authenticity of the long-term static key of the other party, to obtain evidence that the only party that may be capable of computing the shared key is indeed the perceived communicating party.

d) Key confirmation. Each party computes and communicates a message authentication check value over the strings communicated by the other party, to evidence possession of the shared key to the other party. This confirms to each party the true identity of the other party and proves that the other party successfully computed the shared key. This key confirmation message may authenticate an additional string communicated by the party itself as well. The strings and string operations are defined in Annex F.

The protocol assumes that each party has access to the root key of the certificate authority (CA) that signed the certificate received from the other party.

7.4.6.2.3 Security properties of the scheme

Successful execution of the complete scheme results in security properties, including the following:

- Mutual entity authentication. Each party has assurances as to the true identity of the other party and that that party was alive during the execution of the protocol.
- Mutual implicit key authentication. Each party has assurances that the only party that may have been capable of computing the shared key is indeed the intended communicating party.

- Mutual key confirmation. Each party has evidence that its intended communicating party successfully computed the shared key.
- Perfect forward secrecy. Compromise of the static key does not compromise past shared keys.
- No unilateral key control. Each party has assurance that neither party was able to control or predict the value of the shared key.
- Additional security properties, such as unknown key-share resilience and known-key security. For details, See ANSI X9.63:2011, Table H.2.

The security services provided by each scheme are assured after successful completion of the complete scheme in question (and if the prerequisites of the scheme are satisfied). From the schemes themselves, it is not clear a priori what properties are provided during execution of the protocol steps of the scheme. The security services provided include:

– Processing of random key contributions does not offer any security services, since these messages are independent.

– From the perspective of the joining device A, the protocol is finished after completion of the processing steps resulting from receipt of the key confirmation message MAC_B, whereas from the perspective of the security manager B, the protocol is only finished after completion of the processing steps resulting from receipt of the key confirmation message MAC_A. In particular, security manager B does not have any assurances prior to receipt and processing of the key confirmation message MAC_A. Thus, any actions by B triggered prior to completion of the entire protocol with A are premature, in the sense that these cannot logically be based on any security assurances (as there are none). In contrast, any actions by B triggered after successful completion of the entire protocol with A may be well-founded, in the sense that these may be based on the security services resulting from the completion of the protocol.

NOTE This re-emphasizes the importance of considering the effect of cryptographic schemes in their entirety.

7.4.6.3 Key distribution scheme

7.4.6.3.1 Overview

Key distribution is based on the shared key resulting from the asymmetric-key key agreement scheme executed between the joining device and the security manager, as described in 7.4.6.2.

7.4.6.3.2 Description of the scheme

The mechanism for distribution of keying material from the security manager to the newly joined device and the system manager is the same as that described in the symmetric-key join process. For details, see 7.4.4.

7.4.6.3.3 Security properties of the scheme

Successful execution of the key distribution scheme results in security properties including the following:

- Secure and authentic transfer of the D-key and associated keying information from the security manager to the newly joined device.
- Secure and authentic transfer of the T-key and associated keying information from the security manager to the newly joined device and to the system manager selected by the security manager.
- In either case, the distributed keying material is generated by the security manager, thereby offering unilateral key control.

7.4.6.3.4 Formats of protocol messaging

The mechanism for distribution of keying material from the security manager to the newly joined device and the system manager is the same as that described in the symmetric-key join process. For details, see 7.4.4.

7.4.6.3.5 Asymmetric-key-based join protocol

The asymmetric-key-based join protocol can be viewed as a protocol that combines the asymmetric-key key agreement scheme discussed in 7.4.6.2 and the key distribution scheme discussed in 7.4.6.3, the main difference being in the actual organization of messaging in TPDUs.

The asymmetric-key-based join protocol and the symmetric-key-based join protocol only differ in the use of an asymmetric-key key agreement scheme, rather than a symmetric-key key agreement scheme. Thus, all other aspects of the specification of the symmetric-key-based join protocol (see 7.4.4) apply to the asymmetric-key-based join protocol as well.

7.4.6.4 Asymmetric-key join process messages

7.4.6.4.1 General

Figure 47 and Figure 48 illustrate the messaging involved in the asymmetric-key join process by which a new device shall join an operating network in which it has not recently been a participant. The flow shows the normal case in which no errors or timeouts occur. The timeouts are specified in 7.4.7.3.

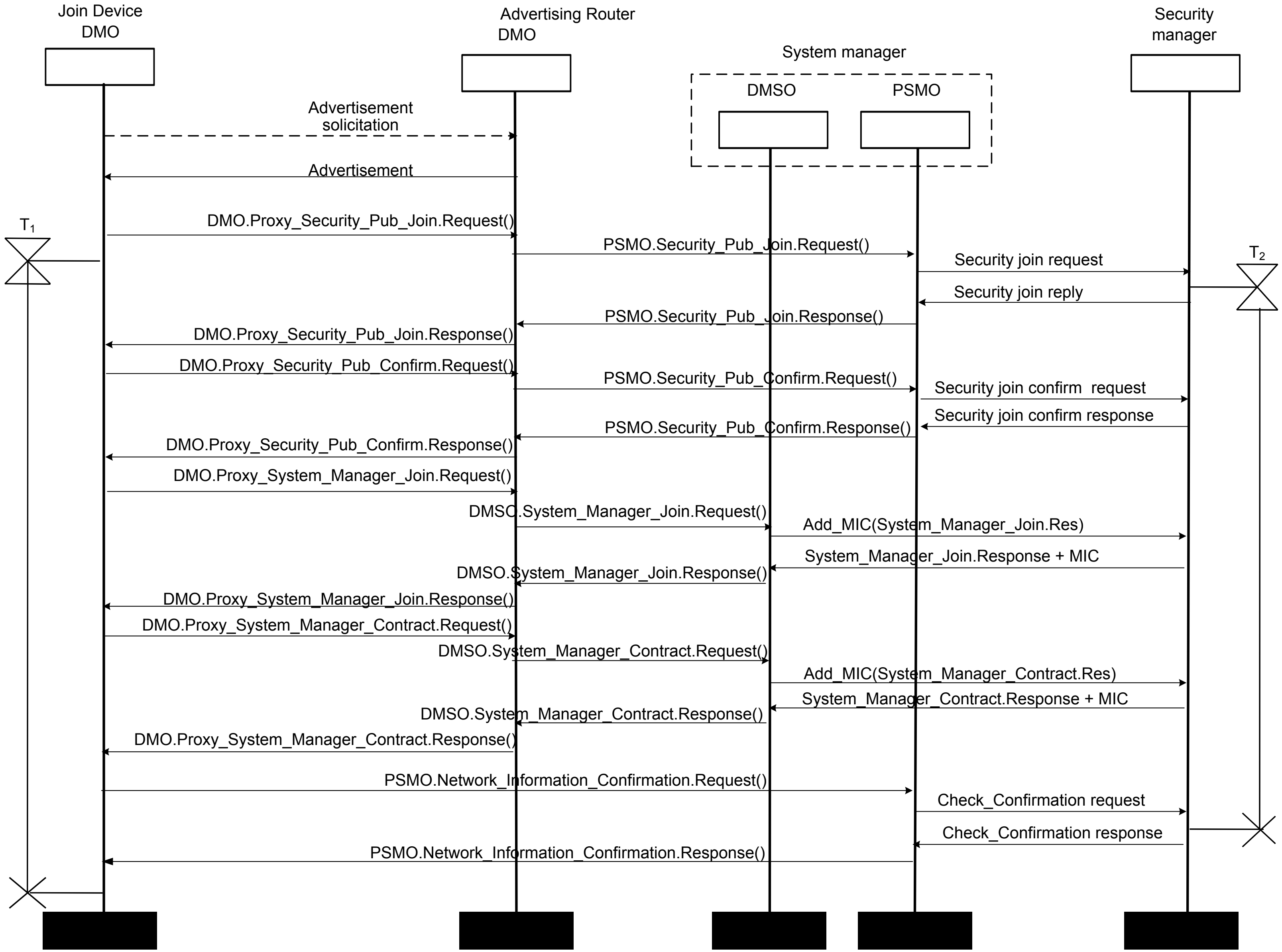


Figure 47 – Example: Overview of the asymmetric-key join process for a device with a DL

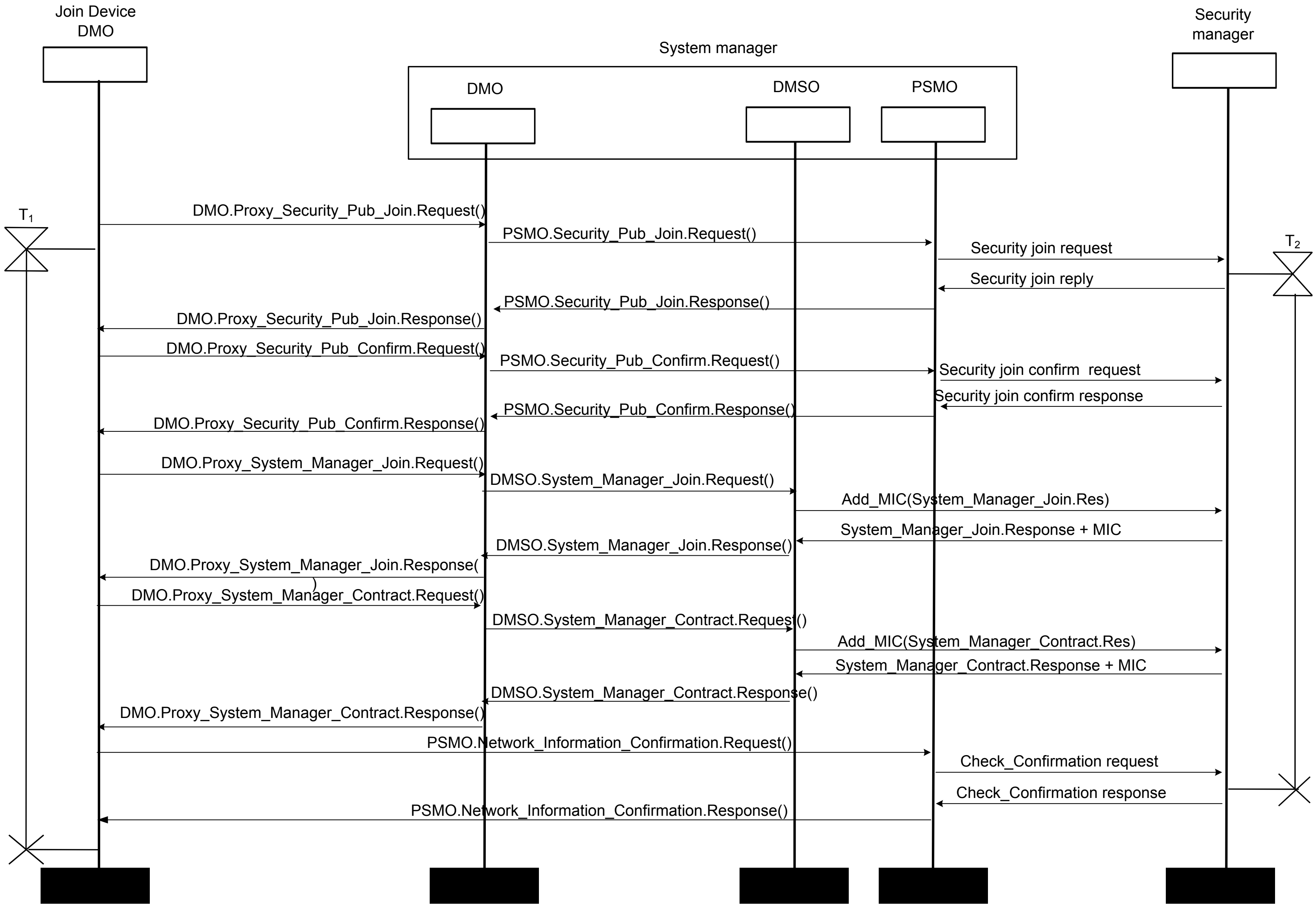


Figure 48 – Example: Overview of the asymmetric-key join process of a backbone device

On the joining device, the asymmetric-key join process shall be initiated by transmitting DMO.Proxy_Security_Pub_Join().Request and finalized by receiving a valid PSMO.Network_Information_Confirmation().Response. On the security manager, the asymmetric-key join process shall be initiated by receiving valid message derived from PSMO.Security_Pub_Join().Request and finalized by a transmitting message derived to be PSMO.Network_Information_Confirmation().Response.

As shown in Figure 48, a new device shall use the methods defined for the advertising router's DMO to send and receive the join request and join response messages. The methods related to the non-security information are described in 6.3.9.2. The DMO methods related to the security information for the asymmetric join method are described in 7.4.6.4.2.

The advertising router shall use the methods defined for the system manager's DMSO to send and receive the non-security related join request and response messages. These DMSO methods are described in 6.3.9.5. Methods defined for the system manager's proxy security management object (PSMO) shall be used by the advertising router to send and receive the security related join request and response messages. The PSMO methods related to the security information are described in 7.4.5.2.

7.4.6.4.2 Device management object and proxy security management object methods related to the asymmetric-key join process

The new device shall use the Proxy_Security_Pub_Join method defined for the advertising router's DMO in the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response. After transmitting the DMO.Proxy_Security_Pub_Join().Request, the new device shall start the join timer JT_1. After receiving the DMO.Proxy_Security_Pub_Join().Request, the security manager shall start the join timer T_2.

Table 69 describes the Proxy_Security_Pub_Join method.

Table 69 – Proxy_Security_Pub_Join method

Standard object type name: DMO (Device management object)				
Standard object type identifier: 127				
Method name	**Method ID**	**Method description**		
Proxy_Security_Pub_Join	6	Method to use advertising router as proxy to send security join request and get security join response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_Request	Security_Pub_Join_Request; see 7.4.6.4.3	Security join request based on public keys from new device that needs to be forwarded to security manager
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_Response	Security_Pub_Join_Response; see 7.4.6.4.3	Security join response based on public keys from security manager that needs to be forwarded to new device; this is protected using the join key

The advertising router shall use the Security_Pub_Join method defined for the system manager’s PSMO for sending the security information that is part of the join request on behalf of the new device and to get the security information that is part of the join response.

The source object of the DMO.Proxy_Security_Pub_Join().Request shall be the DMO in the joining device’s DMAP.

Table 70 describes the Security_Pub_Join method.

Table 70 – Security_Pub_Join method

Standard object type name: PSMO (Proxy security management object)				
Standard object type identifier: 105				
Method name	**Method ID**	**Method description**		
Security_Pub_Join	3	Method to use the PSMO in the system manager to send security join request and get a security join response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_Request	Security_Pub_Join_Request; see 7.4.6.4.3	Security join request from new device to security manager
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Join_Response	Security_Pub_Join_Response; see 7.4.6.4.3	Security join response from security manager to new device that is protected using the join key

After receiving the Proxy_Security_Pub_Join().Response message, the new device shall use the Proxy_Security_Pub_Confirm() method defined for the advertising router's DMO for sending a security confirmation to the advertising router. Table 71 describes this method. The source object of the DMO.Security_Pub_Join().Request shall be the DMO in the joining device's DMAP.

The advertising router shall use the Security_Pub_Confirm method defined for the system manager's PSMO for sending this security confirmation to the security manager. Table 72 describes this method.

The security manager is responsible for checking the confirmation message. If the test fails, the join state and cached information for the new device shall be initialized or dropped. If the test succeeds, then the security manager stops the join timer T_2, and sends a confirmation response and the non-security information responses to the new device.

If the new device receives a valid response against its confirmation request, then the new device stops the timer JT_1.

Table 71 – Proxy_Security_Pub_Confirm method

Standard object type name: DMO (Device management object)				
Standard object type identifier: 127				
Method name	**Method ID**	**Method description**		
Proxy_Security_Pub_Confirm	7	Method to use advertising router as proxy by new device for sending security confirmation		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Security_Pub_Confirm_Request	Security_Pub_Confirm_Request; see 7.4.6.4.3	Security confirmation from new device to security manager through advertising router
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Security_Pub_Confirm_Response	Security_Pub_Confirm_Response; see 7.4.6.4.3	Security confirmation from security manager to new device through advertising router

Table 72 – Security_Pub_Confirm method

Standard object type name: PSMO (Proxy security management object)				
Standard object type identifier: 105				
Method name	**Method ID**	**Method description**		
Security_Pub_Confirm	4	Method to send security confirmation of the new device to the security manager through the PSMO		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Security_Pub_Confirm_Request	Security_Pub_Confirm_Request; see 7.4.6.4.3	Security confirmation from new device to security manager through advertising router
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Security_Pub_Confirm_Response	Security_Pub_Confirm_Response; see 7.4.6.4.3	Security confirmation from security manager to new device through advertising router.

After receiving the DMO.Proxy_System_Manager_Join() and DMO.Proxy_System_Manager_Contract() response, the join device invokes PSMO.Network_Information_Confirmation() method. Table 73 describes the method.

The Confirm field in PSMO.Network_Information_Confirmation().Request is the MACTag generated with following operation:

$$MACTag = HMAC\text{-}MMO_{K\_join}[MIC_1 \,||\, MIC_2 \,||\, Challenge_{joining\_device}]$$

where:

MIC_1: MIC field in DMO.Proxy_System_Manager_Join().Response (see Table 19);

MIC_2: MIC field in DMO.Proxy_System_Manager_Contract().Response (see Table 20).

Table 73 – Network_Information_Confirmation method

<table>
<tr><th colspan="5">Standard object type name: PSMO (Proxy Security Manager Object)</th></tr>
<tr><th colspan="5">Standard object type identifier: 105</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="7">Network_Information_Confirmation</td><td>5</td><td colspan="3">Method to make sure that correct network information was received by Join Device.</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>Confirm</td><td>OctetString16</td><td>Confirmation message to make sure the join device received correct network information from the system manager</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

7.4.6.4.3 Formats of protocol messaging

7.4.6.4.3.1 Format of the join request internal structure (PK-join-1)

The Security_Pub_Join_Request data type used in the Security_Pub_Join method and Proxy_Security_Pub_Join method has the following structure and represents the first message flow of the asymmetric-key key agreement scheme (7.4.6.2). This data type is used by the new device and its proxy router in the corresponding methods of the DMO of the proxy router and the PSMO of the system manager respectively. The PK-join-1 data shall be formatted as illustrated in Table 74.

Table 74 – Format of asymmetric join request internal structure

Standard data type name: Security_Pub_Join_Request (PK-Join-1)		
Standard data type code: 415		
Element name	**Element identifier**	**Element type**
New_Device_EUI64	1	Type: EUI64Address Classification: Constant Accessibility: Read only
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 1000 0000
Ephemeral elliptic curve point X	3	Type: OctetString37 Classification: Static Accessibility: Read only
Implicit certificate of new device	4	Type: OctetString SIZE(37..66) Classification: Static Accessibility: Read/write
NOTE 1 The format of the implicit certificate used in this standard is defined in 7.4.6.2.1.2. NOTE 2 The total size of the asymmetric-key join request ranges from 83..112 octets. If the user employs this approach, the request sometimes will require more than one conveying DPDU.		

The protocol control field is 1 octet in size and specifies which algorithm is used for the asymmetric-key join protocol and which stage of the protocol is currently being executed. This subfield shall be formatted as specified in Table 75.

Table 75 – Format of the protocol control field

7	6	5	4	3	2	1	0
Algorithm ID = "10"		reserved				Join subprotocol phase	

The Algorithm ID is 2 bits in size and indicates the asymmetric-key join algorithm in use. The algorithm defined in this standard has ID 0b'10'.

The join subprotocol phase is 2 bits in size and indicates the current phase of the protocol:

0: Asymmetric-key Join Request;

1: Asymmetric-key Join Response;

2: Asymmetric-key Join Confirm Request;

3: Asymmetric-key Join Confirm Response.

7.4.6.4.3.2 Format of the asymmetric join response internal structure (PK-join-2)

The Security_Pub_Join_Response data type used in the Security_Pub_Join method and Proxy_Security_Pub_Join method has the following structure and represents the first message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-2 data shall be formatted as illustrated in Table 76.

Table 76 – Format of asymmetric join response internal structure

Standard data type name: Security_Pub_Join_Response (PK-Join-2)		
Standard data type code: 416		
Element name	**Element identifier**	**Element type**
New_Device_EUI64	1	Type: EUI64Address Classification: Constant Accessibility: Read only
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 1000 0001
Ephemeral elliptic curve point Y	3	Type: OctetString37 Classification: Static Accessibility: Read only
Implicit certificate of security manager	4	Type: OctetString SIZE(37..66) Classification: Static Accessibility: Read/write
NOTE 1 The format of the implicit certificate used in this standard is defined in 7.4.6.2.1.2. NOTE 2 The total size of the asymmetric-key join request ranges from 83..112 octets. If the user employs this approach, the request sometimes will require more than one conveying DPDU.		

7.4.6.4.3.3 Format of the first join confirmation internal structure (PK-join-3)

The Security_Pub_Confirm_Request data type used in the Security_Pub_Confirm method and Proxy_Security_Pub_Confirm method has the following structure and represents the third message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-3 data shall be formatted as illustrated in Table 77.

Table 77 – Format of first join confirmation internal structure

Standard data type name: Security_Pub_Confirm_Request (PK-Join-3)		
Standard data type code: 417		
Element name	**Element identifier**	**Element type**
New_Device_EUI64	1	Type: EUI64Address Classification: Constant Accessibility: Read only
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 1000 0010
Message_authentication_tag_MAC	3	Type: OctetString16 Classification: Static Accessibility: Read only
Size of text	4	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0 Valid range: 0..31
Text	5	Type: OctetStringN (SIZE : see element 4) Classification: Static Accessibility: Read/write

The Message_authentication_tag_MAC is generated with the following formula:

$$Message\_authentication\_tag\_MAC = MACmackey(02_{16} \mid\mid U \mid\mid V \mid\mid QEU \mid\mid QEV)$$

where:

- U is the EUI64Address of new device;
- V is the 8-octet ID of the security manager;
- QEU is the octet string of the ephemeral public key of the new device;
- QUV is the octet string of the ephemeral public key of the security manager.

This is part of the ECMQV key agreement scheme. The MACmackey is defined in ANSI X9.63:2011, 5.7. In this specification, the keyed hash function HMAC-MMO with the master key shall be used for the MACmackey function.

The text field is used to store any additional information that needs to be authenticated. Users may use this field for any information that requires protection during the asymmetric-key join process.

7.4.6.4.3.4 Format of the second join confirmation internal structure

The Security_Pub_Confirm_Response data type used in the Security_Pub_Confirm method and Proxy_Security_Pub_Confirm method has the following data structure and represents the fourth message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-4 data shall be formatted as illustrated in Table 78.

Table 78 – Format of join confirmation response internal structure

Standard data type name: Security_Pub_Confirm_Response (PK-Join-4)		
Standard data type code: 418		
Element name	**Element identifier**	**Element type**
Protocol control field	1	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 1000 0011
Message_authentication_tag_MAC	2	Type: OctetString16 Classification: Static Accessibility: Read only
Size of Text	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0
Text	4	Type: OctetString (SIZE : see element 3) Classification: Static Accessibility: Read/write
Master_Key_HardLifeSpan	5	Type: Unsigned16 Classification: Static Accessibility: Read/write
Encrypted D-key	6	Type: SymmetricKey Classification: Static Accessibility: Read/write
DL Crypto Key Identifier	7	Type: Unsigned8 Classification: Static Accessibility: Read/write
DL_Key_HardLifeSpan	8	Type: Unsigned16 Classification: Static Accessibility: Read/write
Encrypted system manager T-key	9	Type: SymmetricKey Classification: Static Accessibility: Read/write
System_Manager_Session_Key_HardLifeSpan	10	Type: Unsigned16 Classification: Static Accessibility: Read/write

The Message_authentication_tag_MAC is generated with the following formula:

Message_authentication_tag_MAC = MACmackey(03_{16} || U || V || QEU || QEV)

where:

- U is the EUI64Address of new device;
- V is the 8-octet ID of the security manager;
- QEU is the octet string of the ephemeral public key of the new device;
- QUV is the octet string of the ephemeral public key of the security manager.

This is part of ECMQV key agreement scheme. The MACmackey is defined in ANSI X9.63:2011, 5.7. In this specification the HMAC-MMO with the master key shall be used for the MACmackey function.

The text field is used to store user-determined information that needs to be authenticated. Users may use this field for any information to be protected during the asymmetric-key join process.

The key material and policy fields are the same as for the symmetric-key join process. Specifically, the following shall be the same as in 7.4.5:

- Master key compressed policy
- Encrypted D-key
- DL Crypto Key Identifier
- D-key compressed policy
- Encrypted system manager T-key
- System manager T-key compressed policy

7.4.7 Join process and device lifetime failure recovery

7.4.7.1 General

At any point during the join process, there is a possibility that a PDU will be dropped. In this case, the system should be able to recover and proceed. The following state definition and transition outline the recovery mechanism, along with triggered side effects.

7.4.7.2 Device states during the join process and device lifetime

The device states during the join process are:

- Provisioned: No master key, not in the process of getting the master key.
- Joining: No master key, in the process of getting the master key.
- Joined: Having the current master key, not in the process of getting the next master key.
- Updating: Having the current master key, in the process of getting the next master key.
- Overlapped: Having both the current master key and the next master key.

7.4.7.3 State transitions

The state transitions for a device joining the network shall be as outlined in Table 79 and Figure 49. The timeout values for the join process join_timeout (ID 4) in Table 92, shall be configurable using DL advertisements. Erasing of keys should be at least equivalent to clearing of confidential data, as defined in NIST SP800-88:2012, rev 1, Table 2-1.

Table 79 – Join process and device lifetime state machine

Transition	Current state	Event(s)	Action(s)	Next state
T1	Provisioned	DMO initiates the join process	Advertising router DMO.Proxy_Security_Sym_Join().Request or DMO.Proxy_Security_Pub_Join().Request	Joining
T2	Joining	DMO.Proxy_Security_Sym_Join().Response or DMO.Proxy_Security_Pub_Join().Response received & crypto check ok	Populate appropriate entries in DSMO and KeyDescriptor Call PSMO.Security_Confirm().Request (may be delayed), or Call PSMO.Network_Information_confirmation().Request	Joined
T3	Joined	SoftExpirationTime of master key expired	Call PSMO.Security_New_Session().Request with the security manager	Updating
T4	Updating	DSMO.New_Key().Request(master_key) from security manager via the PSMO and crypto check ok	Save master key material and policy. Set Key_ID of session to the value assigned by the security manager. Return a DSMO.New_Key().Response	Overlapped keys
T5	Joined	DSMO.New_Key().Request(master_key) from security manager via the PSMO and crypto check ok	Save master key material and policy. Set Key_ID of session to the value assigned by the security manager. Return a DSMO.New_Key().Response	Overlapped keys
T6	Overlapped keys	ValidNotAfter of old master key expired.	Remove expired master key	Joined
T7	Updating	Timeout or PSMO.Security_New_Session().Response&& crypto check ok && SESSION_DENIED	Set the next retry time	Joined
T8	Updating	ValidNotAfter of master key expired	Remove expired master key	Provisioned
T9	Joined	ValidNotAfter of master key expired	Remove expired master key	Provisioned
T10	Joining	Timeout	Reset state machine	Provisioned

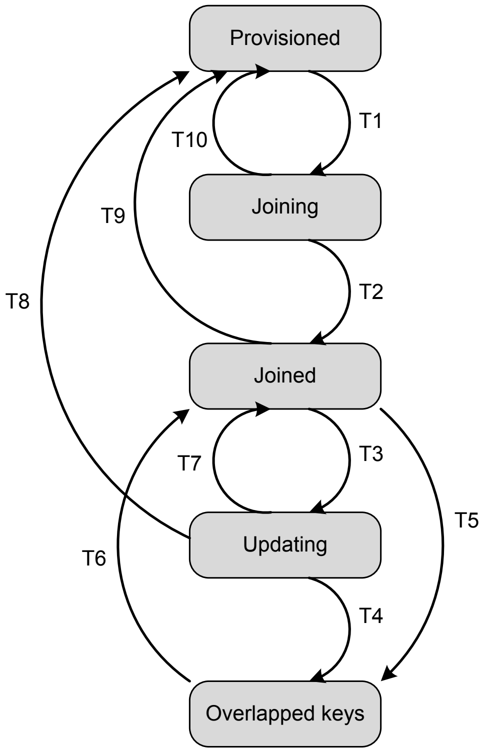


Figure 49 – Device state transitions for join process and device lifetime

7.5 Session establishment

7.5.1 General

The session establishment occurs in support of an end-to-end secure communication between two UAPs. The end point of a session is defined as the concatenation of the IPv6Address and the transport port. The security manager is responsible for granting or denying the cryptographic material used to establish the end-to-end secure channel between the two devices.

7.5.2 Description

Figure 50 provides a high-level example of session establishment.

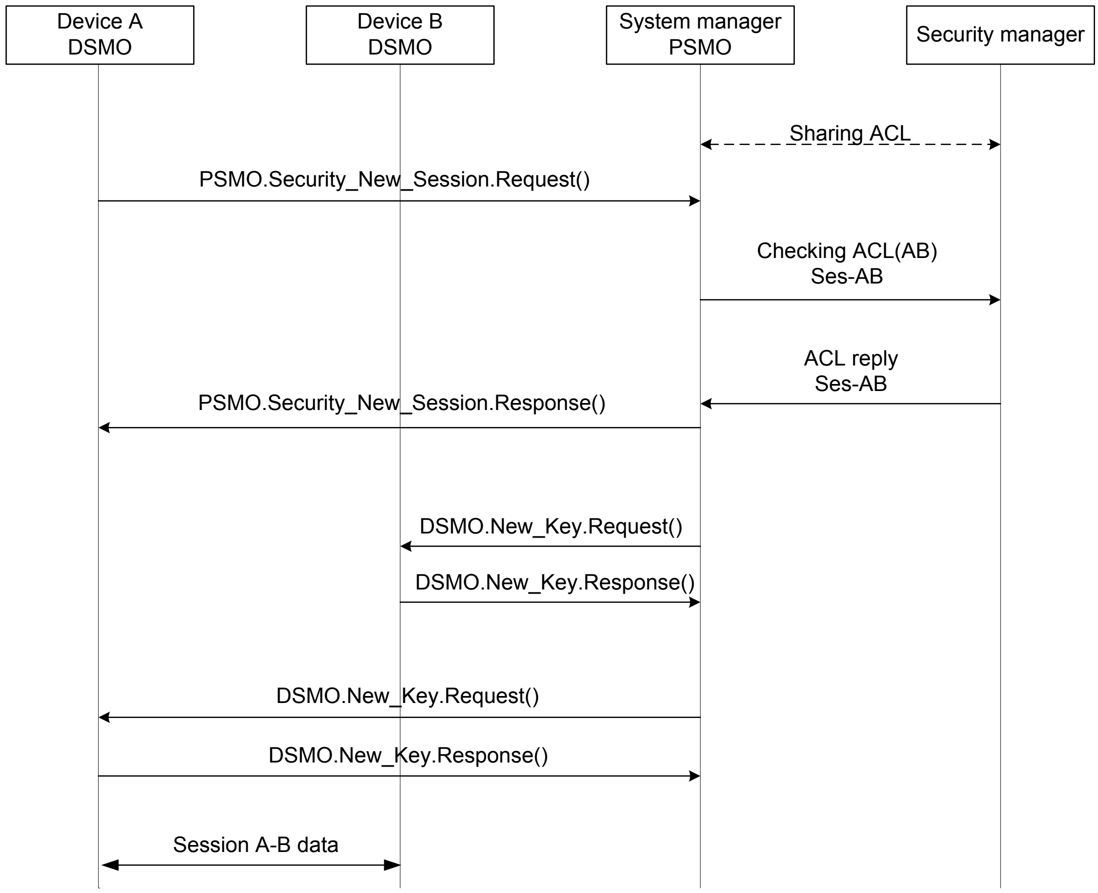


Figure 50 – High-level example of session establishment

In the high-level example shown in Figure 50, a UAP on device A establishes a session with a UAP on device B. The DSMO of device A sends the request to the security manager via the PSMO object of the system manager. The system manager then forwards the request to the security manager, which authenticates the request and may perform a check to verify if the session is allowed. If the session is granted, the security manager generates a single T-key for both end points, encrypts a copy for device A and another copy for device B, and forwards the messages to the system manager's PSMO. The system manager's PSMO can then send the response to the session request. The system manager's PSMO then calls the DSMO method to add a new T-key on device A and device B.

The session establishment may be initiated by a field device or by the security/system manager.

- The field device initiates a session establishment using the PSMO.Security_New_Session() method in Table 80.
- The system/Security manager initiates a session establishment using the DSMO.New_Key() method in Table 83.

The security manager shall assign the same key and Crypto Key Identifier among devices which participate in the secure session. In the case of overlapped keys, the security header needs to convey the Crypto Key Identifier of the key selected to protect the PDU. At the receiver, the device looks for the KeyDescriptor with the Crypto Key Identifier specified in the incoming PDU security header. If the Crypto Key Identifier value does not have a match, the receiving device will not be able to decrypt and/or authenticate the incoming PDU.

7.5.3 Application protocol data unit protection using the master key

7.5.3.1 General

The request is made from the device's DSMO to the system manager's PSMO acting as a proxy to the security manager. The APDU shall be protected with using almost the same PDU security mechanism as the TL. The cryptographic key shall be the master key, and the nonce shall be constructed in the same manner as the TL in 7.3.3.7, but the TAITimeRounded value shall be used for the Nominal TAI creation time field. See Table 363 for coding rules applied to TAITimeRounded values.

NOTE 1 Since the join process is done at the AL, there is no security at the TL during that process, except confirmation messages.

Since the granularity of the Time_Stamp field is in seconds, two cryptographic operations using the master key shall not be permitted within the same second.

NOTE 2 If the message rate using the master key exceeds the rate of once per second, there will be a nonce collision.

7.5.3.2 Replay protection for application protocol data unit protected with the master key

Upon reception of the APDU protected with the master key, the security procedure shall check for any nonce duplicates with a valid MIC in the nonce cache with the corresponding KeyDescriptor. If a duplicate nonce is detected, the procedure shall discard the PDU before processing the ASDU, otherwise the procedure shall store the nonce into nonce cache in the KeyDescriptor.

7.5.4 Proxy security management object methods related to the session establishment

Table 80 describes the Security_New_Session method.

Table 80 – Security_New_Session method

Standard object type name: PSMO (Proxy security management object)				
Standard object type identifier: 105				
Method name	**Method ID**	**Method description**		
Security_New_Session	6	Method to use the PSMO in the system manager to send security session request and get a security session response		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	New_Session_Request	Security_New_Session_Request; see Table 81	Security new session request from a device to security manager
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	New_Session_Response	Security_New_Session_Response; see Table 82	Security new session response from security manager to the requesting device, protected using the master key

The Security_New_Session_Request data structure that is used to form the session request is defined in Table 81.

Table 81 – Security_New_Session_Request data structure

Standard data type name: Security_New_Session_Request		
Standard data type code: 420		
Element name	**Element identifier**	**Element type**
Originator_IPv6Address	1	Type: IPv6Address Classification: Static Accessibility: Read/write
Originator_Port	2	Type: Unsigned16 Classification: Static Accessibility: Read/write
Destination_IPv6Address	3	Type: IPv6Address Classification: Static Accessibility: Read/write
Destination_Port	4	Type: Unsigned16 Classification: Static Accessibility: Read/write
Algorithm_Identifier	5	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = AES_CCM*
Protocol_Version	6	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = IEC 62734 Ed.1.0 (i.e., this standard)
Security_Control	7	Type: Unsigned8 Classification: Static Accessibility: Read/write
Crypto_Key_Identifier	8	Type: Unsigned8 Classification: Static Accessibility: Read/write
Time_Stamp	9	Type: TAITimeRounded Classification: Static Accessibility: Read only
MIC	10	Type: OctetString (SIZE = 4, 8, 16) Classification: Static Accessibility: Read only

This data structure consists of a plaintext section only protected using the master key shared between the requester of the session and the security manager. The EUI64Address of the requester shall be used in the nonce construction to protect this structure.

- Originator_IPv6Address shall be the IPv6Address of the first end device (usually the source) in the session.

- Originator_Port shall be the T-port of the first end UAP (usually the source) in the session.
- Destination_IPv6Address shall be the IPv6Address of the second end device (usually the destination) in the session.
- Destination_Port shall be the T-port of the second end UAP (usually the destination) in the session.
- Algorithm_Identifier defines the algorithm and mode of operation supported in this session. In the current release this shall be set to 0x1 = AES_CCM*.
- The protocol version identifies the protocol used for this security association. In this standard, this octet shall be 0x01.
- Security_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the master key security level assigned in the join process. The Crypto Key Identifier Mode shall be ‘01’ corresponding to a Crypto_Key_Identifier Field size of 1 octet.
- Crypto_Key_Identifier shall be the Crypto_Key_Identifier of the master key used in protecting this structure.
- Time_Stamp shall be the full 32-bit truncated representation of TAI time used in the T-nonce construction.
- MIC shall be the integrity code generated by the AES_CCM* computation. The size of the MIC is assigned in Security_Control field.

The nonce used to generate the MIC is formed as outlined in Table 57 with:

– EUI64Address: EUI64Address of the device transmitting the Security_New_Session Request message.

– Nominal TAI time: The Time_Stamp field in the Security_New_Session Request message.

The Security_New_Session_Response data structure that is used to form the new session response is defined in Table 82.

Table 82 – Security_New_Session_Response data structure

Standard data type name: Security_New_Session_Response		
Standard data type code: 421		
Element name	**Element identifier**	**Element type**
Status	1	Type: Unsigned8 Classification: Static Accessibility: Read/write
Security_Control	2	Type: Unsigned8 Classification: Static Accessibility: Read/write
Crypto_Key_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read/write
Time_Stamp	4	Type: TAITimeRounded Classification: Static Accessibility: Read only
MIC	5	Type: OctetString (SIZE = 4, 8, 16) Classification: Static Accessibility: Read only

Fields include:

- Status shall be the status of the session where 0x1 = SECURITY_SESSION_GRANTED and 0x0 = SECURITY_SESSION_DENIED.
- Security_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the master key security level assigned during the join process. The Crypto Key Identifier Mode shall be ‘01’ corresponding to a Crypto Key Identifier field size of 1 octet.
- Crypto_Key_Identifier shall be the Crypto_Key_Identifier of the master key used in protecting this structure.
- Time_Stamp shall be the 32-bit representation of truncated TAI time used in the nonce construction.
- MIC shall be the integrity code generated by the AES_CCM* computation. The size of the MIC is assigned in the Security_Control field.

The nonce to generate the MIC is formed as outlined in Table 57 with:

– EUI64Address: EUI64Address of the device transmitting Security_New_Session Request message.

– Nominal TAI time: The Time_Stamp field from the Security_New_Session Request message.

If the session is granted, the security manager via the PSMO of the system manager shall call the DSMO New_Key method defined in 7.6.3 to write a new T-key in the devices specified in the session request.

7.6 Key update

7.6.1 General

T-keys have a limited lifetime and are updated periodically to ensure that the session is kept alive. The key update process may be initiated by a device, although it should be pushed from the security manager between the SoftExpirationTime and the HardExpirationTime of a T-key.

7.6.2 Description

The key update process is summarized in Figure 51. A TLE may request that the security manager update a T-key. The security manager will then issue a call to the DSMO of the endpoint TLEs, via the PSMO of the system manager, to update the T-key for those TLEs. Each message is protected under the active master key shared between the security manager and the specific TLE’s DSMO.

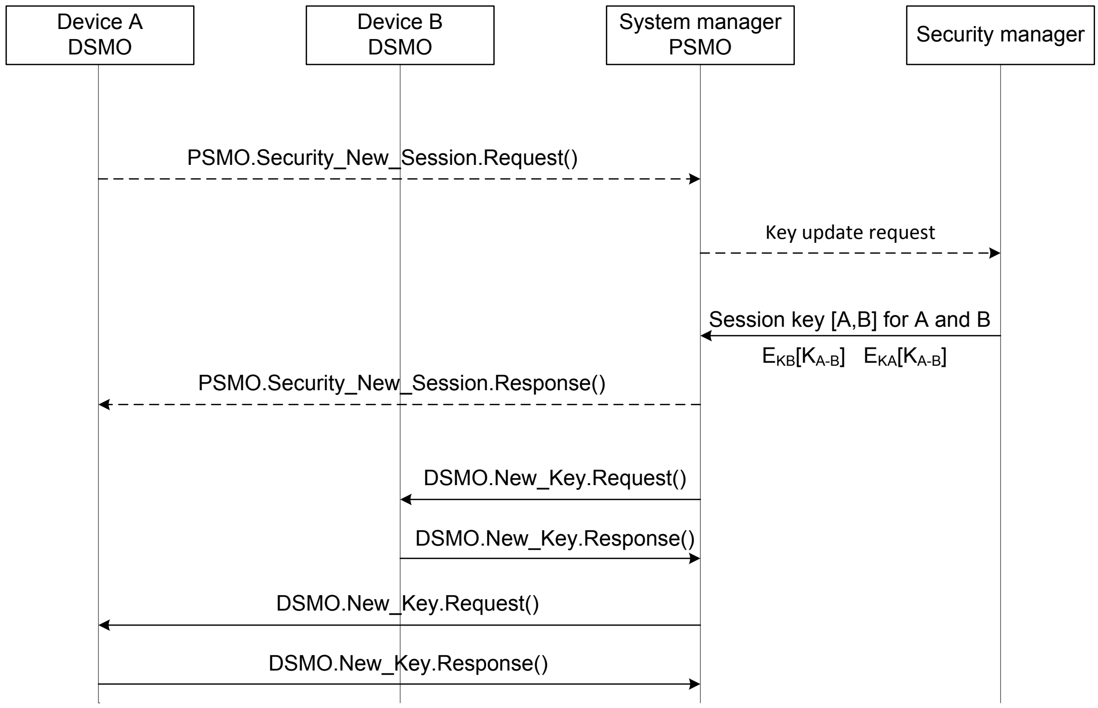


Figure 51 – Key update protocol overview

A TLE participating in a session may initiate the key update process by making a call to the PSMO Security_New_Session method. The request is forwarded from the system manager's PSMO to the security manager, which authenticates the request using the master key of the requesting device. If the check is successful, the security manager recognizes that the session already exists and simply proceeds with the key update protocol exactly as if the SoftExpirationTime of the T-key has expired. The nonce construction for protecting APDU using master key is described in 7.5.3.

If the SoftExpirationTime of an active T-key has passed, the security manager shall call the New_Key method on the DSMO of the end devices to write a new key and accompanying policies.

Key Update may also be used to update the DL and master key.

7.6.3 Device security management object methods related to T-key update

Table 83 describes the New_Key method.

Table 83 – New_Key method

Standard object type name: DSMO (Device security management object)				
Standard object type identifier: 125				
Method name	**Method ID**	**Method description**		
New_Key	1	Method to use the DSMO in the device to send a protected security key and accompanying policies.		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Key_And_Policies	Security_Key_and_Policies; see Table 84	Security key and polices to be authenticated, decrypted and stored by a device participating in a session
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Key_Update_Status	Security_Key_Update_Status; see Table 85	Status of the key update, authenticated with the master key

The Security_Key_and_Policies data structure that is used to form the New T-key request is defined in Table 84.

Table 84 – Security_Key_and_Policies data structure

Standard data type name: Security_Key_and_Policies		
Standard data type code: 422		
Element name	**Element identifier**	**Element type**
Key_Policy	1	Type: OctetString (see Table 88) Classification: Static Accessibility: Read only
End_Port_Source (elided for DL or master key)	2	Type: Unsigned16 Classification: Static Accessibility: Read/write
EUI64_remote (elided for DL, EUI64Address of security manager for master key)	3	Type: EUI64Address Classification: Static Accessibility: Read/write
128_Bit_Address_remote (elided for DL or kaster key)	4	Type: IPv6Address Classification: Static Accessibility: Read/write
End_Port_remote (elided for DL or master key)	5	Type: Unsigned16 Classification: Static Accessibility: Read/write
Algorithm_Identifier	6	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = AES_CCM*
Security_Control	7	Type: Unsigned8 Classification: Static Accessibility: Read/write
Crypto_Key_Identifier	8	Type: Unsigned8 Classification: Static Accessibility: Read/write
Time_Stamp	9	Type: TAITimeRounded Classification: Static Accessibility: Read only
New_Key_ID	10	Type: Unsigned8 Classification: Static Accessibility: Read only
Key_Material	11	Type: SymmetricKey Classification: Static Accessibility: Read only
MIC	12	Type: OctetString (SIZE = 4, 8, 16) Classification: Static Accessibility: Read only

This data structure consists of a plaintext section only protected using the master key shared between the requester of the session and the security manager. The EUI64Address of the requester shall be used in the nonce construction to protect this structure.

Fields include:

- 128_Bit_Address_remote shall be the IPv6Address of the remote endpoint TLE in this session. In the case of the D-key or master key, this field shall be elided.
- End_Port_remote shall be the T-port of the remote endpoint UAP in this session. In the case of the D-key or master key, this field shall be elided.
- Algorithm_Identifier defines the algorithm and mode of operation supported in this session. In the current release this shall be set to 0x1 = AES_CCM*.
- Security_Control shall be as defined in 7.3.1.2. The security level is chosen from ENC-MIC-32, ENC-MIC-64 and ENC-MIC-128 with master key security level assigned in join process. And the Crypto Key Identifier Mode shall be '01' corresponding to a Key Index Field size of 1 octet.
- Crypto_Key_Identifier shall be the Crypto_Key_Identifier of the master key used in protecting this structure.
- Time_Stamp shall be the 32-bit representation of truncated TAI time used in the nonce construction.
- Key_Policy shall be as described in Table 88 and populated by the security manager based on its security policies for this session.
- New_Key_ID shall be the 8-bit Crypto_Key_Identifier assigned to this key material by the security manager.
- Key_Material shall be a symmetric key used for this session.
- MIC shall be the integrity code generated by the AES_CCM* computation. The size of the MIC is specified in the Security_Control field.

The security level for the master key shall be set to at least the strength of the highest key used.

The Security_Key_and_Policies data structure is protected by AES-CCM* with the following parameters:

– Authentication part: element 1..10
– Encryption part: element 11
– Key: master key
– Nonce: formed Table 57 structure with:
– EUI64Address: EUI64Address of Security manager
– nominal TAI Time: Time Stamp element conveyed in Security_Key_and_Policies

Upon receipt of the DSMO.New_Key().Request method call, the DSMO of the end device shall decrypt and do an integrity check on the PDU using the same incoming PDU processing step as defined in the TL (see 7.3.3.9) with the nonce constructed with the EUI64Address of the security manager and the 32 bits of time included in the PDU. The key used shall be the current master key as identified by the Crypto Key Identifier.

Upon successful completion of the check, the appropriate KeyDescriptor shall be populated using the fields in the Security_Key_and_Policies data structure. In this release, the issuer is always the security manager.

The DSMO shall then generate a status message as defined in Table 85 to notify the security manager of the status of the method call.

The Security_Key_Update_Status data structure that is used to form the response to the key update request is defined in Table 85.

Table 85 – Security_Key_Update_Status data structure

Standard data type name: Security_Key_Update_Status		
Standard data type code: 423		
Element name	**Element identifier**	**Element type**
Status	1	Type: Unsigned8 Classification: Static Accessibility: Read/write
Security_Control	2	Type: Unsigned8 Classification: Static Accessibility: Read/write
Crypto_Key_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read/write
Time_Stamp	4	Type: TAITimeRounded Classification: Static Accessibility: Read only
MIC	5	Type: OctetString (SIZE = 4, 8, 16) Classification: Static Accessibility: Read only

Fields include:

- Status shall be the status of the session, where 0x1 = SECURITY_KEY_UPDATE_FAILURE and 0x0 = SECURITY_KEY_UPDATE_SUCCESS.
- Security_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the master key security level assigned during the join process. The Crypto Key Identifier Mode shall be ‘01’ corresponding to a Crypto Key Identifier Field size of 1 octet.
- Crypto_Key_Identifier shall be the Crypto_Key_Identifier of the master key used in protecting this structure.
- Time_Stamp shall be the 32-bit representation of truncated TAI time used in the nonce construction.
- MIC shall be the integrity code generated by the AES_CCM* computation. The size of the MIC is assigned in the Security_Control field.

The nonce used to generate the MIC is formed as outlined in Table 57 with:

– EUI64Address: EUI64Address of the DLE transmitting the New_Key response message.

– Nominal TAI time: The Time_Stamp element from the Security_Key_Update_Status message.

7.6.4 Failure recovery

7.6.4.1 General

At any point during the session establishment or key update process, there is a possibility that a PDU will be dropped. In this case, the system should be able to recover and proceed. The following state definitions and transitions outline the recovery mechanism, along with the triggered side effects.

7.6.4.2 T-key and D-key states

T-key and D-key states include:

- Idle: No key, not in the process of getting the current T-key.
- Establishing: No key, in the process of geting the current T-key.
- Established: Having the current key, not in the process of getting the next key.
- Updating: Having the current key, in the process of getting the next key.
- Overlapped: Having the current key and the next key

7.6.4.3 T-key and D-key state transition

The state transitions shown in Table 86 and Figure 52 show the state of the device initiating a T-key or D-key retrieval, and of its peer accepting the request. Both start in the idle state, and both end in the established state with a valid session or D-key and relevant cryptographic elements.

Table 86 – T-key and D-key state transition

Transition	Current state	Event(s)	Action(s)	Next state
T1	Idle	DSMO requested new session	Call the following method on the system manager: PSMO.Security_New_Session. Request()	Establishing
T2	Establishing	PSMO.Security_New_Session() .Response && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed	Established
T3	Idle	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed. Return a DSMO.New_Key.Response()	Established
T4	Established	Session or D-key SoftExpirationTime expired	Call the following method on the system manager: PSMO.Security_New_Session. Request()	Updating
T5	Updating	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed. Return a DSMO.New_Key.Response()	Overlapped keys
T6	Established	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Store new keys in memory	Overlapped keys
T7	Overlapped keys	ValidNotAfter of current session or D-key expired.	Remove expired key	Established
T8	Updating	Timeout OR PSMO.Security_New_Session. Response() && crypto check ok && SESSION_DENIED	Set time of next retry	Established
T9	Updating	ValidNotAfter of last session or D-key expired	Remove expired key	Idle
T10	Established	ValidNotAfter of last session or D-key expired	Remove expired key	Idle

Transition	Current state	Event(s)	Action(s)	Next state
T11	Establishing	Timeout	Reset state machine and set next retry time if necessary	Idle

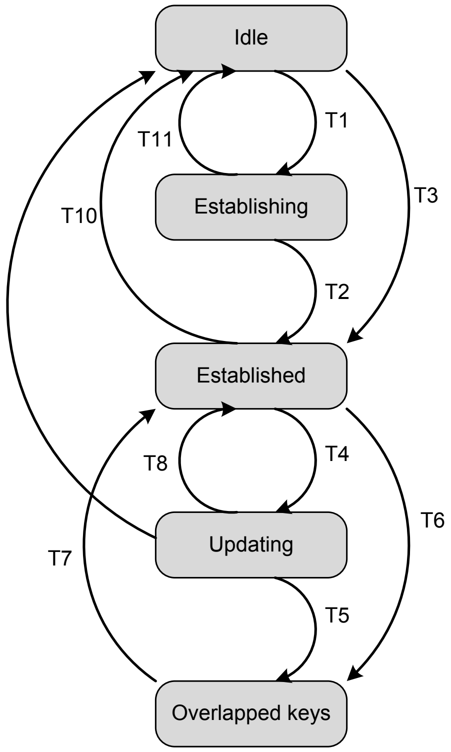


Figure 52 – Device key establishment and key update state transition

NOTE 1 If a device receives the DSMO.New_Key() request while it is in the Updating or Overlapped state, the device will discard the master key, which is not used to encrypt the new master key.

NOTE 2 If the device receives through a DSMO.New_Key() request a new master key which was encrypted using an unknown master key, the device is able to query the security manager for the needed master key for decryption with PSMO.New_Session_Request() request, to re-synchronize the master keys. The security manager has the necessary information to infer, select and use the appropriate master key.

7.7 Functionality of the security manager role

7.7.1 Proxy security management object

The attributes of the PSMO are given in Table 87.

Table 87 – Attributes of PSMO in the system manager

Standard object type name: Proxy security management object (PSMO)				
Standard object type identifier: 105				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved for future editions of this standard	1..63	—	—	—

7.7.2 Authorization of network devices and generation or derivation of initial master keys

The security manager maintains a database containing:

- a list of devices whose credentials have been established through a provisioning process or upon first attempt to join the network, and that have not been revoked; and
- a list of valid join keys and their associated lifetimes that have been issued to provisioning agent devices, which might be provided by new devices that attempt to join the network.

When a new device attempts to join the network and its request comes to the security manager via the system manager's PSMO as described in 7.4.5, the security manager examines the first two lists for a quick accept/reject decision. Otherwise, the procedure shall be as described in 7.4.6.

7.7.3 Interaction with device security management objects

A security manager interacts with a device's DSMO via the PSMO:

- during the join process;
- when distributing new keys;
- when receiving new keys that have been established by other devices through use of key agreement protocols;
- during a join process as described in 7.4.4; and
- during key recovery when a new network security manager replaces a failed one.

7.7.4 Management of operational keys

7.7.4.1 General

The security manager maintains the current master key and associated key-generation and policy attributes for each device it manages.

All symmetric keys are maintained in the security manager's operational storage, which in higher-security implementations should be physically protected within the security manager crypto module.

Where the participating devices do not both implement the set of asymmetric cryptography primitives specified in 7.4.6, which is a construction option for each device, the security manager may also generate shared-secret symmetric data keys for their unicast DL and TL associations.

NOTE Many devices do not have a high-entropy source of random bits. Without such a source, any key component generated by the device is potentially susceptible to inference.

7.7.4.2 Key archiving

Regulation or policy may require that keys be archived to permit concurrent or subsequent decryption of encrypted messaging.

7.7.4.3 Key recovery

The security manager should support two forms of key recovery:

- recovery by a field device that has lost keys that were maintained in volatile storage (e.g., RAM) due to power failure or uncorrected memory error; and
- recovery by a new network security manager of the operational keys currently in use in the network.

Each field device that supports asymmetric-key cryptography shall keep in non-volatile storage:

– its EUI64Address; and
– its public/private key pair, with the latter as a signed certificate if that is available.

Each field device that supports only symmetric-key cryptography shall keep in non-volatile storage:

– its EUI64Address;
– its join key; and
– its current join key and related keying information, if it has previously been a member of the network.

All other operational keying information may be kept in volatile storage, subject to loss upon device power failure or memory corruption, since this information can be regenerated by a security manager once that security manager has determined the EUI64Address of the device.

7.7.4.4 Security policy administration

Primary deployment options affecting network-wide security policy are selected during initial setup of the security manager for a network. Other policy deployment options may be selected at a later time.

7.8 Security policies

7.8.1 Definition of security policy

In this standard, the security policy is defined as a combination of the following parameters:

- Key Type defined in Table 89;
- Key Usage defined in Table 90;
- Key Lifetime defined in 7.2.2.4; and
- Security Level defined in 7.3.1.1.

Keys are distributed with the above parameters specified explicitly or reconstructed implicitly at the recipient. A corresponding KeyDescriptor is generated with those parameters.

7.8.2 Policy extent

Security policies constrain the security choices that individual programs and devices can make. These policies exist at the following levels:

- subnet-wide, across all devices participating in a given D-subnet, which may encompass the entire networked system;
- device-wide, across all application programs and supporting communications layers within the device;
- key-wide, across all PDUs secured with a given key; and
- link-wide, across all PDUs transmitted over a given connection defined by a source and a destination, which may include UAP ports, thus providing UAP-wide policies, across all service invocations by a given application.

Some system-wide policies shall be established before system operation begins; others can be changed dynamically while the system is operating, without interrupting ongoing sessions.

7.8.3 Unconstrained security policy choices

System security policy choices may be made during system operation. The new policy will go into effect with the next rekeying of the affected devices by a security manager. Thus, operation with a given symmetric key always has a fixed set of attributes.

7.8.4 Policy structures

The format of the policies is outlined in Table 88.

Table 88 – Structure of policy field

Octet	Bits							
	7	6	5	4	3	2	1	0
0	Key_Type			Key_Usage			Granularity	
1..4	Nominal ValidNotBefore							
5..6 (7, 8 opt)	HardLifeSpan							
9	Security_Level			Reserved				

Fields include:

- Key_Type: type of key defined in Table 89.
- Key_Usage: usage of key defined in Table 90.
- Granularity: unit in which Nominal HardLifeSpan is interpreted as defined in Table 91.
- Nominal ValidNotBefore in seconds: absolute TAI time in TAITimeRounded form, when the key recipient can start to use the key.
- HardLifeSpan: duration of time for which that key is valid. The key valid duration starts from ValidNotBefore. If the ValidNotAfter field is filled with 0x00 for any granularity, that key has an infinite lifetime and thus the key will never expire. Unless ValidNotAfter is infinite, the actual time duration of the KeyHardLifeSpan must not exceed 48,5 days (see 7.3.2.4.10) in any granularity.
- Security_Level: security level for each key, as defined in Table 35.
- Reserved: reserved field should be 0.

The possible values for the key types are outlined in Table 89.

Table 89 – Key_Type

Key_type value	Description
0	Reserved
1	Symmetric-key keying material, encrypted
2	ECC manual certificate
3	ECC implicit certificate
4..7	Reserved

The possible values for the key usage are outlined in Table 90.

Table 90 – Key_Usage

Key_Usage value	Description
0	Group key for PDU processing (i.e., D-key)
1	Link key for PDU processing (i.e., T-key)
2	Master key for session establishment
3	Join key
4	Public-key for ECMQV scheme
5	Root key CA for ECQV scheme
6	Reserved
7	Fixed global non-secret key

The granularity of the HardLifeSpan in the key policy is outlined in Table 91.

Table 91 – Granularity

Granularity	SI time unit (Note 1)	Common name	Scale factor	HardLifeSpan (octets)
0	s	second	1 s	4
1	min	minute	60 s	3
2	h	hour	3 600 s	2
3	d	day	86 400 s	2
(Note 1) Although "s" is the only official SI time unit, the other units listed in the second column are accepted for use with the SI system.				

The following policies shall be available to a network specified by this standard. The variable *k* indicates a variable that may be set by the security manager of a given network.

- Alerts and logging:

  A device keeps track of the number of failed cryptographic computation over a period of time. If a configurable threshold is exceeded, an alert is generated. Alerts include Data DPDU failure rate exceeded, TPDU failure rate exceeded, and key update failure rate exceeded. See alerts in 7.11.4.

- Device policy:

  D-authentication shall always be active with an authentication tag size of 32 bits. The default key used by a joining device is the well-known K_global used to detect random errors only. A secret key protects the higher-level APDU during a secure join. See 7.4.

NOTE 1 The DL MIC size is specified in 7.3.1.1 and the constraints for the DMIC are specified in 7.3.2.

- Key policy:

  The link, security association and PDU policy are applied through the Key policy. All users of a given key shall have the same policy; see Table 88. The configurable elements include:

  – types of key; see Table 89;

  – granularity of TAI time used in the key lifetime; see Table 91;

  – MIC size (32, 64 or 128 bits); see Table 35;

  – DMIC size of 32, 64 or 128 bits, set to 32 by default, set in the DMXHR; see 7.3.2.2;

  – TMIC size of 0, 32, 64, or 128, set to 0 if security is off, set to 32 by default if security is on. Soft lifetime; see Table 93;

  – payload encryption on/off;

- DL encryption on/off, set to off by default, set in the DMXHR; see 7.3.2.2;
- TL encryption on/off, set to off if ‘security’ is off, set to on if ‘security’ is on);
- HardLifeSpan (see Table 88), expressed as an absolute value of TAI time, limited by the maximum duration from the time of key generation that will prevent a rollover of the nonce;

  NOTE 2 This issue is linked to the TAI time granularity in the nonce (see 6.3.10); 48,5 days if used at 1 024 PDU/s.

- key originator: the EUI64Address of the generator of a given key (usually the security manager) which shall set the policy for a given key;
- allowed sessions in the security manager.

• Access control policy:

  The access control function for a session establishment is required only in the security manager. The security manager decides to grant or deny a session in the session establishment phase. The result is returned by the PSMO.Security_New_Session() method. If a security manager has both an Allowed and a Disallowed list, the security manager may indicate which one has precedence.

  - Allowed list: The security manager may have a list of allowed devices identified by valid information (e.g., EUI64Address and TAG name) listed in Table 372.
  - Disallowed list: The security manager may have a list of disallowed devices identified by valid information (e.g., EUI64Address and TAG name).

7.9 Security functions available to the AL

7.9.1 Parameters on transport service requests that relate to security

UAPs are permitted to establish application associations dynamically by requesting a session to be established. See 6.3.11.2.5.2. After a session is established, all communications from that UAP with those peers is handled securely until a period of non-use or until a need to reuse storage for security state information causes the TL to terminate the prior transport security association. If a subsequent transport service request from the UAP to those peers occurs after the transport security association has been discontinued, that subsequent request shall be treated as a new request, resulting in a new transport security association.

There is intentionally no ability to carry unsecured TPDUs on the transport security association once it has been established, since such a mechanism would be trivially easy to attack simply by altering selected authenticated TPDUs to indicate that they employed no authentication.

To support stateless AL services, the least-recently-used policy may be applied by underlying layers for recycling any resource commitments (e.g., security connection state) that they might make.

It would assist efficient security system operation if each transport service request on an association had the ability to hint at the expected interval before next use of the association. Such hinting provides guidance to the management of the implicit transport security connections needed for secured transport communications, permitting intelligent caching of established security connections and minimizing the thrashing that occurs when an implicit security connection is closed and then re-opened after a new key is established at all association participants.

The permitted security levels (see 7.3.1.1) on a transport service request are:

• encryption of the TPDU upper-layer payload: on/off;
• authentication of the TPDU with a TMIC of size 32, 64 or 128 bits.

In an API, these may be conveyed jointly as a single signed integer (e.g., an Integer8), where the sign was used to designate encryption (–) or not (+), and the magnitude was used to specify the requested size nn, with the value zero representing a request for no authentication and no encryption.

7.9.2 Direct access to cryptographic primitives

7.9.2.1 General

UAPs may use any of the cryptographic services available to a device. These include:

- unkeyed and keyed hash functions;
- pseudo-random or true-random bit string generation;
- symmetric-key cryptography;
- block cipher encryption;

NOTE 1 Exclusion of block cipher decryption makes it more likely that an implementation is able to use hardware assistance.

- stream cipher functions for processing data strings that include authentication, encryption, extended authentication with encryption, decryption, and decryption with extended authentication.

The available cryptographic primitives also may include a single construction option:

- asymmetric-key cryptography:
  - encryption with a public key and decryption with a private key of a private/public key pair;
  - signing with a private key and signature authentication with a public key of a private/public key pair;
  - key pair generation;
  - certificate generation, signing, and self-signing;
  - two-party Menezes-Qu-Vanstone key agreement;
  - Pintsov-Vanstone digital signatures.

NOTE 2 The single asymmetric-key-cryptography construction option provides all of these capabilities.

Abstract service definitions for all of the primitives of 7.9.2 are specified in 6.2.3.

7.9.2.2 Unkeyed hash functions

A secure unkeyed (or fixed-key) one-way hash function shall be provided.

The default unkeyed hash shall be Matyas-Meyer-Oseas (MMO) as specified in ISO/IEC 10118-2, based on the block cipher of 7.9.3.2.

NOTE Use of the MMO algorithm makes it more likely that an implementation is able to use hardware assistance..

Other unkeyed hash functions may be used where needed, either due to national requirement or because a larger output hash size is required for some application or to counter a threat.

An alternate cryptographic algorithm package may be required for US government systems, because MMO is not authorized for US government use. Other governments may have similar policies.

7.9.2.3 Random bits

Each device shall provide a high-quality source of random bits from a deterministic random bit generator. This may be a properly-seeded generator that is compliant with ANSI X9.82 or FIPS 186-3. Where available, the high-entropy source should be a non-deterministic random bit generator.

A high quality source of random bits shall be used in the asymmetric-key join. A properly seeded deterministic random bit generator may be used in generating challenge values in the symmetric-key join.

NOTE 1 Non-deterministic random bit generators are not suitable for direct use due to the inability to prove any statistical properties of such a source other than its non-determinism. Instead, they are used to seed and provide continuing high-entropy input to deterministic random bit generators, whose statistical properties are quantifiable. Certification of the entropy source (as the certification of the security implementation), being a highly specialized function, is best delegated to an accredited entity. NIST SP 800-22 is useful in testing non-deterministic and deterministic random bit generators.

NOTE 2 In the symmetric-key join process, it is possible to generate a seed by using the block cipher (whose default is AES) to encrypt the TAI time under the join key (i.e., Seed = Encrypt[K_join, TAI]). Such a join key is presumed to arise from a high entropy source, having been generated in the security manager and distributed during the provisioning phase.

7.9.3 Symmetric-key cryptography

7.9.3.1 Keyed hash functions

The default keyed hash shall be HMAC, based on the unkeyed hash of 7.9.2.1 (see FIPS 198).

7.9.3.2 Block cipher encryption and decryption functions

The default block cipher shall be AES-128, which has a 16 B block size and a 16 B key size (see FIPS 197).

Alternate block ciphers may be used with appropriate algorithm identifier where needed, either due to national requirements or because a larger key size or block size is required for some application or to counter some threat.

7.9.3.3 Stream cipher functions for encryption, decryption, authentication, extended authentication with encryption, and decryption with extended authentication

The security of this system is based in part on the availability of a stream cipher mode of operation of a block cipher that provides encryption/decryption, authentication, or both. When both are provided, the authentication can extend to data that is not included in the encryption/decryption process.

NOTE Encryption/decryption without authentication is avoided within TPDUs and DPDUs because there are a number of published cryptanalytic attacks that apply to all such schemes. However, the encryption-only and decryption-only modes of CCM* are available to UAPs for their use, such as for protection of the data in place.

The default stream cipher mode of operation of the block cipher of 7.9.3.2 shall be CCM* (see ISO/IEC 19772, mechanism 3). CCM* may be used for authentication-only, for extended-authentication-with-encryption, or for decryption-with-extended-authentication.

7.9.3.4 Secret key generation primitive

A secret key generation (SKG) primitive shall be used by the symmetric-key key agreement schemes specified in this standard.

This primitive derives a shared secret value from a challenge owned by an entity U_1 and a challenge owned by an entity U_2 when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

- Prerequisites: the prerequisites for the use of the SKG primitive are:
  - each entity shall be bound to a unique identifier (e.g., the EUI64Address of the device). All identifiers shall be bit strings of the same size. Entity U_1's identifier will be denoted by the bit string U_1. Entity U_2's identifier will be denoted by the bit string U_2;
  - a specialized MAC scheme shall have been chosen, with tagging transformation as specified in ANSI X9.63:2011, 5.7.1. The size in bits of the keys used by the specialized MAC scheme is denoted by macKeySize.
- Input: the SKG primitive takes as input:
  - a bit string MacKey of size macKeySize bits to be used as the key of the established specialized MAC scheme;
  - a bit string QEU_1 provided by U_1;
  - a bit string QEU_2 provided by U_2.
- Actions: the following actions are taken:
  - form the bit string consisting of U_1's identifier, U_2's identifier, the bit string QEU_1 corresponding to U_1's challenge, and the bit string QEU_2 corresponding to U_2's challenge.
- MacData = U_1 || U_2 || QEU_1 || QEU_2
  - calculate the tag MacTag for MacData under the key MacKey using the tagging transformation of the established specialized MAC scheme:
- MacTag = MACMacKey(MacData)
  - if the tagging transformation outputs invalid, also output invalid and stop;
  - otherwise, set Z=MacTag.
- Output: the bit string Z as the shared secret value.

7.10 Security statistics collection, threat detection, and reporting

Major security-related events logged by the security manager should include:

- authorizations of new devices;
- first joining of new devices to the network; and
- prolonged disappearance of devices from the network, particularly when they are expected to have a stationary presence.

NOTE Required logging of other security events is a potential subject for future standardization.

The following security-related events shall both be logged and alerted:

- MIC failure rates on received DPDUs that appear to be properly-formed, specifying the proper network-ID, that exceed a range specified in attribute 5 of the DSMO;
- MIC failure rates on received TPDUs that exceed a range specified in attribute 6 of the DSMO; and
- any integrity failure detected when unwrapping a wrapped symmetric key that exceeds a range specified in attribute 9 of the DSMO.

7.11 DSMO functionality

7.11.1 General

The device security management object (DSMO) is part of the DMAP and is the local security management application in each device. It is responsible for the agreement and exchange of cryptographic material along with associated policies. It communicates with the DSMO of the security manager via the proxy security manager object (PSMO) of the system manager. Therefore, TL security shall be used to protect the DSMO traffic, except during the join process which requires alternative special measures.

7.11.2 DSMO attributes

Table 92 describes the DSMO.

Table 92 – DSMO attributes

Standard object type name: Device security management object (DSMO)				
Standard object type identifier: 125				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
DPDU_MIC_Failure_Limit	1	The threshold of DPDU MIC failures per time unit beyond which an alert will be sent to the security manager	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 5	The value is reset to 0 after an alert is generated
DPDU_MIC_Failure_Time_Unit	2	The time interval in seconds used to determine the DPDU MIC failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 60 s	
TPDU_MIC_Failure_Limit	3	The threshold of TPDU MIC failures per time unit beyond which an alert will be sent to the security manager	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 5 Valid range: > 0	The value is reset to 0 after an alert is generated
TPDU_MIC_Failure_Time_Unit	4	The time interval in seconds used to determine the TPDU MIC failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 5	
DSMO_KEY_Failure_Limit	5	The threshold beyond which an alert will be sent to the security manager	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 1	The value is reset to 0 after an alert is generated
DSMO_KEY_Failure_Time_Unit	6	The time interval in hours used to determine the DSMO key failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 1	

Standard object type name: Device security management object (DSMO)				
Standard object type identifier: 125				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Security_DPDU_Fail_Rate_Exceeded_AlertDescriptor	7	Used to change the priority of Security_DPDU_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 6]	See alert definition
Security_TPDU_Fail_Rate_Exceeded_AlertDescriptor	8	Used to change the priority of Security_TPDU_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 6]	See alert definition
Security_Key_Update_Fail_Rate_Exceeded_AlertDescriptor	9	Used to change the priority of Security_Key_Update_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Default value: [FALSE, 6]	See alert definition
pduMaxAge	10	The maximum amount of time in seconds a PDU is allowed to stay in the network. If a PDU is received in a time window exceeding this period, it shall be rejected at the receiver	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 510 Valid range: 0..600	Set to 510 s by default.

7.11.3 KeyDescriptor

7.11.3.1 General

The information associated with a key is summarized in Table 93.

Table 93 – KeyDescriptor

Element name	Element identifier	Element scalar type
KeyLookupData	1	Type: OctetString36 Classification: Static Accessibility: Read/write See Table 94
KeyUsage	2	Type: Unsigned8 Classification: Static Accessibility: Read/write Valid range: 0..7 See Table 90
ValidNotBefore	3	Type: TAITimeRounded Classification: Static Accessibility: Read/write
SoftExpirationTime	4	Type: TAITimeRounded Classification: Static Accessibility: Read/write
ValidNotAfter	5	Type: TAITimeRounded Classification: Static Accessibility: Read/write
Issuer	6	Type: IPv6Address or EUI64Address Classification: Static Accessibility: Read/write
CryptoKeyIdentifier	7	Type: Unsigned8 or Unsigned64 Classification: Static Accessibility: Read/write
KeyMaterial	8	Type: OctetString Classification: Static Accessibility: Read/write
Security level	9	Type: Unsigned8 Classification: Static Accessibility: Read/write Valid range: 0..7 See Table 35
Counter	10	Type: Unsigned8 Classification: Static Accessibility: Read/write
NonceCache	11	Type: OctetString Classification: Dynamic Accessibility: Read/write
MICFailures	13	Type: Unsigned16 Classification: Static Accessibility: Read/write
NOTE * indicates an index field.		

The T-keyLookupData OctetString fields are listed in Table 94.

Table 94 – T-keyLookupData OctetString fields

Field name	Field scalar type
SourceAddress	Type: IPv6Address
SourcePort	Type: Unsigned16
DestinationAddress	Type: IPv6Address
DestinationPort	Type: Unsigned16

NOTE 1 Since the internal representation of the Key Descriptor is not observable, any representation aspects in the following are purely for exposition.

Key Descriptor fields include:

- KeyLookupData:
  - at the TL, used as index to find a key for a given association;
  - at the DL, this field is not used and shall be set to all 0x00;
- KeyUsage: identifies whether the key is usable as a D-key or a T-key or both;

  NOTE 2 Using a 2-bit bitmap allows a key to be defined to be used as a DL and a T-key at the same time, if allowed by the key policy.
- ValidNotBefore: time (TAI) at which the key becomes valid;
- ValidNotAfter: time (TAI) at which the key becomes invalid;
- SoftExpirationTime: time (TAI) at which an updated key is needed;
- Issuer: address of the issuer of the key; this can be an IPv6Address or an EUI64Address;
- CryptoKeyIdentifier: Crypto Key Identifier, set by the key issuer, used to distinguish keys when multiple keys are valid concurrently;
- KeyMaterial: key data for encryption/decryption and/or MIC generation;
- SecurityLevel: as described in Table 88;
- Counter: if KeyUsage bit0 is 0 (this key is not a D-key), this field is not used, so is set to 0;
- NonceCache: if KeyUsage bit1 is 0 (this key is not a T-key), this field is not used, so is set to NULL;
- MICFailures: number of MIC authentication failures after which an alarm should be generated.

7.11.3.2 Additional device security management object methods to support key management

Table 95 describes the delete key method. The result of the method invocation is stored into ServiceFeedbackCode in the application sublayer header and returned to the requesting device. The nonce construction for protecting APDU using a master key is described in 7.5.3.

Table 95 – Delete key method

<table>
<tr><th colspan="5">Standard object type name(s): Device security management object (DSMO)</th></tr>
<tr><th colspan="5">Standard object type identifier: 125</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description :</th></tr>
<tr><td rowspan="16">Delete_key</td><td>2</td><td colspan="3">This method is used to delete a symmetric key on a device. This method is evoked by the PSMO of the security manager. The method shall be protected by the current master key shared between the device and the security manager.</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>KeyUsage</td><td>Unsigned8</td><td>KeyUsage defined in Table 90</td></tr>
<tr><td>2</td><td>Crypto_Key_Identifier</td><td>Unsigned8</td><td>The Crypto_Key_Identifier used to uniquely identify keys overlapping in validity period</td></tr>
<tr><td>3</td><td>Source_Port</td><td>Unsigned16</td><td>Source port; if KeyUsage is not 0x01 (i.e. T-key), this field should be elided</td></tr>
<tr><td>4</td><td>Destination_Address</td><td>IPv6Address</td><td>Destination Address; if KeyUsage is not 0x01 (i.e. T-key), this field should be elided</td></tr>
<tr><td>5</td><td>Destination_Port</td><td>Unsigned16</td><td>Destination Port; if KeyUsage is not 0x01 (i.e. T-key), this field should be elided.</td></tr>
<tr><td>6</td><td>MasterKeyID</td><td>Unsigned8</td><td>Crypto Key Identifier for the master key used for generating MIC</td></tr>
<tr><td>7</td><td>Time_Stamp</td><td>Unsigned32</td><td>Time of creating this message in TAITimeRounded form. This argument is time portion of the nonce used for generating MIC to protect this method call</td></tr>
<tr><td>8</td><td>MIC</td><td>OctetString
(SIZE = 4, 8, 16)</td><td>The integrity check using AES_CCM*. The MIC size is chosen from MIC-32, MIC-64 and MIC-128 with master key security level assigned in join process</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type (data type and size)</th><th>Argument description</th></tr>
<tr><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

The MIC is generated by an AES-CCM* operation with the following parameters:

- authentication part: Element 1..7;
- encryption part: none;
- key: master key, which has Crypto Key Identifier = MasterKeyID;
- nonce: formed Table 57 structure with:
  - EUI64Address: EUI64Address of Security manager;
  - nominal TAI time: Time Stamp field conveyed in Delete_Key() request.

The Key_Policy_Update method is described in Table 96. The result of the method invocation is stored into ServiceFeedbackCode in the application sublayer header and returned to the requesting device. The nonce construction for protecting APDU using master key is described in 7.5.3.

Table 96 – Key_Policy_Update method

Standard object type name(s): Device security management object (DSMO)				
Standard object type identifier: 125				
Method name	**Method ID**	**Method description**		
Key_Policy_Update	3	This method is used to update a policy associated with a symmetric key on a device. This method is evoked by the PSMO of the security manager. The method shall be protected by the current master key shared between the device and the security manager.		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	KeyUsage	Unsigned8	KeyUsage defined in Table 90
	2	Crypto_Key_Identifier	Unsigned8	The Crypto_Key_Identifier used to uniquely identify keys overlapping in validity period
	3	Source_Port	Unsigned16	Source port; if KeyUsage is not 0x01 (e.g. T-key), this field should be elided
	4	Destination_Address	IPv6Address	Destination Address; if KeyUsage is not 0x01 (e.g. T-key), this field should be elided
	5	Destination_Port	Unsigned16	Destination Port; if KeyUsage is not 0x01 (e.g. T-key), this field should be elided
	6	SoftLifeSpanRatio	Unsigned8	The percentage of the HardLifeSpan beyond which a key update will be initiated
	7	Security_Level	Unsigned8	Security level specified in Table 35.
	8	MasterKeyID	Unsigned8	Crypto Key Identifier for the master key used for generating MIC
	9	Time_Stamp	Unsigned32	Time of creating this message in TAITimeRounded form. This argument is time portion of the nonce used for generating MIC to encrypt and protect this method call
	10	MIC	OctetString (SIZE = 4, 8, 16)	The integrity check using AES_CCM*. The MIC size is chosen from MIC-32, MIC-64 and MIC-128 with master key security level assigned in join process
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	—	—	—	—

The MIC is generated by an AES-CCM* operation with the following parameters:

- authentication part: element 1..7;

- encryption part: none;
- key: master key, which has Crypto Key Identifier = MasterKeyID;
- nonce: formed Table 57 structure with:
  - EUI64Address: EUI64Address of security manager;
  - nominal TAI time: Time Stamp field conveyed in Key_Policy_Update() request.

The SoftExpirationTime in the Key Descriptor is updated following a successful MIC check on the parameters of this method call. All the parameters shall be concatenated from the first element to the one the before last element (thus excluding the integrity check). The key SoftLifeSpanRatio is the percentage of the difference between the ValidNotAfter and the ValidNotBefore time. For example, a SoftLifeSpanRatio of 50% would cause a key update half way between the ValidNotBefore and the ValidNotAfter.

7.11.4 DSMO alerts

Table 97 describes the DSMO alerts.

Table 97 – DSMO Alerts

Standard object type name(s): Device security management object (DSMO)					
Standard object type identifier: 125					
Description of the alert: Security alerts on the state of the communication					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: urgent, high, med, low, journal)**	**Value data type**	**Description of value included with alert**
0 = Event	2 = Security	0 = Security_DPDU_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Default value: 0	Alert generated after the preconfigured DPDU failure rate threshold is exceeded. The value conveys the number of failures during the time period
0 = Event	2 = Security	1 = Security_TPDU_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Default value: 0	Alert generated after the preconfigured TPDU failure rate threshold is exceeded. The value conveys the number of failures during the time period
0 = Event	2 = Security	2 = Security_Key_Update_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Default value: 0	Alert generated after the preconfigured updating security key failure rate threshold is exceeded. The value conveys the number of failures during the time period

8 Physical layer

8.1 General

The physical layer (PhL) is responsible for converting the digital data information into, and from, radio frequency energy emitted, and captured, by a device's antenna. Clause 8 also specifies the operating frequencies, transmission power levels, and modulation methods used. As described in 5.2.6.2, this standard uses IEEE 802.15.4:2011 2,4 GHz DSSS as the default PhL, which it refers to as the Type A field medium. Future versions of this standard may define alternate physical layers.

The PhL provides two services, the PhL data service and the PhL management service. These services are collectively accessible via the PhSAP. The PhL data service (PhD) enables the transmission and reception of actual user data (PhPDUs) across the physical radio channel. The PhL management service is used to control the operating functions of the radio such as channel selection, transmit power selection, etc.

The structure of PhPDUs used by this standard is defined in IEEE 802.15.4:2011. Each PhPDU consists of a PHY synchronization header (PSH), a PHY coding header (PCH), and a PHY payload that is a single PhSDU. A start frame delimiter (SFD) in the SHR is commonly used as an observable timing reference.

A device employing a certified IEEE 802.15.4:2011-compliant 2,4 GHz DSSS radio generally will be allowed to operate without a site license in most countries around the world. Type licensing of the device that is acceptable to the country(ies) of intended use may be required.

8.2 Default physical layer

8.2.1 General requirements

The default physical layer shall be the Type A PhL, which shall be based on IEEE 802.15.4:2011 2,4 GHz DSSS with additional requirements and exceptions as specified herein.

Devices on mobile platforms such as ships and trains and even trucks may move between different regulatory jurisdictions. In all cases the regulations for the current locale of the device apply. One of the device configuration parameters, dlmo.CountryCode (9.1.15.6), provides the locale and regulatory-constraint guidance needed to drive conformance to the relevant regulations; thus for mobile platforms the value of this parameter shall be changed during system operation, in a timely manner, as necessary to comply with relevant regulations. See Annex V.

The device vendor and the end user are responsible for certifying that these devices are compliant with this standard and any country- or region-specific regulations.

8.2.2 Additional requirements of IEEE 802.15.4:2011

8.2.2.1 Over-the-air data rate

The PhL shall support a raw (over-the-air) data rate of 250 kbit/s.

8.2.2.2 Timing requirements

This standard requires that the PhLE support changing the channel for every PhPDU transmitted. Timing requirements are specified in Table 98.

Table 98 – Timing requirements

Event	Requirement
Time to change RF channels	< 200 μs
Time to switch from receive to transmit (with PA on)	< 200 μs
Time to switch from transmit (with PA on) to receive	< 200 μs
Inter-reception preparation time	< 200 μs
where PA refers to any RF power amplifier in the apparatus	

8.2.2.3 Carrier sense mode selection

IEEE 802.15.4:2011 2,4 GHz DSSS physical layer supports the use of a CSMA/CA scheme to reduce collisions and increase coexistence. This scheme can delay transmission of a PhPDU excessively, due to repeated random back-off delays during channel acquisition.

The PhLE shall select the mode of CSMA/CA operation on a D-transaction by D-transaction basis as requested by the DLE, where that selection depends on system configuration, including the regulatory regime under which the wireless system is operating, as constrained by dlmo.CountryCode (9.1.15.6).

8.2.2.4 Number of channels

The PhLE shall support as a minimum IEEE 802.15.4:2011 2,4 GHz DSSS channels 11..25. Support of IEEE 802.15.4:2011 channel 26 is optional where its use is permitted by regulatory constraints, and prohibited where regulations prohibit its use.

NOTE Use of channel 26 is optional due to commonly encountered regulatory constraints near the band edge.

8.2.2.5 Transmit power limits

As specified by IEEE 802.15.4:2011, the PhLE shall support a minimum full power level of -3 dBm, measured in accordance with the regulations to which the device is being certified.

The PhLE shall provide adjustable transmit power from -5 dBm to the maximum power of the device, in increments as specified by the PhE’s TXPowerTolerance attribute, as specified in the IEEE 802.15.4:2011 PHY PIB.

As required by IEEE 802.15.4:2011, 8.1.5, the maximum radiated power level shall not exceed the regulatory requirements that apply where the device is deployed, as constrained by dlmo.CountryCode (9.1.15.6).

8.2.3 Exceptions to the IEEE 802.15.4:2011 physical layer

8.2.3.1 General

The requirements of this standard that are deviations or omissions from the IEEE 802.15.4:2011 physical layer are listed here.

8.2.3.2 Limitation of frequency bands and modulation classes

Although the IEEE 802.15.4 physical layer supports multiple frequency bands and modulation classes, a device compliant with this standard shall operate in the license-exempt 2 400..2 483,5 MHz band using DSSS modulation (and coding at 250 kbit/s), which is specified in IEEE 802.15.4:2011, Table 66, as 2 450 DSSS. This standard does not support any of the other frequency bands or data rates or modulation and coding techniques specified in IEEE 802.15.4.

9 Data-link layer

9.1 General

9.1.1 Overview

The data-link layer (DL) in this standard is designed with the general goal of constraining the range of recognized construction options for a field device, while enabling flexible and innovative system solutions.

The DL specification provides a set of capabilities that are well defined and verifiable for each device that participates in a D-subnet. The DLE can be conceptualized as a table-driven state machine that operates independently on each device. A D-subnet is a group of DLEs provided with a matched set of table-driven configurations by the system manager.

The DL’s building blocks include timeslots, superframes, links, and graphs. The system manager may assemble these building blocks to configure a DLE in one of three general operational alternatives, slotted-channel-hopping, slow-channel-hopping, and hybrid slotted/slow-channel-hopping. (See 9.1.7.2 for a discussion of channel-hopping.)

A timeslot is a single, non-repeating period of time. The timeslot durations in this standard are configurable to a fixed value such as 10 ms or 12 ms. Once a timeslot duration is selected, all timeslots generally have the same duration and they are re-aligned to a 4 Hz cycle at each 250 ms clock interval. (See 9.1.9 for a discussion of DLE timekeeping.)

A superframe is a collection of timeslots repeating on a cyclic schedule. The number of timeslots in a given superframe determines how frequently each timeslot repeats, thus setting a communication cycle for DLEs that use the superframe. The superframe also has an associated reference channel-hopping pattern. (See 9.1.8 for a discussion of superframes.)

Links are connections between DLEs. When the system manager defines paths between DLEs, the DLEs receive link assignments. A link assignment repeats on a cyclic schedule, through its connection to an underlying superframe. Each link refers to one timeslot or a group of timeslots within a superframe, its type (transmit and/or receive), information about the DLE’s neighbor (the DLE on the other end of the link), a channel offset from the superframe’s underlying channel-hopping pattern, and transmit/receive alternatives.

This standard supports graph routing as well as source routing. A directed graph is a set of directed links that is used for routing Data DPDUs within a D-subnet. Each directed graph within the D-subnet is identified by a graph ID. In source routing, the originating DLE designates the hop-by-hop route that a Data DPDU takes through a D-subnet. Graph routing and source routing may be mixed. (See 9.1.6 for a discussion of routing.)

9.1.2 Coexistence strategies in the DL

This standard incorporates several strategies that are used simultaneously to optimize coexistence with other users of the 2,4 GHz radio spectrum, as described in 4.6.10. Most of these strategies are handled adaptively by the DLE in conjunction with the system manager.

9.1.3 Allocation of digital bandwidth

The DLE is a table-driven state machine that provides prioritized access to digital bandwidth for directional communication among DLEs within a D-subnet. The state machine operates on one timeslot at a time.

Digital bandwidth is allocated by a system management function. For example, a field device may need to report every 10 s. A level of service is arranged through the system management function, to ensure that the digital bandwidth is available when needed. The system management function, in turn, arranges the digital bandwidth available to the field device's DLE and any required intermediate router DLEs.

A link is the basic unit of service within the DL. A link may be incoming, outgoing, or bidirectional. It may be unicast or broadcast (see 9.1.9.4.2).

DL digital bandwidth may be allocated to deliver an average level of service, for example, 10 Data DPDUs of available digital bandwidth (multi-hop) per minute. Alternatively, DL digital bandwidth may be allocated to support a reporting interval and a level of service, for example, one Data DPDU (including retries) every 15 s, with 2 s maximum latency to a backbone connection.

DL digital bandwidth may be organized as a pool that can be shared by a collection of DLEs using the corresponding links. A level of service may be delivered by ensuring that sufficient shared, contention-based capacity is available.

For more granular channel allocation, links may be tied to particular groups of Data DPDUs.

A service level may be delivered with a combination of specific link allocations and generally available shared digital bandwidth. For example, for each report, a dedicated link may be allocated for the first transmission to each of two neighbors, with retries using shared digital bandwidth.

9.1.4 Structure of the DPDU

The general structure of a data-link protocol data unit (DPDU) in this standard is shown in Figure 53.

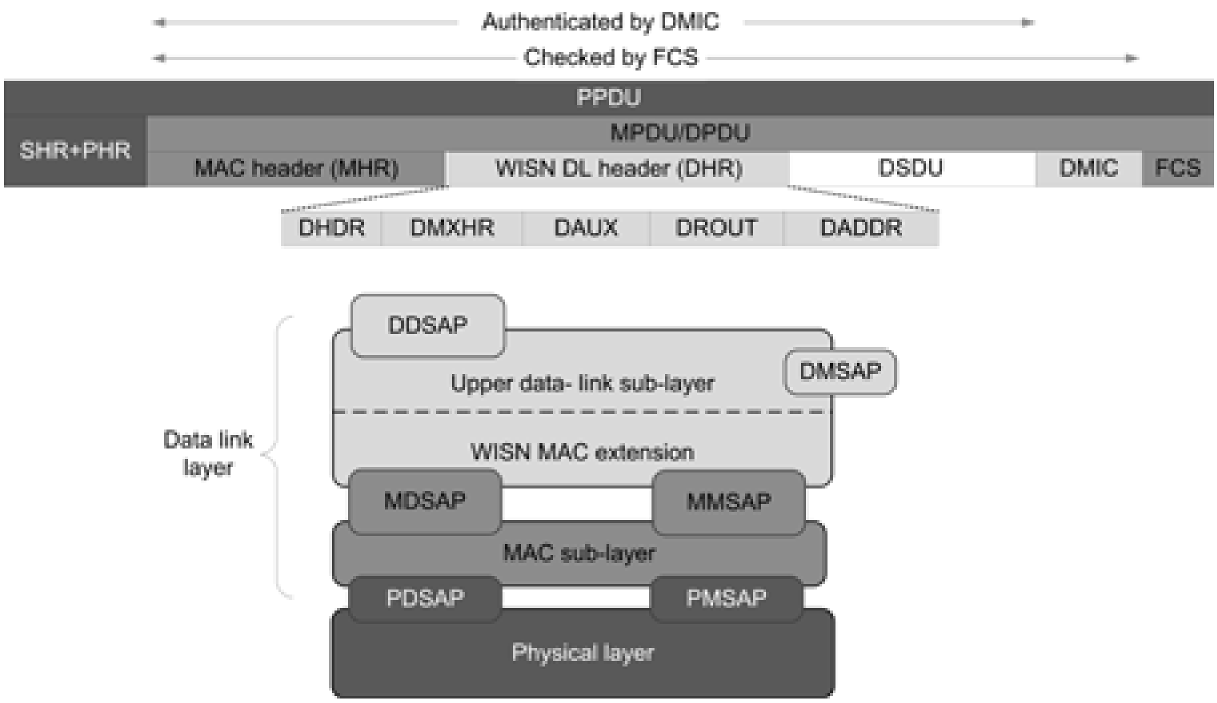


Figure 53 – DL protocol suite and PhPDU/DPDU structure

The DL specified by this standard includes:

- A subset of the IEEE 802.15.4 MAC, as described in 9.1.5. This handles the low-level mechanics of sending and receiving individual DPDUs (all of which are classified as "data"

DPDUs by IEEE 802.15.4:2011). The SHR, PHR, MHR, and frame check sequence (FCS) of every DPDU are as described and specified in IEEE 802.15.4.

- An extension to the MAC, including aspects of the DL that are not specified by IEEE but are logically MAC functions.
- An upper-DL protocol that handles link and mesh aspects above the MAC level.

Components of the DPDU header in this standard are described in 9.3.1.

9.1.5 The DL and the IEEE 802.15.4:2011 MAC

This standard uses the IEEE 802.15.4:2011 MAC (called IEEE MAC herein). Only IEEE MAC data frames are used. The formats used are as specified by IEEE 802.15.4:2011, with the two exceptions explicitly enumerated in 9.1.5. See 9.3.3 for detail.

The IEEE MAC describes various features that are not used by this standard's DL (called "the DL" herein). In summary, only IEEE MAC data frames are used by the DL.

A DLE compliant with this standard never associates with a coordinator in the sense defined by the IEEE MAC. None of the IEEE MAC functions involving FFDs are used by this standard.

Within the limited context of this standard's DPDUs, there are some features that are not supported for IEEE MAC data frames. These features are implemented via the MAC extension of this standard, which enhances the IEEE MAC with features that are logically MAC functions but that are not included in IEEE 802.15.4:2011.

The DL and the IEEE 802.15.4 MAC each specify an entity called a superframe (see 9.1.8), but the DL uses no aspects of the IEEE 802.15.4 MAC superframe specification.

ACK/NAK DPDUs are used to convey time information for clock correction, in addition to providing authenticated acknowledgment. These features are not available when using the IEEE 802.15.4:2011 MAC immediate acknowledgment MPDU. For this and other reasons, the MAC-level immediate acknowledgments specified in IEEE 802.15.4:2011 are not used; instead, MAC-level immediate acknowledgments compliant with this standard are provided within this standard as short IEEE 802.15.4:2011 data frames, usually using an address field structure combination not used for such purposes in IEEE 802.15.4:2011.

NOTE ACK/NAK DPDUs are short control signaling (SCS) as specified in ETSI EN 300 328.

The IEEE 802.15.4:2011 MAC includes active and passive scans, which are not used in this standard. This standard has alternative active and passive scans, using IEEE 802.15.4:2011 data frames.

The IEEE 802.15.4:2011 MAC backoff and retry mechanism is not used by the DL. Instead, the DL implements its own retries, involving spatial diversity (retries to multiple DLEs), frequency diversity (retries on multiple radio channels), and time diversity (delaying the Data DPDU). The manner and degree of these elements of diversity are not fixed, but configured by the system manager. More generally, this standard's DL uses CSMA/CA, but the details are different from CSMA/CA use as defined in IEEE 802.15.4:2011. Various aspects of the IEEE 802.15.4:2011 MAC's CSMA/CA behavior are not used, and CSMA/CA functions are handled in this standard's DL.

The standard includes two exceptions to the MAC-PDU addressing combinations specified in IEEE 802.15.4:2011:

- Solicitation Data DPDUs and most ACK/NAK DPDUs, which are technically data frames in IEEE 802.15.4:2011, use a destination-addressing mode of 00 and a source-addressing mode of 00. In IEEE 802.15.4:2011, this combination is limited to IEEE 802.15.4:2011 beacons and trivially-spoofed IEEE 802.15.4:2011 immediate acknowledgments.

- Advertisement Data DPDUs and secondary duocast/N-cast ACK/NAK DPDUs, which are technically data frames in IEEE 802.15.4:2011, use a destination-addressing mode of 00 and a source-addressing mode of 10 (DL16Address). In IEEE 802.15.4:2011, this combination implies that the frame is directed to the PAN coordinator, which does not exist in this standard (so therefore that meaning cannot apply).

9.1.6 Routes and graphs

9.1.6.1 General

Routes are configured by the system manager, based on reports from DLEs that indicate instantaneous and historical quality of wireless connectivity to their immediate neighbors. The system manager accumulates these reports of signal quality to make routing decisions. The signal quality reports are standardized, but the routing decision process within the system manager is not standardized. Once the system manager makes its routing decisions, it uses standard Data DPDUs to configure routes within each DLE in the D-subnet. (See 9.1.13 and 9.1.14 for a review of neighbor discovery.)

DL routing is adaptive at two levels:

- DLEs make instantaneous adaptive forwarding decisions. DLEs are normally configured with path diversity, so that if one link fails somewhere along the route, the DLE can immediately send the Data DPDU along an alternative path.
- If, over time, certain links have consistent connectivity issues, this is reported to the system manager, which can then reconfigure the DLE to use different links.

Within each Data DPDU, DL routing instructions are placed in the Data DPDU's DROUT subheader (see 9.3.3.6). When a Data DPDU is addressed to an immediate neighbor, such as during the D-subnet join process, the route is simply the address of that neighbor. When the Data DPDU is being sent to a more distant DLE, a single graph number can indicate how the Data DPDU's payload should be conveyed through the D-subnet to reach that address. These two approaches may be combined. For example, a DROUT subheader may contain two entries, the first identifying an immediate neighbor for the first hop, and the second indicating a graph that is used for the rest of the route through the D-subnet.

Routes that identify specific D-addresses, also known as source routing, are not as adaptive as routes based on graphs. When a route is based on a series of D-addresses, each DLE along the route becomes a single point of failure. Graphs, on the other hand, should be configured with multiple branches at each hop, so that if there is a connectivity problem with one neighbor, the DLE can send the payload of that Data DPDU to a different neighbor.

Source routing is useful for quick, transitory communications between DLEs, such as during the join process. Source routing may also be used when graph route resources are scarce.

The DROUT subheader is constructed by the DLE that injects the Data DPDU into the DL from a NLE. Routes are selected by table lookup based on contract ID, destination D-address, or by default. Once a route is selected the Data DPDU's conveyed payload follows that route until it arrives at the D-subnet termination point, which may be its ultimate destination, or alternatively may be a waypoint along the route, such as a backbone router or the system manager. At the D-subnet termination point, the received Data DPDU's payload is passed to the collocated NLE.

The DROUT subheader includes a forwarding-limit field which is used to limit the number of times that a Data DPDU can be forwarded within a D-subnet. The forwarding limit is initialized by the originating DLE when the route is assigned, and decremented with each hop until it reaches zero, triggering discard of the Data DPDU if a subsequent hop was required.

NOTE This forwarding limit ensures that Data DPDUs cannot circulate forever via an unintended circular route.

9.1.6.2 Graph routing

A graph is a set of directed links that is used for routing messages within a D-subnet. Each graph designated by the system manager for routing within a D-subnet is identified by a graph ID.

The links associated with each graph are configured by the system manager. A D-subnet may have multiple graphs, some of which may overlap. Each DLE may have multiple graphs going through it, even to the same neighbors.

Figure 54 illustrates an example of graph routing.

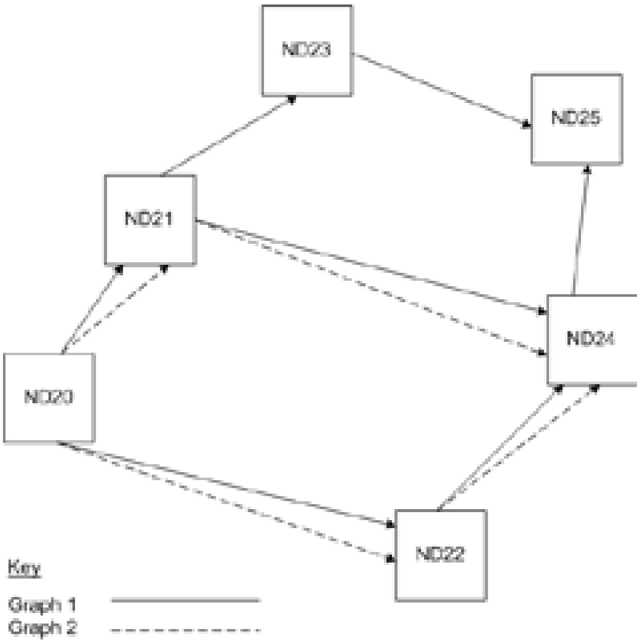


Figure 54 – Graph routing example

In Figure 54, ND20 communicates with ND25 using graph 1. To send a Data DPDU's payload on that graph, ND20 may forward it to ND21 or ND22. From those DLEs, the Data DPDU's payload may take several alternate routes, but either way, following graph 1, the Data DPDU's payload will arrive at ND25. Similarly, to communicate with ND24, ND20 may send Data DPDUs on graph 2 through ND21 or ND22, either of which in turn will forward the Data DPDU's payload to ND24.

Figure 54 shows all graphs originating from ND20, but the same graphs may be used by any node. For example, the system manager may configure ND21 to use graph 1 for its communication with ND25.

NOTE 1 The DL routing information in the Data DPDU's payload often is updated as that payload moves through the D-subnet.

Table 99 and Table 100 reflect the contents of graph tables on ND20 and ND21. These graph tables roughly correspond to data structures within each DLE for the topology shown in Figure 31. For example, a Data DPDU following graph 2 will look up graph ID 2 in each router along the route to find out which neighbors it can use for the next hop.

Table 99 – Graph table on ND20

Graph ID	Neighbor address
1	21, 22
2	21, 22

Table 100 – Graph table on ND21

Graph ID	Neighbor address
1	23, 24
2	24

Each graph within a D-subnet is identified by a graph ID. A Data DPDU usually originates within the D-subnet, at a field device or at a gateway or system manager or backbone router. To send a message on a graph, the originating DLE includes a graph ID in the Data DPDUs DROUT subheader. The Data DPDU travels along the paths corresponding to the graph ID until it reaches its destination or is discarded.

In order to route Data DPDUs over a graph, each DLE along the path needs to maintain a graph table containing entries that include the graph ID and next-neighbor(s)' D-address(es). A DLE routing a Data DPDU performs a lookup based on graph ID and sends the Data DPDU to any one of the applicable neighbors. Once a neighbor acknowledges receipt of the Data DPDU, the DLE releases it from a Data DPDU forwarding buffer.

Diverse graph paths (branches) may be established by configuring more than one neighbor associated with the same graph index. A branch may be configured with a preferred neighbor, indicating that the DLE should attempt to transmit the Data DPDU to the preferred neighbor first, even if there is an earlier-occurring opportunity to transmit to other neighbors. If no preferred neighbor is designated, the DLE should treat all branches equally, transmitting the Data DPDU at the first opportunity that presents itself. If the first transmission does not result in an ACK DPDU, the DLE normally is configured to use alternative branches for retries.

Figure 55 provides examples of routing graphs that are inbound (toward the backbone) and outbound (away from the backbone). The basic organization of inbound and outbound routing graphs may be very similar to each other, but pointing in opposite directions, as shown in Figure 55. The system manager configures routing relationships among DLEs in a D-subnet.

The top half of Figure 55 shows an inbound graph in an example of DL routing configuration. An inbound graph enables a set of DLEs to send Data DPDUs toward the backbone or system manager. A single graph may be used for inbound routing in a small D-subnet, as shown in the top half of Figure 55. It is possible and often desirable for the system manager to define multiple inbound graphs, particularly as the number of DLEs in the D-subnet increases. As shown in the top half of Figure 55, if DLEs have multiple neighboring routers, then diversity is often inherent in the inbound graph.

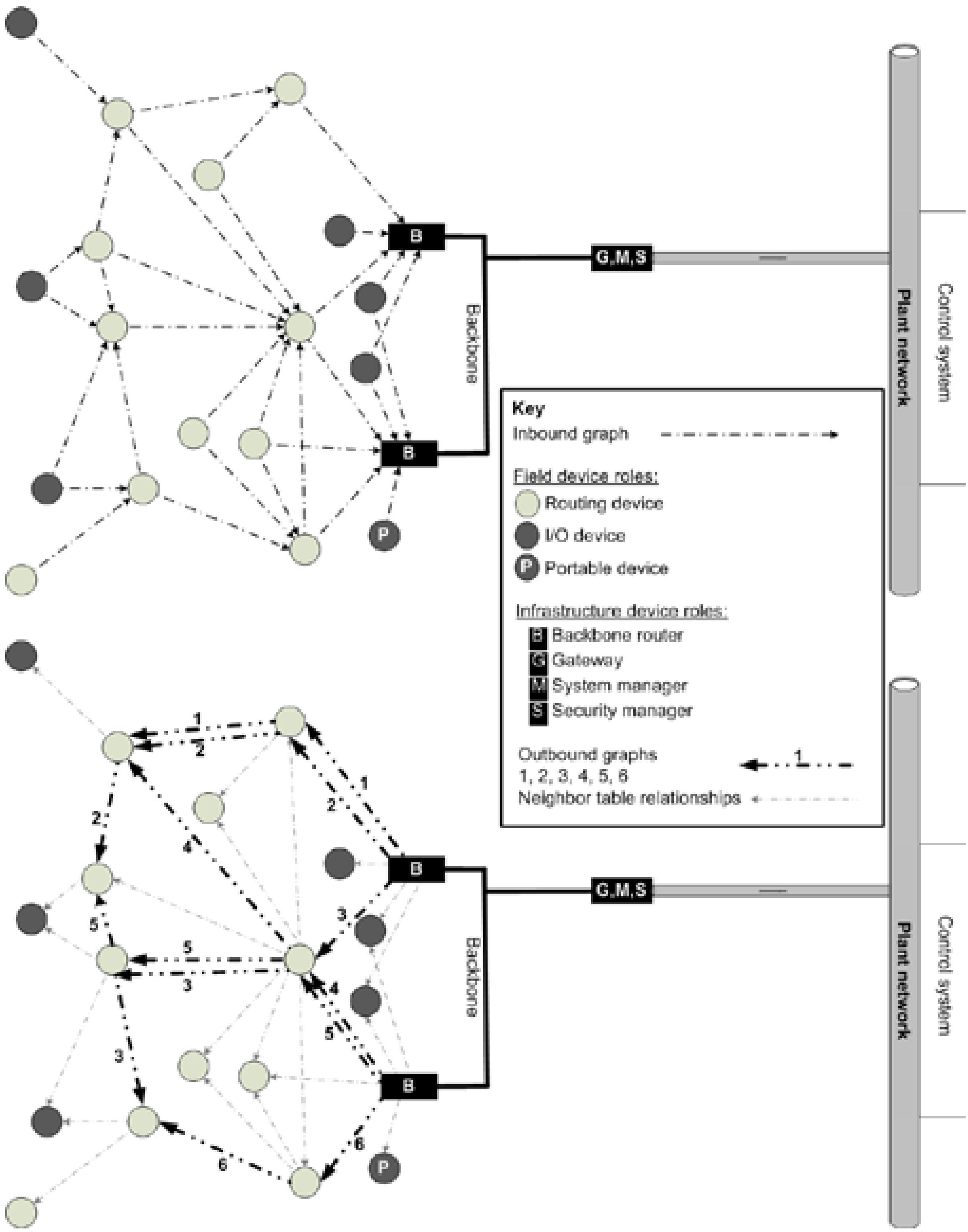


Figure 55 – Inbound and outbound graphs

The illustrative inbound graph in Figure 55 can be problematic for fragmented NPDUs, because fragments arriving at different backbone routers may pose a reassembly challenge. The following considerations apply:

a) Each contract specifies a maximum NPDU size. Most contracts covering communication toward the backbone, including those used for generally higher priority traffic such as publish/subscribe communication of process variables and source/sink communication of alerts (i.e., process alarms and device events), specify maximum APDU sizes that will not trigger NPDU fragmentation, which thus does not occur. Most communications fit in a single Data DPDU, which is determinable for a particular flow in terms of the maximum supported payload size that is agreed when the contract is established.

b) Those contracts that permit NPDUs of a size that can require fragmentation toward the backbone are typically low-priority background transfers of large blocks of information, such as captured waveform upload. A problem occurs when the fragments of a fragmented NPDU are delivered to different BBRs.

   That problem (of uncoordinated fragment delivery) can be avoided in three different ways:

   1) The contract may specify or be tied to a route that terminates in a single BBR, thus avoiding delivery of different fragments of a single NPDU to differing devices.

   2) The contract may specify or be tied to a route that terminates in multiple BBRs from the same vendor, that are known by the system manager to have an inter-BBR fragment reassembly protocol to manage such reassembly via their shared backbone network.

      NOTE 2 It is common practice for plant owner-operators to purchase identical-function infrastructure equipment from only one vendor, so that problems with that class of equipment will be referable directly to the responsible vendor, thus reducing the need of plant personnel to determine which specific vendor's equipment is at fault. Such elimination of inter-vendor “finger pointing” typically leads to faster problem resolution and a quicker return to normal plant productivity. It also makes more usable any vendor-proprietary features or diagnostics that the infrastructure equipment may provide.

   For DLE contracts supporting payload sizes that do not fit in a single DSDU, the selected graphs that are directed toward the backbone often use reduced path diversity to ensure that a set of fragmented NPDUs are all delivered to the same BBR (alternative a) or to a subset of BBRs that are known to jointly provide a shared fragment-reassembly capability (alternative b).

The bottom half of Figure 55 shows a set of outbound graphs in an example of routing configuration. An outbound graph is usually used to send Data DPDUs from backbone DLEs to field DLEs. As shown in the bottom half of Figure 55, multiple graphs may be used for outbound routing, with each outbound graph corresponding to a group of DLEs within radio range of the graph.

In this example, the inbound DL graph routing ends at the backbone, at which point the NL takes over routing responsibilities. The relationship between a WISN D-subnet and a backbone N-subnet is described in 5.5.

Although all of the examples above show inbound and outbound graphs, these are not actually different types of graphs; they are just graphs that happen to point in opposite directions. Peer-to-peer routing is also supported by this standard. The system manager may arrange a graph to follow any route where connectivity exists.

A DSDU is forwarded along a graph until the graph is terminated. If the Data DPDU's destination address matches the DLE's address, then the DSDU has reached its destination and the graph is terminated. Alternatively, if the graph number in the Data DPDU does not have a corresponding entry in the lookup table dlmo.Graph (see 9.4.3.6), the graph has reached its termination point.

9.1.6.3 Graph extensions

The bottom half of Figure 55 shows that outbound graphs do not necessarily extend to all DLEs on the periphery of a D-subnet. Nonetheless, such DLEs are covered by the outbound graphs implicitly, through a graph extension mechanism. A DLE automatically extends graphs by checking the Data DPDU's destination address for a neighbor table entry, thus indicating that the Data DPDU's destination is one hop away. If it is, the router treats the neighbor as if it were listed in the graph, thereby extending the graph for that Data DPDU. More formally, when the Data DPDU's destination address is in a DLE's neighbor table, and the Data DPDU is being routed with a graph, the DLE shall treat that graph as including that neighbor for the purpose of routing that Data DPDU even if the graph does not explicitly refer to the neighbor. All routers shall support this basic form of implicit graph extensions.

An explicit graph extension field in the neighbor table provides an additional degree of control. If a graph is specifically designated in a neighbor table, as described in 9.4.3.4.2, the neighbor is not only treated as being covered by the graph; the neighbor is also given preferential treatment. If the neighbor is designated as the graph's last hop, a Data DPDU following that graph shall be forwarded exclusively to that neighbor. If the neighbor is designated as a preferred branch, the DLE should attempt to forward an applicable Data DPDU to that neighbor before other neighbors.

Support for the explicit graph extension field in the neighbor table is a device construction option; its support status is reported to the system manager through dlmo.DeviceCapability when the DLE joins the D-subnet. All routers support the basic implicit graph extension capability, but it is expected that only some routers will fully support the explicit-last-hop and preferred-branch indicators in the neighbor table.

9.1.6.4 Source routing

Source routing is a general method of routing supported by this standard. In source routing, the originating DLE may be configured to designate a hop-by-hop route for a Data DPDU to follow through a D-subnet. A simple use of source routing is a Data DPDU directed one hop away to a specific neighbor, such as for joining. When a source-routed Data DPDU arrives at an intermediary DLE, the intermediary DLE examines the path information in the Data DPDU to determine the neighbor to which it should forward the Data DPDU.

A source route is a list of entries specifying the route that a Data DPDU shall follow through the D-subnet. The first entry in the list specifies the next hop, and the list is shortened as the Data DPDU moves through the D-subnet. Source routing entries can specify graphs or D-addresses, thus allowing graphs to be chained. 12-bit graph numbers within a source route are encoded in binary as 0x1010 gggg gggg gggg.

The Data DPDU header compresses a source route to a single octet in the common case where a single graph route is specified and the graph number is ≤ 255 (encoded in binary as 0x1010 0000 gggg gggg). This is commonly referred to as graph routing, but formally it is a source route containing a single graph.

In the provisioning or joining process, an EUI64Address is used to address a DLE that has not yet received an IPv6Address. This case is encoded as a route with a graph of zero and a DADDR D-address of zero (see 9.3.3.6 and 9.3.3.7), indicating that the EUI64Address can be found in the Data DPDU's MAC header (MHR).

When a Data DPDU is received by the DLE through its wireless link, the following processing steps shall be followed, in order to determine whether the DL route has terminated and to update the source route in the Data DPDU header:

- The DADDR subheaders destination D-address is checked to see if it matches the D-address of the receiving DLE. If there is a match, the DSDU has reached its final destination and the DSDU shall be passed to the collocated NLE as described in 9.2.4. (The DADDR destination D-address is encoded as zero, in the case where the destination D-address is duplicated in the MHR. See 9.3.3.7. In that case, the match is indirect, based on the MHR destination D-address.)
- The first entry in the source route is deleted if appropriate, thus shortening the source route by shifting the second and subsequent entries (if any) into the prior positions (shift left). The first entry shall be deleted unless it is a graph number of a graph that has not reached its termination point.
- If the route has no remaining entries, the route has terminated and the DSDU shall be passed to the collocated NLE as described in 9.2.4.

If the DSDU is not passed to the collocated NLE, the Data DPDU shall be discarded if the forwarding limit (in the DROUT subheader) is zero. If the forwarding limit is positive, it is decremented and the Data DPDU placed on the DLE's forwarding message queue.

When a DSDU is intended to be routed through the backbone, the DL route should terminate at the backbone router. If a route does not terminate in a collocated backbone router, it is forwarded by the DLE and never processed by the collocated backbone router's NLE, thus allowing peer-to-peer messaging to occur within the D-subnet through the auspices of a backbone router.

These routing methods are examples of different ways to configure the DL routing capability that is resident in all field routers compliant with this standard. The system manager shall configure all graphs within a D-subnet. The ability to configure routing in any of several ways such as graph routing and/or source routing enables device interconnectability.

9.1.6.5 Route selection

The route through the D-subnet for a Data DPDU is selected when the message enters the DL (see 9.2.2). The route is stored in the DROUT subheader for use by other DLEs that will route the Data DPDU. The initial selection of a route is based on decision rules in the initiating DLE. The following list shows the route selection criteria in order, whereby the route shall be selected based on the first condition that applies:

- The Data DPDU has a destination EUI64Address. A destination EUI64Address is used only during the join process, when a router is sending a response to an immediate neighbor that has not yet received a DL16Address from the system manager. In that case, the EUI64Address from the IEEE MAC is used, with a Graph ID of 0 in the D-route.
- The ContractID is associated with a particular D-route. This may be used when a particular graph or source D-route is intended to provide a defined level of service.
- The destination DL16Address within the D-subnet is associated with a particular route.
- The destination DL16Address within the D-subnet is an immediate neighbor, such as during the join process.
- Otherwise, use the default route. Normally, the default will direct the message to the nearest backbone router, or to the system manager if there is no backbone router.

A single route may be designated as the default by the system manager, by designating a particular D-route as a default. The default D-route is usually configured to route messages to the system manager, or to a backbone router if there is no system manager on the D-subnet. A default D-route may sensibly be configured in conjunction with the establishment of a DLE's contract with the system manager. Additional routes may be configured as needed, such as to provide enhanced quality of service or to route messages to a peer DLE on the D-subnet.

9.1.7 Slotted-channel-hopping, slow-channel-hopping, and timeslots

9.1.7.1 General

Three general operational alternatives are supported by the DL:

– slotted-channel-hopping;

– slow-channel-hopping; and

– hybrid combinations of slotted-channel-hopping and slow-channel-hopping.

These three operational alternatives are different ways for a system manager to configure a slotted-channel-hopping capability that is supported by every DLE in a D-subnet. Channel-hopping schedules are configured by the system manager through advertisement Data DPDUs and the dlmo.Superframe attribute.

Slotted-channel-hopping and slow-channel-hopping provide different ways to configure a series of timeslots. The system manager determines the mode of operation and assigns the use of superframes, which are cyclic collections of timeslots. (See 9.1.8 for further discussion of superframes.) This provides flexibility and interoperability of the relevant communication functions (i.e., interconnectability) without requiring excessive complexity within the devices.

From the perspective of a field device, the DLE can be visualized as a player piano with several keys. Each key corresponds to a D-transaction, which specifies a specific channel and timeslot. One key is used to send a pending Data DPDU from the outbound queue, another key is used to listen for an incoming Data DPDU, and so forth. The system manager provides a piano roll for the DLE to play over and over again. The playing style may be slow-channel-hopping, slotted-channel-hopping, or a hybrid of the two. The DLE does not differentiate; it simply mechanically plays each key at a specified time on a specified channel, based on the instructions on the piano roll.

The note details within a timeslot are configurable, using timeslot templates provided by the system manager. There is a constrained series of operations that can be performed within a timeslot – transmit, listen, wait, timeout, and acknowledge – but those simple building blocks may be assembled with different timings. These definitions are flexible, under the control of the system manager.

The duration of timeslots in a D-subnet is set to a specific value by the system manager when a DLE joins the D-subnet. A timeslot duration of 10 ms to 12 ms is expected to be typical. Timeslot duration is configurable to enable:

- optimized coexistence with other systems, such as other D-subnets conforming to this standard, to IEC 62591, and to IEC 62601;
- longer timeslots to accommodate extended message wait times;
- shorter timeslots to take full advantage of optimized implementations;
- longer timeslots to accommodate serial ACK/NAK DPDUs from multiple devices (e.g., duocast, N-cast);
- longer timeslots to accommodate long-duration CSMA/CA at the start of a timeslot (e.g., for prioritized access to shared timeslots);
- longer timeslots to accommodate slow-hopping periods of extended duration;
- imeslots to be synchronized with other non-standard-compliant D-subnets to facilitate inter-routing.

Figure 56 illustrates a slotted-channel-hopping operation.

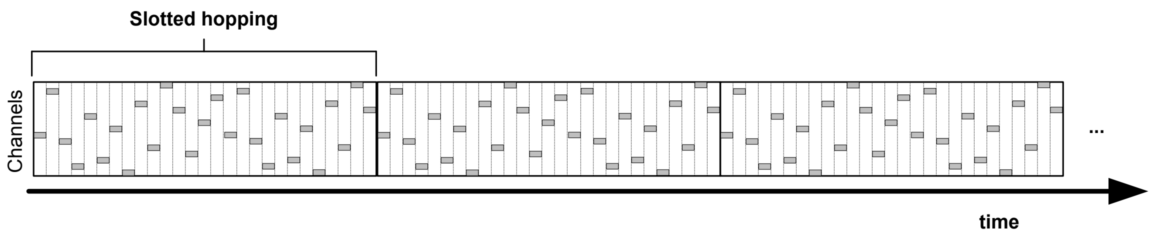


Figure 56 – Slotted-channel-hopping

In slotted-channel-hopping, channel-hopping timeslots of equal duration are used. Each timeslot uses a different radio channel in a channel-hopping pattern. In slotted-channel-hopping, each timeslot is intended to accommodate a single D-transaction consisting of one Data DPDU and its ACK/NAK DPDU acknowledgment(s).

Figure 57 illustrates a slow-channel-hopping operation.

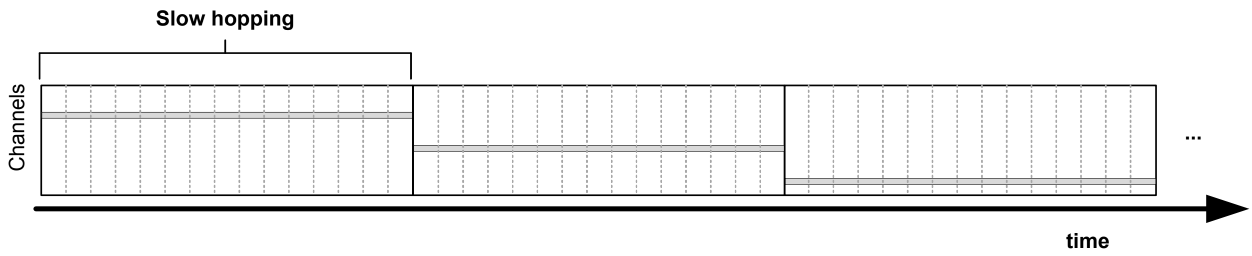


Figure 57 – Slow-channel-hopping

In slow-channel-hopping, a collection of contiguous timeslots is grouped on a single real radio channel. Each collection of timeslots is treated as a single slow-channel-hopping period; however, as shown in Figure 57, timeslots still underlie slow-channel-hopping. Slow-channel-hopping periods are configurable, usually on a scale of about 100 ms to 400 ms per hop. Longer slow-channel-hopping periods, potentially multiple seconds in duration, may be used to support DLEs with imprecise timekeeping and/or DLEs that have temporarily lost contact with the D-subnet. To enable DLE interworkability, all DLEs compliant with this standard shall support configuration of timeslot duration, as designated by the system manager.

In some regulatory domains, slow-channel-hopping of an IEEE 802.15.4:2011 2,4 GHz radio is not permitted to exceed 400 ms per hop. However, in other regulatory domains there is no such constraint. The slow-channel-hopping period is set by the system manager, not by this standard, but its use is constrained in each DLE by dlmo.CountryCode (9.1.15.6). Thus this regulatory constraint is explicitly enforced where it exists, similar to other regulatory constraints.

The structure, duration, and assignment of timeslots in slow-channel-hopping periods are configured by the system manager, which determines the mode of operation and assigns the use of timeslots. This provides flexibility without requiring excessive complexity within the DLEs, facilitating DLE interworkability.

Figure 58 illustrates a hybrid mode of operation.

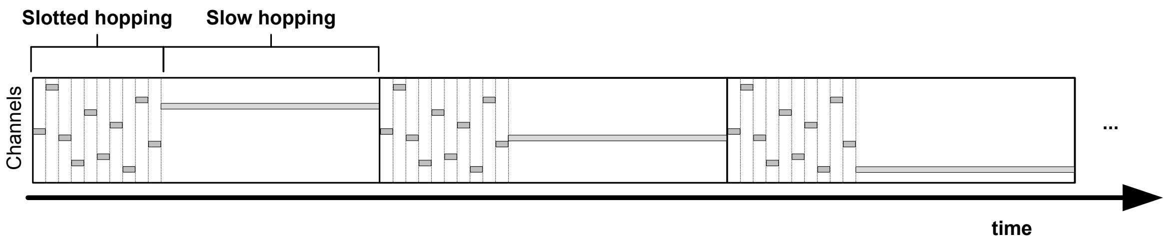


Figure 58 – Hybrid operation

Hybrid operation uses slotted-channel-hopping and slow-channel-hopping periods in a configured combination. For example, in Figure 58, a number of timeslots using slotted-channel-hopping is followed by a period of slow-channel-hopping.

9.1.7.2 Channel-hopping

9.1.7.2.1 General

DL communications are intended to be distributed across multiple radio channels. The system uses defined channel-hopping patterns which provide a specific sequence of channels for communication among collections of devices. Channel-hopping begins at a designated offset in the channel-hopping pattern, continuing through the pattern sequentially until the end, then repeating the pattern indefinitely.

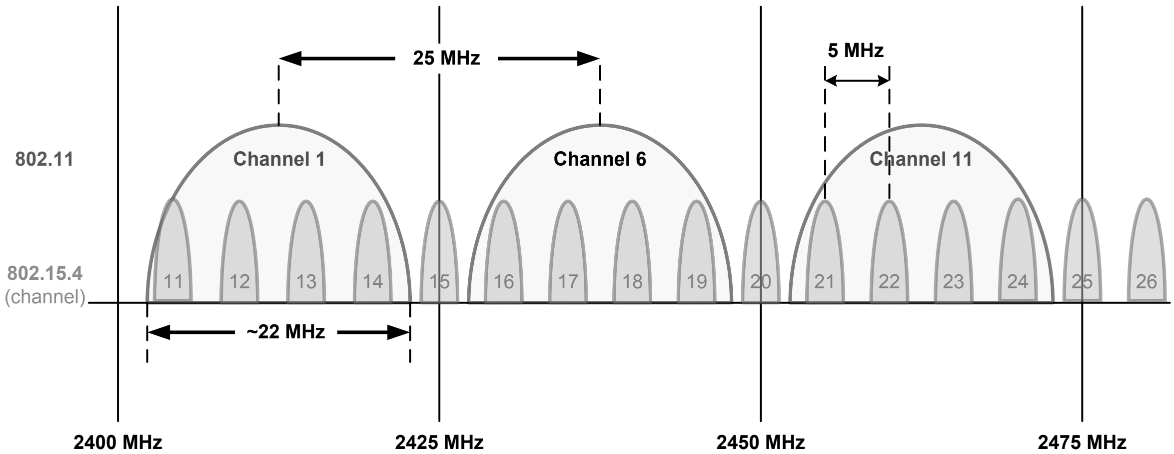
IEEE 802.15.4:2011 DSSS channels 11..26 are mapped to nominal channel numbers 0..15 in this standard. This overview refers to them by their IEEE 802.15.4 nomenclature, as channels 11..26.

This standard is based on devices that use one channel at a time. Multiple radios that are co-packaged are able to operate multiple instances of the PhLE and DLE simultaneously on different channels. The details of such multi-channel operation are not specified by this standard, but such operation is intentionally supported in the D-nonce.

9.1.7.2.2 Radio spectrum considerations

For radio communication, this standard uses IEEE 802.15.4:2011 DSSS channels in the 2,4 GHz band. The IEEE 802.15.4:2011 physical layer (2,4 GHz, DSSS) includes sixteen channels, numbered 11 through 26.

Figure 59 illustrates the sixteen IEEE 802.15.4:2011 DSSS channels along with three overlapping, commonly used IEEE 802.11 channels.



NOTE Channel numbers shown are those of IEEE 802.11 and IEEE 802.15.4:2011, rather than those of this standard.

Figure 59 – Radio spectrum usage

In Figure 59, the narrow channels 11..26 are IEEE 802.15.4:2011 2,4 GHz DSSS channels; these channels are substantially non-overlapping. Also shown are the wider IEEE 802.11 channels 1, 6, and 11; these three channels are common choices for IEEE 802.11 communication and each overlaps a number of IEEE 802.15.4:2011 2,4 GHz DSSS channels.

As Figure 59 shows, IEEE 802.15.4:2011 2,4 GHz DSSS channels 15, 20, and 25 do not substantially overlap any of the three common IEEE 802.11 channels. Therefore, IEEE 802.15.4:2011 2,4 GHz DSS channels 15, 20, and 25 may reasonably be designated as slow-channel-hopping channels. These slow-channel-hopping channels may be configured for purposes such as neighbor discovery. For example, information about the D-subnet may be advertised by field routers on pre-designated slow-channel-hopping channels, so that any DLE seeking to join the D-subnet can limit its active and passive scans to these channels when discovering nearby field routers.

This illustration demonstrates how spectrum management techniques supported by this standard may be used to account for a commonly encountered scenario. Alternative configurations of WiFi, other users of the radio spectrum, and local regulatory restrictions may justify alternative configurations in actual installations.

Most DL communications are intended to be distributed across all available IEEE 802.15.4:2011 DSSS channels (11..26). This standard defines a number of predefined channel-hopping patterns, each providing a specific sequence of channels to use. Other channel-hopping patterns are configurable by the system manager. The channel-hopping patterns that are predefined in this standard are selected to have certain properties intended to minimize the occurrence of unmanaged and repeated collisions with co-located wireless devices, particularly IEEE 802.11 devices.

9.1.7.2.3 Channel 26 and other blocked channels

Support for IEEE 802.15.4:2011 2,4 GHz DSSS channel 26 is optional in this standard, because some implementations may encounter regulatory restrictions at the upper band edge. In addition, a DLE may be blocked from using other channels due to regulatory restrictions, but not for other reasons. After restriction or forced-enabling for the regulatory domain specified by dlmo.CountryCode (9.1.15.6), a list of channels that the DLE supports is reported to the system manager during the process of the DLE joining the D-subnet. This is done through the attribute dlmo.DeviceCapability.ChannelMap. A DLE may be configured with links that use such unsupported channels; in that case the DLE shall treat those links as unselectable.

Since support for IEEE 802.15.4:2011 2,4 GHz DSSS channel 26 is optional in the standard, a system manager may sensibly limit D-subnet operation to IEEE 802.15.4:2011 2,4 GHz DSSS channels 11..25. The channel-hopping patterns predefined by this standard include IEEE 802.15.4:2011 2,4 GHz DSSS channel 26, but these predefined channel-hopping sequences are designed so that they can be shortened by excluding IEEE 802.15.4:2011 2,4 GHz DSSS channel 26 from the channel map in each superframe.

9.1.7.2.4 Spectrum management and selective channel utilization

Multiple methods are available for limiting use of busy or undesirable radio channels, including clear channel assessment (CCA), spectrum management, and selective channel utilization.

Timeslots are normally configured to check for a clear channel before transmitting, using the different modes of the CCA mechanism defined in IEEE 802.15.4:2011. CCA causes a DLE that is about to initiate transmission to relinquish a timeslot if use of the channel by another DLE is detected prior to transmission. See 4.6.11, 9.1.9.4.3 and 9.1.9.4.8.

Spectrum management is a form of selective channel utilization. Spectrum management limits the DL configuration to a subset of channels. Limiting slow-channel-hopping to IEEE 802.15.4:2011 2,4 GHz DSSS channels 15, 20 and 25 is an example of spectrum management. Another example is when a system manager blocks (blacklists) certain radio channels that are not working well or are prohibited by regulation or local policy, or whitelists channels that are mandated by regulation or local policy. Spectrum management is handled by the system manager, through the way that it configures a DLE and the associated PhLE. See 9.1.8.4.7.

Additionally, a DLE may autonomously treat transmit links on problematic channels as idle, thus reducing unnecessary interference and wasted energy on channels with a history of poor connectivity. A DLE skipping links in this manner should periodically test the links to verify that they remain problematic. Such selective channel utilization can be disabled by the system manager on a link-by-link basis, through the attribute dlmo.Link[ ].Type.SelectiveAllowed. See 9.4.3.7.2, Table 182.

9.1.7.2.5 Repeating channel-hopping-patterns

This standard supports five predefined IEEE 802.15.4:2011 2,4 GHz DSSS repeating channel-hopping patterns, which shall be supported in every DLE:

- pattern1: 19, 12, 20, 24, 16, 23, 18, 25, 14, 21, 11, 15, 22, 17, 13 (, 26);

- pattern2: pattern1 in reverse order;
- pattern3: 15, 20, 25 (intended for slow-channel-hopping channels);
- pattern4: 25, 20, 15 (pattern3 in reverse order);
- pattern5: 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21. 22, 23, 24, 25 (, 26).

NOTE 1 IEEE 802.15.4:2011 2,4 GHz DSSS channel 26 is shown in parentheses, as it is supported by this standard but is not necessarily used due to commonly encountered regulatory constraints at the band edge. Figure 60, through Figure 66 and Figure 70 through Figure 74 mostly include channel 26, even though its use is commonly masked out by the superframe that uses the hop sequence.

NOTE 2 In this standard channels are numbered 0..15, as described in 9.4.3.2. However, for tutorial purposes and to ease comparison with IEEE 802.11 (WiFi) and other uses of the same frequency band, channel-hopping patterns are expressed as their IEEE 802.15.4:2011 DSSS channel numbers.

NOTE 3 Pattern5, which is based on IEC 62591, is intended to facilitate coexistence with that IEC standard.

The system manager can configure a DLE to use any of these channel-hopping patterns for slotted-channel-hopping, slow-channel-hopping, or hybrid channel-hopping.

Any channel or set of channels in a channel-hopping pattern may be disabled (masked out) by configuration of the superframe that uses the channel-hopping pattern, with the effect of shortening the channel-hopping pattern for spectrum management.

The predefined channel-hopping patterns of this standard are designed to have certain properties, whether channel 26 is included in the pattern or not. Specifically, successive channels in most of the predefined channel-hopping-patterns are separated by at least 15 MHz, the same bandwidth as three IEEE 802.15.4:2011, 2.4 GHz DSSS channels. This property mitigates the effects of interference and multipath fading in industrial environments.

As shown in the example in Figure 60, predefined channel-hopping-pattern1 is arranged so that consecutive hops do not overlap the same IEEE 802.11 channel.

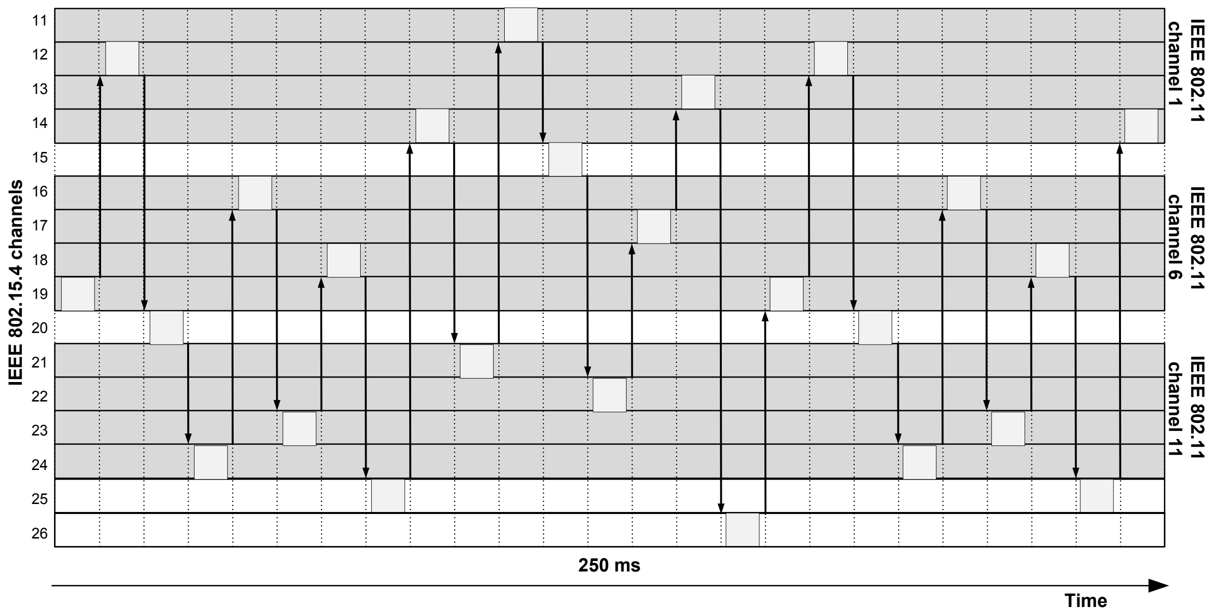
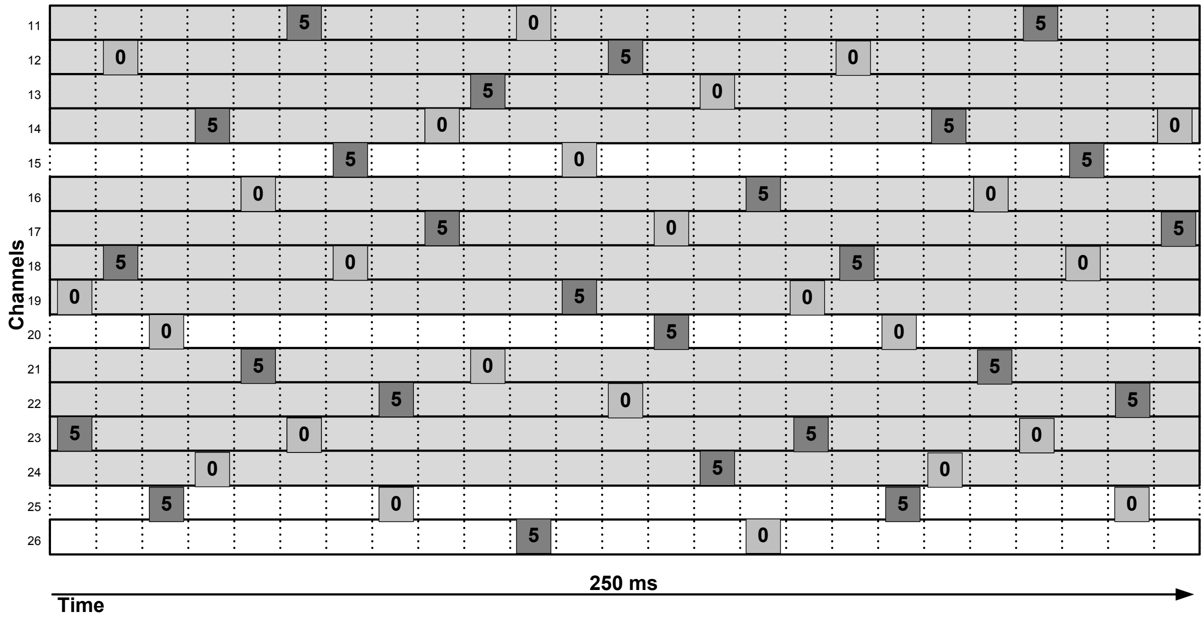


Figure 60 – Predefined channel-hopping-pattern1

At least three channels separate each consecutive hop in pattern1, resulting in frequency shifts of at least 20 MHz. When retries occur in consecutive hops, they will not encounter or cause interference in the same IEEE 802.11 channel.

For different groups of DLEs, it is desirable for DLEs to hop on non-interfering patterns.

Each channel-hopping pattern is combined with a hopping pattern offset. If the hopping pattern offset is zero, then the specified baseline channel-hopping pattern is used. If the hopping-pattern offset is 5, then an offset of 5 is used when indexing the baseline channel-hopping pattern. Figure 61 shows how two groups of DLEs with different hopping-pattern offsets into channel-hopping pattern1 may be used together without competing for the same radio channel at the same time.



NOTE Channel numbers shown are those of IEEE 802.15.4:2011, rather than those of this standard.

Figure 61 – Two groups of DLEs with different channel-hopping-pattern-offsets

A superframe’s channel-hopping-offset is determined indirectly from the dlmo.Superframe[ ].ChBirth attribute, called simply ChBirth, which gives the starting reference of the channel-hopping pattern at TAI time zero. This provides a baseline channel-hopping sequence. Offsets from a baseline channel-hopping sequence, given in the dlmo.Link[ ].ChOffset attribute, called simply ChOffset, are as described here and shown in Figure 61. While the same result is achievable by using ChBirth or ChOffset, it should be noted that the two attributes are essentially reversed. Details of channel-hopping-pattern-offset calculations are found in 9.4.3.5.3.

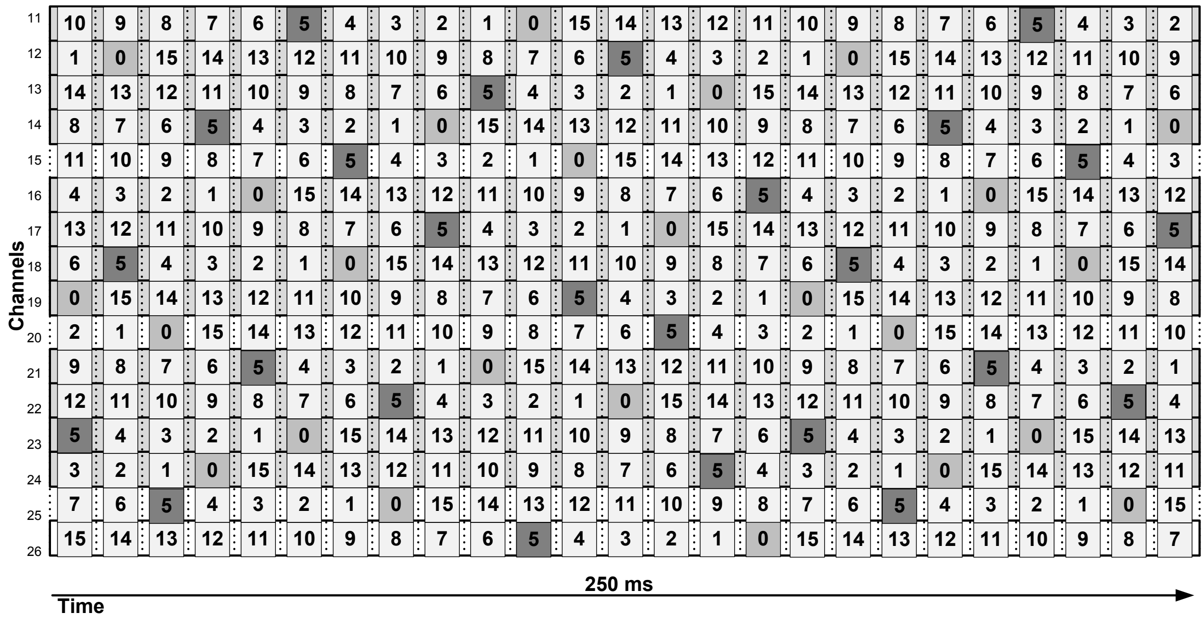
In Figure 61, boxes numbered 0 represent a group of DLEs using predefined channel-hopping-pattern1; its repeating channel-hopping-pattern (using the nomenclature of IEEE 802.15.4:2011) is:

19, 12, 20, 24, 16, 23, 18, 25, 14, 21, 11, 15, 22, 17, 13, 26

Boxes numbered 5 in Figure 61 represent another group of DLEs using channel-hopping-pattern1 with a hopping-pattern offset of 5. A channel-hopping-pattern-offset of 5 has the effect of essentially rotating the channel-hopping sequence to the left by 5, resulting in a repeating channel-hopping sequence (using the nomenclature of IEEE 802.15.4:2011) of:

23, 18, 25, 14, 21, 11, 15, 22, 17, 13, 26, 19, 12, 20, 24, 16

Figure 62 extends this principle, illustrating how different channel-hopping-pattern-offsets may be used for a larger number of DLEs.



NOTE Channel numbers shown are those of IEEE 802.15.4:2011, rather than those of this standard.

Figure 62 – Interleaved channel-hopping-pattern1 with sixteen different channel-hopping-pattern-offsets

In Figure 62, sixteen channels are available. Therefore, up to sixteen DLEs may use channel-hopping-pattern1, each with a different channel-hopping-pattern-offset of 0..15.

As illustrated in Figure 62, a given channel-hopping-pattern may be used concurrently by assigning different channel-hopping-pattern-offsets to various DLEs, superframes (see 9.1.8), or groups of DLEs. As a simple example, each of two DLE clusters may use the same channel-hopping-pattern with different channel-hopping-pattern-offsets, so that the two clusters can share the same radio spectrum without mutual interference. The clusters may be in the same D-subnet or in different D-subnets; as long as they accurately share a consistent timeslot duration and synchronized sense of time, their channel-hopping patterns can be interleaved as shown in Figure 62.

The five predefined channel-hopping-patterns shall exist in every DLE compliant with this standard. This enables routers to advertise concisely the channel-hopping-pattern that is being used and the current channel-hopping-pattern-offset, when it is one of these patterns. This standard also supports customized channel-hopping-patterns in every DLE, in addition to the five predefined patterns, so that the system manager can configure additional channel-hopping-patterns for use by already-joined DLEs.

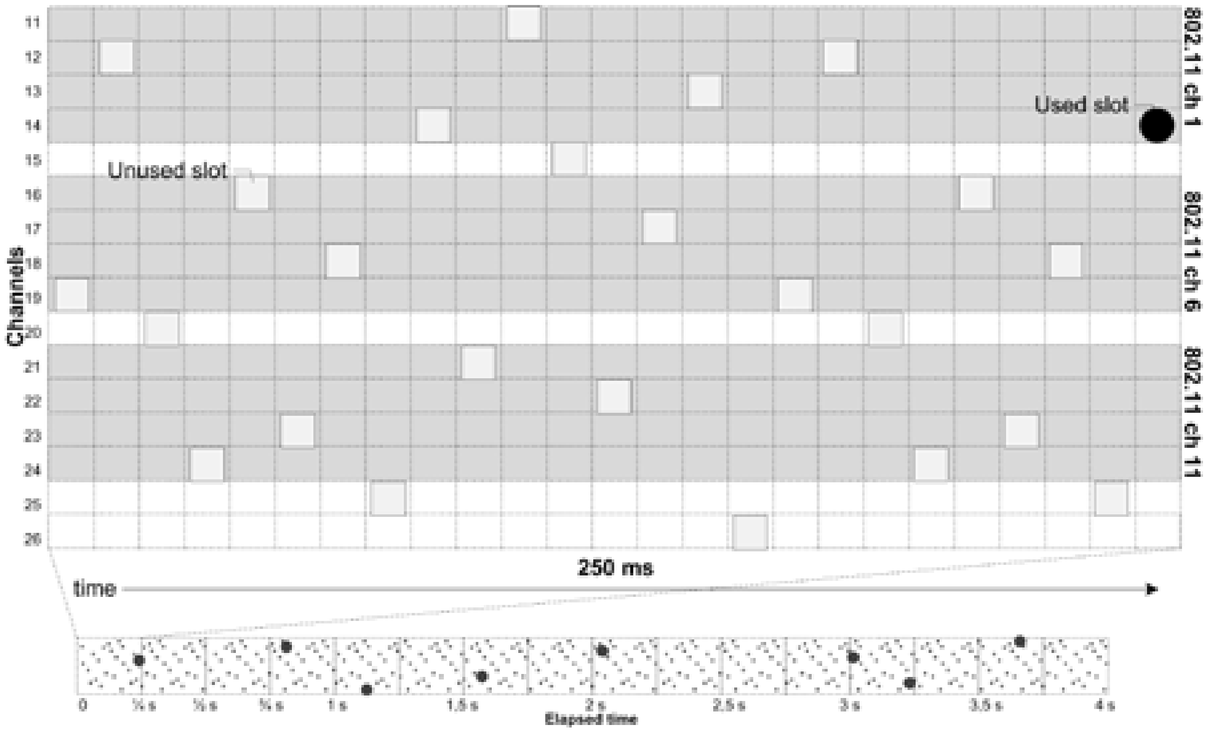
Each predefined channel-hopping-pattern is the same size as the number of channels being used. Thus, for example, channel-hopping-pattern1 uses 16 channels and is 16 hops long. This property allows the channel-hopping-pattern to be interleaved at different offsets as shown in Figure 62.

NOTE 4 Since some DLEs might not support channel 26, which is optional, systems often limit operation to 15 channels (11..25) with essentially the same result.

9.1.7.2.6 Timeslot and channel use

A system shall use slotted-channel-hopping, slow-channel-hopping, or a hybrid combination of the two.

Figure 63 illustrates the use of slotted-channel-hopping. Each timeslot is used with the next successive channel in the channel-hopping pattern.

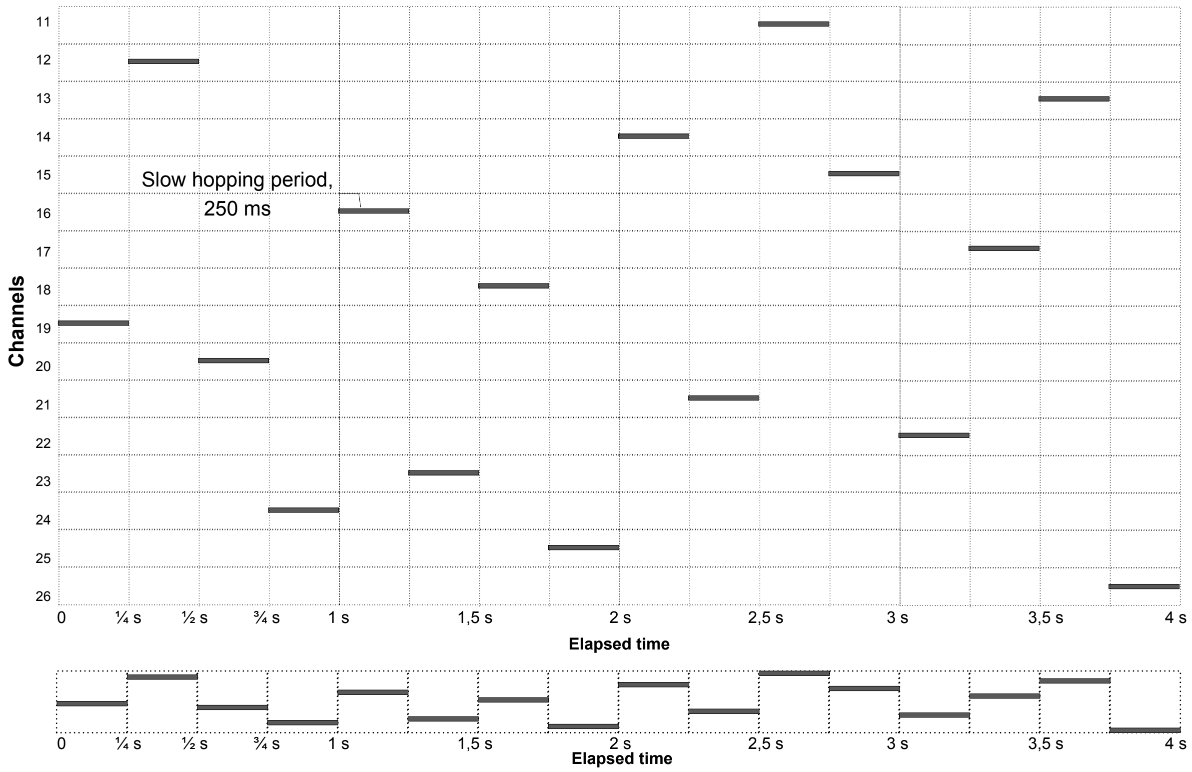


NOTE Channel numbers shown are those of IEEE 802.11 and IEEE 802.15.4:2011, rather than those of this standard.

Figure 63 – Example timeslot allocation for slotted-channel-hopping

The bottom portion of Figure 63 illustrates that the channel-hopping-pattern can be used repeatedly as time progresses. Superframe size and timeslot usage by the DLE is not necessarily tied to lower cyclical DL constructs such as channel-hopping-patterns or 250 ms timeslot alignment intervals. (See 9.1.9.1.3 for a discussion of alignment intervals.)

Figure 64 illustrates the use of slow-channel-hopping. Each channel is used over multiple timeslots.

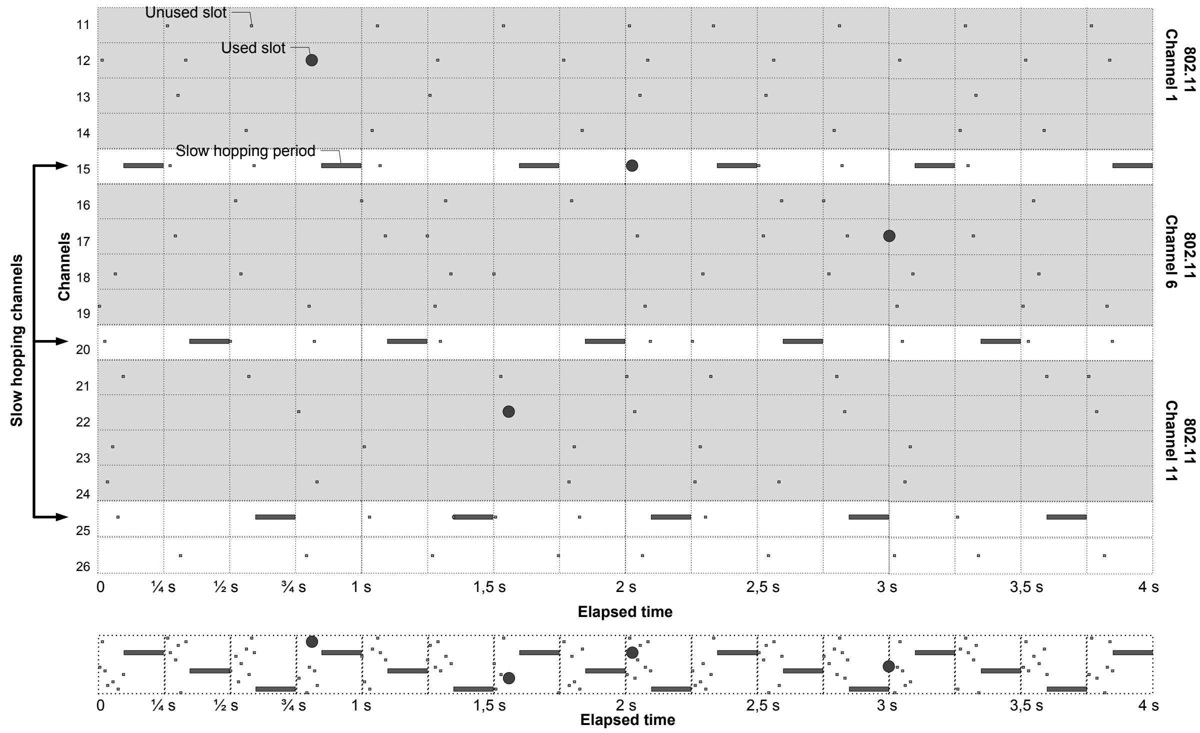


NOTE Channel numbers shown are those of IEEE 802.15.4:2011, rather than those of this standard.

Figure 64 – Example timeslot allocation for slow-channel-hopping

Slow-channel-hopping periods can span a 250 ms timeslot alignment interval. (See 9.1.9.1.3 for a discussion of timeslot alignment intervals.) In such cases, slow-channel-hopping-periods are not interrupted by idle periods, that is, if a slow-channel-hopping-period traverses the edge of a timeslot alignment interval, the radio does not turn off during the otherwise-required idle period.

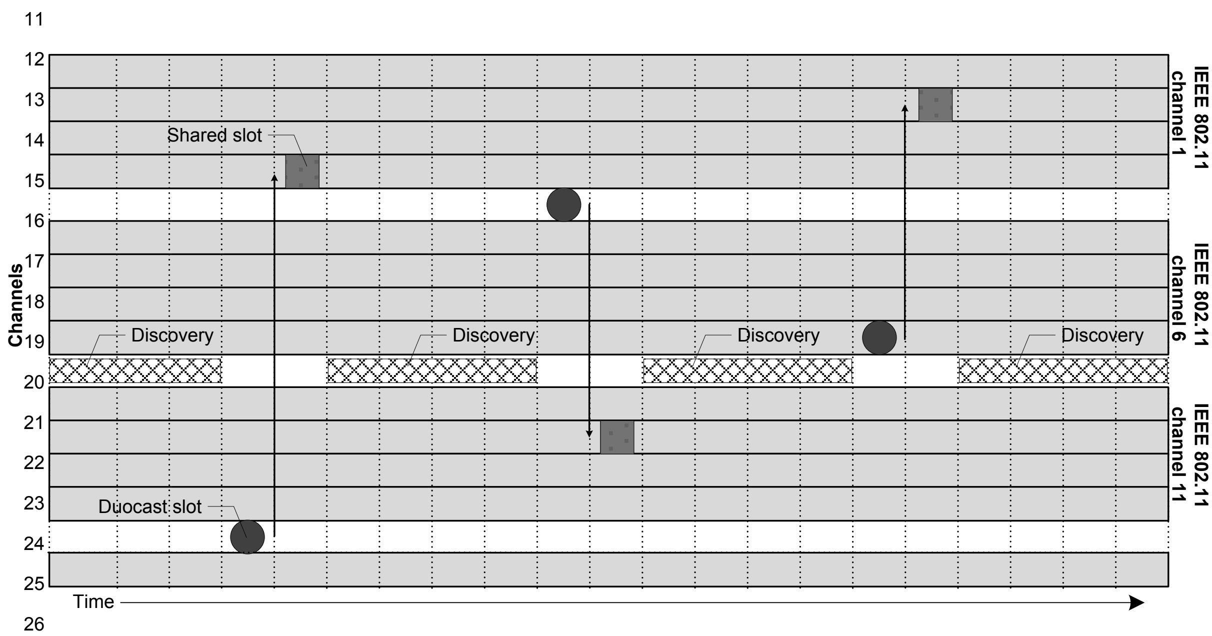
Figure 65 illustrates a hybrid system that combines slotted-channel-hopping and slow-channel-hopping. In this example, within each 250 ms alignment interval, a number of timeslots, each assigned to a different channel, are followed by a slow-channel-hopping-period on a single channel.



NOTE Channel numbers shown are those of IEEE 802.11 and IEEE 802.15.4:2011, rather than those of this standard.

Figure 65 – Hybrid mode with slotted-channel-hopping and slow-channel-hopping

The order in which slotted-channel-hopping and slow-channel-hopping can be combined is flexible; slow-channel-hopping periods need not follow slotted-channel-hopping timeslots. Rather, the two may be used in any sensible combination. For example, Figure 66 shows an example configuration, where a DLE switches between slow-channel-hopping and slotted-channel-hopping.



NOTE Channel numbers shown are those of IEEE 802.11 and IEEE 802.15.4:2011, rather than those of this standard.

Figure 66 – Combining slow-channel-hopping and slotted-channel-hopping

In the example of Figure 66, slotted-channel-hopping is used when broadcast/multicast, duocast/n -cast, or contention-based communication timeslots are allocated explicitly. When a DLE does not have a timeslot allocation, it listens on channel 20, which facilitates neighbor discovery. (See 9.1.9.4.7 for a discussion of duocast/N-cast.)

9.1.8 Superframes

9.1.8.1 General

A superframe is a repeating sequence of timeslots. The number of timeslots in each superframe cycle (its size) and the duration of each of those timeslots determines the period of the superframe cycle. This establishes the structure of the communication schedule for DLEs that use the superframe. For example, a superframe that cycles every 500 ms will allow each DLE that uses a single timeslot within the superframe to communicate every 500 ms.

When a superframe is created, it is given a superframe ID. Figure 67 shows how DLEs may communicate in an example of a three-timeslot superframe.

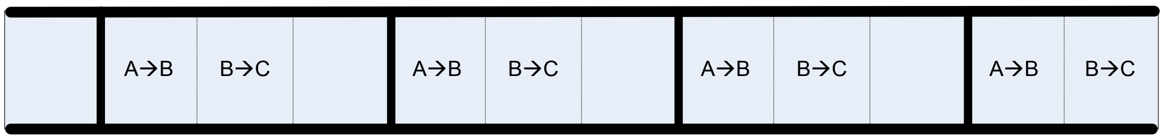


Figure 67 – Example of a three-timeslot superframe and how it repeats

In Figure 67, DLE A may communicate with DLE B during the first timeslot of each superframe cycle. DLE B may communicate with DLE C during the second timeslot of each cycle. The third timeslot of each cycle is unassigned (idle). The cycle repeats every three timeslots.

Figure 67 shows timing cycles and communication links within the same structure, which is a conceptual view. Superframe cycles and communication links are represented as separate but related configurable objects within the DLE. Figure 68 illustrates this data structure, clarifying the distinction between superframes and links, for the same three-timeslot superframe as in Figure 67.

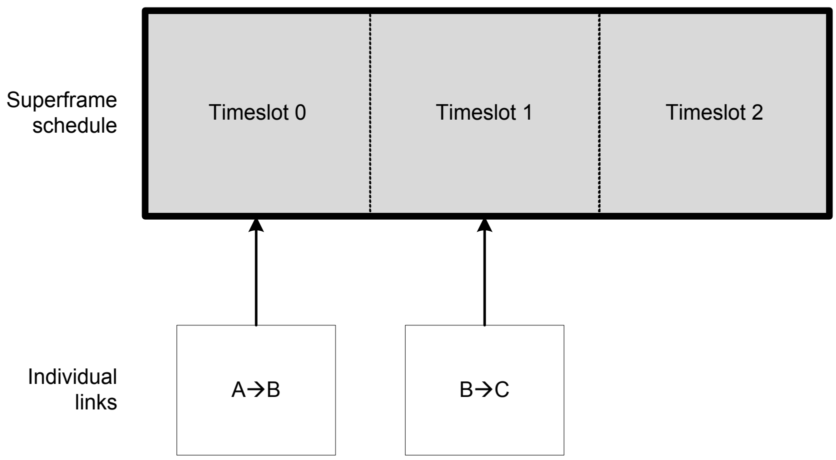


Figure 68 – Superframes and links

As shown in Figure 68, superframes refer to a collection of timeslots. Links refer to the use of superframe timeslots for communication between a specific pair of DLEs. A timeslot is a period of time. A superframe is a cyclic schedule of timeslots and associated channels. A link describes a specific activity that repeats within a superframe cyclic schedule.

The system manager configures matching sets of links among a collection of DLEs that communicate with each other. For example, a link on DLE A may be configured to transmit Data DPDUs to DLE B on a particular superframe cycle. On the same cycle, DLE B should have a link that is configured to listen for an incoming Data DPDU. These two links are matched in the sense that the system manager has configured these two related operations to occur concurrently on the same channel.

Several performance parameters are determined by superframe period and how links are assigned to superframes. In general, shorter-period superframes result in lower Data DPDU latency and increased digital bandwidth, at the expense of increased energy consumption and more concentrated allocation of digital bandwidth. Longer-period superframes generally result in higher latency and lower digital bandwidth, but with reduced energy consumption and less concentrated allocation of digital bandwidth. These tradeoffs should be carefully considered when determining superframe period and link density within a superframe.

A given DLE may be configured to use concurrently several superframes of different sizes. A link with one timeslot within a superframe of length L slots repeats twice as frequently as a similar one-timeslot link within a superframe of length $2 \times L$ slots, thus allowing for twice the throughput per second.

A DLE may use more than one superframe simultaneously. Also, not all DLEs in a D-subnet need to participate in each superframe. By configuring a DLE to participate in multiple concurrent superframes of different lengths, it is possible to establish multiple communication schedules with incommensurate periods that all operate simultaneously.

Superframes are numbered for identification, but these superframe numbers are limited in scope to the DLE where the superframe is used. Since the scope of a superframe number is a single DLE, a neighboring DLE can use the same superframe number for a completely different purpose. Superframes can be added, removed, activated, and deactivated while the D-subnet is running.

Figure 69 shows how timeslots in different superframes are aligned, even though the superframes may cycle at independent rates.

Figure 69 – Multiple superframes with aligned timeslots

NOTE Timeslot alignment is a result of defining all slots to have identical durations and of realigning timeslots to TAI time every 250 ms (see 9.1.9.1.3). Superframes with timeslots of different durations are usable simultaneously within a D-subnet. However, the designers of this standard considered only configurations wherein a single timeslot duration is used during operation of a given D-subnet (changeable at D-subnet reinitialization).

A DLE with multiple links in a timeslot may encounter link collisions when two or more links coincide. To address such situations, each link is assigned a priority. A higher-numbered link priority means the link takes precedence over a link with a lower-numbered priority. In the event of a link collision, the link with the higher-numbered priority is used. If two links have the same priority, a superframe priority is used. See 9.1.8.5.

In addition to link priority, each Data DPDU is assigned a priority by its originating DLE. Once a link is selected based on link priority, the priority of the Data DPDU is used to weight the relative importance of all queued Data DPDUs that can use the link.

9.1.8.2 Exponential backoff

Links may be shared or dedicated. On the receiver side, there is no structural difference between shared versus dedicated links. On the transaction initiator side, an exponential-backoff bit in the link configuration specification indicates whether the link is shared or dedicated. If a link is shared, as indicated by the link's exponential-backoff bit being set (to 1), the transaction initiator shall use exponential backoff for retries using that link. If a link is dedicated, as indicated by the link's exponential-backoff bit being reset (to 0), the transaction initiator shall not use exponential backoff in that link.

It is possible, and sometimes reasonable, for a system manager to configure multiple DLEs to transmit at the same time and on the same radio channel, without setting an exponential-backoff bit in the applicable links. The term "dedicated link" is used merely to indicate that exponential backoff is not applied to such links.

Exponential backoff shall be applied when, and only when, a DLE transmits a unicast Data DPDU on a shared link and does not receive an error-free ACK/NAK DPDU, which implies a possible collision. A unicast transmission that is aborted due to CCA sensing shall be treated as equivalent to an unsuccessful transmission in the context of exponential backoff. Exponential backoff is intended to resolve such collisions. Exponential backoff shall operate on a per-neighbor basis, and applied to all Data DPDUs in the message queue addressed to that neighbor, regardless of Data DPDU priority.

For each neighbor, the DLE maintains a backoff exponent and a backoff counter, called BackoffExponent[Neighbor] and BackoffCounter[Neighbor] herein. BackoffExponent[ ] and BackoffCounter[ ] are inaccessible implementation internals, and therefore are not included in the DL object model.

BackoffExponent[Neighbor] and BackoffCounter[Neighbor] are set to zero every time an ACK/NAK DPDU is received for a unicast Data DPDU that was sent to a particular neighbor in a shared link.

A BackoffCounter[Neighbor] value of zero allows the DLE to send a Data DPDU at the next shared-link transmission opportunity. Following an unsuccessful transmission to the neighbor in a shared link, if the current value of BackoffExponent[Neighbor] is less than dlmo.MaxBackoffExp, the DLE increments BackoffExponent[Neighbor] and then sets BackoffCounter[Neighbor] by selecting a value uniformly from the interval $0..2^{(\text{BackoffExponent[Neighbor]})}-1$. For each transmit opportunity in a shared link, BackoffCounter[Neighbor] is decremented until it reaches zero. If a transmit opportunity is in a dedicated link (no exponential backoff indicator), the DLE may use the link regardless of the value of BackoffCounter[Neighbor]. The attribute dlmo.MaxBackoffExp limits the maximum value of BackoffExponent[Neighbor].

Retry behavior can be configured by the system manager. DLMO attributes that relate to retries include:

- dlmo.MaxBackoffExp. The maximum value for BackoffExponent[Neighbor].
- dlmo.MaxLifetime and dlmo.Graph.MaxLifetime: maximum lifetime of a Data DPDU. A Data DPDU that is being forwarded shall be deleted if held in a DLE's message queue for longer than MaxLifetime. dlmo.MaxLifetime provides a default value for the DLE. A non-null value for dlmo.Graph[ ].MaxLifetime indicates that dlmo.MaxLifetime shall be overridden and set to the specified value for Data DPDUs following that particular graph.
- Operation of exponential backoff is illustrated in the following pseudocode:

```
// For each neighbor, independently
BExp[Nei] = 0; // BackoffExponent[Neighbor] in text
BCnt[Nei] = 0; // BackoffCounter[Neighbor] in text
For each timeslot (
If (transmit link and Data DPDU match)( // See 9.1.8.5
   If (not exponential backoff link) (
      // Dedicated link
      Attempt to transmit Data DPDU using link;
      If (transmit was successful) remove Data DPDU from queue;
      )
   Else (
      // Shared link
      If (BCnt[Nei] > 0) BCnt[Nei]--
      Else (
         Attempt to transmit Data DPDU in link;
         If (transmit was successful) (
            Remove Data DPDU from message queue;
            BExp[Nei]=0;
            BCnt[Nei]=0;
            )
         Else (
            // Transmit failed; exponential backoff
            If (BExp[Nei] < MaxBackoffExp) BExp[Nei]++;
               BCnt[Nei] = Random (0, 2^(BExp[Nei]-1));
            )
         )
      )
   )
Delete all messages beyond MaxLifetime;
If (no queued Data DPDU for neighbor) (BCnt[Nei]=0; BExp[Nei]=0;)
```

NOTE As described in 9.1.8.5, it is possible for a link to be configured as a Transmit/Receive (T/R) link, which is a compressed representation of a paired transmit link and receive link. Logically, T/R links are processed as two independent links.

9.1.8.3 Superframe channel use

Timeslots within a superframe are associated with a slow or slotted-channel-hopping pattern, as well as an offset into that pattern.

From the perspective of each DLE using a superframe, there is a baseline channel-hopping pattern offset, which may vary from DLE to DLE and which may be overridden with an alternative offset applied to a link or collection of links within a superframe.

A given unicast D-transaction occurs on a single channel, with the Data DPDU and ACK/NAK DPDU(s) all transmitted on the same channel.

A superframe is not limited to one channel at a time; rather, a superframe is a two-dimensional structure indicating time and channel, as was previously illustrated in Figure 62 (see 9.1.7.2.5).

Figure 62 shows a superframe, with time on the horizontal axis and channel on the vertical axis. The superframe spans all of the channels over the length (duration) of that superframe. As shown in Figure 62, sixteen DLEs may use sixteen different offsets from channel-hopping pattern A; the superframe encompasses all of the channel assignments for all of the superframe timeslots.

The default channel offset may be different for transmitting versus receiving and may vary by link.

The period of the channel-hopping pattern is not necessarily related to the length of the superframe. Referring to Figure 62, a superframe might be configured as 25 timeslots long, even though channel-hopping pattern A is only 16 hops long.

For frequency diversity, superframe length and channel-hopping pattern size may be configured to be relatively prime, that is, with no common factors. As a counter-example, consider a configuration wherein superframe length is 25 timeslots, with a channel-hopping pattern repeating on a 15-channel cycle, resulting in a superframe schedule where only 3 of the 15 available channels are ever used. Such an arrangement can cause regulatory issues in situations where use of all channels is required by each device.

9.1.8.4 Organizing superframes

9.1.8.4.1 General

Two general superframe types are supported:

- Slotted-channel-hopping, which makes optimal use of available digital bandwidth and supports battery-powered routers.
- Slow-channel-hopping, intended for routers with available energy to run their receivers continuously during a given period. Slow-channel-hopping allows neighboring DLEs to operate with less exacting time synchronization requirements, particularly during the neighbor discovery process.

Hybrid configurations may be arranged by combining superframes, for example, one slotted and one slow. Slotted- and slow-channel-hopping will be discussed separately, followed by some examples of hybrid configurations.

NOTE In the marketplace, slow-channel-hopping is sometimes referred to as CSMA, and slotted-channel-hopping as TDMA. These terms are not used in this standard, except to the extent that CSMA/CA is supported by the standard in a literal sense. (See 9.1.9.4.8.) Slow-channel-hopping is built on a TDMA base, slotted-channel-hopping includes CSMA aspects. The solution designer is free to mix the approaches.

9.1.8.4.2 Superframe scope

Superframes are commonly discussed as abstractions that span several DLEs. Nonetheless, while the superframe may be conceptualized at the D-subnet level, the scope of the superframe data structure is limited to each DLE. A superframe is instantiated as a data structure on a single DLE that independently drives its DLE state machine. A DLE's superframe definitions need to relate to those of its neighbors so that DLEs communicate at the same time. Superframe definitions within each DLE are numbered, but that numbering is only needed by the DMAP for table read/write operations and by other objects and attributes within the DLE that refer to the superframe.

A superframe may be contrasted with a routing graph's scope. A graph ID, unlike a superframe ID, is carried in a Data DPDU's DROUT header, and a graph ID shall be consistent and unique in all DLEs that use the graph. No such constraints apply to a superframe.

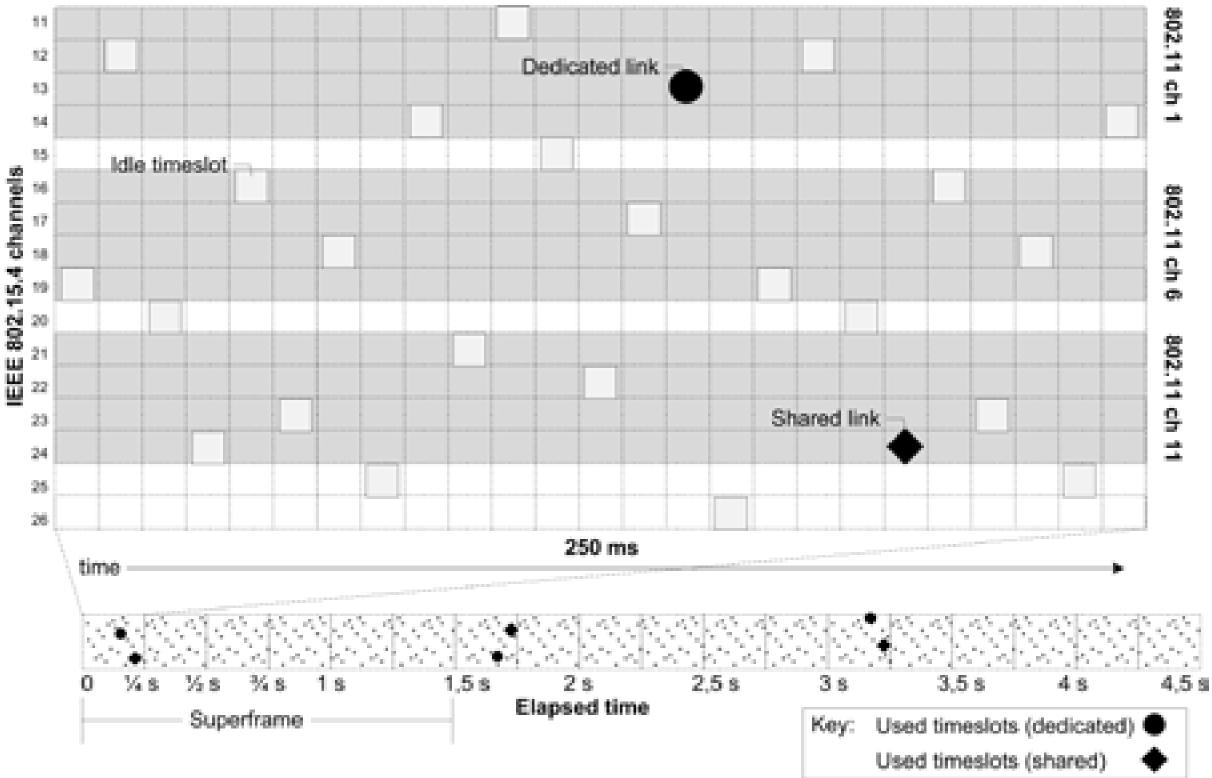
9.1.8.4.3 Blocks of contention-based capacity

Mains powered routers may sensibly operate their receivers continuously. As its lowest priority superframe, such a router may support a superframe comprised partially or entirely of receive links. The router's neighbors may maintain corresponding shared transmit links to the router. Such a configuration results in blocks of contention-based digital bandwidth available to the routers neighbors. Slow-channel-hopping or slotted-channel-hopping may be used in a superframe of that type. Channel-hopping-offset may be selected to avoid collisions between dedicated links versus a general inventory of shared links.

9.1.8.4.4 Slotted-channel-hopping

Slotted-channel-hopping uses channel-hopping superframe timeslots of equal duration. Each superframe timeslot uses a different radio channel in a hopping pattern. In slotted-channel-hopping, each superframe timeslot is intended to accommodate one D-transaction, including a Data DPDU and its ACK/NAK DPDU(s).

Figure 70 illustrates a slotted-channel-hopping superframe from the perspective of one DLE, which may be a router.



NOTE Channel numbers shown are those of IEEE 802.11 and IEEE 802.15.4:2011, rather than those of this standard.

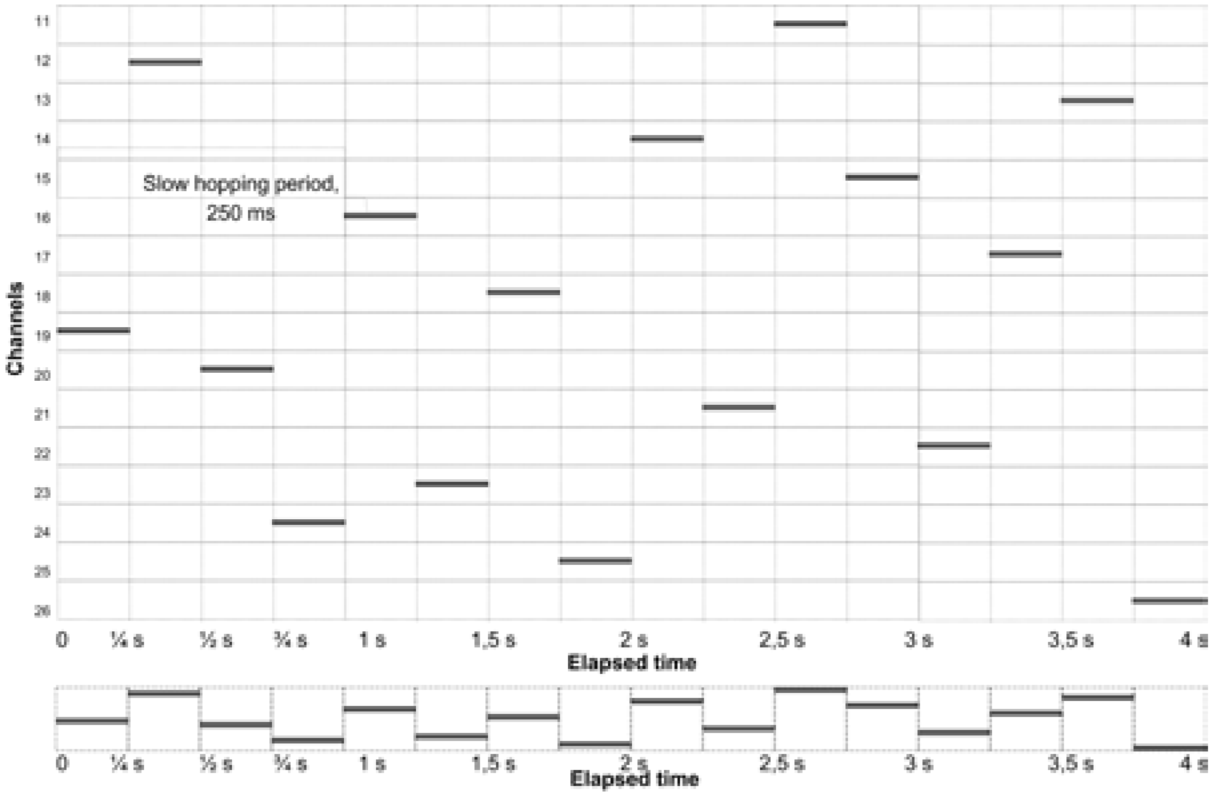
Figure 70 – Example superframe for slotted-channel-hopping

In this example, the superframe is 1,5 s long. Timeslots with link assignments are depicted with circles (dedicated links) and diamonds (shared links). As Figure 70 shows, from the perspective of a single DLE, many superframe timeslots might be left idle. Timeslots with link assignments repeat at a fixed interval defined by the superframe length.

9.1.8.4.5 Slow-channel-hopping

In slow-channel-hopping, a collection of contiguous superframe timeslots is grouped on a single radio channel. Each such collection of superframe timeslots is treated as a single slow-channel-hopping period.

Figure 71 illustrates slow-channel-hopping.



NOTE Channel numbers shown are those of IEEE 802.15.4:2011, rather than those of this standard.

Figure 71 – Example superframe for slow-channel-hopping

Timeslots in a slow-channel-hopping superframe are generally shared, providing immediate, contention-based channel bandwidth on demand to a router's immediate neighbors.

Figure 72 shows the main components of a slow-channel-hopping superframe.

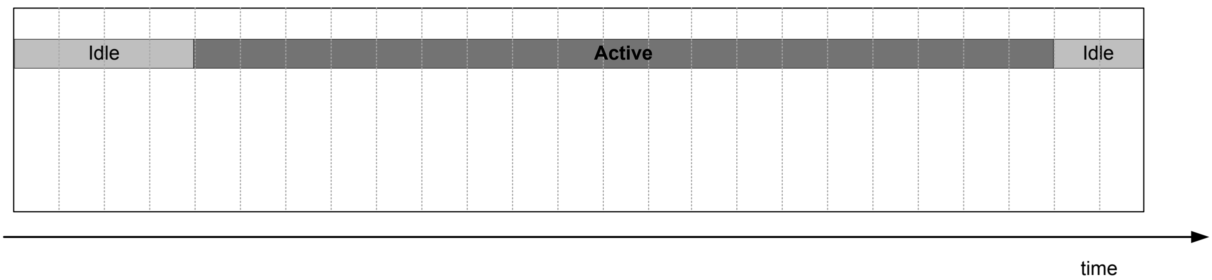
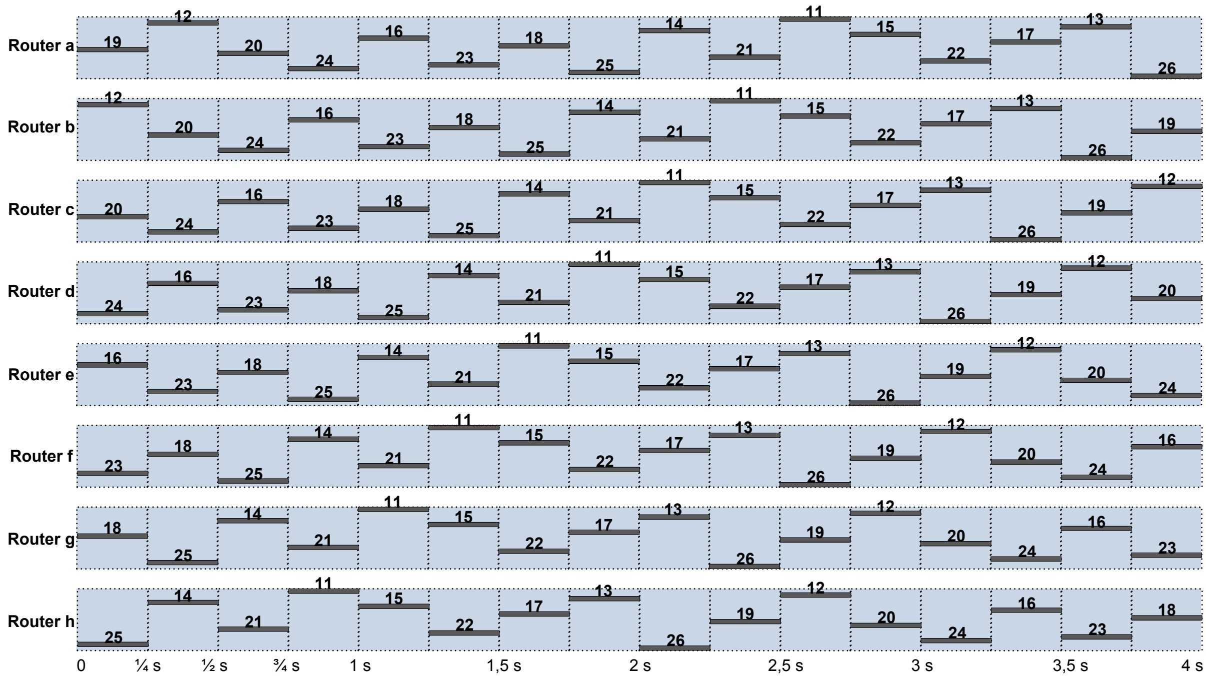


Figure 72 – Components of a slow-channel-hopping superframe

A baseline slow-channel-hopping period has a fixed duration, with a fixed number of idle timeslots (which may be zero) at the beginning and end of the hop. The example of a slow-channel-hopping period shown in Figure 72 is comprised of 25 timeslots, including four idle timeslots at the beginning, nineteen active timeslots in the middle, and two idle timeslots at the end. The idle timeslots are intended to support hybrid configurations where slow-channel-hopping superframes are paired with slotted-channel-hopping superframes, with the slotted-channel-hopping timeslots scheduled for use during the idle periods of the slow-channel-hopping superframe.

NOTE 1 Idle periods as described here are configurable by matching superframe and channel-hopping phases, and defining links that match the desired active range.

It is not necessary for all routers in a common area to hop together. Figure 73 shows how many routers can be assigned slow-hopping patterns that are disjoint from each other, thus avoiding collisions. It does not imply that all routers in a system use disjoint communication channels. illustrates.



NOTE Channel numbers shown are those of IEEE 802.15.4:2011, rather than those of this standard.

Figure 73 – Example configuration for avoiding collisions among routers

Each router may use a different offset into the channel-hopping pattern. With a 16-channel hopping pattern, each of up to 16 routers may be configured with a different offset into the pattern, so that no two routers use the same channel at the same time.

NOTE 2 Although the above example purports to show how collisions can be avoided through disjoint assignments of channel hopping patterns, a realistic system would require at least one shared channel-hopping pattern via which the routers could communicate with each other.

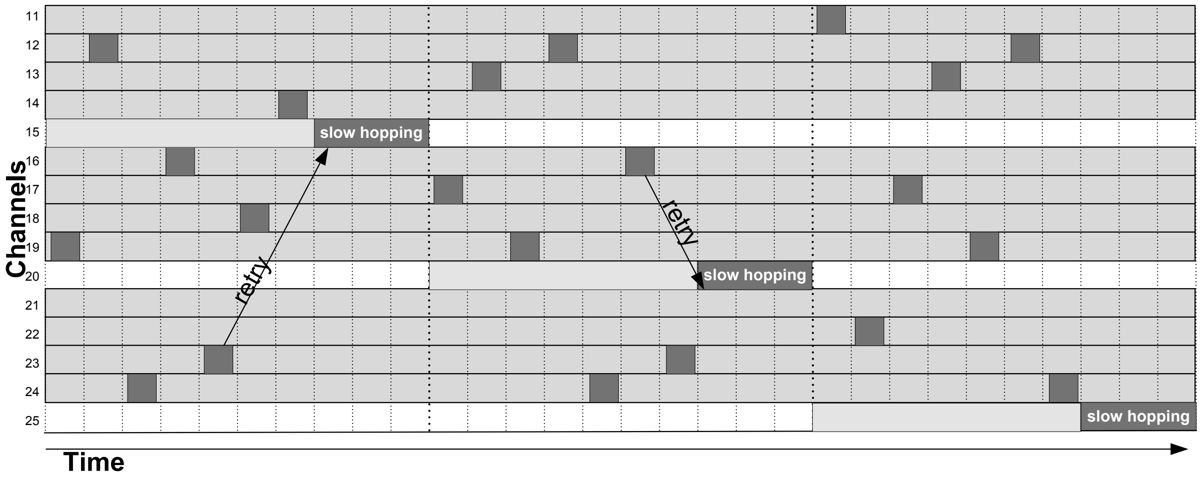
See 9.4.3.5.5 for more detail on slow-channel-hopping.

9.1.8.4.6 Hybrid channel-hopping configurations

Hybrid configurations may use combinations of the slotted-channel-hopping and slow-channel-hopping superframes.

Hybrid configurations are usually arranged so that slotted-channel-hopping links are allocated for scheduled, periodic messaging. This may leave blocks of lightly-used slow-channel-hopping capacity, available on a contention basis, for less predictable uses such as alarms and retries.

The example in Figure 74 illustrates how slotted-channel-hopping and slow-channel-hopping superframes may be combined.



NOTE Channel numbers shown are those of IEEE 802.15.4, rather than those of this standard.

Figure 74 – Hybrid configuration

In Figure 74, slotted-channel-hopping has been overlaid on a slow-channel-hopping background. The slow-hopping periods fill in the time between periodic collections of dedicated superframe timeslots.

In this configuration, if an attempted transmission in a dedicated timeslot fails, the succeeding slow-channel-hopping period can be used to retry the transmission. In Figure 74, two of the superframe timeslots are shown with retries on different channels during the subsequent slow-channel-hopping period.

9.1.8.4.7 Superframes and spectrum management

The term spectrum management refers to the ability of the system manager to configure a D-subnet to block unwanted channels from operation in a D-subnet.

Each superframe includes a channel map, called Superframe[ ].ChMap, that is a bit mask of channels that shall be included and excluded in the hop sequence that is referenced by the superframe. Excluded channels have the effect of shortening the hop sequence.

For example, a superframe channel map that includes channels 11..25 and excludes channel 26 has the effect of shortening the hop sequence by removing channel 26 from the hop sequence. More generally, the system manager may eliminate any collection of channels from the hop sequences that are referenced by superframes in a D-subnet, with the result of removing those channels from operation.

There is also a channel map in the DeviceCapability attribute that is reported to the system manager when the DLE joins the D-subnet. The DeviceCapability channel map does not shorten any channel-hopping sequence used by the DLE, but rather is a signal to the system manager that, for regulatory reasons, any link using one of the excluded channels will be treated as idle.

The system manager may also block use of certain channels through the attribute dlmo.IdleChannels. Unlike the channel map in the superframes, dlmo.IdleChannels does not cause hop sequences to be shortened; rather, it causes links on designated channels to be treated as idle. dlmo.IdleChannels is intended to provide a quick way for the system manager to disable certain channels in a way that does not require D-subnet-wide coordination of revised hop sequences.

Channel-specific diagnostics, as described in 9.4.2.27, provide the system manager with information to support spectrum management.

9.1.8.5 DLE message queue operation

DL routers compliant with this standard shall support a DLE message queue. This message queue has some attributes that can be configured by the system manager, affecting how the queue operates.

The standard does not generally specify internal DLE mechanisms. However, to a limited degree, the DLE message queue is specified by the standard. The system manager can configure the DLE message queue to achieve particular quality of service objectives, and a limited model of message queue behavior is implicit in the configuration alternatives provided to the system manager.

When a DLE receives a unicast Data DPDU from its neighbor, it first assesses whether the DSDU should be passed to the NL or forwarded to another DLE through the DL, as described in 9.3.3.6.

If the Data DPDU needs to be forwarded, the DLE then evaluates whether the Data DPDU should be accepted or NAKed, in part based on the available capacity of the DLE message queue. Data DPDUs shall be NAKed if the DLE message queue has run out of capacity for Data DPDUs of that type.

The DL reports its forwarding queue capacity to the system manager when the DLE joins the D-subnet, through the attribute dlmo.DeviceCapability, field QueueCapacity. This externally reported queue capacity does not include portions of the queue that are reserved for the DLE's internal use. For example, the DLE message queue in a particular DLE might report that it has a queue capacity for five Data DPDUs. The system manager is then able to configure those five positions in the queue. This nominal capacity refers only to a portion of the message queue that is exclusively used to route Data DPDUs through the DLE. In practice, the actual DLE has additional message queue capacity space that it does not report to the system manager, because the DLE also needs to handle Data DPDUs on its own behalf. Unreported message buffer capacity, for the DLE's own use, is considered an internal DLE matter and is not allocated by the system manager. The system manager assumes that the DLE has sufficient message queue capacity for its own use when contracts are granted.

Consider the example of a field router with a reported DLE message buffer capacity of eight Data DPDUs. The actual buffer capacity may be twelve Data DPDUs, in which case the difference between reported and actual capacity (four Data DPDUs) is for the DLE's own use. In this example, the system manager might reasonably configure the DLE's nominal buffer capacity as follows:

- No more than three of the eight buffers will be used to forward Data DPDUs with priority ≤ 2.
- No more than five of the eight buffers may be used to forward Data DPDUs with priority ≤ 5.

See 9.4.2.26 for further discussion of queue buffer capacity and priority levels.

In addition, for a finer grained degree of control, the system manager can designate a certain number of buffers exclusively to forward Data DPDUs that are being routed along a specific graph, as described in 9.4.3.7.

Figure 75 provides an overview of how links interact with an implicit DLE message queue within a DLE.

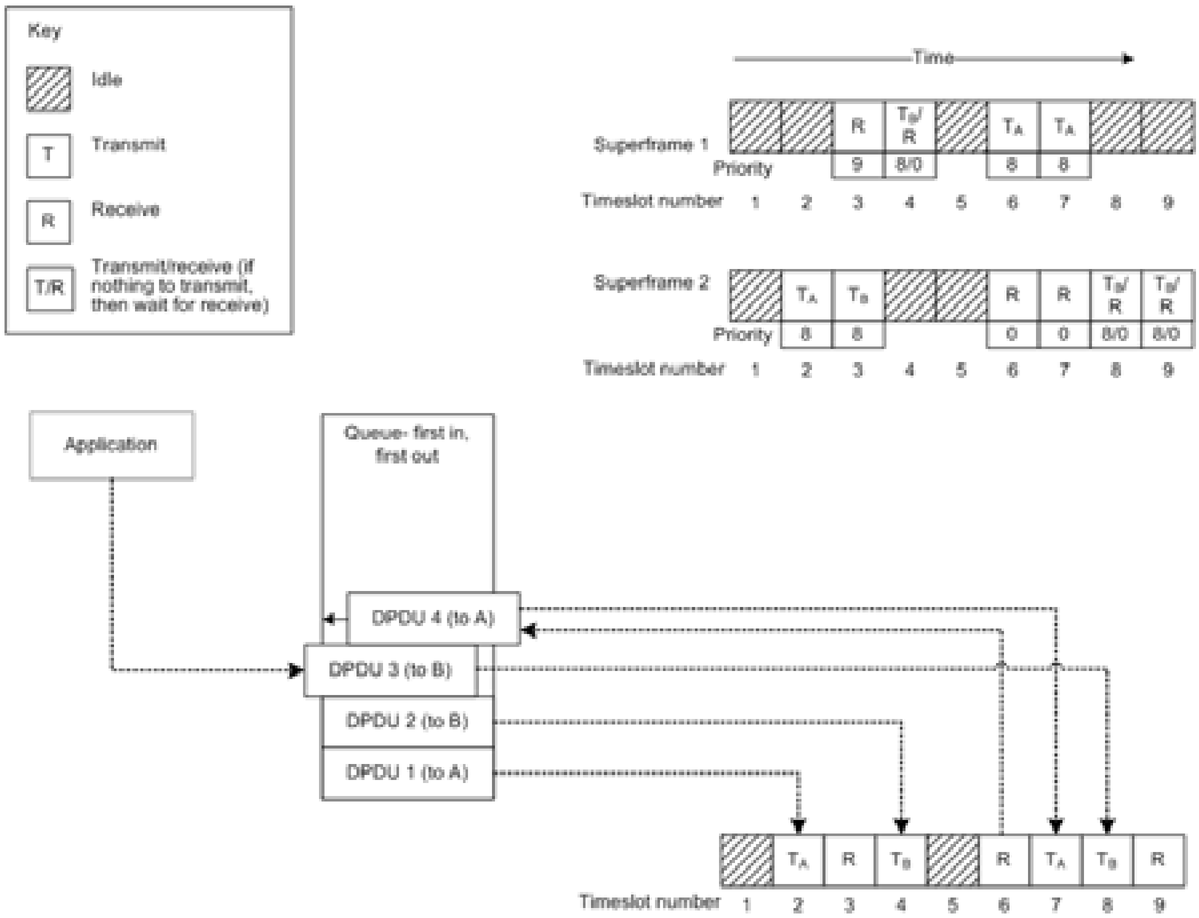


Figure 75 – Timeslot allocation and message queue

As shown in Figure 75, Data DPDUs are held in a message queue until transmit links become available. Data DPDUs are placed in the queue in the order they are received. Generally, retrieval of Data DPDUs from the queue is first in, first out (FIFO).

NOTE This simplified example shows only a single destination address for each Data DPDU. In practice, each Data DPDU on the message queue is actually a candidate for links to multiple neighbors.

Consider the example of Data DPDU 3. Data DPDU 3 originates in an application and enters the DL through the DDSAP (Figure 53). When the Data DPDU is processed by the DLE, it is placed in the DLE’s message queue. Based on the Data DPDU's ultimate destination, the DLE determines that the Data DPDU needs a link to DLE B for its next hop. Data DPDUs 1 and 2 are already on the queue, so Data DPDU 3 is queued behind them. When a link to DLE B becomes available, Data DPDU 2 will be sent before Data DPDU 3.

Each Data DPDU on the queue is assigned a priority by the originating DLE. This simplified tutorial assumes that all Data DPDUs have equal priority. In actual practice, link priority takes precedence over Data DPDU priority, in the sense that Data DPDUs are not considered for transmission until after a transmit link is selected. Once a transmit link has been selected, the Data DPDU on the queue is selected based first on priority, and then on a FIFO basis if multiple candidate Data DPDUs have the same priority.

Data DPDU 4 shows an example where a Data DPDU is received in one timeslot and is available to be forwarded in the next timeslot. In general, a Data DPDU received in one timeslot (timeslot N) shall be available on the queue for forwarding in next timeslot (timeslot N+1). An exception is allowed when default timeslot template 3 is used (see Figure 157). In that case, the D-transaction might be incomplete at the start of the next timeslot. A Data DPDU received starting in one timeslot (timeslot N), using default timeslot template 3, shall be available on the queue for forwarding in the timeslot following the next timeslot (timeslot N+2).

In Figure 75, two superframes are shown, and each superframe is associated with a series of links. For example, timeslot 3 in superframe 1 is associated with a receive link (R), while timeslot 3 in superframe 2 is associated with a transmit link to DLE B (T_B). Idle slots do not have defined links.

Each link has a priority. The attributes dlmo.LinkPriorityXmit and dlmo.LinkPriorityRcv provide default link priorities, which may be overridden for any particular link. The example in Figure 75 mostly shows the default priorities of 0 for receive links and 8 for transmit links.

In most cases the system manager should assign lower priority to receive links (R) than to transmit links (T), thus giving precedence to servicing outgoing Data DPDUs on its queue. However, this is not a strict requirement. For example, if a latency-critical incoming flow is scheduled for a particular timeslot, the system manager may configure receive links with higher priority in that case. As an illustration in Figure 75, the third link in superframe 1 is assigned a priority of 9, giving this particular receive link a higher priority than a transmit link at the same time.

Transmit/receive (T/R) timeslots use a compressed format to combine a transmit link and a receive link. Logically, a T/R link is two links, with the receive part of the link having a priority of dlmo.LinkPriorityRcv which defaults to zero. For example, if a timeslot has a high priority T_A/R link and lower priority T_B link, the T_B link has higher priority than the R part of the T_A/R link. Baseline operation is that a Data DPDU queued for transmission and a link are matched at the start of a timeslot, and the timeslot is assigned to a D-transaction that will be run to completion according to the link configuration. In the event that a D-transaction is aborted due to CCA detection of competing channel activity, an optimized implementation may complete the timeslot using a receive link that is valid for the same time interval.

Once a queued Data DPDU has been transmitted and acknowledged, the D-transaction is deemed successful and the Data DPDU is deleted from the queue. The following example assumes that all transactions are successful.

The box at the bottom right of Figure 75 illustrates how links are used in the following example.

- Timeslot 1: The first link in both superframes 1 and 2 are idle; therefore, the first timeslot is idle.
- Timeslot 2: The second link in superframe 1 is idle, but there is a transmit link for DLE A (T_A) in superframe 2. There is also a Data DPDU to DLE A in the queue; therefore, the second timeslot is assigned for transmission to DLE A, and Data DPDU 1 is sent.
- Timeslot 3: Superframe 1 has a receive link, and superframe 2 has a transmit link. The link in superframe 1 takes precedence due to its priority, so timeslot 3 is used to listen for incoming Data DPDUs. (As described above, receive links are usually assigned lower priority than transmit links. This example illustrates how a system manager can give priority to a receive link, for example, to service an incoming flow.)
- Timeslot 4: Superframe 1 has a T_B/R link, indicating that it should transmit if there is a Data DPDU for DLE B in the queue. Data DPDU 2 is sent.
- Timeslot 5: Both superframes are idle in timeslot 5, so the timeslot is idle.
- Timeslot 6: Superframe 1 designates a transmission link to DLE A, but there is no longer a Data DPDU for DLE A in the queue. Since there is nothing useful for the transmit link to do in this slot, the link in superframe 2 is used. The DLE receives an inbound Data DPDU, determines that its next hop is to DLE A, and places the Data DPDU on the queue.
- Timeslot 7: Now that there is a Data DPDU for DLE A on the queue, the transmit link in superframe 1 gets priority and Data DPDU 4 is sent. Note that Data DPDU 3 was skipped over because no T_B link has become available yet.
- Timeslot 8: Now that a T_B link is available, Data DPDU 3 is sent.

- Timeslot 9: The T_B/R link results in a receive slot, because there is no Data DPDU to DLE B on the queue.

This example was simplified in some essential respects.

– As noted above, Data DPDU priorities were assumed to be equal. In practice, if two Data DPDUs on a message queue both match a given link, the Data DPDU with the higher priority is transmitted. The FIFO queue position is relevant only for Data DPDUs of equal priority.

– If a unicast Data DPDU does not receive an acknowledgment, it stays on the queue and the DL retry strategy is applied. See 9.1.8.2.

– A link may be configured for use by a particular graph. In that case, Data DPDUs with that graph ID shall be granted prioritized or exclusive access to the link. See 9.4.3.7.

– A DLE may be designed to skip links on radio channels with a history of subpar connectivity. A DLE may also skip links in order to retry on an alternative link. See 9.1.7.2.4.

Operation of queue processing is illustrated in the following pseudocode:

```
For each timeslot (
   Order Data DPDUs on queue by priority;
   // FIFO within priority

   Order links by priority;
   // Treat T/R link as two links, with receive side
   // of link assigned link priority dlmo.LinkPriorityRcv;
   // Within link priority, order by superframe priority,
   // then superframe number (highest first);

   For each link, in priority order
      If it is a receive link (
         Use the receive link;
         Done with timeslot;
         )
      Else // it is a transmit link
         For each Data DPDU, in priority order
            If link matches Data DPDU (
                    Use the link;
                    Done with timeslot;
                    )
)
```

9.1.9 DL time keeping

9.1.9.1 Timing

9.1.9.1.1 General

The DL propagates and uses international atomic time (TAI) for its internal operation, and also provides TAI time as a service (through the DMAP) to wireless devices compliant with this standard.

DLEs within a D-subnet may be configured to track a shared sense of time to within a few milliseconds of each other, to support sequence of event reporting and other application-layer coordinated operations within the scope of a D-subnet. Synchronized time among immediate neighbors is also essential for operation of the wireless protocol.

Slotted-channel-hopping requires tight time synchronization among immediate neighbors. However, the internal clock of a non-routing DLE that has been disconnected from D-subnet operation for more than a few minutes may have drifted by tens or hundreds of milliseconds or more in relation to the overall D-subnet clock. Slow or hybrid channel-hopping configurations support the continued operation of such DLEs.

9.1.9.1.2 International atomic time

In this standard, time is based on international atomic time (TAI) as the time reference. See 5.6.

9.1.9.1.3 Alignment intervals

In this standard, one-second TAI increments are divided into 250 ms (1/4 s) alignment intervals, wherein the 250 ms (2^{-2} s) cycles shall align with nominal TAI s as shown in Figure 76.

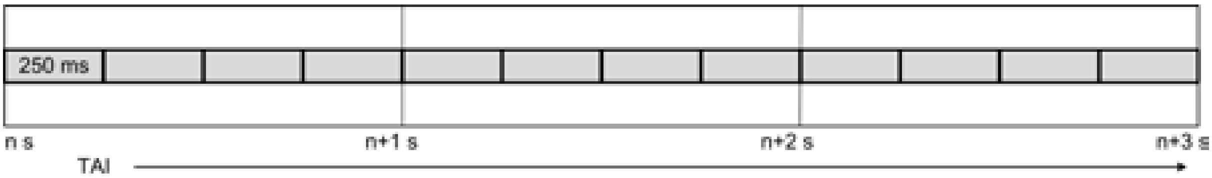


Figure 76 – 250 ms alignment intervals

Continuous control loops in the process industries, which have been a traditional focus of IEC TC65, are frequently based on fixed-period computation of control outputs. These process loops usually repeat at rates of 4 Hz (250 ms), 1 Hz (1 s) or multiples of either 4 s or 5 s. Process monitoring is usually mapped onto this same 4 Hz (and slower) structure.

Applications with slightly higher loop rates, such as compressor surge control loops running at 12 Hz, are supportable by scheduling multiple communications opportunities per 250 ms cycle.

9.1.9.1.4 Timeslot duration, timeslot alignment, and idle periods

Within 250 ms alignment intervals, time is divided into timeslots of configurable but equal duration. A timeslot is a time interval of predefined duration used to send or receive a Data DPDU and any corresponding ACK/NAK DPDU(s). Timeslots are generally shared by at least one pair of DLEs that communicate during the allocated time. The DL organizes these timeslots into superframes, which are collections of timeslots with a common period and potentially other common attributes.

Timeslot durations are configured during D-subnet setup. Normally, during a given period of operation, all timeslots within a D-subnet have the same duration. Timeslot duration is configured in units of 2^{-20} s (~0,95 μs).

NOTE 1 The DL binds timeslot duration to superframes, and nothing in the standard prevents multiple superframes with different timeslot durations from being active simultaneously within a D-subnet. However, this standard considers only configurations wherein a single timeslot duration is used in a given D-subnet. A DLE supporting multiple timeslot durations simultaneously, such as a backbone router or a bridge between two D-subnets, is modelable as containing multiple DLEs running in parallel.

Timeslots align with the 250 ms alignment intervals, but timeslot durations do not necessarily divide evenly into 250 ms. Therefore, the system inserts a short idle period every 250 ms as needed, thus realigning the timeslots to a 4 Hz cycle. This is shown in Figure 77 with two illustrative examples using different timeslot durations of 9,999 ms and 11,719 ms, respectively.

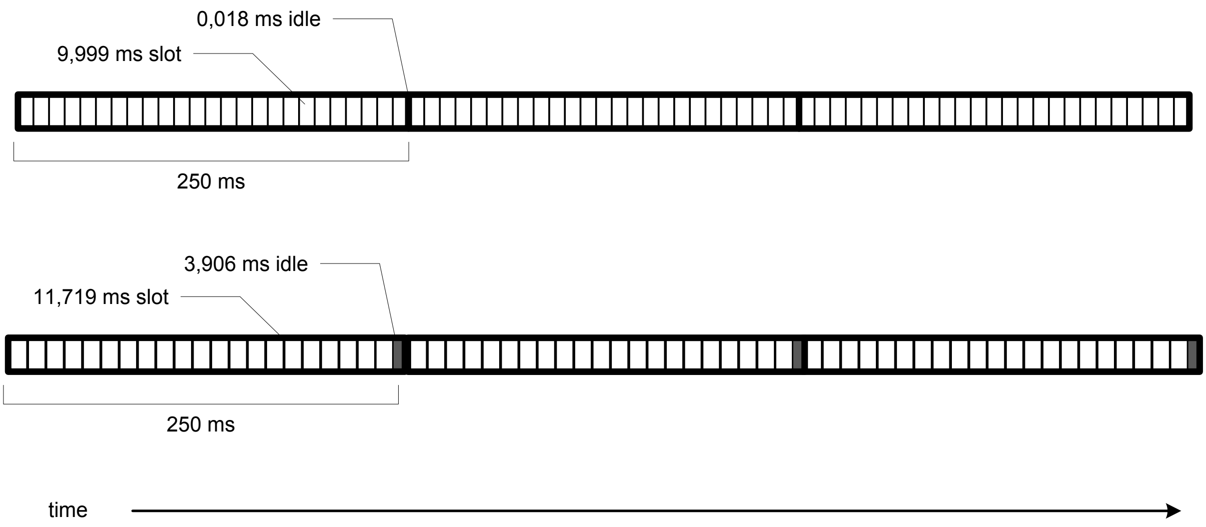


Figure 77 – Timeslot durations and timing

NOTE 2 Calculations for the idle times in Figure 77:

a) 10 485 × 25 × 2^{-20} s = 249,982 ms. Subtract from 250 ms to get 0,018 ms;

b) 12 288 × 21 × 2^{-20} s = 246,094 ms. Subtract from 250 ms to get 3,906 ms.

NOTE 3 A timeslot intended closely to approximate 10 ms is 10 485 units, or 9,999 275 ms, in duration.

The idle period is not used by the system for scheduled operations; it is simply a short period inserted to ensure that timeslots align and repeat at 250 ms intervals. This makes it straightforward for the system manager to organize collections of timeslots that repeat at exact multiples of 250 ms.

Slow-hopping periods can span an alignment interval. In such cases, slow-hopping periods are not interrupted by idle periods; that is, if a slow-hopping period traverses the edge of an alignment interval, the receiver's radio continues operation through the idle period.

9.1.9.1.5 Scheduled timeslot time

Each DL timeslot has a scheduled fractional-second start time, which is called the scheduled timeslot time. Both the DL and higher-layer protocols use this coordinated time sense as security material, thus providing replay protection. See 7.3.2.6.

Since the 250 ms intervals align with TAI quarter-seconds, scheduled timeslot timing can be derived from the timeslot duration. For example, if timeslot duration is 9,999 ms, scheduled timeslot times in the second quarter-second of TAI time t are t + 0,250 000 s, t + 0,259 999 s, t + 0,269 998 s, etc.

NOTE This mixed-radix time representation facilitates inter-conversion at higher layers, where PDUs often span D-subnets with different data rates, different timeslot durations and/or different DL protocols.

The scheduled timeslot start time is in units of 2^{-20} s (microsecond precision). DLEs commonly use internal clocks based on a 2^{15} Hz (32 KiHz) very-precise very-low-power "watch" crystal. Implementations may, and commonly will, round the scheduled timeslot start time to the nearest 32 KiHz clock tick. This rounding is permitted on a per-timeslot basis, without adjusting the underlying timeslot schedule based on units of 2^{-20} s. However, this rounding to 32 KiHz is not permitted to accumulate over a 250 ms period, as shown in Table 101. This consideration is included within the ±96 µs jitter allowed by this standard; see 9.4.3.3.1.

Table 101 – Approximating nominal timing with 32 KiHz clock

Nominal timeslot offset (ms)	2^{-20} s (clock counts)	2^{-15} s (clock counts, rounded)	Actual timeslot offset (ms)
0	0	0	0
10	10 485×1	328	10,010
20	10 485×2	655	19,989
30	10 485×3	983	29,999
•••	•••	•••	•••
200	10 485×20	6 553	199,982
210	10 485×21	6 881	209,991
220	10 485×22	7 208	219,971
230	10 485×23	7 536	229,980
240	10 485×24	7 864	239,990

9.1.9.2 DL time propagation

9.1.9.2.1 General

The DLE uses TAI time for its internal operation, and provides a notion of TAI time as a service (through the DMAP) to wireless neighbors compliant with this standard.

In a D-subnet, DLEs may take on three functions in the time propagation process:

- DL clock recipient, a receiver of periodic clock updates through the DL; or
- DL clock source, a provider of periodic clock updates to DL neighbors; or
- DL clock repeater, a DL clock recipient that also acts as a DL clock source to some of its neighbors.

Additional information about time propagation can be found in 6.3.10.

9.1.9.2.2 DLE clock stability

The clock stability of the DLE is the nominal clock stability of a wireless device, in the presence of periodic time corrections from the D-subnet.

The DLE, when configured as a clock recipient, relies on the D-subnet to provide periodic time updates wirelessly. It may also use these time updates in calibration of its internal clock to account for conditions such as temperature, aging, and voltage.

The time reference for a D-subnet originates from one or more clock masters, which provide time that is monotonically increasing at a rate that closely tracks real time.

When the DLE joins the D-subnet, the DLE reports its clock stability to the system manager as dlmo.DeviceCapability.ClockStability (called ClockStability here), which is in units of parts per million (1×10^{-6}). ClockStability applies to any arbitrary 30 s period during the expected life of the device, under all conditions that a user might reasonably expect from the device's published specifications.

For example, if ClockStability reports that a DLE has a clock with maximum instability of 10×10^{-6} s/s, then the DLE's clock shall be stable to within ±300 μs during any arbitrarily selected 30 s period.

ClockStability is reported as an envelope. For example, if ClockStability reports that a DLE has a maximum clock instability of 10x10^{-6} s/s, then the clock shall be stable to within ±300 μs at all instants during any arbitrarily selected 30 s period. This is intended to allow for devices that are subjected to occasional environmental shocks that can cause small clock discontinuities. Small discontinuities are acceptable, as long as they do not add up to more than ±ClockStability × 30 μs at any time over any 30 s period.

NOTE 1 ClockStability, specified in units of 1x10^{-6} s/s, is equivalently 1 μs/s.

NOTE 2 It is possible that neighboring DLEs have clocks that drift in opposite directions. Therefore, in the worst case, ClockStability is additive between pairs of neighboring DLEs. In practice clock repeaters are corrected periodically, which lessens that effect.

The standard assumes that clock drift is negligible within a timeslot.

ClockStability is reported to the system manager without caveats. If the device is specified to work under environmental stress, such as extremes of temperature or mechanical shock, then ClockStability shall reflect performance under such stress.

The attribute dlmo.ClockExpire, configured by the system manager, provides the maximum number of seconds that the DLE can safely operate in the absence of a clock update. Normally, the system manager arranges that a DLE will maintain clock synchronization as a by-product of normal communication. However, when the DLE fails to receive a clock update for an extended period of time, defined by ClockExpire, the DLE should actively interrogate a DL clock source for a time update. Failure to do so will eventually result in loss of time synchronization with the D-subnet. ClockExpire defaults to a value that is appropriate for use during the join process.

A clock repeater should not send clock corrections to its neighbors through ACK/NAK DPDUs if it has not itself received a clock correction for a period that exceeds the ClockExpire attribute. If a clock repeater's clock has expired and it is polled for a time update, it should respond with a NAK1.

A DLE with an expired clock should not be used as a clock repeater, but it may continue to operate in the D-subnet, albeit with a potential risk of losing synchronization with its neighbors. The attribute dlmo.ClockTimeout provides the maximum amount of time that a DLE may reasonably continue operating in a D-subnet in the absence of a clock update. If the DLE has not received a clock update for a period of time that exceeds DLTimeout, the DLE may reasonably reset itself to the provisioned state and initiate a search for a new D-subnet.

NOTE 3 A DLE operating in a slow-channel-hopping configuration is capable of being configured to retain a D-subnet connection for extended periods of time, even with a clock that has drifted across timeslots.

9.1.9.2.3 Preferred and secondary clock sources

The system manager configures each DL clock recipient to treat one or several of its neighbors as DL clock sources. Such DL clock sources may be designated as preferred or secondary, based on the attribute dlmo.Neighbor[ ].ClockSource. Multiple neighbors may be designated as preferred DL clock sources. A DL clock recipient should adjust its clock value whenever it has an interaction with a preferred DL clock source.

The attribute dlmo.ClockStale defines a period of time that shall pass before a DLE begins accepting clock updates from secondary DL clock sources. If, after a period of ClockStale, no clock update is received from any preferred source, the DL clock recipient should accept clock updates in its interactions with neighbors that are designated as secondary DL clock sources. Once a DL clock recipient accepts a clock update from a secondary DL clock source, it should continue to use that secondary DL clock source until either (a) it receives a clock update from a preferred DL clock source, or (b) the secondary DL clock source times out.

The attribute dlmo.ClockStale determines the timeout interval. For example, if ClockStale is set to 45 s by the system manager, then a DL clock source should not accept clock updates from a secondary DL clock source until it has not received a clock update from any preferred DL clock source for at least 45 s.

Each DL clock recipient can be configured to retain and periodically report statistics (see 9.4.3.9) for any of its preferred DL clock sources, including:

- A count of clock timeout events.
- A running average of clock corrections, a signed integer in units of 2^{-20} s, indicating a bias if nonzero.
- The standard deviation of clock corrections, estimated in units of 2^{-20} s, for example, a value that roughly accounts for approximately 68% of clock corrections.
- A count of clock corrections in excess of three such standard deviations.

9.1.9.2.4 Shared time sense during D-subnet operation

D-subnet transactions nominally occur in a DL timeslot, on a schedule known to both sender and receiver. This shared sense of time is used as security material, both for header compression and for replay protection.

DL clock sources may be configured to periodically transmit DL advertisements embedded in a Data DPDU's DAUX subheader. Each such advertisement provides a TAI time reference for the D-subnet. All standard-compliant DLEs that receive an advertisement are assumed to be capable of participating in time-synchronized communication with the advertising DLE. DL clock recipients with relatively imprecise clocks may have a limited capability to communicate only with routers that use slow-channel-hopping.

MAC layer authentication requires shared security keys, shared time sense, and knowledge of a neighbor's EUI64Address. This information is acquired stepwise during the join process, with security keys provided at the end. The DL uses the standard block cipher (usually AES-128), together with a well-known security key, during the join process in order to provide enhanced integrity checking (but not security). This is described in more detail in 7.4.

In slotted-channel-hopping, both transaction initiators and transaction recipients share an intrinsic sense of which timeslot is being used; otherwise, the DLEs would not be able to communicate. Every timeslot has a scheduled start time that is known by all participating DLEs.

In slow-channel-hopping, a DLE with an inaccurate sense of DL time will nominally transmit in a particular timeslot, based on its own sense of time. However, such a DLE is permitted to miss its target, and may actually transmit in any timeslot within the slow-channel-hopping period. Therefore, unicast Data DPDUs that are transmitted using a slow-channel-hopping superframe shall include an extra octet in the Data DPDU's header to indicate which timeslot within the slow-channel-hopping period was intended. This allows the receiving DLE to reconcile its timeslot sense with that of the transmitting DLE, applying time correction across timeslots to validate the accuracy of the transaction initiator's clock, and to use the timeslot's scheduled start time as security material.

For security purposes, a timeslot's scheduled start time (which is not the start time of a resulting Data DPDU) is passed to the DSC for use in DL-related security operations. See 9.1.11. In most cases, the timeslot of the Data DPDU and the timeslot of any responding ACK/NAK DPDU are the same. There is one exception, which occurs for slow-channel-hopping when the clock of one of the DLEs has drifted into a different timeslot. In that case, the channel-hopping-offset of the acknowledging DLE shall be included in the ACK/NAK DPDU's DMXHR (Table 117) to identify unambiguously the exact time that is to be used for security operations, even if the acknowledging DLE is not acting as a DL clock source for the ACK/NAK DPDU recipient. Absence (i.e., non-inclusion) of the channel-hopping-offset field in the ACK/NAK DPDU implies that the DLE originating the Data DPDU and the DLE originating the ACK/NAK DPDU are using the same timeslot.

9.1.9.3 Pairwise time synchronization

9.1.9.3.1 General

Clock updates are propagated through the D-subnet in the course of normal communications within a timeslot or slow-channel-hopping period. These building blocks are leveraged by the system manager to arrange the propagation of D-subnet time.

DL clock sources use three general mechanisms to propagate clock updates to their neighbors.

- A DL clock source originates a Data DPDU that includes an advertisement. The time is conveyed based on when the Data DPDU is transmitted.
- A DL clock source acknowledges a received Data DPDU in a timeslot. The time is conveyed by measuring when the DL clock source detects the start of the Data DPDU and echoing the result of this measurement in the ACK/NAK DPDU(s).
- A DL clock source acknowledges a Data DPDU within a slow-channel-hopping period. The process is similar to acknowledgment within a timeslot, with the addition of an octet to identify the timeslot uniquely if needed.

The standard relies on stable clocks, particularly in field routers, for reliable D-subnet operation. For DL clock stability requirements, see Table B.8.

A DL clock recipient maintains a list of valid neighboring DL clock sources. If it receives a clock update from a designated DL clock source, it uses the data to update its clock.

When a DLE discovers a D-subnet, it acquires the D-subnet's TAI time and uses that reference for subsequent communication. The DLE shall then periodically re-synchronize its internal clock to the D-subnet clock. These re-synchronization operations are incremental, based on offsets into a timeslot with a scheduled time reference known to both DL clock source and DL clock recipient.

Whenever a DLE interacts with one of its designated DL clock sources, it receives updated clock information. Clock adjustments are included in ACK/NAK DPDUs, thus conveying time information to the recipient. Clock updates are also included in the DAUX subheader of advertisement Data DPDUs originating from DL clock sources.

Thus, a DLE may receive clock updates as a by-product of routing data through a DL clock source. Alternatively, if a DLE needs more frequent clock updates, it may be configured to receive clock updates by enabling its receiver to operate at times coinciding with periodic scheduled Data DPDUs from DL clock sources that include the DAUX subheader with an advertisement.

A DLE may acquire a clock update by sending a Data DPDU with a zero-length DSDU to a DL clock source. A DL clock source shall acknowledge such a Data DPDU and then discard it.

9.1.9.3.2 Clock source acknowledges receipt of a Data DPDU within a timeslot

When a DLE transmits a unicast Data DPDU (see Figure 81) to a DL clock source, it may receive a clock update in the ACK/NAK DPDU, as shown in Figure 78.

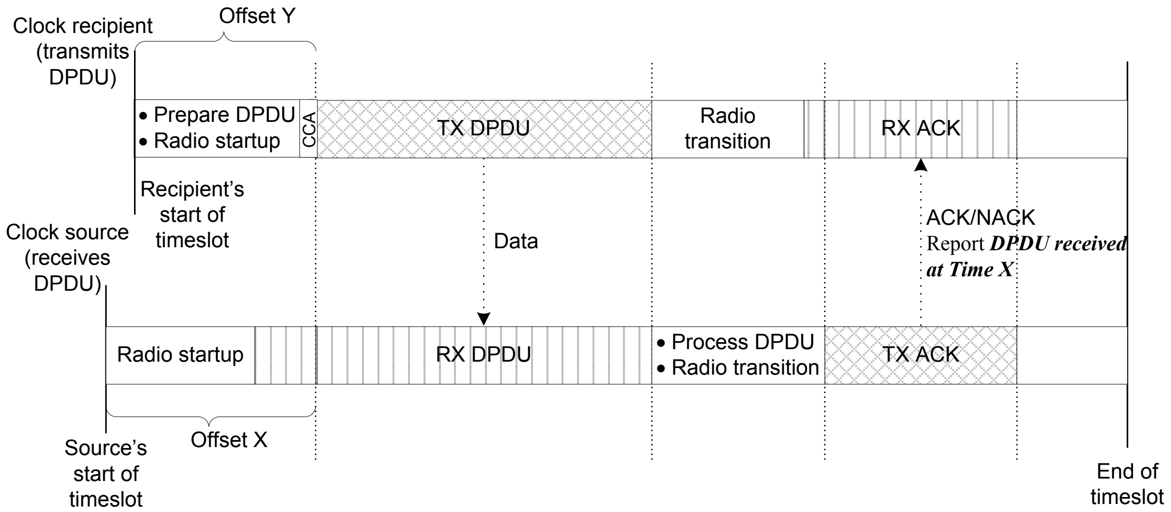


Figure 78 – Clock source acknowledges receipt of a Data DPDU

The ACK/NAK DPDU from the DL clock source includes the start time of the original transmission reported as a time offset (with microsecond precision, in units of 2^{-20} s) of the Data DPDU's start time (i.e., the start time of the PhSDU) from the scheduled timeslot start time as measured by the DL clock source.

A given unicast D-transaction occurs on a single channel, with the Data DPDU and ACK/NAK DPDU(s) all transmitted on the same channel.

This clock synchronization information is targeted at the DLE that originated the Data DPDU. While other DLEs may overhear and update their clocks based on the same information, the design is not optimized for that usage.

While a DLE's internal details of clock synchronization are not specified by this standard, the transaction is intended to support a clock synchronization process that operates generally as follows:

a) The DL clock recipient sends a Data DPDU to the DL clock source and records that it sent the Data DPDU at time offset Y.

b) The DL clock source receives the Data DPDU at virtually the same instant and records that it received the Data DPDU at time offset X.

a) The diagram shows the case where the DL clock recipient's clock is running faster than the DL clock source. In that case, X > Y. If the DL clock source is running faster, X < Y. The time of the DL clock source is assumed to be correct.

b) In the ACK/NAK DPDU, the DL clock source reports that it received the Data DPDU at time offset X.

c) The DL clock recipient applies a time correction that is computed as (Y – X).

The result shown in the Figure 78 is that the timeslots start at different times but end at the same time. An actual implementation may delay application of time corrections, such as by adjusting the clock gradually. See 9.1.9.2.2.

9.1.9.3.3 Clock source originates a Data DPDU that includes an advertisement

A DL clock source is often the originator of a Data DPDU. This is generally the case when a DL clock source transmits a scheduled advertisement that includes the TAI time. The payload of the Data DPDU may be unrelated to time synchronization; the TAI time information is contained within the DAUX subheader, so that it may be overheard by any of the DL clock source's neighbors.

Multiple DLEs may be configured to enable their receivers simultaneously in anticipation of a scheduled Data DPDU conveying an advertisement.

This standard requires that a DL clock source be capable of precisely controlling when advertisement Data DPDUs are transmitted, with a precision of ±96 μs (3 octets), referenced to the DLE's internal clock. See 9.4.3.3.1.

9.1.9.3.4 Clock source acknowledges a Data DPDU within a slow-channel-hopping period

As noted previously, DL clock sources use ACK/NAK DPDUs to report time with microsecond resolution (2^{-20} s), as offsets relative to the scheduled nominal start time of a timeslot. Identification of the timeslot is assumed to be unambiguous, known to both sender and recipients. Thus only the offset is communicated in clock updates.

However, within slow-channel-hopping periods, a DL clock recipient's internal clock may have drifted tens or hundreds of milliseconds or more relative to the sender's clock, so that the presumption of shared identification of the timeslot might be incorrect. Therefore, an extra octet, identifying a specific timeslot, is added to both Data and ACK/NAK DPDU headers within slow-channel-hopping-periods, as specified in the DPDU's DHDR. The initial timeslot within a slow-channel-hopping period is defined as having an offset of zero.

Within slow-channel-hopping-periods, DLEs with an accurate sense of time should operate within timeslot boundaries. However, DLEs without an accurate sense of time might not be capable of respecting timeslot boundaries, and might not even know which timeslot they are actually using. A timeslot offset in each DPDU header allows the receiver to reconstruct the sender's time sense and vice versa (see 9.3.3.3).

9.1.9.3.5 Auditing the quality of a neighbor's clock

When a DLE joins the D-subnet, it reports its clock stability specifications to the system management function. In their normal interactions with these DLEs, DL clock sources and DL clock recipients can be configured by the system to collect diagnostics, on a per-neighbor basis, to audit these nominal specifications (see 9.4.3.9).

9.1.9.3.6 Discontinuous clock adjustments

Under some conditions, it may be necessary for the system manager to adjust all of its DLE's clocks by tens or hundreds of milliseconds or more, for example, when two D-subnets are being joined. The system manager may schedule a discontinuous clock adjustment to occur at a particular TAI time, by setting the dlmo.TaiAdjust attribute on each of its DLEs. At the designated time, all DLEs shall adjust their clocks forward or backward by the specified amount of time.

There are some system impacts of a clock adjustment that should be considered whenever this feature is used.

- The security model in this standard does not allow time to run backward. Time is used in the security nonce, which can never be repeated in any standard communication layer. Consequently, there shall be an interruption in service, equal to the magnitude of the adjustment plus at least one timeslot, if time is adjusted to an earlier time.

- Phased reports will shift in time by the amount of the clock adjustment, unless corresponding contracts are revised in tandem with the clock adjustment.
- DLEs that do not receive the time correction before the cutover time may lose synchronization with D-subnet operation, and may need to re-discover their neighbors.

9.1.9.4 Transactions within timeslot templates

9.1.9.4.1 General

NOTE Transactions are also described in 4.6.11.

Transaction timing within a DL timeslot is specified by timeslot templates. This standard defines default timeslot templates that are needed by the DLE in its interactions with a wireless provisioning DLE. Additional timeslot templates may be added by the system manager during or following the join process. (See 9.4.3.3 for more information on timeslot templates.)

Figure 79 illustrates some aspects that are addressed by DL timeslot templates.

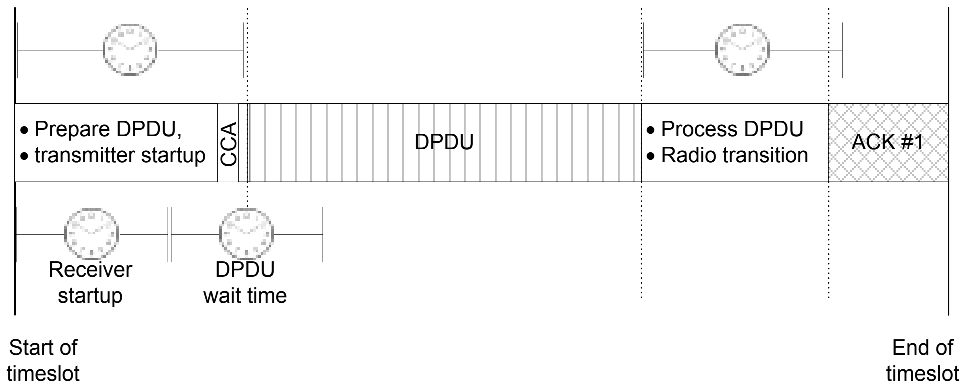


Figure 79 – Transaction timing attributes

As shown in Figure 79, timeslot templates define the timing for operations such as Data DPDU reception wait time, and the turnaround time (Data DPDU reception processing and radio transition) between receiving a Data DPDU and transmitting an ACK/NAK DPDU.

Generally, there are two types of transaction templates: transmit and receive. A transaction initiator template specifies a time range to begin Data DPDU transmission, to check the channel for activity before transmission, and the relative placement of any ACK/NAK DPDU(s) in the timeslot. A transaction receiver template specifies the interval during which a received Data DPDU can begin arriving, and thus when to timeout if no Data DPDU start is detected. It also specifies the timing of any ACK/NAK DPDUs sent by transaction responders.

Default timeslot templates cover baseline transactions needed to interact with a provisioning DLE, which are usable during the join process. Default templates may also be used by joined DLEs for general transactions. Additional templates may be provisioned into the DLE to accommodate features such as:

- carrier sense multiple access with collision avoidance (CSMA/CA) periods at the start of a timeslot (see 9.1.9.4.8); and
- extended timing of ACK/NAK DPDUs relative to the end of the Data DPDU or the end of the timeslot (see 9.1.9.4.6) .

By convention in this standard, timeslot template timing is specified based on the start and end times of both Data and ACK/NAK DPDUs and the end of the timeslot. PhPDU timing, dependent on the details of the physical layer that conveys each DPDU, can be inferred from those DPDU start and end times (see 9.4.3.3).

9.1.9.4.2 D-transaction overview

As shown in Figure 80, the DL supports both unicast and broadcast transactions.

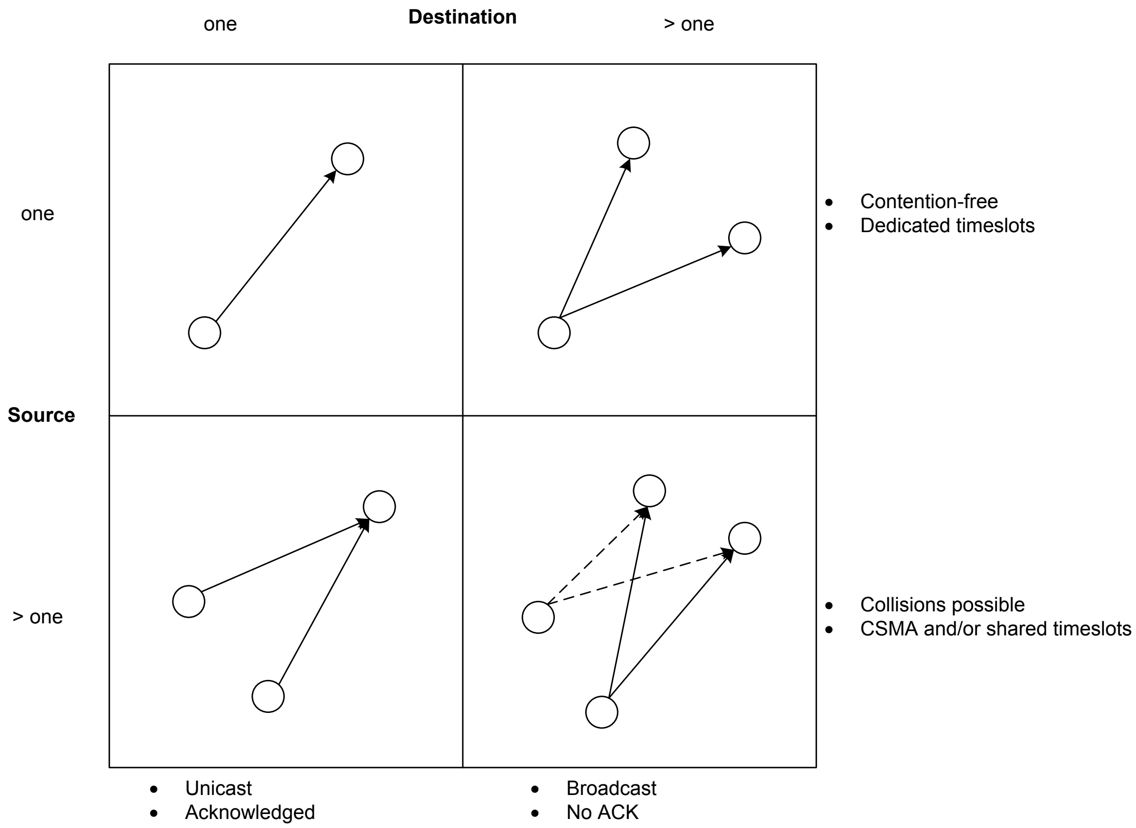


Figure 80 – Dedicated and shared transaction timeslots

Unicast transactions, indicated in the left half of Figure 80, may use dedicated timeslots, for example, when a DLE reports repetitively on a schedule. Duocast is a variant of unicast, wherein a second receiver is scheduled to overhear the Data DPDU and provides a second ACK/NAK DPDU. Duocast is shown graphically in Figure 84.

Receipt of an ACK (positive acknowledgment) DPDU by the transaction initiator indicates that the transaction recipient has successfully received the Data DPDU and that the transaction initiator should mark the transaction as complete. Unicast and duocast transactions require the transmission of ACK/NAK DPDUs in response to such receipt.

Broadcast transactions, indicated in the right half of Figure 80, may also use dedicated timeslots, such as for scheduled advertisements. Broadcast transactions cannot use a DL immediate acknowledgment (i.e., via an ACK/NAK DPDU).

As shown in the lower left quadrant of Figure 80, unicast and duocast transactions may use shared timeslots, such as within slow-hopping periods. Shared timeslots are commonly used for retries, join requests, exception reporting, and burst traffic.

As shown in the lower right quadrant of Figure 80, broadcast transactions such as solicitations may also use shared timeslots.

The term contention-free in Figure 80 is relevant only within the scope of D-subnet timeslot allocation. If two D-subnets, or DLEs within the same D-subnet, are allocating timeslots in an uncoordinated fashion, then access is not contention-free. Likewise, other users of the 2,4 GHz spectrum, such as WiFi, Bluetooth®[7], ZigBee, IEC 62591 and IEC 62601, potentially may also interfere. Improved coexistence with these uncoordinated systems is achieved by using CCA (see 9.1.9.4.3) to check the channel before transmission.

The use of broadcast in this standard is limited to these DL operations: advertisements and solicitations. Advertisements and solicitations, used for D-subnet discovery, are described in 9.1.13.

NOTE Broadcast as described here does not use broadcast MPDUs defined in IEEE 802.15.4. See 9.1.5. Broadcast and multicast, as AL services, are not supported in this version of the standard.

9.1.9.4.3 Unicast transaction

Figure 81 illustrates a unicast transaction.

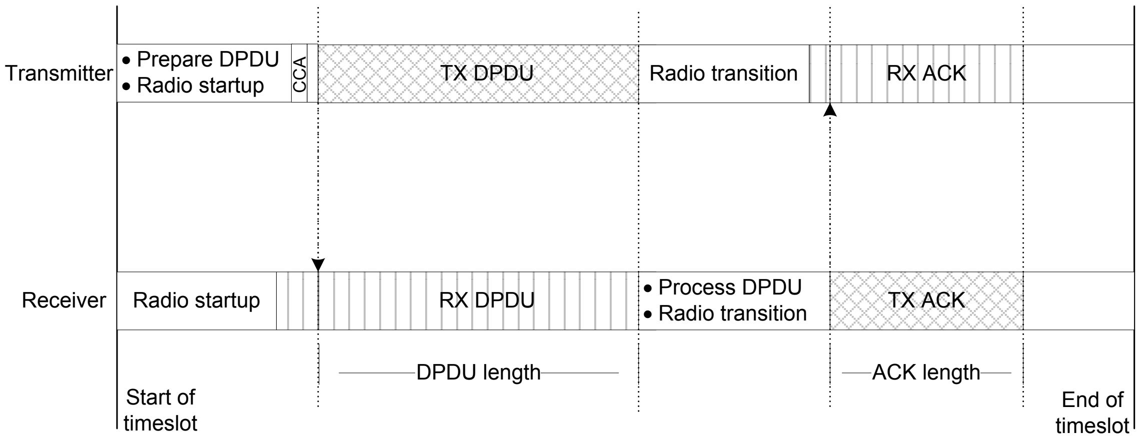


Figure 81 – Unicast transaction

Before a DLE transmits a Data DPDU in a timeslot, it prepares the Data DPDU for transmission, generally with DL security. Prior to transmission, the DLE is normally configured to perform a clear channel assessment (CCA) of the radio space.

CCA shall be implemented as described in IEEE 802.15.4:2011. That standard specifies a detection period of 8 symbols. CCA shall be performed as configured in the timeslot template field dlmo.TsTemplate[ ].CCAmode (see Table 163), where the choices, listed by their coding, are:

- CCA Mode 4 (Aloha);
- CCA Mode 1 (energy above threshold);
- CCA Mode 2 (carrier sense only);
- CCA Mode 3 (carrier sense and/or energy above threshold).

Compliance to IEEE 802.15.4:2011 requires that at least one of the CCA modes be supported. Compliance to EN 300 328 requires that CCA Mode 1 be supported. If a non-zero value for dlmo.TsTemplate[ ].CCAmode is selected and the selected mode is not supported by the DLE, the DLE may choose a different CCA mode that is supported by the DLE and permitted by the applicable regulatory regime. CCA modes supported by the DLE are indicated in dlmo.DeviceCapability.SupportedCCAmodes; see 9.4.2.23.

[7] Property of the Bluetooth Special Interest Group

IEEE 802.15.4:2011 permits CCA Mode 3 to be implemented either as the AND or the OR of CCA Mode 1 and CCA Mode 2. When CCA Mode 3 is implemented as the AND of CCA Mode 1 and CCA Mode 2, it may be used in lieu of CCA Mode 1 in regulatory regimes that require CCA Mode 1. When CCA Mode 3 is implemented as the OR of CCA Mode 1 and CCA Mode 2, it cannot be used in lieu of CCA Mode 1 in regulatory regimes that require CCA Mode 1.

If a specific CCA mode is required by regulation, but is not supported by implementation, then the DLE shall not initiate transactions.

If CCA reports a busy medium, the transmission transaction shall be aborted.

Use of CCA as defined by IEEE is not intended to exclude additional CCA checks that might be supported by advanced devices. For example, if a backbone router is capable of detecting IEEE 802.11 modulation, the DLE may reasonably leverage this capability to detect and report a busy medium.

CCA should be complete 192 μs prior to the start of the physical layer header.

If the D-transaction requires an ACK/NAK DPDU, the transmitting DLE enables its receiver after the transmission is completed, at a time specified in the TsTemplate. If an ACK DPDU is received, the transmitted Data DPDU is deleted from the DLE's transmit queue.

When a DLE is scheduled to receive a Data DPDU, it enables its PhLE's receiver at the time specified in the TsTemplate and waits for the expected PhPDU. If it detects a valid IEEE 802.15.4:2011 SHR and PHR, it continues to attempt to receive the entire PhPDU. It then processes the contained Data DPDU (including DMIC authentication) and determines if the Data DPDU requires an acknowledgment. To send an ACK/NAK DPDU in acknowledgment, a DLE that is a transaction responder enables its PhLE's transmitter and sends the ACK/NAK DPDU within the same timeslot, such that the ACK/NAK DPDU is transmitted at the time specified by the timeslot template for the primary (or a secondary, etc.) responder in that timeslot, depending on the DLE's role as primary, etc., responder.

The time window for each expected ACK/NAK DPDU is defined by the timeslot template. If there is a substantial delay between the end of the Data DPDU and the scheduled start of the expected ACK/NAK DPDU, an implementation may sensibly power down its receiver during the delay.

9.1.9.4.4 Negative acknowledgments

A transaction recipient shall respond to receipt of a unicast Data DPDU with a NAK DPDU when it cannot accept the Data DPDU at that time, but has successfully received it without other error. Time synchronization information may be included in NAK DPDUs. Similarly, a NAK DPDU ensures that RF statistics correctly log a clean transmission. A NAK DPDU can be used to exert back-pressure as a simple flow control mechanism.

The DL supports two types of NAK DPDUs: NAK0 and NAK1. A NAK0 DPDU is intended to indicate resource limitations in the router, while a NAK1 DPDU is intended to signal downstream connectivity problems in the D-subnet.

The DLE shall respond to a unicast Data DPDU with a NAK0 DPDU when it correctly receives the Data DPDU but cannot accept it due to lack of capacity in its message queue. See 9.1.8.5 for a discussion of the DLE's message queue. A DLE may also respond with a NAK0 when it is configured in excess of its forwarding capability (ForwardRate), as described in 9.4.2.23.

The DLE may respond to a unicast Data DPDU with a NAK1 DPDU to apply back pressure in the event of lost downstream connectivity. For example, when the DLE loses downstream connectivity to all of its next neighbors in a specific graph, and then receives a Data DPDU that is following the same graph, the DLE may sensibly generate an immediate response of a

NAK1 DPDU to indicate the lack of ability to forward Data DPDUs that are directed to the same graph.

When a DLE receives a NAK0 or a NAK1 DPDU from a neighbor, it shall back off by not transmitting more Data DPDUs to that neighbor for a period of dlmo.NackBackoffDur. This backoff delay does not include delay of Data DPDUs without a payload, which allows the DLE to poll a neighbor that is a DL clock source for a time update even though the neighbor is not accepting Data DPDUs at that time.

As described in 9.1.9.2.2, if a DL clock repeater's clock has expired and it is polled for a time update, it should respond with a NAK1.

9.1.9.4.5 Explicit congestion notification

The standard supports explicit congestion notification (ECN) as described in IETF RFC 3168. ECN provides a mechanism for a router to affect AL flow control.

As described in 12.12.4.2.3, there is a limited number of data source requests that can be simultaneously awaiting response from a data sink. Flow control at the data source operates by incrementally increasing the limit on outstanding requests, based on receipt of timely data sink acknowledgments.

ECN provides a mechanism whereby flow control can be effective without driving the D-subnet to the point of failure. Any router along the path from data source to data sink may set ECN to indicate that the router is nearing its capacity. When the data sink receives the ECN, it echoes it back to the data source, which then accounts for the ECN in its flow control logic.

An ECN indicator in the Data DPDU header may be set by any field router that is experiencing congestion, following the guidelines in IETF RFC 3168, as a signal to a data source that it should apply flow control to reduce its use of D-subnet resources. This ECN indicator is propagated to the data sink, such as a gateway, and eventually works its way back to the data source through a TL acknowledgment.

In addition, the DL provides a special type of acknowledgment called ACK/ECN. A DLE receiving an ACK/ECN treats it as a normal DL acknowledgment. In addition, the DLE may treat the ACK/ECN as an early indication that the data sink's acknowledgment will include an ECN.

Use of the ACK/ECN should be limited to Data DPDUs with a priority of seven (7) or less, corresponding to best effort queued and real time sequential flows.

9.1.9.4.6 Data DPDU wait times

The clock times of transmitting and receiving DLEs are rarely in perfect synchronization. Therefore, a transmitting DLE is unlikely to transmit a PDU (protocol data unit) at exactly the time that a receiving DLE expects it. If a PDU is transmitted too early, the receiver might not yet be enabled. If a PDU is transmitted too late, the receiving DLE may have disabled its radio in order to save energy. PDU wait time (PWT) is the time period when the receiver is expected to listen for incoming PDUs. A particular degree of timing accuracy between the DLEs is implicit in the system manager's selection of PWT.

DLEs compliant with this standard accommodate configurable PWTs and configurable timeslot durations. PWT is not directly specified in this standard, but can be inferred within timeslot templates from the earliest and latest times that PDU reception begins. See Table 161.

NOTE 1 This tutorial does not make a distinction between PhPDU and DPDU timings. As specified in 9.4.3.3, the DL follows a convention of specifying timeslot timing in reference to the DPDU (PhSDU), and not to the PhPDU. In implementations based on IEEE 802.15.4:2011 (2,4 GHz), PhPDU header detection involves receipt of a preamble, start frame delimiter (SFD), and frame length, for a total PhPDU header of 192 μs (6 octets) before the DPDU begins. Similarly, radio transmission of a PhPDU begins 192 μs prior to the nominal DPDU start time.

PDU wait times in a unicast PDU are illustrated in Figure 82. The same principles apply to other types of PDUs.

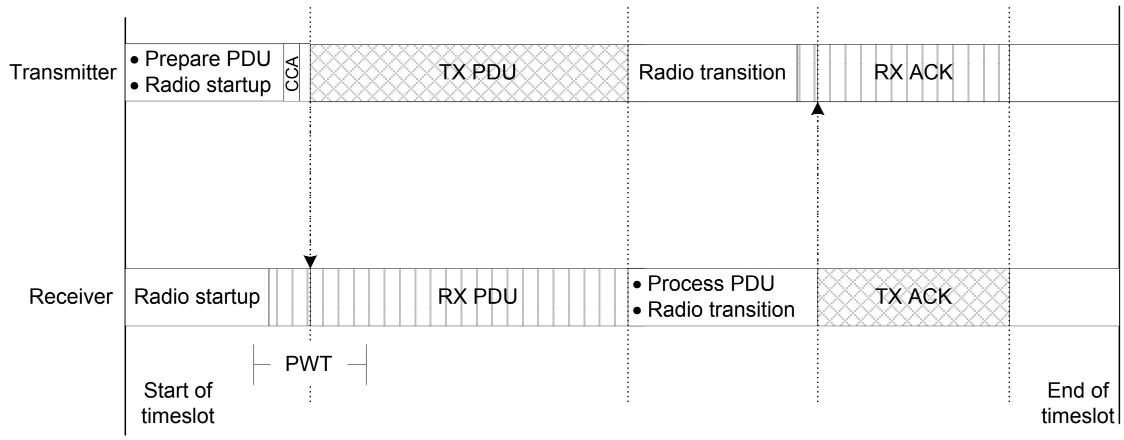


Figure 82 – PDU wait time (PWT)

The duration of the PWT is configured by the system manager accounting for the intrinsic stability of transmitting and receiving DLE's clocks, and the guaranteed maximum time between clock updates.

For example, if transmitting DLEs are accurate to ±1 ms relative to the receiver within the clock expiration period (dlmo.ClockExpire), the receiver's PWT should be 2 ms long. A less accurate transmitting DLE will require a longer receiver PWT or a shorter clock expiration period. (In this example, the radio's listening time should be slightly longer than the 2 ms PWT, to account for the PhPDU's SHR and PHR durations.)

If the receiver does not begin receiving the expected PDU by the end of its PWT, the receiver is permitted to disable its radio for the duration of the timeslot.

Timeslot durations of 10 ms have a timing budget that can allocate about 2 ms to PWT. For a longer PWT, either other allocations have to be adjusted or the timeslot duration has to be increased. For example, if the D-subnet supports DLEs that sleep for up to two minutes, timing errors of about ±2 ms may accumulate between reports. This can be accommodated by increasing the PWT from (for example) 2 ms to 4 ms, with a corresponding increase in timeslot duration and receiver energy consumption.

NOTE 2 DLEs with infrequent reporting intervals are capable of being configured to check the D-subnet periodically for receivable Data DPDUs. These DLEs are capable of receiving clock corrections through the DAUX subheader as part of the same process.

9.1.9.4.7 Duocast/N-cast transactions

Duocast/N-cast is a variant of unicast, wherein one or more additional receivers are scheduled to overhear the Data DPDU and provide additional acknowledgments. Duocast/N-cast provides support for latency-controlled access to a backbone with an increased probability of first-try success. Duocast/ N-cast transactions are intended primarily for DLEs with links to two or more infrastructure DLEs, particularly to backbone routers, as shown in Figure 83.

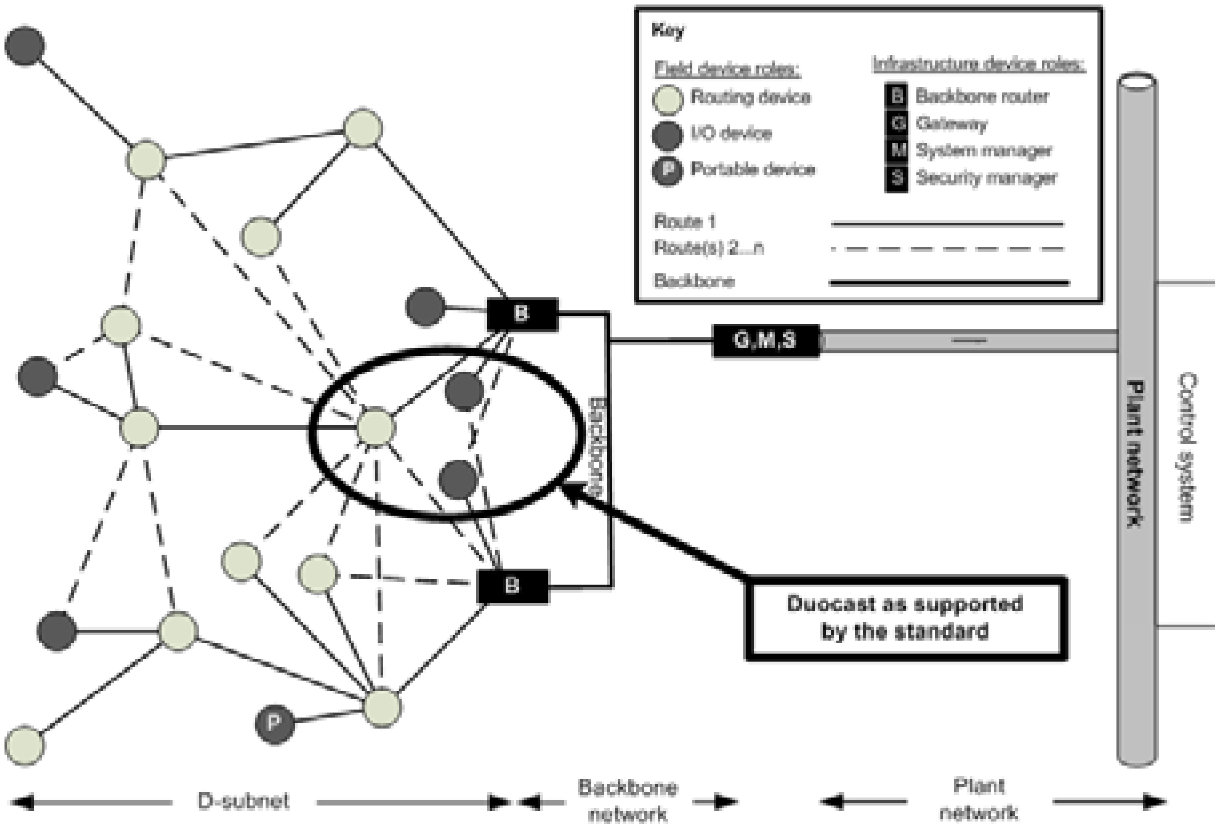


Figure 83 – Duocast support in the standard

DLEs receiving the duocast/N-cast acknowledgments, circled in Figure 83, need to be configured with timeslot templates with an extended listening window for the additional acknowledgment(s). DLEs sending the secondary duocast/N-cast acknowledgments (e.g., one of those gray DLEs circled in Figure 83) need to be configured individually with timeslot templates with appropriately delayed/deconflicted acknowledgment windows for their individual acknowledgments, and with the address of the primary intended recipient to enable them to respond only to the expected Data DPDU. Duocast/N-cast support usually involves increased timeslot duration of approximately 1 ms to 2 ms per secondary receiver, as configured by the system manager.

NOTE 1 Coordination of the duocast/N-cast response often involves back-channel coordination between the responding infrastructure DLEs (since such responders usually are backbone routers that are also connected to a higher-throughput backbone), but coordination via standard configuration messaging is also possible, depending on the design of the infrastructure DLEs.

NOTE 2 If the probability of success of a single acknowledged-unicast transaction is p, the probability of failure for such a unicast transaction is $(1-p)$. For the corresponding duocast transaction it is typically $(1-p)^2$, while in the general case for an N-cast transaction it is $(1-p)^N$. Thus when p = 95%, the probability of failure for a unicast transaction is 5%, for a duocast transaction it is 0,25%, for a 3-cast transaction it is 0,0125%, etc. Thus, in most relatively planar environments duocast suffices, though in highly metallic and obstructed three-dimensional environments such as offshore oil platforms 3-cast (“triocast”) may be worth consideration.

NOTE 3 In many cases duocast is for contracts with a small maximum APDU size, such that the transaction-initiating Data DPDU and both acknowledging ACK/NAK DPDU subslots would require no greater duration than a maximal-payload unicast transaction with a single-acknowledgment Data DPDU and its subsequent single acknowledging ACK/NAK DPDU. In that case a single timeslot duration is usable for both unicast and restricted duocast transactions, albeit with different transaction templates.

Duocast/N-cast timeslots may be scheduled in conjunction with available digital bandwidth for a fast retry on an alternate channel, as shown in Figure 66.

Figure 84 illustrates a transaction involving duocast transmission and reception.

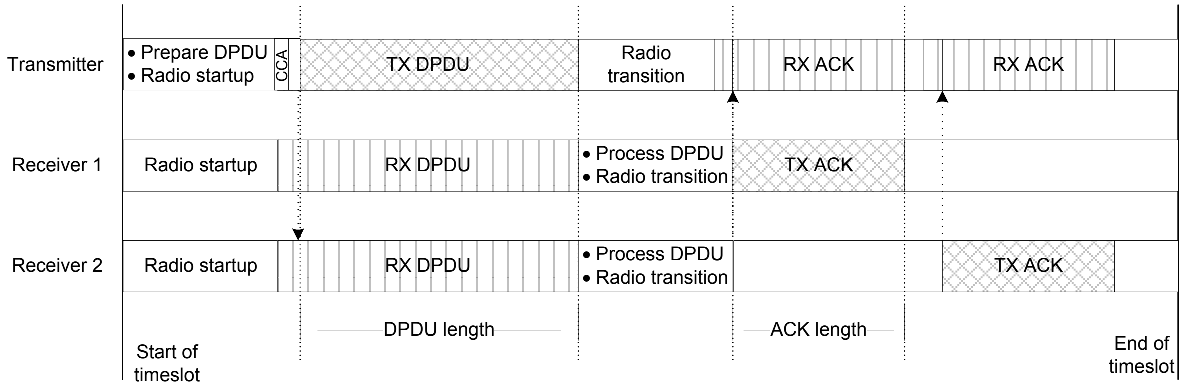


Figure 84 – Duocast transaction

In the example in Figure 84, the Data DPDU is addressed to receiver 1, the primary recipient, and is overheard by receiver 2, a secondary recipient. For duocast/N-cast transactions, the destination address of the Data DPDU is set to that of the primary recipient (receiver 1 in Figure 84), and an acknowledgment is expected from at least one recipient during the same timeslot. As illustrated in Figure 84, the primary recipient transmits an ACK/NAK DPDU upon receipt of the Data DPDU. Secondary recipients also transmit an ACK/NAK DPDU, but after an additional recipient-dependent delay to allow time for any preceding ACKNAK DPDUs to complete.

If an ACK DPDU is received from any acknowledging recipient, the transaction is complete and the Data DPDU is deleted from the transaction-originator's DLE's message queue.

If an ACK DPDU is received from a recipient, the DLE that originated the transaction is not required to expend energy receiving and processing any subsequent ACK/NAK DPDU(s) in that transaction. However, the DLE that originated the transaction should periodically verify that it is able to receive an ACK/NAK DPDU from each expected acknowledging recipient, to confirm the continuing availability of the secondary receiver(s).

As noted in 9.1.5 and described in 9.3.4, a duocast/N-cast acknowledgment from a secondary recipient includes the acknowledging DLE's own address field in the source address field of the MHR. This enables the transaction initiator to identify the acknowledgment's source and occasionally to report the same to the system/security manager to facilitate detection of cyber-attacks.

9.1.9.4.8 Shared timeslots with CSMA/CA

Unicast transactions may occur in timeslots that are dedicated to a specific link. Alternatively, shared timeslots may be designated to provide bandwidth on demand to a collection of DLEs. Shared timeslots are usually configured to transmit only near the start of the timeslot.

Timeslot templates specify transmission time as a range as per 9.4.3.3. At a minimum, the transmit time shall be configured to a range of at least 192 μs, thereby allowing for ±96 μs (±6 PHY symbol periods) of jitter that is permissible in a transaction initiator. If the transmission time range is configured to be larger than 200 μs, the DLE shall select a randomized time within that range to begin its transmission, making reasonable accommodation for the DLE's actual transmission jitter characteristics.

Figure 85 illustrates the use of a shared timeslot with active CSMA/CA.

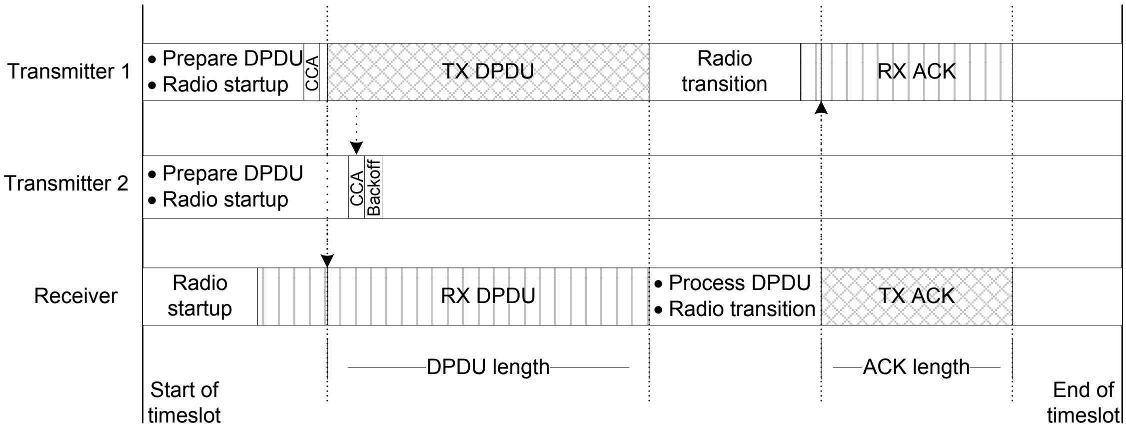


Figure 85 – Shared timeslots with active CSMA/CA

In the example in Figure 85, two DLEs are contending for use of the channel in a shared timeslot. This approach is used for all shared slots, configurable to be used in various ways.

Priorities within a shared timeslot may be managed by configuring DLEs with different timeslot templates. A DLE with a high-priority Data DPDU, such as a retry for a failed duocast transaction, may be configured to transmit its Data DPDU as early as possible within the timeslot. A DLE with less critical requirements may be configured to delay its transmission to slightly later in the timeslot, such as 2 ms later. If another DLE has already claimed the timeslot, as shown in Figure 85, the CCA of the delayed DLE might (or might not) detect that the channel is in use and consequently defer its transmission to another timeslot.

Use of active CSMA/CA within shared timeslots may involve configurations with longer timeslots and longer Data DPDU wait times, and use of more receiver energy.

9.1.9.4.9 Transactions during slow-channel-hopping periods

Some DLEs do not have a sufficiently stable time base to communicate at their normal messaging rate within short timeslots. These DLEs may need to use slow-hopping periods for their communication. Slow-channel-hopping periods are simply a set of concatenated timeslots on the same channel, wherein the receiver runs its radio continuously and the transaction initiator is not required to respect timeslot boundaries within the slow-hopping period.

As shown in Figure 86, a transaction during a slow-channel-hopping period is very similar to a unicast transaction in a shared timeslot, except that Data DPDU transmission can occur anywhere within the slow-channel-hopping period. Transmitting DLEs target the beginning of a specific timeslot within a slow-channel-hopping period, based on the transaction initiator's own sense of time, which is not required to be very well synchronized with that of the intended receiver(s). Transmitting DLEs use CCA to check the channel before transmission. In the absence of higher priority operations (such as forwarding of Data DPDUs), a receiver hosting the slow-channel-hopping period runs its radio receiver continuously except when responding to Data DPDUs that it receives.

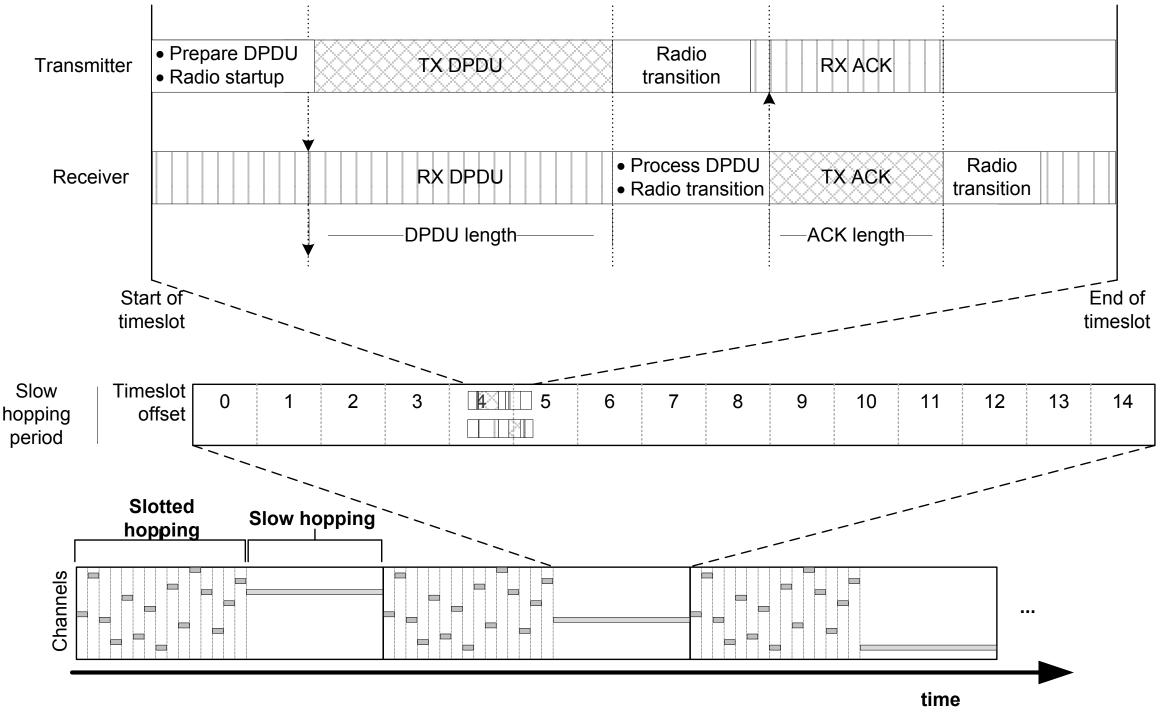


Figure 86 – Transaction during slow-channel-hopping periods

DLEs can target any timeslot within a slow-channel-hopping period, with one major caveat: scheduled Data DPDU timeslot time is required to increase with each transaction.

DLEs should respect timeslot boundaries within slow-channel-hopping periods to the best of their ability. If all DLEs within slow-channel-hopping periods are well behaved, with well-synchronized clocks, the resulting performance is approximately comparable to that of the slotted Aloha protocol.

A DLE whose time sense differs from that of its intended receiver(s) will nominally transmit near the start of a particular timeslot, based on its own sense of time. However, such a DLE may actually initiate transmission in any phase of any timeslot within the slow-channel-hopping period.

For example, a DLE that has a clock source with a stability of $\pm 50 \times 10^{-6}$ over a few-minute interval, due to uncompensated environmental fluctuations, may sleep for 3 minutes between transactions, corresponding to a clock drift of about ±9 ms. A DLE of this type might wait for a scheduled advertisement to get itself resynchronized prior to transmission. Alternatively, it might transmit during a slow-channel-hopping period (if available) and receive time synchronization in the acknowledgment. Continuing with this example, a DLE with ±9 ms accuracy would select one of the slots within the slow-channel-hopping period, and transmit using the appropriate link template. The DLE would nominally be transmitting in a particular timeslot, based on the DLE's own sense of time, but may actually be transmitting in a different timeslot.

In this example, the DLE should not attempt to transmit in the first or last timeslot in the slow-channel-hopping period, because it may actually transmit outside of the available time range. It is the DLE's responsibility to avoid selecting timeslots that are inconsistent with the DLE's time-keeping capabilities, accounting for the DLE's own uncalibrated clock drift in combination with the 10×10^{-6} clock drift allowed for a neighboring router or the 100×10^{-6} clock drift allowed for a neighboring non-routing field device.

All Data DPDUs in slow-channel-hopping periods shall include an extra octet in the Data DPDU header to indicate which timeslot offset within the slow-channel-hopping period was intended. This enables the receiving DLE to reconstruct timeslot information from the transmitting DLE, to apply time correction across timeslots, to validate the accuracy of the transaction initiator's clock, and to use the scheduled timeslot start time as security material for message authentication. (This is specified in the Data DPDU MAC subheaders, DMXHR; see 9.3.3.4.)

If necessary, a corrected timeslot offset is provided in the acknowledgment (see 9.3.4). Unambiguous shared timeslot identification is needed for both the transaction initiator and transaction receivers to authenticate the Data DPDU and to resynchronize time. Even if the transmitting DLE has a highly accurate sense of time, the receiving DLE(s) might not; therefore the channel-hopping-offset octet is required for all Data DPDUs using a slow-channel-hopping superframe.

The initial timeslot within a slow-channel-hopping period is defined as having an offset of zero.

9.1.10 D-subnet addressing

9.1.10.1 Address types

DL16Addresses shall always be used within a D-subnet, except that EUI64Addresses are used in a limited way during the initial phases of the join process to communicate with the joining device.

Every DLE in a D-subnet is identified in three ways:

- Each D-subnet DLE has an EUI64Address identifier that is presumed to be unique.
- Each DLE compliant with this standard shall be assigned at least one IPv6Address when it joins the D-subnet. However, within the DL, only the DL16Address alias for this IPv6Address (described next) shall be used.
- Each DLE or foreign device that is accessible through a D-subnet has a D-subnet-unique DL16Address, which is an alias for its IPv6Address. The scope of any DL16Address shall be limited to a particular D-subnet.

The EUI64Address shall be used as a new DLE's address for immediate neighbor addressing prior to and during the join process. Once a DLE has joined the D-subnet and received its IPv6Address, it shall be addressed by either its IPv6Address or a D-subnet-local DL16Address alias of that IPv6Address.

The EUI64Address is also used in each DPDU security nonce. Whenever a DL16Address is used in a Data DPDU or an ACK/NAK DPDU, the DPDU's recipient(s) need(s) a-priori knowledge of the corresponding EUI64Address. This neighbor information is provided by the system manager as part of the link establishment process. An exception is made for a new DLE that is communicating with a neighbor that has advertised its DL16Address. In that case, the DLE of the joining DLE shall acquire the EUI64Address of the advertising DLE by initiating a transaction with that neighbor, sending it a Data DPDU with a null payload while requesting the EUI64Address in the replying ACK/NAK DPDU, as described in 9.3.3.3. For that bootstrapping transaction, the applicable D-key is K_global, used with a DMIC-32, which is the same as for other Data DPDUs involving an EUI64Address. (See 9.1.11 for discussion of DL security.)

When a DL16Address refers to a DLE within a D-subnet, the DL16Address is always the 16-bit MAC address of the DLE. When a Data DPDU contains a DL16Address that refers to a device not within the scope of the D-subnet, the Data DPDU is routed to a DLE that is serving as a backbone router, which maps the logical DL16Address to its IPv6Address counterpart.

Most Data DPDUs include two source DL16Address and two destination DL16Address . One pair of source/destination DL16Address is for next-hop addressing at the MAC sublayer. A second pair of source/destination DL16Addresses (actually D-aliases) are found within the Data DPDU's DADDR subheader, where they specify the D-subnet-local ultimate source and destination NLEs via their D-aliases.

Routing is usually specified by graphs, not by D-addresses. However, when source routing is used within a D-subnet, DL16Addresses are normally used. Again, the one exception is that EUI64Addresses are used during the join process for communication with an immediate neighbor within the D-subnet.

9.1.10.2 Subnet identifier and uniqueness of DL16Addresses

The scope of a DL16Address is a D-subnet. A neighboring D-subnet may use the same DL16Address to reference a different DLE. Different D-subnets may use different DL16Addresses to refer to the same backbone device.

Each D-subnet has a 16-bit D-subnet identifier (dlmo.SubnetID; see 9.4.2.1), which has the same value as the PAN ID found in the IEEE 802.15.4:2011 MPDU header. Within the scope of a network, each D-subnet shall have a unique dlmo.SubnetID. However, different networks are not necessarily coordinated, so dlmo.SubnetIDs across co-located standard-compliant networks are not guaranteed to be unique. Thus, it is possible, though unlikely, that a DPDU from a different standard-compliant D-subnet will be received with what appears to be a valid DL16Address and PAN ID. The DLE relies on the DSC to discard such DPDUs on receipt due to DMIC mismatch or DMIC non-authentication, where that mismatch occurs due to non-identical D-security symmetric keys.

SubnetID=0x0000 and SubnetID=0xFFFF shall not be used as D-subnet IDs in this standard. SubnetID=0x0001 is reserved for provisioning D-subnets (see 13.1) and shall not otherwise be used.

9.1.11 DL management service

9.1.11.1 General

Management messages to a DLE are fully secured at the AL, using the end-to-end relationship between the system/security manager and the DLE's DMAP.

The DL's MAC, based on IEEE 802.15.4:2011, is not accessed directly by the DMAP; instead it is configured indirectly through the DMSAP. This isolates the rest of this standard from evolutionary changes to IEEE specifications, facilitates future adoption of alternative physical layer specifications (e.g., radios) that may have alternative or enhanced associated MACs, and enables some MAC and PHY operational aspects (such as CCA) to be used or not on a timeslot-by-timeslot basis.

As shown in Figure 87, DL management commands generally flow through the full communication protocol stack defined by this standard.

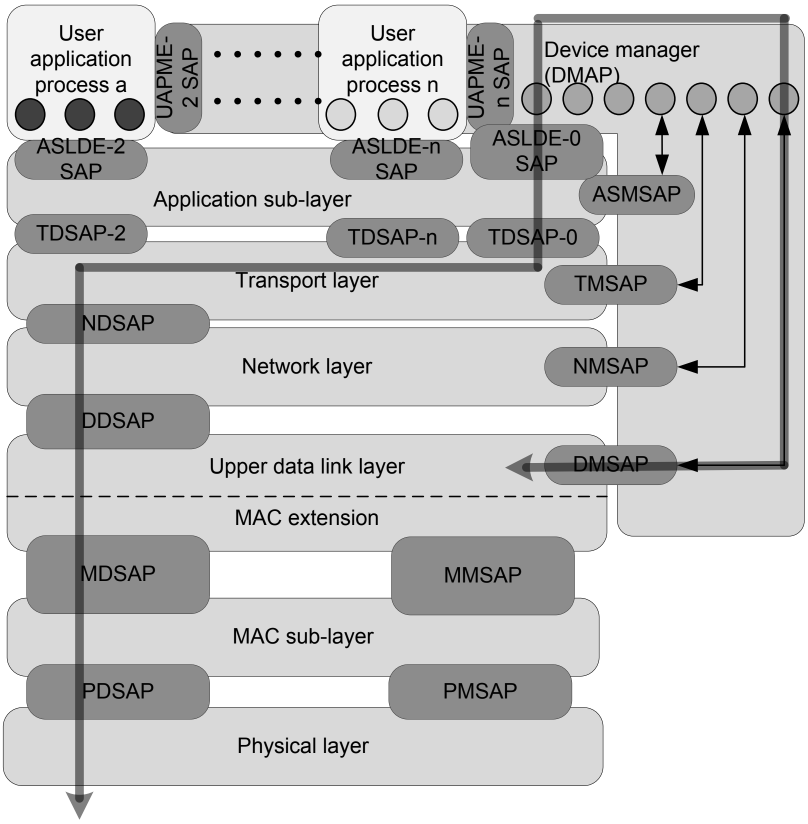


Figure 87 – DL management SAP flow through standard protocol suite

A DLE is configured via its DMAP through its DMSAP. Most management SAPs are generic, simply reading and setting data structures within the layer. Those data structures generally define how a DLE operates its state machine. The DMAP communicates with the system manager through the application sublayer, using end-to-end security.

For information on the general handling of standard management objects, see 6.2.5 and 6.2.6.

9.1.11.2 Management attributes and indexed attributes

DMSAPs involve the manipulation of the DL management object (DLMO). The DLMO includes a variety of attributes that are used to configure the DLE and/or report its status.

Some DLMO attributes apply to specific values. For example, attribute dlmo.SubnetID provides the subnet-ID for the D-subnet that the DLE has joined.

Some DLMO attributes can be visualized as tables with a collection of indexed rows. Each such attribute is specified as an indexed OctetString. For example, the DLE includes the attribute dlmo.Neighbor, which is an indexed OctetString attribute containing a collection of neighboring DLEs. Each entry of the dlmo.Neighbor attribute includes a set of fields for that neighbor, such as an indicator of whether that neighbor is a DL clock source. Each entry of the dlmo.Neighbor table is uniquely identified by the neighbor’s DL16Address. Indexed OctetString attributes in the DLMO include:

- dlmo.Ch: channel-hopping patterns;
- dlmo.TsTemplate: timeslot templates;
- dlmo.Neighbor: DLE neighbors;
- dlmo.NeighborDiag: diagnostics for DLE neighbors;

- dlmo.Graph: graphs for routing;
- dlmo.Superframe: superframes specifying common attributes for associated links;
- dlmo.Link: links, each of which is associated with a superframe;
- dlmo.Route: routes usable in DROUT subheaders.

Relationships among these attributes are described in 9.4.3.1.

DLMO attributes shall be maintained through the DMAP using methods that are described in Clause 9 and Clause 12. The ASLE services Read and Write described in Table 271 provide a general framework for reading and writing DLMO attributes. Methods for writing attributes that are to become active at a TAI cutover time, as well as methods for reading and writing indexed OctetString attributes, are described in 9.5.

The format of the DLMO attributes is defined in the DLE specification. When these objects are embedded within over-the-air messages, the specified formats shall be used. This standard does not (and cannot) constrain how data is stored within a DLE, or define how corresponding information is relayed internal to a DLE.

9.1.11.3 Management messages from immediate neighbors

Any Data DPDU may include a DAUX subheader, for example carrying advertisement information from an immediate neighbor. The DAUX subheader information is not propagated to higher layers of the communication protocol suite. In most cases, the content of the DAUX subheader is intended for the recipients DLE’s management process, which in turn may use this information to configure the DLE state machine. For example, the DAUX subheader may include superframe definitions that are intended to be used by the DLE as a starting point in the join process and/or for neighbor discovery.

The DAUX subheader is modeled as being instantly visible to the DMAP and immediately acted on if necessary.

NOTE Since standards apply only to externally-visible aspects, the internal DMSAP interface is not subject to standardization.

9.1.11.4 Multiple D-subnets

DL management objects support only one active D-subnet at a time. All DL16Addresses shall be unique within the scope of that single D-subnet. This constraint does not prevent a DLE from participating in multiple D-subnets simultaneously. Multiple D-subnets might be modeled as multiple instances of a DLE, but such operation is not specified by this standard.

If dlmo.SubnetID=0, the DLE is not yet participating in a D-subnet.

9.1.11.5 Multiple PhLEs (radios)

Each DSAP of a DLE is assumed to be associated with only one PhLE (radio). This does not preclude implementations with multiple radios, which might be modeled as multiple instances of a DLE. Such operation is not specified by this standard.

9.1.12 Relationship between DLE and DSC

The relationship between the DLE and the DSC is described in 7.3.2. Careful review of 7.3.2 is essential for anyone who wishes to fully understand DLE operation.

Generally, the DLE relies on the DSC for authentication, integrity and conditional confidentiality. All DPDUs include a DMIC that unambiguously validates that an MPDU originates from a DLE that shared the same D-subnet key, vs. those that implement another protocol using similar modulation, coding and channels. The DMIC uses a non-secret security key during the join process but then is usually configured to use a shared-secret key once the DLE is joined.

ACK/NAK DPDUs are expanded by the DLE, by inserting the Data DPDU’s DMIC into the ACK/NAK DPDU’s header, as a virtual field before being processed by the DSC. This virtual field, which is not transmitted, is included in the ACK/NAK DPDU’s DMIC calculation. Thus the Data DPDU’s DMIC is essentially echoed in the ACK/NAK DPDU without actually transmitting the field. In this manner, the ACK/NAK DPDU is unambiguously bound to the corresponding Data DPDU of the D-transaction.

9.1.13 DLE neighbor discovery

9.1.13.1 General

Wireless D-subnets compliant with this standard are detected by the DLE. A new DLE may hear advertisements from neighboring DLEs that have already joined a D-subnet of interest. This is called neighbor discovery. Following neighbor discovery, the DLE can join the D-subnet as described in 7.4.

The DL advertisement and neighbor discovery processes are the building blocks that enable a DLE to learn the DL16Address and EUI64Address of a neighboring router, and the set of scheduled links that have been allocated for use when joining. That information is then used by the DLE and DME to join the D-subnet.

DLEs discover D-subnets through advertisement Data DPDUs received from one or more advertising DLEs that periodically announce their presence. An advertising DLE is capable of acting as a proxy in the provisioning or joining process. An advertisement contains information that enables a new DLE to send a join request to the advertising DLE, for relay to a system manager, and some time later to receive a join response.

After joining a D-subnet, the DLE receives advertisement Data DPDUs from neighboring DLEs in the D-subnet and builds a local list of candidate neighbors with which it may have reasonable quality communications. This list of candidates is reported to a system manager through the attribute dlmo.Candidates. The system manager uses this information to determine how the DLE fits into the D-subnet topology. In turn that information is used to establish communication relationships between the new DLE and its neighbors.

Advertising DLEs may be discovered using passive scanning, active scanning, or a combination of passive and active scanning.

In passive scanning, the DLE periodically listens for advertisements on a series of channels. Generally, a battery-powered passive scanning DLE will listen frequently when first powered on. If a D-subnet is not discovered quickly, the DLE may extend its battery life by scanning less frequently and/or by allocating shorter time intervals for each scan. Such reductions can result in substantial delays in D-subnet joining and/or D-subnet formation, so are used only after the rapid join strategy has failed.

Active scanning overcomes some disadvantages of passive scanning. DLEs that are configured for active scanning will search for a D-subnet by periodically transmitting solicitation Data DPDUs, which trigger advertisement Data DPDUs from neighboring routers in response. The DLE transmitting the solicitation Data DPDU is called an active scanning interrogator, while the responding DLE is called an active scanning host. Active scanning hosts expend energy operating their radio receivers while listening for solicitation Data DPDUs. Some active scanning hosts have energy available for continuous receiver operation. Active scanning hosts with more limited energy sources may be configured to listen continuously for certain periods of time, such as during D-subnet formation.

Link schedules used for joining are not necessarily related to superframe schedules used for normal D-subnet operation. Thus, little information about D-subnet operation needs to be conveyed in the advertisements.

9.1.13.2 Auxiliary subheader and advertisements

DL advertisements and solicitations are conveyed in a Data DPDU's auxiliary subheader (DAUX) within the DPDU header (DHR). See 9.3.1 for an overview of the DHR. A Data DPDU that conveys a solicitation is known as a solicitation DPDU. Similarly, a Data DPDU that conveys an advertisement is known as an advertisement DPDU.

The DAUX subheader is usually absent from a DHR, but shall be included in any DHR if so configured for a particular link. A Data DPDU containing a DAUX subheader may also carry a higher layer payload that is unrelated to the neighbor discovery function.

Transmission of an advertisement is triggered by an advertisement flag as configured within a link definition. The advertisement flag indicates that an advertisement shall be transmitted in a superframe's timeslot, in the absence of a higher priority link.

The same link definition may also include a transmission flag, in which case the DLE shall check the message queue for matching outbound Data DPDUs. Thus, the Data DPDU may simultaneously carry:

- an advertisement; and
- a DSDU payload that is entirely unrelated to the advertisement.

The payload capacity of the Data DPDU is reduced when an advertisement is embedded within the DHR. Some Data DPDUs on the message queue, particularly messages that have been fragmented at the NL, may be too long to be combined with an advertisement. Such messages are not candidates for links that are shared with an advertisement, effectively giving the advertisement priority access to those timeslots.

NOTE When messages are fragmented by the NL, the fragment size is set by the NL before being passed to the DLE, with fragment size being configured by the system manager. In configurations where advertisements are infrequently combined with transmit links, the maximum fragment size often is limited by the Data DPDU payload capacity without considering the combined advertisement.

In general, Data DPDUs containing an advertisement use a DMIC based on the D-subnet's security key, thereby providing an advertisement that can be trusted by DLEs after they have joined the D-subnet. This DMIC cannot be validated prior to joining, because an unjoined DLE does not yet have the D-subnet security key. Therefore, an unjoined DLE that is scanning for a D-subnet is permitted to process an advertisement in a DAUX subheader even if it is unable to authenticate the Data DPDU containing the DAUX.

Without the benefit of a DMIC, an unjoined DLE receiving an advertisement can still use the IEEE 802.15.4:2011 FCS as an integrity check. However, the IEEE 802.15.4:2011 FCS alone does not filter MPDUs from other systems that use IEEE 802.15.4:2011. Therefore this standard provides an additional integrity check, not involving a security key, specifically covering the advertisement subheader, as specified in 9.3.5.2.4.3.

9.1.13.3 Active scanning solicitation and response

In active scanning a new DLE, acting as an active scanning interrogator, periodically solicits advertisements from active scanning hosts that happen to be in radio range. A DLE receiving the solicitation can be configured to respond with an advertisement.

Active scanning is intended for configurations in which an advertising DLE, in its capacity as an active scanning host, is able to operate its receiver more or less continuously in a slow-channel-hopping configuration. Active scanning hosts may be continuously powered.

Alternatively they may be energy-constrained DLEs that run their receivers in a slow-channel-hopping configuration for limited periods of time, such as during D-subnet formation.

The solicitation request is encoded in the DAUX subheader of the solicitation Data DPDU. A solicitation Data DPDU shall not contain an NL payload.

A solicitation Data DPDU, when received by an active scanning host DLE, causes that DLE to transmit an advertisement Data DPDU in the next timeslot if the DLE is so configured. A router receiving a solicitation Data DPDU shall respond by transmitting an advertisement Data DPDU in the next full timeslot if, and only if:

- the default receive link for scanning (Table 165) applies in the next timeslot and occurs on the same radio channel as the solicitation; and
- the DLE attribute dlmo.ActScanHostFract is configured for response to solicitation Data DPDUs. Dlmo.ActScanHostFract indicates the fraction of time that the DLE should respond when it receives an active scanning solicitation, where
  - a value of 0 indicates that the DLE is not configured as an active scanning host and that will not respond to solicitations,
  - a value of 255 indicates that the DLE should always respond to solicitations, and
  - a value in the range 1..254 indicates that the DLE makes a uniformly-random selection from the range 1..255 each time it receives a solicitation, and does not respond with a solicitation if the result is greater than dlmo.ActScanHostFract, thus generating a solicitation response Data DPDU with probability dlmo.ActScanHostFract / 255; and
- the DLE is configured to respond to the D-subnet ID included in the advertisement Data DPDU, as described in 9.4.2.20. A solicitation Data DPDU may be configured to include a D-subnet ID to limit respondents to a desired set of D-subnets.

The next full timeslot, in this context, shall be defined as the next timeslot that starts following the end of the solicitation's PhPDU plus 1 ms . Within that next timeslot, the advertisement Data DPDU shall be transmitted using timing as defined in the default transaction initiator template in Table 166, even if that default template is overwritten during DLE configuration.

An active scanning interrogator, which is presumably not synchronized with D-subnet timing, should enable its radio to receive an advertisement Data DPDU starting as early as 3 212 μs following the end of the solicitation Data DPDU, which is 1 ms plus the 2 212 μs from Table 166. Following that time, the active scanning interrogator should keep its receiver enabled long enough to receive an advertisement Data DPDU beginning at any time during a full timeslot duration. The active scanning interrogator should use its own timeslot duration as the assumed timeslot duration of the active scanning host. Therefore, when solicitation Data DPDUs are used, the active scanning interrogator should be configured with a timeslot duration that matches or exceeds the timeslot duration of the target D-subnet.

A solicitation Data DPDU is required for a link that is configured as a solicitation link.

To support passive scanning by new DLEs, active scanning hosts may also be configured to transmit advertisement Data DPDUs periodically.

It may be necessary to suppress solicitations, to account for situations where it is unsafe or illegal for a DLE to operate its radio transmitter without authorization. To address such situations, the DLMO attribute dlmo.RadioSilence provides a mechanism to disable solicitations along with all other DLE transmissions.

Solicitations are disabled by default. Solicitations are not used in the default configuration, and dlmo.ActScanHostFract defaults to zero.

9.1.13.4 Continuous scanning

Neighbor discovery should be an ongoing process even after a DLE has joined the D-subnet. Ongoing scans can, over time, help to form a more optimal D-subnet. Mains-powered routers that spend a substantial portion of their time listening can, over time, receive many advertisements from nearby routers. Battery-powered devices cannot spend a high percentage of their time listening, but even if they just sample the channel periodically, they can, over extended periods of time, build a comprehensive picture of neighboring routers. If the D-subnet is configured with a coordinated schedule of advertisements, such scanning can be performed more efficiently.

A system management function may establish an overall D-subnet schedule for advertisements, and a joined DLE may be provisioned with a schedule of receive links to ensure that such advertisements are heard over time. When used, this approach enables DLEs on the D-subnet to find neighbors efficiently. Additionally, a low-duty-cycle DLE may be configured to use these scheduled advertisements to remain time-synchronized with the D-subnet.

Advertisements are authenticated with a DMIC, to enable DLEs that have joined the D-subnet to rely on scheduled advertisements as a trusted source of timing and connectivity information.

9.1.14 Neighbor discovery and joining – DL considerations

9.1.14.1 General

The DL provides a configurable mechanism to discover neighboring DLEs.

During provisioning, a DLE is configured to scan for neighbors that can act as proxies in the join process. When an advertisement is received from a candidate neighbor, the DLE uses information in the advertisement to create communication links to and from that neighbor, and then provides the neighbor’s addressing information to the DMAP. For the remainder of the join process, the DLE provides communication support to upper layers by passing Data DPDUs to and from the neighbor.

After joining the D-subnet, the DLE is implicitly or explicitly instructed by the system manager to scan for new neighbors for a designated period of time, using superframes and links provided by the system manager. The result of this scan is reported as neighbor information to the system manager that uses the information to provide the DLE with an optimal configuration within the DLE’s mesh. The DLE then continues to accumulate information about new candidate neighbors throughout its lifecycle, and this information is periodically reported to the system manager to facilitate configuration of improved and adaptive mesh configurations.

The D-subnet joining process from the DLE’s point of view can be informatively summarized as follows:

- The DLE, possibly in its factory state, searches for an advertisement from the provisioning DLE. This search is built into the DLE as defined by this standard.
- The DLE receives an advertisement from the provisioning DLE. This advertisement, with D-subnet ID = 1, provides a compressed but fully functional configuration for the DLE’s state machine. The DLE uses this configuration to communicate with the provisioning DLE. During provisioning, a different DLE configuration, that is subsequently used to search for the target D-subnet, is written to the device provisioning object (DPO).
- At the end of provisioning, the provisioning DLE sets Join_Command=1, indicating successful completion of the provisioning process and causing the DLE to reset to the provisioned state. The DLE defines that reset as first resetting the DLE to its factory state, thus erasing the material from the provisioning DLE’s advertisement, and then initializing the DLE with the settings stored in the DPO.

– The DLE then commences operation of its state machine, using the configuration as initialized by the DPO. This configuration from the provisioning DLE should be matched to the target D-subnet to facilitate efficient D-subnet discovery. For example, if the target D-subnet is configured to transmit advertisements on three channels, the DPO might reasonably configure the DLE to scan those same three channels.

The discovery process is successful when the DLE receives an advertisement from the target D-subnet. This advertisement contains a compressed but fully functional DLE configuration that can be used for communication with the system manager through the router that transmitted the advertisement.

If the join process times out, the DLE is reset to the provisioned state. The configuration derived from the advertisement is erased, the DPO configuration is re-established, and the DLE resumes its search for a target D-subnet.

If the join process is successful, the DLE's joining configuration persists until explicitly updated by the system manager. Thus, at the end of the join process, the DLE usually retains an interim connection with the system manager through the advertising router.

Following a successful join, the DLE searches for a set of candidate routers that can be used for communication. After a configurable period of time, defaulting to 60 s, the DLE reports this list of candidates to the system manager. This information is used by the system to replace the interim connection with a more permanent and resilient DLE configuration.

If the DLE's intended reporting rate will be so low that the time synchronization required for slotted-channel-hopping will not be maintainable, and if the local D-subnet provides slow-channel-hopping intervals, then the DLE can shift to using slow-channel-hopping for most of its infrequent communications.

NOTE The join process, which is intended for infrequent, acyclic use by any given DLE, uses slotted-channel-hopping; slow-channel-hopping during the join process is not supported.

9.1.14.2 DLE states

When a device is manufactured, the DLE is in its default state. Attributes in the default state are defined by this standard.

In the default state, the DLE shall periodically scan for a provisioning D-subnet, using a search procedure defined by this standard. When the DLE receives one or more advertisements from a provisioning DLE in a provisioning D-subnet, the DLE shall use information in one of the advertisements to establish a superframe with links that can be used to communicate with the selected provisioning DLE. The DLE then informs the DPO (device provisioning object) that a D-association has been established, and switches the DLE to the provisioning state.

In the provisioning state, the DLE halts the search procedure defined by this standard. Instead, the DLE operates its state machine according to a superframe with links that were provided by the provisioning DLE's advertisement. During the provisioning process, the DLE provides communication services to upper layers by operating its state machine according to the advertised superframe and links, so that provisioning APDUs can be passed between the DPO and the provisioning DLE via the DLE that is being provisioned.

If the provisioning process times out, the DLE reverts to its default state and resumes its default search procedure for a provisioning D-subnet.

If the provisioning process is successful, the DPO provides the DLE with a set of attributes, including D-subnet information, superframes, and links, that the DLE can use to search for the target D-subnet(s). The DLE then switches from the provisioning state to the provisioned state, and operates its state machine as configured in the superframes and links that were provided by the DPO.

The switch from the provisioning state to the provisioned state is triggered when the provisioning DLE sets Join_Command=1 (Table 10). When that occurs, the DLE is reset to its provisioned state and then operates its state machine as configured in order to search for a target D-subnet.

In general, a DLE reset to a provisioned state is accomplished in two steps. First, the DLE is reset to its default state. Then, information from the DPO's Target_DL_Config attribute is applied to the DLE. This provides a set of attributes, including D-subnet information, superframes, and links, that the DLE uses to search for the target network and corresponding D-subnets. See 13.8.

If the DLE is provisioned through an out-of-band mechanism, the provisioning state may be bypassed and in that case the DLE transitions directly from the unprovisioned state to the provisioned state.

The DPO retains a copy of the information that was used to provision the DLE, providing a means to reset the DLE back to its provisioned state by putting the DLE into its default state and then adding the provisioned attributes from the DPO.

A DLE in the provisioned state operates its state machine as configured in its provisioned superframes and links. The provisioned superframes and links should be matched to the operating characteristics of the target D-subnet(s), so that a target network is efficiently discovered when the DLE and a target D-subnet are in proximity to each other. The result is that the DLE receives at least one advertisement from at least one proxy DLE that is participating in one of those target D-subnets. The DLE uses the information conveyed by one of the received advertisements to establish a superframe with links that can be used to communicate with the sending proxy DLE. The DLE that is attempting to join the D-subnet then informs its DMAP that a D-association has been established to a neighboring proxy, and that DLE then switches from the provisioned state to the joining state.

When the DLE enters the joining state, it retains its configuration from the provisioned state (see 13.8), and adds the superframe, links and other attributes defined or implied by the advertisement. The DLE then provides communication services to upper layers during the joining process, by operating its state machine according to the advertised superframe and links with the result that joining APDUs are passed between the DMAP and proxy DLE through the DLE that is being provisioned.

If the DMAP's joining process times out, the DLE resets to its provisioned state and resumes scanning for a D-subnet using the provisioned superframes and links.

If the DMAP's joining process is successful, the advertised superframes and links continue to be used temporarily for communication with the system manager. The DPO retains the information needed to reset the DLE back to the provisioned state in the event of a DLE reset.

When the DLE is first placed in its joined state, it has a single connection to the D-subnet through the neighboring proxy DLE, using the superframe and links defined in the advertisement. This single connection, selected implicitly when the DLE selected the proxy DLE, lacks path diversity, and may be suboptimal for any number of reasons. Therefore, the system manager shall provide instructions for the DLE to search for several neighbors and to report the result when the search is completed. Simple instructions may be provided through the same advertisement that established the initial connection to the proxy DLE, or more elaborate search instructions may be provided by the system manager immediately after the DLE joins the D-subnet. In either case, the DLE provides the system manager with a list of candidate neighbors soon after it joins the D-subnet, with 60 s being the default reporting time as controlled by the attribute dlmo.DiscoveryAlert. The system manager analyzes this list of candidate neighbors and then provides the DLE with an updated configuration that includes path diversity (mesh) and generally provides a more optimal D-subnet connection.

To support security processing, the DMAP needs the EUI64Address of the advertising DLE during both the provisioning and joining process. Since the advertisement Data DPDU provides only the advertising router's DL16Address, the DLE shall acquire the EUI64Address from the neighbor when communication begins. The neighbor's EUI64Address shall be acquired by using a transmit link to interrogate the neighbor using a Data DPDU with a null payload, setting the DHDR request EUI64Address bit (bit 5, Table 118) to a value of 1, causing the neighbor's EUI64Address to be returned in the ACK/NAK DPDU.

9.1.14.3 Consolidated DL configuration information

The DPO maintains the attribute Target_DL_Config that includes the settings for various attributes in the DLE. This OctetString is provided to the DLE at the end of the provisioning process, and is retained by the DPO in the event that it needs to reset the DLE back to its provisioned state.

The DL_Config_Info OctetString contains a collection of attributes that the DPO provides to the DLE in order to establish the provisioned state. Each attribute is expressed as a tuple including the attribute number, followed by an OctetString that contains the new attribute value. The structure of DL_Config_Info is shown in Table 102.

Table 102 – DL_Config_Info structure

Octets	Bits							
	7	6	5	4	3	2	1	0
1 octet	N (number of attributes)							
1 octet	$AttributeNumber_1$ (Unsigned8)							
–	$NewAttribute_1$ (OctetString)							
…	…							
1 octet	$AttributeNumber_N$ (Unsigned8)							
–	$NewAttribute_N$ (OctetString)							

Several of the attributes in DL_Config_Info are indexed OctetStrings. In these cases, $NewAttribute_X$ is a new row entry. By DL convention, each row entry in an indexed OctetString attribute includes the row index as its first field.

DL_Config_Info can be used to configure any read/write attribute in the DLE. At a minimum, DL_Config_Info shall configure:

- AdvFilter, which provides a filter so that the DLE can select superframes that are of interest.
- At least one superframe, and at least one link, that can be used by the DLE in searching for advertisements.

Timeslot templates used for searching for the D-subnet, if different from the default timeslot templates, shall be provided to the DLE during the provisioning process.

DL_Config_Info shall not be used except through the DPO.

The DLE's provisioned attributes shall be retained by the DPO so that the DLE can be reset to its provisioned state.

Superframe operation may be delayed or disabled by setting the IdleTimer field within the superframe. For example, two superframes may be provisioned in a DLE, with superframe number1 searching aggressively for the D-subnet and superframe number2 searching on a low-duty cycle. The IdleTimer of superframe number1 might cause that superframe to time out a few minutes after the DLE is configured. Alternatively, a superframe might be configured with an IdleTimer so that it is idle until some future time. If the DLE is reset to the provisioned state, the superframe idle timers shall be reset to the originally provisioned state as well.

A DLE in the provisioned state shall operate its DL clock and increment TAI time. This enables the DLE to operate its superframes as provisioned to discover candidate D-subnets. During the join process, a revised TAI time will be received in advertisements and the DL clock reset accordingly.

The DLE might completely lose its time sense while in the provisioned state, for example due to removal of a battery. Complete loss of time sense in a provisioned DLE shall trigger a reset of the DLE to its provisioned state.

9.1.14.4 Scanning for neighbors in the unprovisioned state

An unprovisioned DLE begins in the system default state. Its DLE configuration includes the five default channel-hopping patterns and the three default timeslot templates. Its superframes, links, graphs, and routes are blank.

Before attempting to join the plant network, the unprovisioned DLE needs to establish contact with a provisioning DLE and be transitioned to the provisioned state. This may occur through an out of band mechanism, such as a wired modem or an infrared link.

This standard defines an unprovisioned DLE's search procedure for a provisioning D-subnet's advertisement. The search procedure is radio silent, not involving solicitations or any other transmission until an advertisement is received from a provisioning D-subnet.

An unprovisioned DLE shall scan for a provisioning D-subnet's advertisements on channels 4 and 14, corresponding to IEEE 802.15.4:2011 channels 15 and 25, if radio regulations so permit (see Figure 59). In each of these channels, the unprovisioned DLE shall scan for an advertisement at a fixed interval of exactly 0,25 s, 0,5 s, 1 s, 2 s, 4 s, 8 s, 16 s, or 32 s. When the DLE is powered on or physically reset, it shall scan at the shortest interval of 0,25 s for at least 10 s, and then may gradually increase the interval as necessary to preserve its energy supply.

SubnetID=1 is reserved for the provisioning D-subnet. Therefore, only advertisements with SubnetID=1 shall be considered by a DLE in the unprovisioned state.

9.1.14.5 Scanning for neighbors in the provisioned state

Once a DLE is in the provisioned state, it shall use its provisioned superframes and links to scan for a D-subnet of interest. The DLE may discover multiple advertisement routers and then select one to be used as a proxy in the join process.

9.1.14.6 Scanning for neighbors after joining the D-subnet

A DLE joins the D-subnet through a single neighbor that is sending advertisements. Initial contracts are established based on a route that is not necessarily optimal, through the joining proxy DLE. Having established a single connection through the join process, the DLE shall be configured by the system manager to immediately search for additional and alternative neighbors in order to support a more optimized mesh configuration.

A DLE in the joined state can be configured by the system manager to passively scan for advertisements by configuring the DLE with links and superframes that enable the DLE's radio receiver at scheduled times. The DLE may be configured to actively scan for advertisements using solicitations.

The NeighborDiscovery alert (see 9.6.2 and 9.4.2.24) provides a mechanism for the DLE to transfer this information to the system manager. The alert is intended for the following scenarios:

- Within 15 s of entering the joined state, the DLE shall be configured with superframes and links that search for advertisements from routers that are in range, and select the most promising candidates. The configuration can be accomplished through the advertisement itself, prior to joining, as described in 9.3.5.2.4.2. Alternatively, the configuration can be provided by configuring DLMO attributes after joining. After a configurable elapsed amount of time, as defined by the attribute dlmo.DiscoveryAlert, the DLE shall use the neighbor discovery alert to send the list of candidates to the system manager. A superframe dedicated to this initial search can be configured to automatically time out through its IdleTimer field.
- The DLE should be configured by the system manager to continuously passively and/or actively scan for advertisements on an ongoing basis. Over time, this enables the DLE to build a list of candidate neighbors. The HRCO may be configured to periodically transmit this candidate neighbor table, along with neighbor diagnostics, to the system manager. The system manager may alternatively read the candidate neighbor table, dlmo.Candidates, on its own schedule.
- A DLE shall also use the neighbor discovery alert whenever a connectivity issue is reported through the DL_Connectivity alert (see 9.6.1). This provides an up-to-date picture of neighborhood connectivity, enabling the system manager immediately to consider alternative solutions to the reported DLE connectivity problem.

9.1.15 Radio link control and quality measurement

9.1.15.1 General

Signal quality information is accumulated in the DLE and reported through the DMAP. In support of these higher-level functions, the DLE provides primitives that report signal quality information. The DLE also provides attributes that enable the system manager to control radio emissions.

9.1.15.2 Performance metrics

Numerous performance metrics can be accumulated by the DLE on a per-neighbor basis, configured by the system manager and reported through the DMAP (see 9.4.3.9).

The DLE can be configured by the system manager to accumulate the following types of performance data on a per-neighbor basis:

- Received signal strength indicator (RSSI) and received signal quality indicator (RSQI).

  NOTE This standard defines practices for RSSI and RSQI reporting to facilitate consistency, so that signal characteristics are somewhat comparable among devices of differing construction.

- For transmissions: counts of successful transmissions, CCA backoffs, unicast errors, and NAKs.
- Count of received Data DPDUs.
- Diagnostics of clock corrections.

Per-channel diagnostics are also collected and consolidated for all neighbors.

RSSI shall be reported as a signed 8-bit integer, reflecting an estimate of received signal strength in dBm. RSSI reports shall be biased by +64 dBm to give an effective range of -192 dBm to +63 dBm. For example, a reported RSSI value of -16 corresponds to a received signal strength of -80 dBm.

The actual received signal strength for a device depends on the receiver's noise floor, which is device-construction and operating-temperature dependent, as well as on the receiver's minimum sensitivity. These can be accounted for within the receiving device, as they are quite repeatable for a given device design and device operating temperature. Thus device designers should approximately map available indications of received signal strength and device temperature (where known) to a reasonable estimate of RSSI, so that system managers have a consistent basis for making routing decisions among DLEs.

RSQI shall be reported as a qualitative assessment of signal quality, with higher number indicating a better signal. A value of 1..63 indicates a poor signal, 64..127 a fair signal, 128..191 a good signal, and 192..255 an excellent signal. A value of zero indicates that the chipset does not support any signal quality diagnostics other than RSSI.

RSSI is a quantitative measurement, mapped to physical units. RSQI is a qualitative measurement. RF devices from different manufacturers, or with different part numbers, or even with an improved die layout or photolithography shrink, may generate different raw RSQI values. Since the IEEE 802.15.4 PHY does not specify a common measurement and reporting methodology for the underlying hardware, no superior software sublayer can create it; the information simply is not present consistently across different devices.

RSQI metrics are intended to be particularly useful when comparing different entries in dlmo.Candidates, where the assessment is among signaling of differing origins received by a single device. A DLE may use innovative techniques to report comparisons of the likely link quality of different candidate neighbors. Because inter-device variances at the receiving device are removed, RSQI entries in dlmo.Candidates reported from a single given DLE may be reasonably compared to each other, and fine distinctions can be taken as meaningful. For example, differences within the range of good signals can be reasonably taken into consideration if the RSQI metrics are from the same DLE.

RSQI may also be compared across different DLEs, but fine distinctions are unlikely to be as meaningful. For example, the distinction between a fair and an excellent link is likely to be meaningful even if reported from unlike devices, but distinctions between different levels of good links has no standard meaning if reported from different devices.

Since any reported RSQI value is a qualitative measurement, comparison of such values must necessarily take the RF-chip-specific nature of such measurements into account. Given the specified interpretation of reported values, any two non-zero RSQI values reported by different DLEs that differ by an amount of 32 or more can be ranked as “better” and “worse”, where the confidence in the ranking increases with increasing numeric difference.

Similarly, differing RSSI values reported by the same device at different times or for different remote correspondents may be compared reliably, as can RSSI values among devices that are known (by vendor and other device-specific model identification) to provide callibrated RSSI estimates. In other cases such comparisons are at best approximate, somewhat similar to RSQI though presumably more closely approximating the actual strength of received signaling at the reporting device. In particular, magnitude ordering among reports from the same reporting device are always reliable in terms of their ordering and approximate magnitude of difference.

9.1.15.3 Accumulating and reporting diagnostic information

The system manager establishes a DL communication relationship between a DLE and its neighbor by adding an entry to the DLE’s dlmo.Neighbor attribute. Each such entry specifies a level of diagnostics to be collected, through the field dlmo.Neighbor[ ].DiagLevel. For each neighbor, diagnostics may be collected at a baseline level, or at a detailed level including clock diagnostics.

Per-channel diagnostics are accumulated and consolidated for all neighbors, in the attribute dlmo.ChannelDiag.

When the dlmo.Neighbor[ ].DiagLevel field is set for a particular neighbor, the DLE shall create corresponding entries in the read-only attribute dlmo.NeighborDiag. NeighborDiag values are accumulated from the time that the dlmo.NeighborDiag entry is created.

Three mechanisms are provided for reporting diagnostic information contained in dlmo.NeighborDiag and dlmo.ChannelDiag:

- The health reports concentrator object (HRCO), described in 6.2.7.7, can be configured to report any attribute in the DLE on a periodic basis. dlmo.NeighborDiag entries and dlmo.ChannelDiag can be reported through that mechanism.
- Diagnostic information can be retrieved at any time by the system manager, by reading the applicable attributes.
- Diagnostic information can be reported by the DLE on an exception basis, through the DL_Connectivity alert.

Diagnostics include a combination of levels, such as RSSI, and counters, such as a count of acknowledgments.

Levels are accumulated as exponential moving averages (EMAs). The level is initialized with the first data value, after which each new data value is accumulated into the EMA level as follows, where:

$$EmaLevel_{NEW} = EmaLevel_{OLD} + (\alpha \ / \ 100) \times (NewData - EmaLevel_{OLD})$$

The smoothing factor α is expressed as an integer in the range of 0..100, representing a percentage; it is configured by the system manager through the attribute dlmo.SmoothFactors (see 9.4.2.25).

Counters in dlmo.NeighborDiag are accumulated as ExtDLUint unsigned integers, which internally are 15-bit integers. When a counter reaches its maximum value, of 32 767 (0x7FFF), it shall "stick" and continue to report that maximum value. Counters shall be reset to zero whenever the row is reported through the HRCO or retrieved through a read operation. Reporting the value through the DL_Connectivity alert shall not reset any counters.

9.1.15.4 Radio silence

The DLE can be configured to transmit only when actively participating in a D-subnet. This behavior is configured by the dlmo.RadioSilence attribute, which designates a timeout period for D-subnet participation, in seconds. For example, if the dlmo.RadioSilence attribute is set to the default of 600 s (10 min), the DLE silences its radio transmitter 10 min after losing communication with the D-subnet. When all DLEs on a D-subnet are configured for radio silence, it is possible to disable the D-subnet entirely, even if some DLEs do not receive an explicit command to disable communications.

When a valid time update is accepted by the DLE from an advertisement or an acknowledgment, the DLE internally records the current time as the radio silence time reference. If the DLE does not accept another time update in the subsequent time period designated by dlmo.RadioSilence, the DLE shall become silent by ignoring all of its configured transmit links, including solicitations. In the radio silent state, the DLE continues to operate its radio receiver as per its scheduled receive links, but without transmitting acknowledgments in the absence of a clock update.

For example, suppose the DLE receives a time update at 01h:02m:03s, and dlmo.RadioSilence is set to 600 s (10 min). If the DLE does not receive another time update by 01h:12m:03s, i.e., 10 min later, it will silence its radio at that time.

If dlmo.RadioSilence is configured as zero, the feature is disabled.

Radio silence is the default. The default D-subnet discovery procedure does not use solicitations, and dlmo.RadioSilence defaults to 600 s.

Support of the dlmo.RadioSilence attribute is required for all DLEs.

The radio silence profile limits the permitted range of the dlmo.RadioSilence attribute. The radio silence profile is reported to the system manager on joining through dlmo.DeviceCapability. A DLE with the radio silence profile shall reject updates to dlmo.RadioSilence that are greater than 600 s, thus ensuring that such a DLE will never spontaneously transmit a DPDU once it has lost contact with the D-subnet for 600 s.

Temporary radio silence can be accomplished with another attribute, dlmo.RadioSleep. When dlmo.RadioSleep is set to a positive value, the DLE treats all links, including receive links, as idle for the designated number of seconds. Activation of dlmo.RadioSleep shall be slightly delayed to allow for transmitting an AL acknowledgment for the DMAP APDU that causes the attribute to be set. When the sleep period is over, dlmo.RadioSleep is automatically reset to zero, indicating that the feature is disabled.

9.1.15.5 Radio transmit power

This standard provides the system manager with a degree of control over radio transmit power, through the attribute dlmo.RadioTransmitPower.

dlmo.RadioTransmitPower is used to control the DLE's radio transmit power level, in dBm EIRP. It defaults to the DLE's maximum supported power level, and is always constrained by dlmo.CountryCode (9.1.15.6) to the regulatory constraints of the locale of use. This constrained default value is also reported to the system manager during the join process through dlmo.DeviceCapability.

When dlmo.RadioTransmitPower is changed by the system manager, the DLE shall not transmit at an output power level in excess of dlmo.RadioTransmitPower.

In addition, a DLE may autonomously calibrate its output power level to the minimum level needed to maintain reliable connectivity. To enable this, the DLE supports the echoing of signal quality information in acknowledgments, so that an implementation can calibrate the received signal quality at various power levels. See 9.3.5.5.

NOTE 1 An accurate calculation of the DLE's actual output level from correspondent reports of received RSSI depends on design information for the reporting correspondent DLE that is not known to the self-calibrating DLE. Those factors include the correspondent DLE's receiver noise floor and minimum receive sensitivity. Thus any self-adjustment of transmit levels based on received RSSI is at best approximate, particularly when the corresponding DLEs' RF subsystems do not share a common design (such as when they are from different manufacturers).

It is possible for DLEs to provide local correction of the RSSI values that they report to account for the influence of their own receiver's noise floor and minimum receive sensitivity. Such self-correction, which is not addressed by the IEEE 802.15.4 standard, typically can be performed in a piecewise-linear manner. The resulting reported values are considered to meet the RSSI requirements of both this standard and those of IEEE 802.15.4, even though they are slightly adjusted from the measurements of the RF subsections of implementing DLEs (since those actual measurements do not take those other relevant characteristics of the RF subsystem into account). See also 9.1.15.2.

9.1.15.6 Country code

The provisioning DLE and/or the system manager can inform the DLE being provisioned of regulatory considerations through its dlmo.CountryCode attribute, Table 103, which is a 16-bit packed structure consisting of a 10-bit country code and six Booleans:

- Bits 0..9 provide a 10-bit country code as an Unsigned10 integer, using ISO 3166-1 numeric three-digit country codes.
- Bits 10..15 specify a six-element Boolean array:
  - Bit10 (Index 0), FCC, indicates whether FCC rules apply. A DLE shall operate in compliance with FCC rules when Index0 (Bit10) is TRUE.
  - Bit11 (Index 1), ETSI, indicates whether ETSI rules apply. A DLE shall operate in compliance with ETSI rules when Index1 (Bit11) is TRUE.

– Bit12 (Index 2), LP, indicates whether a 10 dBm EIRP limit applies. A DLE shall limit its emissions to ≤ 10 dBm EIRP when Index2 (Bit12) is TRUE.

– Bit13 (Index 3), LBT, indicates whether the DLE shall operate under adaptive modulation rules, using LBT to sense the channel when initiating a transaction, ceasing use of the slot if activity is detected: FALSE=non-adaptive, TRUE=adaptive.

– Bit14 (Index 4), FHSS, indicates whether the DLE shall operate under frequency-hopping spread-spectrum rules: FALSE=not-FHSS-rules, TRUE=FHSS-rules.

– Bit15 (Index 5), Locked, indicates whether the value of this attribute is fixed while the DLE is operational. Once this “sticky” bit is set, any subsequent attempt to modify this attribute shall be rejected except when the DLE is reset to the factory default state during (re)provisioning.

Table 103 – CountryCode

Octet number	Bits							
	7	6	5	4	3	2	1	0
1	Locked	FHSS	LBT	LP	ETSI	FCC	bits 9..8	
2	ISO 3166-1 CountryCode bits 7..0							

When Bit11 and Bit13 (ETSI and LBT) are both TRUE, each D-transaction shall begin with a LBT observation interval of at least 20 us, using CCA Mode 1, thus supporting modes V.4 3) and V.4 6).

NOTE 1 CCA Mode 3 is also acceptable under EN 3003 328 v1.8.1 when it is implemented as the AND of CCA Mode 1 and CCA Mode 2, but not when it is implemented as the OR of CCA Mode 1 and CCA Mode 2. See 9.1.9.4.3.

When Bit11, Bit13 and Bit14 (ETSI, LBT and FHSS) are all TRUE, operation switches momentarily to the non-adaptive rules of ETSI EN 300 328 while sending an ACK/NAK DPDU (as short control signaling) within a transaction and for the immediately following Tx-gap-time of EN-mandated non-transmission, thus supporting mode V.4, 6).

Bit15 (Locked) supports device operation (when TRUE) in regulatory regimes that prohibit the ability to reconfigure a device in such a way that it would violate regulatory restraints, while still supporting devices (when FALSE) on mobile platforms such as ships and trains that may cross regulatory jurisdictional boundaries, and while still permitting (via the reprovisioning exception) the repair or refurbishment of devices with subsequent resale into or reuse in markets where other regulations apply.

When no specific country of intended use has been identified, the default for dlmo.CountryCode shall be 0x3C00, indicating that a device in the default state should comply with FCC rules, ETSI rules, the < 10 dBm EIRP limit, and be classified as an adaptive non-FHSS device. See 5.2.5 and Annex V.

NOTE 2 This default value ensures that the equipment, before it has been provisioned, meets the regulatory requirements of most regions in which it might be deployed, and in particular as such rules would apply to the three-channel default configuration used for out-of-the-box over-the-air provisioning. Such constraint enables the device to participate in short-range provisioning over the Type A wireless medium, at which point the dlmo.CountryCode attribute would be changed to reflect the intended regulatory regime that applies to the device’s initial (and usually only) locale of deployment.

9.1.16 DLE roles and options

The DL specified by this standard is designed with the general goal of constraining the range of construction options for a conforming device, while enabling flexible and innovative system solutions.

The DL framework does not require that all DLEs be equivalent. For example, some routers, designed as dedicated infrastructure devices, might have a continuous source of energy, powerful processors, and essentially unlimited memory capacity. In contrast, some field instruments may have low-capacity batteries and may lack routing capability.

These distinctions among DLEs are covered in three general ways in this standard:

- memory capacity;
- DLE capabilities; and
- DLE roles.

Every DLE has a limited amount of memory that is available for DL operations, and the system manager needs knowledge of these limitations in order to configure the DLE and balance the D-subnet operation. DLE DL memory is not reported as a single block, but rather as specific capacities of memory for specific purposes. For example, each indexed OctetString attribute supports a limited number of entries, with the capacity available to the system manager as metadata. Similarly, buffer capacity for Data DPDU forwarding is reported by the DLE on startup.

Certain DLE capabilities are also reported on startup. For example, the DLE reports the stability of its own clock, as well as a list of radio channels that it can support legally. DLE capabilities reported with the join request are enumerated in 9.4.2.23.

DLE roles describe the general capabilities of a given DLE configuration. For example, a DLE may be capable of routing or not. Distinctions of this type have various implications throughout the DLE, in terms of minimum memory capacity, DLE capabilities, and support of various features. The DLE simply reports which roles it supports, and the system manager is then responsible for mapping this into a portfolio of DLE capabilities. Standard mappings between roles and minimum capabilities are provided in Annex B.

9.1.17 DLE energy considerations

Devices have different levels of available energy. One device may have a continuous energy source. Another device may have a large battery, but may need most of that energy capacity for running a sensor. Yet another device may use energy scavenging as its primary energy source. Different battery chemistries have different characteristics, a given battery chemistry may provide different performance depending on the supplier, and a battery's capacity may vary depending on environmental factors. New battery technologies are likely to emerge with currently unknown performance characteristics. One application might need a 20-year battery life, while a different application might tolerate a 6-month life.

The DLE may be configured by the system manager to consume different amounts of energy. The DLE consumes energy in two general ways:

- The DLE consumes energy by providing wireless service to its own applications. When a DLE establishes a contract to transmit data every 5 s, the DLE consumes a corresponding amount of energy.
- The DLE consumes energy acting as a router on behalf of neighboring DLEs. A DLE may be configured to transmit advertisements every 10 s. A DLE may be configured to operate its receiver almost continuously, listening for solicitations. The D-subnet may be configured so that a DLE forwards up to 100 DSDUs per minute. All of these scenarios consume energy.

The DLE reports a general sense of its capacity to support DL routing operations in certain fields of the dlmo.EnergyDesign attribute. This attribute is reported through the dlmo.DeviceCapability attribute.

dlmo.EnergyDesign indicates the device's designed energy capacity to handle DL operations. This attribute is constant over the life of the device and reflects the device's design, not its current state. A system manager should configure a DLE within these stated energy limitations:

– EnergyLife indicates the device's energy life by design. A positive value provides energy life in days; a negative value provides energy life magnitude in hours. A value of 0x7FFF indicates a continuous power source and no constraining device energy limitations. Other EnergyDesign fields describing DLE energy capacity are based on this target energy life. Configuration of the DLE beyond these stated energy capacities will likely reduce the device's energy life.
– ListenRate indicates the DLE's energy capacity on average, in seconds per hour, to operate its radio's receiver. ListenRate includes time to receive Data DPDUs for the DLE's own application contracts, plus Data DPDUs being forwarded by the DLE on behalf of other DLEs.
– TransmitRate indicates the DLE's energy capacity, in Data DPDUs per minute, to transmit Data DPDUs on its own behalf and to forward Data DPDUs on behalf of its neighbors.
– AdvRate indicates the DLE's energy capacity, in Data DPDUs per minute, to transmit dedicated advertisement (or solicitation) Data DPDUs.

EnergyDesign is a constant, and does not reflect the changing state of a device's energy source. The dlmo.EnergyLeft attribute is a dynamic read-only attribute that can be used to report the device's remaining energy capacity. A positive value indicates the remaining life in days, and a negative value indicates the magnitude of the remaining life in hours. A value of 0x7FFF indicates that the feature is not supported. dlmo.EnergyLeft is reported on startup through dlmo.DeviceCapability, and may also be reported periodically through the HRCO.

9.2 DDSAP

9.2.1 General

The DDSAP supports the multi-hop conveyance of a DSDU (e.g., and NPDU) between DLEs in a D-subnet.

DD-DATA.request takes a DSDU from the NLE, prepends a Data DPDU header, and adds it to the message queue. DD-DATA.confirm subsequently reports whether the DSDU was successfully conveyed to a neighboring DLE in the D-subnet.

DD-DATA.indication indicates the receipt of a Data DPDU that has reached its final destination within the D-subnet, and passes its DSDU to the NLE.

All interfaces between the DLE and adjacent layer entities or management entites are internal interfaces within the device, and thus are unobservable. Therefore they are strictly notional and not subject to standardization.

9.2.2 DD-DATA.request

DD-DATA.request is a primitive that accepts DSDU from the NL, selects the route through the D-subnet, and places a corresponding Data DPDU on the DLE's message queue.

The semantics of the DD-DATA.request primitive are as follows:

```
DD-DATA.request (
                SrcAddr,
                DestAddr,
                Priority,
                DE,
                ECN,
                LH,
                ContractID,
                DSDUSize,
                DSDU,
                DSDUHandle)
```

Table 104 describes the parameters for DD-DATA.request.

Table 104 – DD-DATA.request parameters

Parameter name	Parameter type
SrcAddr (DL source address)	Type: DL16Address or EUI64Address
DestAddr (DL destination address)	Type: DL16Address or EUI64Address
Priority (priority of the payload)	Type: Unsigned4
DE (discard eligible)	Type: Unsigned1
ECN (explicit congestion notification)	Type: Unsigned2
LH (last hop, NL)	Type: Unsigned1
ContractID (ContractID of the payload)	Type: Unsigned16 or null
DSDUSize (payload size)	Type: Unsigned8
DSDU (number of octets as per DSDUSize)	Type: Octets
DSDUHandle (uniquely identifies each invocation of this primitive)	Type: Abstract

DD-DATA.request parameters include:

- SrcAddr is the source address of the NSDU. It is normally the DL16Address alias of the NSDU’s source IPv6Address, except when it is the EUI64Address of an unjoined DLE. Subnet ID is implicit, based on dlmo.SubnetID.
- DestAddr is the destination address of the NSDU. It is normally the DL16Address alias of the NSDU’s destination IPv6Address, except when it is the EUI64Address of an unjoined DLE. Subnet ID is implicit, based on dlmo.SubnetID.
- Priority is copied to the DROUT subheader and indicates the Data DPDU’s priority in DLE message queues.
- DE is copied to the DADDR subheader. DE=1 indicates that the DSDU is eligible to be discarded from a message queue in favor of an incoming Data DPDU with DE=0, and of equal or higher priority.

  NOTE “is eligible” does not mean “is mandatory”. Such discard is an implementation option.
- ECN is copied to the DADDR subheader. See 9.1.9.4.5 for a discussion of ECN.
- LH is copied to the DADDR subheader. A value of 1 indicates that the DSDU entered the D-subnet through a backbone router, and therefore shall not exit the DL through a backbone router to avoid circular routes at the NL. This enables the NL to elide the IPv6 hop limit field. Logically, LH is carried by the DL on behalf of the NL, and LH shall not be changed by the DL.
- ContractID may be used by the DLE in route selection, as discussed in 9.1.6.5.
- DSDUSize indicates the number of octets contained in the Data DPDU payload.
- DSDU is the set of octets forming the payload. It may be implemented as a pointer to memory that is shared among layers.
- DSDUHandle is an abstraction that connects each invocation of DD-DATA.request with the subsequent callback by DD-DATA.confirm.

9.2.3 DD-DATA.confirm

DD-DATA.confirm is a primitive that reports the results of a request to transmit a DSDU that was previously placed on the DLE message queue by DD-DATA.request.

Table 105 describes the parameters for DD-DATA.confirm.

Table 105 – DD-DATA.confirm parameters

Parameter name	Parameter type
DSDUHandle (identifier for the payload)	Type: Abstract
Status (see Table 106)	Type: Unsigned

Table 106 specifies the value set for the status parameter.

Table 106 – Value set for status parameter

Value	Description
SUCCESS	Operation was successful
FAILURE	Operation was unsuccessful; operation timed out

NOTE Error handling between the DLE and collacted NLE is an internal device matter, not visible across any observable interfaces, and therefore is not standardized.

9.2.4 DD-DATA.indication

DD-DATA.indication is a virtual primitive that indicates the receipt of a DSDU. A Data DPDU does not trigger a data indication until it reaches its destination on the D-subnet.

The semantics of the DD-DATA.indication primitive are as follows:

```
DD-DATA.indication(
        SrcAddr,
        DestAddr,
        Priority,
        DE,
        ECN,
        LH,
        DSDUSize,
        DSDU)
```

Table 107 describes the parameters for DD-DATA.indication.

Table 107 – DD-DATA.indication parameters

Parameter name	Parameter type
SrcAddr	Type: DL16Address or EUI64Address
DestAddr	Type: DL16Address or EUI64Address
Priority (priority of the payload)	Type: Unsigned4
DE (discard eligible)	Type: Unsigned1
ECN (explicit congestion notification)	Type: Unsigned2
LH (last hop, NL)	Type: Unsigned1
DSDUSize (payload size)	Type: Unsigned8
DSDU (number of octets as per DSDUSize)	Type: Octets

DD-DATA.indication parameters include:

- SrcAddr is the source address of the NSDU. It is normally the DL16Address alias of the NSDU's source IPv6Address, except when it is the EUI64Address of an unjoined DLE. D-subnet ID is implicit, based on dlmo.SubnetID.
- DestAddr is the destination address of the NSDU. It is normally the DL16Address alias of the NSDUs destination IPv6Address, except when it is the EUI64Address of an unjoined DLE. D-subnet ID is implicit, based on dlmo.SubnetID.

- Priority is included in the DROUT subheader and may be used by the NL for subsequent routing. ContractID, if required by the NL, is not carried within the Data DPDU header.
- DE provides the value of the DE bit copied from the incoming DADDR subheader.
- ECN provides the value of the ECN bit copied from the incoming DADDR subheader, and corresponds to the ECN bit described in IETF RFC 3168. See 9.1.9.4.5 for a discussion of ECN.
- LH provides the value of the LH bit copied from the incoming DADDR subheader.
- DSDUSize indicates the number of octets contained in the Data DPDU payload.
- DSDU is the set of octets forming the payload. It may be implemented as a pointer to memory that is shared among layers.

9.3 Data DPDUs and ACK/NAK DPDUs

9.3.1 General

The structure of DPDUs used by this standard is shown in Figure 88.

Figure 88 – PhPDU and DPDU structure

The DPDU reflects the multi-layer PDU structure described in 9.1.4, including:

- MAC header (MHR): The MHR is a data structure modeled on that of IEEE 802.15.4, as specified in 9.1.4 and 9.1.5, which includes frame-format information and D-subnet addressing information. The FCS, at the end of the DPDU, is logically associated with the MHR.
- DPDU header (DHR): The DL header information follows the MHR. Subheaders within the DHR include:
  – DHR header (DHDR): DHDR includes settings for various DLE selections and a version number.
  – DHR MAC extension subheader (DMXHR): Additional fields, not specified by IEEE 802.15.4:2011, that are needed to send a Data DPDU to an immediate neighbor and to receive an immediate ACK/NAK DPDU. The DMXHR includes information about the cryptographic integrity and confidentiality measures that apply to the DPDU. The DMIC following the DSDU is logically associated with the DMXHR.
  – DHR auxiliary subheader (DAUX): Some DPDUs include auxiliary information to facilitate neighbor discovery, time propagation, information exchange, and command exchange among immediate neighbors. The DAUX subheader is frequently absent. A non-null DAUX field shall be included in dedicated advertisement or solicitation DPDUs, or alternatively it may be embedded in unrelated Data DPDUs.
  – DHR routing subheader (DROUT): The DROUT field contains information needed to route the contained DPDU payload through the D-subnet. A non-null DROUT field shall include a Data-DPDU priority class and forwarding limit, plus either GraphID or source routing information.
  – DHR address subheader (DADDR): The DADDR field contains the source and destination endpoint D-addresses within the D-subnet, along with the NL fields ECN, DE, and LH, all of which are conveyed by and visible to the DL.

- DSDU: The DPDU's higher-layer payload is a single 6LoWPAN NPDU as defined in Clause 10, which is passed to the DLE as a DSDU. The payload is conveyed transparently within the D-subnet.
- DMIC: The DMIC, found near the end of the DPDU, is logically associated with the DMXHR. The DMIC is a cryptographically-strong integrity code that permits determination that the received DPDU
  - was originated by a device that shares the relevant encryption key, and
  - was unaltered before reception.

  NOTE In some cases the relevant DMIC encryption key is static and published, enabling forgery by an uninformed attacker.
- FCS: The FCS, found at the end of the DPDU, is logically associated with the MHR. The FCS is a trivially-forgeable integrity code that enables detection of PhL-induced DPDU errors.

Some classes of DPDUs that are generated by the DLE, such as dedicated advertisements and solicitations, have null DROUT, DADDR and DSDU fields.

9.3.2 Octet and bit ordering

9.3.2.1 General

Except in the DL, this standard uses most significant octet first (MSB or big-endian) transmission and documentation conventions, following the precedent set by ISO, IEC, IETF, and many others. That is:

- for multi-octet values, the most significant octet is transmitted first; and
- octet documentation shows bit 7 on the left and bit 0 on the right.

However, IEEE 802.15.4:2011 uses the least significant octet first (LSB or little-endian) conventions. That is:

- for multi-octet values, the least significant octet is transmitted first.

NOTE The IEEE specification is not entirely consistent on this point: IEEE 802.15.4:2011 security subheaders use MSB transmission and documentation conventions.

Bit transmission order within an octet is handled at the PhL. Within the DL the discussion of MSB and LSB is limited to the ordering of octets.

As a result, the DL is unavoidably mixed-endian, with some sections using big-endian and others using little-endian bit ordering.

Generally, the standard DPDU headers follow IEEE 802.15.4 conventions, as follows:

– This standard, except for DL and MAC headers, follows MSB conventions.

– Standard DL and MAC headers follow LSB conventions, with some clearly indicated exceptions in the DPDU header.

– Within the DPDU, security subheaders follow MSB conventions, following the IEEE 802.15.4:2011 precedent.

– All fields within the DPDU header are documented showing bit 7 on the left and bit 0 on the right, following the convention of this standard.

– DLMO attributes, accessible to the system manager through the DMAP, are AL information, and as such generally use MSB conventions. Some exceptions are made for fields that interact directly with DPDU headers that use the LSB convention.

LSB octet ordering is explicitly noted in various parts of the DL specification. By convention, octet 0 is the least significant octet. LSB indicates that the least significant octet (octet 0) is transmitted first, and the most significant octet (octet n) is transmitted last. When not specified, the reverse ordering is used for transmission.

9.3.2.2 Extensible DL unsigned integers

The DL specification uses a construct called ExtDLUint for compressed transmission of unsigned integers. This is not a type used in other standards, and as such ExtDLUint only appears in DPDU headers and within DL-defined octet strings. It is used to indicate compressed encoding of a 15-bit unsigned integer; it is not used in conveying other data. Since this type is not used outside of the DL, it is not specified as a standard AL-supported data type.

An ExtDLUint shall be transmitted as one octet when its value is in the range of 0..127, and as two octets when its value is in the range of 128..32 767, with encoding as shown in Table 108 and Table 109. Bit 0 in the first octet indicates whether one or two octets are transmitted. Octet ordering is always as shown here, with the size indicated in bit 0 of the first octet transmitted.

Table 108 – ExtDLUint, one-octet variant

Octets	Bits							
	7	6	5	4	3	2	1	0
1	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Selection =0

Table 109 – ExtDLUint, two-octet variant

Octets	Bits							
	7	6	5	4	3	2	1	0
1	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Selection =1
2	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7

9.3.3 Media access control headers

9.3.3.1 General

This standard uses a MAC header format whose component data structures are compliant with the detailed structure and field coding of IEEE 802.15.4:2011, 5.2.1 and 5.2.2.2, followed by extensions that are particular to this standard. Only the Data DPDUs of IEEE 802.15.4:2011, 5.2.2.2 are used by this standard.

This standard does not use IEEE 802.15.4:2011 security. Instead, similar security is handled in the DMXHR. DPDU security in this standard is similar to IEEE 802.15.4:2011 security; the main difference is that this standard incorporates the DLE’s shared sense of time and other usage-context-specific information in the cryptographic nonces, resulting in more compact DPDUs and improved resistance to replay and misdirection attacks. To facilitate that improved resistance to attack, the DPDU sequence numbers of this standard are derived from different content sources than that specified by IEEE 802.15.4:2011, 5.2.1.2.

The ACK/NAK DPDUs of IEEE 802.15.4:2011, 5.2.3 cannot be reliably distinguished from a similarly-timed identical DPDU sent by a device that is not the intended recipient. Such indistinguishability can arise due to concurrent activity on the same physical channel by other devices than the intended recipient, for example in other nearby networks, or due to an attacker’s deliberate spoofing of the ACK/NAK DPDU after jamming reception of the Data DPDU of the transaction. Therefore this standard uses short, authenticatable Data DPDUs to implement similar but extended ACK/NAK functionality. When the destination and source MAC addresses of such ACK/NAK DPDUs are those of the source and destination MAC addresses, respectively, of the soliciting Data DPDU, there is no need to convey those addresses explicitly. Thus this standard permits such ACK/NAK DPDUs to suppress both MAC address fields, which contradicts the constraint of IEEE 802.15.4:2011, 5.2.1.1.8.

NOTE Although the amendments of IEEE 802.15.4:2012 were intended to cover similar issues to those that led to the just-described variances, they often do so in ways that are incompatible with the ANSI/ISA 100.11a standard on which this international standard is based, and thus are also incompatible with this standard, which strives to maintain compatibility with deployed equipment that uses that ANSI/ISA standard.

9.3.3.2 Media access control header

The format of the subset of the standard MHR specified in IEEE 802.15.4:2011, 5.2.1 and IEEE 802.15.4:2011, Figure 35, as used by this standard, is summarized in Table 110.

Table 110 – Data DPDU MHR

Number of octets	**bits**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2	Frame control (LSB ordering)							
1	Sequence number							
0 or 2	PAN ID							
0, 2, or 8	Destination address							
0, 2, or 8	Source address							
NOTE The PAN ID, Destination address and Source address fields are each transmitted in LSB order								

The size of the MHR is usually 9 octets, including a PAN ID and two DL16Addresses, with these exceptions:

- Solicitations Data DPDUs have a null (zero-length) PAN ID and two null MAC addresses, so the MHR is 3 octets.
- Advertisement Data DPDUs have a null destination MAC address, so the MHR is 7 octets.
- Other Data DPDUs to or from an unjoined DLE have one DL16Address and one EUI64Address, so an MHR addressed to or from an unjoined DLE is 15 octets.

  NOTE 1 The default DPDU payload capacity, dlmo.MaxDsduSize, is based on an MHR size of 15 octets, providing a basis for making fragmentation decisions for unjoined DLEs.

- ACK/NAK DPDUs, which are used for immediate acknowledgments (short control signaling) have a null PAN ID, a null destination MAC address, and a source MAC address that is either
  – null (zero-length), so the MHR is 3 octets; or
  – a DL16Address, so the MHR is 5 octets; or
  – an EUI64Address, so the MHR is 11 octets.

  NOTE 2 The PAN ID is suppressed because the ACK/NAK DPDU's security authentication serves to reject any ACK/NAK DPDU intended for a different device, whether on the same PAN or a different PAN.

As shown in Table 110, fields include:

a) Frame control. For this field, subfields are as specified in IEEE 802.15.4:2011, 5.2.1.1:
   – Frame Type shall be Data.
   – Security Enabled shall be FALSE, because IEEE 802.15.4:2011 is not used.

     NOTE 3 The extended security of this standard is handled in the DMXHR.

   – Frame Pending shall be FALSE.
   – AR (ack request) shall be FALSE.

NOTE 4 The readily-spoofed IEEE 802.15.4:2011 immediate-acknowledgement DPDU type is not used by this standard.

– When both a destination D-address and a source D-address are included, PAN ID compression shall indicate that the same PAN ID is used for both D-addresses. Otherwise, PAN ID compression shall be FALSE (because such compression only applies when there are two D-addresses).

– MAC addresses are usually DL16Addresses, with exceptions as described below. EUI64Addresses are used by a DLE when joining a D-subnet. Destination addresses are omitted in dedicated advertisements and in solicitations.

– The frame version shall be 0x01.

b) Sequence number. Used by the DSC, as described in 7.3.2.4.10.

NOTE 5 IEEE 802.15.4:2011 requires that each DLE increment its sequence number after each use, so that the sequence number is unique for all Data DPDUs and ACK/NAK DPDUs originated by that DLE. However, this standard uses the “sequence number” field for a somewhat different purpose and so provides an alternate method (other than cyclic sequentiality) of ensuring that cryptographic nonces generated by each real device are unique within the operational lifespan of the cryptographic key with which they are employed.

NOTE 6 In this standard both the DLE and the TLE generate nonces. The provisions refered to in NOTE 5 ensure that the two sets of generated nonces are disjoint.

c) PAN ID shall match dlmo.SubnetID and shall be absent in solicitation Data DPDUs. If the DPDU conveys both a source and a destination MAC address, the PAN ID shall be the destination’s PAN ID (with the source PAN ID inferred to be identical). If there is no destination address, such as in an advertisement Data DPDU, the PAN ID shall be a source PAN ID. In a solicitation Data DPDU, where there is neither a source nor a destination address, this field shall be null (elided). This field shall be null (elided) in all ACK/NAK DPDUs. See 9.1.10.2.

d) Destination address is normally a DL16Address alias for an IPv6Address. An EUI64Address shall be used to address DLEs that have not yet received a DL16Address. The destination address shall be absent in dedicated advertisement Data DPDUs, in solicitation Data DPDUs, and in ACK/NAK DPDUs.

e) Source address is normally a DL16Address alias for an IPv6Address. An EUI64Address shall be used to identify DLEs that have not yet received a DL16Address. The source D-address shall be absent in solicitation Data DPDUs, and in ACK/NAK DPDUs where the D-address would be identical to the destination D-address of the received Data DPDU that initiated the transaction.

9.3.3.3 Data DPDU subheader

The structure of the DHDR for a Data DPDU is shown in Table 111.

Table 111 – Data DPDU DHDR

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	ACK/NAK DPDU expected	Request signal quality in ACK DPDU	Request EUI64Address in ACK DPDU	Include DAUX	DMXHR includes slow-channel-hopping-offset	Clock recipient	DL version Always 01	

This DHDR is always 1 octet.

As shown in Table 111:

- Bit 7 indicates whether ACK/NAK DPDUs are expected from the explicitly or implicitly addressed recipients.

- Bit 6 indicates whether the receiving DLE should report signal quality information in the ACK/NAK DPDU.
- Bit 5 indicates whether the receiving DLE should include its EUI64Address in the ACK/NAK DPDU. This setting shall be used by the sender whenever an acknowledgment is requested (bit 7 value of 1), and no EUI64Address for the neighbor exists in dlmo.Neighbor. See 9.1.10.1.

NOTE Bit 7 is meaningful only for the initial Data DPDU of a transaction. Bits 6 and 5 are meaningful only when Bit 7 is meaningful and has the value TRUE.

- Bit 4 indicates the presence or absence of a DAUX subheader in the Data DPDU.
- Bit 3 indicates whether a slow-channel-hopping-offset is included in the DMXHR. This value shall be included in unicast Data DPDUs where slow-channel-hopping is used. See 9.1.9.2.4.
- Bit 2 indicates whether the transmitting DLE is a DL clock recipient. This is an implicit request to the receiver to include a clock correction in the acknowledgment.
- Bits 0..1 indicates the DL version number. A value of 0x01 shall be used, with 0x10 being reserved for future use. A value of 0x11 is used in the same location in an ACK/NAK DPDU and helps to distinguish a Data DPDU from an ACK/NAK DPDU (see 9.3.4).

9.3.3.4 DPDU MAC extension subheader

A DMXHR following the DHDR is summarized in Table 112.

Table 112 – Data DPDU DMXHR

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Security control							
1	Crypto Key Identifier							
0..2	Slow-channel-hopping-offset (ExtDLUint)							

NOTE For future PHYs with more than 16 channels, it is likely that the channel number will be added as a virtual field. This is a subject of future standardization, but has been considered in the DMXHR design.

The size of the DMXHR is 2..4 octets. A DMXHR size of 3 octets, corresponding to a slow-channel-hopping rate of about 1,25 s or less, was used to calculate the default dlmo.MaxDsduSize.

As shown in Table 112, attributes include:

- Security control and Crypto Key Identifier. The security fields are left unspecified by the DL, and are set by the DSC. See 9.1.12 for an overview of the relationship between the DL and DSCs. While IEEE allows a Crypto Key Identifier as large as 9 octets, in this standard its size is always 1 octet.
- The slow-channel-hopping-offset specifies the timeslot offset into the slow-channel-hopping period, if necessary to unambiguously identify a timeslot (because the transaction initiator and responder have different local perceptions of the proper timeslot). The presence or absence of this field is indicated in the DHDR. The slow-channel-hopping-offset is described in 9.1.9.4.9.

9.3.3.5 DPDU auxiliary subheader

The DAUX subheader is used for:

- DL neighbor discovery;
- temporarily activating links;

- reporting received signal quality in acknowledgments.

The DAUX subheader, present only when bit 4 of the DHDR octet is set, is described in 9.3.5.

A DAUX size of 0 octets was used to calculate the default dlmo.MaxDsduSize.

NOTE dlmo.MaxDsduSize is used to make fragmentation decisions. The DAUX subheader is usable to activate links in a fragmentation scenario. However, link activation is not possible during the join process, and as such link activation is never combined with EUI64Addresses. Since the calculation of dlmo.MaxDsduSize includes one EUI64Address, it allows for a DAUX link activation subheader when an EUI64Address is not present.

9.3.3.6 DPDU Routing subheader

There are two variants of the DROUT subheader. A compressed variant, 2 octets in size, is used when a single graph is used for addressing. The compressed variant is also used when single-hop routing is used, with the route being implicit in the MAC-level addressing found in the IEEE 802.15.4:2011 MHR. When a series of addresses is needed, an uncompressed variant of the DROUT subheader shall be used.

The DROUT subheader shall be elided in a Data DPDU that has no higher-layer payload, as indicated by a DSDU of zero size.

The compressed variant of the DROUT subheader shall be used in the common case where a single graph, with an index of 255 or less, is used for routing. It is shown in Table 113.

Table 113 – DROUT structure, compressed variant

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Compress=1	Priority				DlForwardLimit		
0..1	DlForwardLimitExt							
1	GraphID							

A DROUT size of 2 octets was used to calculate the default dlmo.MaxDsduSize. MaxDsduSize normally needs to be reduced when source routing is used.

As shown in Table 113, the compressed variant of the DROUT comprises:

- Compress. If this value is set to 1, the compressed variant of the DROUT format shall be used.
- Priority. This shall be set to the Data DPDU's 4-bit priority.
- DlForwardLimit and DlForwardLimitExt (forwarding limit) limit the number of times that a Data DPDU may be forwarded within a D-subnet. If the forwarding limit is less than 7, the value shall be transmitted in DlForwardLimit and DlForwardLimitExt shall be elided. If the forwarding limit is greater than or equal to 7, DlForwardLimit shall be transmitted as 7, and the forwarding limit shall be transmitted in DlForwardLimitExt.

  The forwarding limit is initialized by the DL when the route is selected, based on the value of dlmo.Route[ ].ForwardLimit. When a unicast Data DPDU is successfully received by the DL and needs to be forwarded, the Data DPDU shall be discarded if its forwarding limit is zero. If its forwarding limit is positive, the forwarding limit shall be decremented (possibly to zero) and the Data DPDU shall be placed on the message queue.
- GraphID (8 bits). GraphIDs compliant with this standard are 12-bit unsigned integers. In the common case where the route is a single graph ID in the range of 1..255, the compressed variant of the DROUT subheader shall be used. Additionally, the compressed variant is used in single-hop source routing, wherein GraphID=0 shall indicate that the destination is one hop away. Since the single hop destination address can be found in the

MHR, it does not need to be repeated in DROUT. GraphID=0 shall be used during the join process for addressing to and from a neighboring proxy, and is the only way in this standard to indicate a destination EUI64Address in DROUT.

NOTE It is possible for a system manager to configure a circular D-route, with the Data DPDU being forwarded until the ForwardLimit decrements to zero. The LH field in the DL header, described in 9.3.3.7, is not intended to prevent circular routes within a D-subnet.

The uncompressed variant of the DROUT subheader is shown in Table 114.

Table 114 – DROUT structure, uncompressed variant

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Compress=0	Priority)				DlForwardLimit		
0..1	DlForwardLimitExt							
1	N (number of entries in routing table)							
2*N	Series of N GraphIDs/addresses (Unsigned16, LSB)							
NOTE Each GraphID/address is transmitted in LSB order								

As shown in Table 114, the uncompressed variant of the DROUT subheader comprises:

- Compress. If this value is set to 0, the uncompressed variant of the DROUT format shall be used.
- Priority, DlForwardLimit, and DlForwardLimitExt are as described above with Table 113.
- N. This field shall be set to the number of entries in Route. The entries may be a combination of GraphIDs and DL16Addresses. The value of N shall not exceed 15.
- Route. This field shall be set to a series of GraphIDs and/or DL16Addresses, specifying the route, in order, along which the Data DPDU will travel. IETF RFC 4944 limits unicast address ranges to $1..2^{15}-1$. 12-bit GraphIDs in this field shall be represented disjointly from that address range as 0x 1010 gggg gggg gggg, which is the range $10\times2^{12}..11\times2^{12}-1$.

When source routing is used, the DROUT subheader shall be shortened by the DL of intermediate routers as the Data DPDU proceeds along the route, as described in 9.1.6.

The first entry in the DROUT subheader is used to determine the next hop. For example, the route may be specified at the source as <000 123, 000 456, 000 789>. The first hop address, <000 123>, is used to send the Data DPDU to an immediate neighbor. The DROUT subheader, as received by DLE <000 123>, contains the source route <000 123, 000 456, 000 789>. When received, this route is shortened to <000 456, 000 789> (see 9.1.6.3), indicating that address <000 456> is the next hop.

When a graph is specified as the first entry in a source route, the Data DPDU shall follow that graph until it is terminated, as described in 9.1.6.

9.3.3.7 Addressing subheader

The addressing subheader (DADDR) includes NL source and destination addresses, along with three NL fields that are visible to the DL.

The DADDR subheader shall be elided in a Data DPDU that has no higher order payload, i.e., a DSDU of zero size.

The structure of the DADDR subheader is shown in Table 115.

Table 115 – DADDR structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DE	LH	ECN		Reserved=0			
1..2	SrcAddr (ExtDLUint)							
1..2	DestAddr (ExtDLUint)							

A DADDR size of 4 octets was used to calculate the default dlmo.MaxDsduSize, reflecting an assumption that one or the other of the addresses is encoded in one octet.

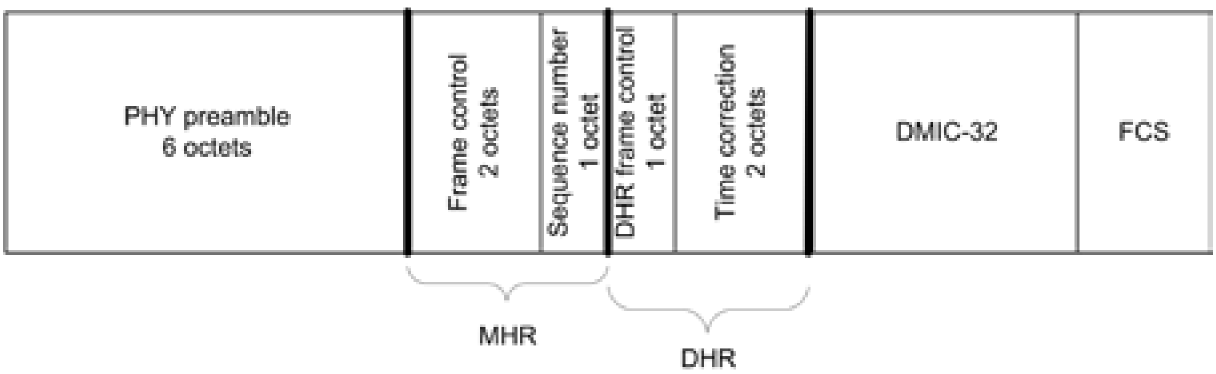
Fields include:

- Discard eligible (DE) shall be set based on the value provided by the NL through DD-DATA.request. DE=1 indicates that the DSDU is eligible to be discarded from the message queue in favor of an incoming Data DPDU with DE=0, and of equal or higher priority.
- Last hop (LH) shall be set based on the value provided by the NL through DD-DATA.request. This bit is carried by the DL to avoid circular routes at the NL, and does not affect DL behavior.
- Explicit congestion notification (ECN) shall be set based on the value provided by the NL through DD-DATA.request. A router experiencing congestion may set ECN as described in IETF RFC 3168. See 9.1.9.4.5 for a discussion of ECN.
- SrcAddr is set based on the value provided from the NL through DD-DATA.request. If the source D-address is duplicated in the MHR source D-address field, SrcAddr shall be encoded as 0x00. This covers the case, during the join process, where the source address is an EUI64Address.
- DestAddr is set based on the value provided from the NL through DD-DATA.request. If the destination D-address is duplicated in the MHR destination address field, DestAddr shall be encoded as 0x00. This covers the case, during the join process, where the destination D-address is an EUI64Address.

By encoding a duplicated DL-address as zero, an octet is compressed on the first and last hop when the address is less than 0x0080, saving energy. Thus a DADDR address encoded as zero references the corresponding DL16Address or EUI64Address in the MHR.

9.3.4 MAC acknowledgment DPDUs

Figure 89 illustrates the structure of ACK/NAK DPDUs.



NOTE This figure includes only the most commonly used fields.

Figure 89 – Typical ACK/NAK DPDU layout

The source address of an ACK/NAK DPDU is the address of the DLE that transmits the DPDU. The destination address is the address of the intended recipient of the DPDU.

Every ACK/NAK DPDU shall be authenticated with a DMIC, but not encrypted. Some fields are virtual, used in creating the DMIC but not actually transmitted.

Although the ACK/NAK DPDU is an IEEE 802.15.4:2011 data frame, it can be distinguished from other such IEEE 802.15.4:2011 data frames based on:

– the ACK/NAK DPDU’s timing, following Data DPDU transmission, as specified in 9.4.3.3; and
– bits 1..0 of its DHR frame control field, as specified in Table 118; and
– a virtual field in the ACK/NAK DPDU that echos the Data DPDU’s original DMIC, as specified in Table 117.

As described in 7.3.2.2, the DMIC in an ACK/NAK DPDU uses the same security policy as the original Data DPDU of the D-transaction to which it is a response, with the exception that the ACK/NAK DPDU’s DMIC size shall always be 32 bits regardless of the Data DPDU’s security policy.

The format of an IEEE 802.15.4:2011 MHR is summarized in Table 116.

Table 116 – ACK/NAK DPDU MHR

Number of octets	Bits							
	7	6	5	6	3	2	1	0
2	Frame control (LSB ordering)							
1	Sequence number							
0	Destination address (null)							
0 or 2	PAN ID							
0, 2, or 8	Source address							
NOTE The PAN ID and Source address fields are each transmitted in LSB order								

The detailed description of these fields is specified in IEEE 802.15.4:2011. As shown in Table 116, these attributes include:

- Frame control attributes for ACK/NAK DPDUs, as follows:
  – Frame type shall be data.
  – Security shall be disabled, as it is handled in the DHR.
  – Frame pending shall be FALSE.
  – Ack.Request shall be FALSE.

    NOTE 1 The above bit requests generation of the unsecurable form of immediate acknowledgment offered by IEEE 802.15.4:2011, which is not used by this standard.

  – Source addressing mode shall be 0x00 (i.e., implicit), except for cases described below where the PAN ID and source address are included in the MHR.
  – Destination addressing mode shall be 0x00 (i.e., implicit).
  – Frame version shall be 0x01.
- Sequence number, used by the DSC, as described in 7.3.2.4.10. As this standard does not use the unsecurable ACK frame type specified by IEEE 802.15.4:2011, its ACK/NAK DPDU does not carry the sequence number of its preceding Data DPDU. Rather the sequence number shall itself be similar to that of a Data DPDU, as specified in 9.3.3.2, and shall be used in construction of the D-nonce for the ACK/NAK DPDU’s DMIC.

- PAN ID, present only when the source address is present (non-null).
- Source address. Normally, a source D-address is not included in an ACK/NAK DPDU, because it matches the destination D-address of the last-received Data DPDU. However, there are two exceptions where it is included:
  - An immediate acknowledger of a received Data DPDU shall include its EUI64Address as the source address of the replying ACK/NAK DPDU's MHR when so requested in the received Data DPDU's DHDR.
  - An immediate acknowledger of a received Data DPDU whose D-address is different than the destination D-address of the last-received Data DPDU shall include its DL16Address as the source address of the replying ACK/NAK DPDU.

  NOTE 2 This second exception occurs in secondary duocast and N-cast acknowledgments.

A prototype DHR following a MHR is summarized in Table 117.

Table 117 – ACK/NAK DPDU DHR

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	ACK/NAK DPDU DHDR							
4 (virtual)	Echoed DMIC of received Data DPDU							
0, 2	Time correction (Unsigned16, LSB) when requested							
0..2	Timeslot offset (ExtDLUint) when needed							
0..3	DAUX subheader usually absent							

As shown in Table 117, attributes include:

- The ACK/NAK DPDU's DHDR is described in Table 118.
- Echoed DMIC of received Data DPDU. For a discussion of handling of this virtual field, see 7.3.2. To unambiguously connect the ACK/NAK DPDU with the Data DPDU to which it is a response, the DMIC of the Data DPDU is included in the ACK/NAK DPDU's DHR as a virtual field, with octet ordering matching the Data DPDU's DMIC. This virtual field is used to calculate the ACK/NAK DPDU's DMIC, but not transmitted. If the received DMIC is longer than 4 octets, only the initial (leftmost) 4 octets of the DMIC are echoed as a virtual field.
- Time correction (LSB). Used by DL clock sources to correct the time of the DL clock recipient, if it is requested in the received DPDU's DHDR. This 2-octet unsigned value, when included in the ACK/NAK DPDU, echoes the time that the Data DPDU was received. The value, in 2^{-20}s (approximately 0,954 μs), reports an offset from the scheduled start time of the current timeslot in the acknowledger's time base. The reported value is based on the Data DPDU's start time. See 9.1.9.3.2.
- Acknowledger's timeslot offset is provided, when needed, within a slow-channel-hopping period. This value, when included in the ACK/NAK DPDU, indicates the current timeslot in the acknowledger's time base. It shall be included only when the received Data DPDU is received in a different slow-channel-hopping timeslot than is used for the acknowledgment. The first timeslot in a slow-channel-hopping period has an offset of zero. When the corrected timeslot offset is non-zero, the time correction (previous field), when included, shall be an offset of the corrected scheduled timeslot time, Security requires that a DLE's time increases from timeslot to timeslot. Therefore, if the timeslot is corrected to an earlier timeslot by a clock recipient, there shall be an interruption in service, equal to the magnitude of the timeslot correction plus at least one timeslot. See 9.1.9.4.9.
- Auxiliary subheader (DAUX). DAUX may be included in an ACK/NAK DPDU, for the limited purpose of echoing received signal quality (see 9.3.5.5).

In an ACK/NAK DPDU, the DHDR octet communicates the ACK/NAK type and other DPDU substructure information, as shown in Table 118.

Table 118 – ACK/NAK DPDU DHDR

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Clock correction included	Slow-channel-hopping-offset included	ACK/NAK type: 0: ACK 1: ACKwithECN 2: NAK0 3: NAK1		DAUX subheader included	Reserved (=0)	ACK/NAK DPDU (=11)	

The DL protocol version number and MAC security key always match those of the received Data DPDU to which the ACK/NAK DPDU is an immediate acknowledgment, and therefore are not included explicitly in the ACK/NAK DPDU.

Bit content is as follows:

- Bit 7 indicates whether the ACK/NAK DPDU includes clock correction information.
- Bit 6 indicates whether the ACK/NAK DPDU includes a slow-channel-hopping-offset.
- Bits 5..4 indicate the class of the ACK/NAK DPDU:

  0b10: a NAK0 negative acknowledgment, signaling that the Data DPDU was received but could not be acknowledged due to message queue congestion (9.1.9.4.4);

  0b11: a NAK1, signaling that the Data DPDU was received but was not accepted due to a recent history of forwarding problems along the route (9.1.9.4.4).

  0b00: an ACK positive acknowledgment;

  0b01: an ACK positive acknowledgment with an ECN (explicit congestion notification) (9.1.9.4.5).

  A router that is signaling ECN in the forward direction should also signal the ECN through ACK/NAK DPDUs when the Data DPDU's priority is 7 or less. A DLE receiving an ECN through an ACK/NAK DPDU may treat this signal as early notification that it is likely to receive an ECN at upper layers;
- Bit 3 indicates whether the ACK/NAK DPDU includes a DAUX subheader, which may be included in an ACK/NAK DPDU for the limited purpose of reporting received signal quality.
- Bit 2 is reserved and shall be set to zero.
- Bits 1..0 are set to ones (11) to distinguish ACK/NAK DPDUs from other DPDUs.

9.3.5 DL auxiliary subheader

9.3.5.1 General

An auxiliary subheader (DAUX) may be included in any Data or ACK/NAK DPDU. Bits 7..5 of the first octet of the DAUX determine its type, with the subsequent subheader format different for each type. Defined types are:

- Advertisement: type 0 in Data DPDU: Provides information needed by new DLEs to synchronize with and join the D-subnet;
- Solicitation: type 1 in Data DPDU: Solicits an advertisement from a neighboring DLE;
- Activate link: type 2 in Data DPDU: Activates an idle link for a period of time;
- Signal quality: type 3 in ACK/NAK DPDU: Reports received signal quality.

All other combinations of type and DPDU-class are reserved for future use.

NOTE Following DL header conventions, DAUX fields use LSB (little-endian) order for transmission. There are some similar structures in the DLMO that use MSB (big-endian) order. For example, superframe structures are specified in both places, using LSB in the DL header and MSB in DLMO attributes.

9.3.5.2 Advertisement auxiliary subheader

9.3.5.2.1 General

Fields within an advertisement DAUX can be grouped logically as:

- advertisement selections;
- time synchronization;
- superframe information;
- join information; and
- integrity check.

Table 119 summarizes the structure of the advertisement DAUX.

Table 119 – Advertisement DAUX structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Advertisement selections; see Table 120							
6	Time synchronization; see Table 122							
6..10	Superframe information; see Table 124							
4..10	Join information; see Table 127							
2	Integrity check; see 9.3.5.2.4.4							
NOTE As described in 9.3.5.2.4.2, join information field size is limited to 10 octets.								

The advertising router's D-subnet ID and DL16Address are conveyed through the MAC sublayer, and do not need to be transmitted redundantly within the DAUX.

An advertisement DAUX may be included within a Data DPDU, but shall not be included within an ACK/NAK DPDU.

An advertisement includes information that enables the receiving DLE to create superframes and links to be used during the join process. This information shall be retained by the DLE at the end of the join process and, along with DL defaults, constitute a starting database of link scheduling information for the DLE. The same links used for joining are temporarily used for general communications until the system manager provides an alternative configuration.

Attributes set by the DLE based on information in the received advertisement include:

- dlmo.SubnetID is set based on the SubnetID in the advertisement.
- TAI time is synchronized by the advertisement.
- dlmo.Superframe number1 is created with fields copied from the advertisement.
- dlmo.Link number1 is created as a transmit link with fields copied from the advertisement.
- dlmo.Link number2 is created as a receive link with fields copied from the advertisement.
- dlmo.Link number3 may be created as passive scanning receive links with fields copied from the advertisement if provided.
- dlmo.Neighbor is initialized by the DLE with an entry corresponding to the advertising router.

- dlmo.Graph number1 is automatically created by the DLE, to provide access to the advertising router.
- dlmo.Route number1 is automatically created by the DLE as the default route using graph number1.

9.3.5.2.2 Advertisement selections

Table 120 specifies the advertisement selections field in the advertisement DAUX.

Table 120 – Advertisement selections elements

Element name	Element encoding
DauxType	Type: Unsigned3 0=advertisement DAUX
ChMapOv	Type: Unsigned1 0=default
DauxOptSlowHop	Type: Unsigned1 0=default
Reserved (octet alignment)	Type: Unsigned3=0

Table 121 illustrates the structure of the advertisement selections field.

Table 121 – Advertisement selections

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DauxType=0			DauxChMapOv	DauxOptSlowHop	Reserved=0		

The advertisement selections field is 1 octet. As shown in Table 120, attributes include:

- DauxType. Always set to 0 for an advertisement DAUX. Indicates the DAUX structure in Table 119.
- DauxChMapOv. TRUE indicates that the DauxChMap field is included in the advertisement DPDU. FALSE selects the default channel map of 0x7FFF. This field corresponds to dlmo.Superframe[ ].ChMapOv.
- DauxOptSlowHop. TRUE indicates that D-subnet offers slow channel-hopping, and that DauxChRate is included in the advertisement DPDU. FALSE indicates the default of slotted-channel-hopping.

  NOTE Slow-channel-hopping can be used during the join process as well as thereafter.

- Bits 2..0 are reserved and shall be set to 0.

9.3.5.2.3 Advertisement time synchronization

Table 122 specifies the time synchronization field in the advertisement DAUX.

Table 122 – Advertisement time synchronization elements

Element name	Element encoding
DauxTAIsecond (current TAI time)	Type: Unsigned32 (LSB) Units: 1 s
DauxTAIfraction (fractional TAI second)	Type: Unsigned16 (LSB) Units: 2^{-15} s

Table 123 illustrates the structure of the advertisement time synchronization field.

Table 123 – Advertisement time synchronization structure

Octets	Bits								
	7	6	5	4	3	2	1	0	Interpretation
1	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	DauxTAIsecond; Integral part of TAI time with granularity of 1 s
2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
3	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
4	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
5	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	DauxTAIfraction; Fractional part of TAI time with granularity of 2^{-15} s
6	0 reserved	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	
NOTE The above representation is radically different from that of TAINetworkTime (Table 362), with the octet ordering reversed in the first four octets, and the ordering both reversed and shifted one bit in the last two octets.									

NOTE 1 The DauxTAIfraction unit of 2^{-15} s was chosen to match the 32 KiHz very-precise very-low-power “watch” crystals commonly used for the continuous clock hardware of WISN devices.

The time synchronization field is 6 octets. As shown in Table 122, subfields include:

– DauxTAIsecond. Current TAI time in units of 1 s.

– DauxTAIfraction. Fractional TAI second in units of 2^{-15} s, with a range of 0..32 667. Within the TAI second, this indicates the advertisement DPDU’s actual start time. (An implementation that actually clocks based on SFD timing should account for a DPDU start time that is nominally 1 octet, or 32 μs, later than the time that the SFD is completely transmitted/received.)

NOTE 2 Although TAI time is normally represented as a 6-octet scaled fixed point binary fraction, modulo 2^{32} s, the above two-part transmittal ordering, where the each part is transmitted separately LSB first and the fractional part has an inserted unused bit at the binary point, does not honor the natural octet ordering of that scaled fixed point fraction.

See 9.1.9 for more information on TAI time and timeslot alignment.

The identity and scheduled timing of the current timeslot can be derived from the Data DPDU’s actual start time combined with the join superframe description. See 9.1.9.1.5.

The time in an advertisement shall be an accurate reflection of the advertiser’s internal TAI clock to within ±96 μs (i.e., the transmission duration of 3 octets), which is the transmission window jitter presumed for devices conforming to this standard.

An ACK/NAK DPDU to a join request includes a clock correction that may be more precise than the original advertisement, and also more current.

9.3.5.2.4 Advertisement join superframe and links

9.3.5.2.4.1 Advertisement join superframe

NOTE The join process, including solicitation and advertisements and use of the information conveyed in advertisements, is described in 7.4.

There are three links specified by the advertisement related to neighbor discovery:

- link number1 for sending join requests, addressed to the neighboring advertising router;
- link number2 for receiving subsequent join responses from the advertising router; and
- link number3 for scanning for additional neighbors after the DLE successfully joins the D-subnet.

All of these links refer to superframe number1, which is also specified in the advertisement.

Field names in the advertisement correspond to equivalent fields in dlmo.Superframe and dlmo.Link. Following DL header conventions, LSB octet ordering is used on certain fields that are transmitted using MSB ordering in the superframe itself. To minimize processing requirements and to compress the DAUX subheader, a subset of superframe and link features is supported through the advertisement.

Table 124 specifies the join superframe information field.

Table 124 – Join superframe information subfields

Subfield name	Subfield encoding
DauxTsDur (timeslot duration)	Type: Unsigned16 Units: 2^{-20} s
DauxChIndex (channel-hopping pattern ID)	Type: ExtDLUint Valid range: 1..5
DauxChBirth (channel-hopping reference starting point)	Type: Unsigned8
DauxSfPeriod (number of timeslots in each superframe cycle)	Type: ExtDLUint
DauxSfBirth (superframe cycle starting point)	Type: ExtDLUint Valid range: 0..127
DauxChRate (length of each slow-channel-hopping period, in number of timeslots)	Type: Unsigned8 Not transmitted and defaults to 1 when DauxOptSlowHop is FALSE
DauxChMap (channel-hopping channel map for spectrum management)	Type: Unsigned16 (LSB) Not transmitted and defaults to 0x7FFF when advChMapOv is FALSE

Table 125 summarizes the structure of the join superframe information field in the advertisement DAUX. ExtDLUint fields are shown as one octet.

Table 125 – Join superframe information structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	DauxTsDur (LSB)							
1	DauxChIndex							
1	DauxChBirth							
1..2	DauxSfPeriod							
1	DauxSfBirth							
0..1	DauxChRate							
0..2	DauxChMap (LSB)							

The join superframe information field is 6..10 octets.

Each subfield of advertisement join superframe information corresponds to a field in dlmo.Superframe. See 9.4.3.5.

When creating superframe number1 from the advertisement, the DL uses values from the advertisement to initialize corresponding superframe fields. Fields in the superframe number1 that do not have equivalently named fields in the advertisement default to fixed values. Table 126 shows the mapping from the advertisement's subfields to superframe number1.

Table 126 – Superframe derived from advertisement

Superframe field name	Value	Notes
* Index	1	—
TsDur	DauxTsDur	—
ChIndex	DauxChIndex	—
ChBirth	DauxChBirth	—
SFType	0	—
Priority	0	—
ChMapOv	DauxChMapOv	—
IdleUsed	0	—
SfPeriod	DauxSfPeriod	Compressed data type used in advertisement. Limits superframes used for joining to a period of approximately 300 s
SfBirth	DauxSfBirth	Compressed data type used in advertisement, consistent with SfPeriod
ChRate	DauxChRate	Compressed data type used in advertisement. Limits superframes used for joining to a slow-channel-hopping rate of approximately one hop per 2,5 s
ChMap	DauxChMap	Convert LSB to MSB
IdleTimer	null	—
rndSlots	null	—

9.3.5.2.4.2 Advertisement join links

NOTE 1 The join process, including solicitation and advertisements and use of the information conveyed in advertisements, is described in 7.4.

There are two sets of links related to joining provided in every advertisement:

- an outbound set of links for transmitting join requests, used to initialize dlmo.Link number1; and
- an inbound set of links for receiving join responses, used to initialize dlmo.Link number2;

in addition, the advertising router may provide a set of links that a DLE can use to scan for advertisements when joining is complete, used to initialize dlmo.Link number3 when provided.

Each device that is attempting to join a subnet, upon receiving an advertisement of a D-subnet that it chooses to join, shall configure its inbound and outbound links based on the information in the received advertisement. It shall then transmit its join request to the advertising router, using those outbound links.

Links used for joining are constrained to a basic set of features. Default timeslot templates are used; see Table 165, Table 166, and Table 167.

Three types of links are identified:

– JoinTx links are used for transmitting join requests to the advertising router;

– JoinRx links are monitored while waiting for a join response;

– AdvRx links, when provided, are activated when joining is complete to passively scan for advertisements from alternative routers.

Table 127 specifies the join information field in the advertisement DAUX.

Table 127 – Join information elements

Element name	Element encoding
DauxJoinBackoff (maximum extent of backoff and retry while joining)	Type: Unsigned4
DauxJoinTimeout (join timeout)	Type: Unsigned4
DauxJoinFldXmit (indicates fields that are transmitted)	Type: Unsigned8
DauxJoinTx (JoinTx link(s))	Type: See Table 184
DauxJoinRx (JoinRx link(s))	Type: See Table 184
DauxAdvRx (Advertisement link(s))	Type: See Table 184 (or null)

The element DauxJoinFldXmit selects the link scheduling parameters used for elements DauxJoinTx, DauxJoinRx, and DauxAdvRx. DauxJoinFldXmit values 0..3 correspond to the semantics described in Table 184.

Table 128 illustrates the structure of the join information field. Fields DauxJoinTx, DauxJoinRx, and DauxAdvRx may be 1..4 octets (or null only for DauxAdvRx), depending on the configuration and selection as described in Table 184.

Table 128 – Join information structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DauxJoinBackoff				DauxJoinTimeout			
1	DauxJoinFldXmit							
1..4	DauxJoinTx							
1..4	DauxJoinRx							
0..4	DauxAdvRx (may be absent)							

Depending on the alternatives selected, it would appear from Table 128 that the total size could be anything between four and fourteen octets. However, alternatives shall be selected such that the total size of the fields shown in Table 128 does not exceed ten octets.

As shown in Table 128, attributes include:

a) DauxJoinBackoff. Maximum extent of exponential backoff on joining. If a join request does not receive an ACK/NAK DPDU due to CCA channel activity detection, or a failed transmission, the DLE shall back off by selecting a uniform-random time interval in the range of 0 s to 1 s for the first retry, and use the first available JoinTx timeslot after that time. Then double the time range with each retry. The DLE may retry up to DauxJoinBackoff times and shall not retry more than DauxJoinBackoff times. At that point, the DLE should abort the attempt to send a message to the advertising router, revert to the provisioned state, and search for another advertisement.

b) DauxJoinTimeout. Guaranteed time, in s, to receive a system manager response to a join request. Expressed as an exponent, to the power of 2. For example, if DauxJoinTimeout=5, then the DLE can expect completion within 2^5 s (=32 s). Following a timeout, the DLE should abort the attempt to join through the advertising router, revert to the provisioned state, and search for another advertisement.

c) DauxJoinFldXmit:

   – Bits 7..6: Unsigned2, describing contents of DauxJoinTx. See Table 184. Corresponds to dlmo.Link[ ].SchedType for link number1. Supported range is 0..2.

   – Bits 5..4: Unsigned2, describing contents of DauxJoinRx. See Table 184. Corresponds to dlmo.Link[ ].SchedType for link number2. Supported range is 0..2.

   – Bit 3: If Bit3=1, transmit DauxAdvRx. If Bit3=0, DauxAdvRx is null and not transmitted.

   – Bits 2..1: Unsigned2, describing contents of DauxAdvRx. See Table 184. Supported range is 0..2. Corresponds to dlmo.Link[ ].SchedType for link number3. If Bit3=0, Bits 2 and 1 are meaningless and shall also be 0.

   – Bit 0 is reserved and shall be set to zero.

d) DauxJoinTx. The join transmission timeslot(s) in each superframe cycle, corresponding to dlmo.Link[ ].Schedule for link number1. These are the transmission opportunities in which to send join requests. Bits 7, 6 of DauxJoinFldXmit specify the format of DauxJoinTx, as defined in Table 184.

e) DauxJoinRx, The join receive timeslot(s) in each superframe cycle, corresponding to dlmo.Link[ ].Schedule for link number2. Bits 5, 4 of DauxJoinFldXmit specify the format of DauxJoinRx as defined in Table 184.

f) DauxAdvRx. Receive links, for scanning for additional neighbors after joining, corresponding to dlmo.Link[ ].Schedule for link number3 when provided. Bits 2, 1 of DauxJoinFldXmit specify the format of DauxAdvRx as defined in Table 184. It is transmitted and meaningful only when DauxJoinFldXmit.Bit 3=1; otherwise its value is null and no corresponding links are created in Table 129.

Links are added to dlmo.Link[ ] based on parameters in the advertisement. Fields are set as shown in Table 129.

Table 129 – Defaults for links created from advertisements

Field name	DauxJoinTx	DauxJoinRx	DauxAdvRx (when DauxJoinFldXmit.Bit3 =1)
* Index	1	2	3
SuperframeIndex	1	1	1
Type-Transmit	1	0	0
Type-Receive	0	1	1
Type-Exponential Backoff	1	0	0
Type –Idle	0	0	1
Type-Discovery	0	0	0
Type-JoinResponse	0	0	0
Type-SelectiveAllowed	1	1	1
Template1	2	1	3
Template2	Null	Null	Null
NeighborType	1	0	0
Graph Type	0	0	0
SchedType	From advertisement DauxJoinFldXmit Bits 7..6	From advertisement DauxJoinFldXmit Bits 5..4	From advertisement DauxJoinFldXmit Bits 2..1
ChType	0	0	0
PriorityType	0	0	0
Neighbor	Address of advertising neighbor	Null	Null
GraphID	Null	Null	Null
Schedule	From advertisement DauxJoinTx	From advertisement DauxJoinRx	From advertisement DauxAdvRx
ChOffset	Null	Null	Null
Priority	Null	Null	Null

Link number3, intended to be used to scan for neighbors after joining, is configured as an idle link. Normally, idle links are enabled through a DAUX subheader as described in 9.3.5.4. In addition, when the DL changes to the join state, it shall activate link number3 for a period of time equal to the initial value of dlmo.DiscoveryAlert.Duration (default 60 s). This causes the DL to collect information into the dlmo.Candidates table and then report it to system manager through the NeighborDiscovery alert, unless the DL is reconfigured by the system manager during the interval for a different result.

NOTE 2 GraphType in Table 129 is set to zero, indicating that the feature is not applicable in this context. See 9.4.3.7.2.

The links created from the advertisement also need entries in dlmo.Neighbor, dlmo.Graph, and dlmo.Route. These entries are automatically added by the DL at the same time as the links, with values as shown in Table 130, Table 131, and Table 132 below.

Table 130 – dlmo.Neighbor entry created from advertisements

Field name	Value
*** Index**	**Address of advertising router**
EUI64Address	Acquired from the advertising router as described in 9.1.10.1
GroupCode	0
ClockSource	2
ExtGrCnt	0
DiagLevel	1
LinkBacklog	0
ExtendGraph	Null
LinkBacklogIndex	Null
LinkBacklogDur	Null

NOTE 3 ExtendGraph in Table 130 is set to null, indicating that the feature is not applicable in this context. See 9.4.3.4.2.

Table 131 – dlmo.Graph entry created from advertisements

Field name	Value
*** Index**	**1**
PreferredBranch	0
NeighborCount	1
Queue	0
MaxLifetime	0
Neighbors	Address of advertising router

Table 132 – dlmo.Route entry created from advertisements

Field name	Value
*** Index**	**1**
Size	1
Alternative	3
ForwardLimit	16
Route	One entry: 0xA001 (Graph number1)
Selector	Null

NOTE 4 Route information in Table 132 is intended to be used as the default route after the DLE joins the D-subnet. Join messages to the neighboring proxy use source routing as described in 9.3.3.6. Sample DL headers for join messages are provided in Annex T.

DLMO updates from DAUX join information are made at two points in the DL lifecycle. First, when a DLE in the default state receives an advertisement from a provisioning mini-D-subnet (SubnetID=1), it needs to update DLMO attributes with DAUX join information in order to join the mini-D-subnet. Second, when a DLE in the provisioned state joins a target D-subnet via an advertising router, it needs to update the DLMO with DAUX join information in order to join the target D-subnet.

There are various other times when a DL might receive and process advertisements, such as when searching for multiple candidate neighbors in the provisioned state, searching for candidate neighbors in the joined state, or receiving D-subnet time updates in any state. The receipt and processing of such advertisements may trigger a join request only in the default or provisioned state, and DAUX join information in the advertisement is posted to the recipient's DLMO only when the DLE attempts to join a mini-D-subnet from the default state or attempts to join a target D-subnet from the provisioned state.

9.3.5.2.4.3 Slotted-hopping, slow-hopping and the join process

An advertisement conveys a compressed form of a superframe definition. DauxChRate in the advertisement exactly corresponds to ChRate in the superframe, which is what distinguishes a slow-hopping superframe from a slotted-hopping one. If DauxChRate=1 the advertisement specifies a slotted-hopping superframe; when DauxChRate>1, the advertisement specifies a slow-hopping superframe.

An advertisement can specify a range of links within a superframe, per Table 184. That provides a mechanism to specify and activate a contiguous set of timeslots.

The slotted-hopping or slow-hopping specified by an advertisement DPDU is a functional subset of the slotted-hopping or slow-hopping specified by superframe and link data structures. The primary difference s that the information in the advertisement DPDU is somewhat compressed. That functional identity (for the subset) permits the information conveyed in a received advertisement DPDU to be used to initialize the superframe/link data structures in the device that is attempting the join operation.

The handling of advertisement superframes and links is described in 9.1.14. The data cross-mapping is specified in Table 126 and Table 129. The only significant difference is that, due to the compressed representation, the advertisement DPDU structure limits the slow-hopping response to 255 timeslots (typically about 2,5 s), which is noted in the description of the DPDU structure. That 2,5 s upper bound is not considered a significant limitation.

The use of slotted hopping, slow hopping and hybrid-hopping is described in 9.1.8.4.4 through 9.1.8.4.6. Figure 74 shows that slow-hopping can be combined with slotted-hopping to provide hybrid-hopping. In an advertising router configured for hybrid operation, an advertisement can instruct the joining device to use slow-hopping links, slotted-hopping links, or a combination. For example, Tx links (to the advertising router) might use slow hopping while Rx links (from the advertising router) might use slotted hopping. Such a configuration is reasonable for a router that is configured as an active scanning host (see 9.1.13.3), where slow-hopping links can perform double-duty as a vehicle for listening for solicitation DPDUs and other Data DPDUs.

The slotted-hopping or slow-hopping or hybrid-hopping pattern is defined by superframe attributes in Table 174. The mapping is shown in Table 126. The timing of the links within the slotted-hopping or slow-hopping or hybrid-hopping superframe is defined in Table 181 and Table 184. Table 127 shows the mapping to Table 181 while referring specifically to Table 184.

In a hybrid configuration, such as in Figure 74, slow-hopping links may be limited to a particular range of timeslots. That is the intended use of the "range" in Table 184. It is also possible to designate specific timeslots (links) within a slow-hopping period, not just a range of timeslots, thus providing more flexibility for the designer of an actual deployed WISN.

9.3.5.2.4.4 Integrity check

DPDUs that embed a DAUX include the IEEE 16-bit ITU-T CRC (FCS) as an integrity check on the overall MPDU, plus a DMIC for authentication as a further integrity check. However, that DMIC cannot be authenticated by a receiving DLE without a shared sense of time, a shared security key, and knowledge of the sending DLE's EUI64Address, which are not available to all DLEs that may overhear the DPDU and derive time from its DAUX subheader.

NOTE 1 Time within the DAUX is usable as a shared sense of time for authentication of the DPDU that contains the DAUX.

This standard permits a DLE to view and use the DAUX even if it cannot authenticate the overall DPDU. It was deemed insufficient to rely on the IEEE 802.15.4:2011 FCS as the only integrity check for advertisements. For this reason, an additional 16-bit integrity check is included within the DAUX, covering only the contents of the DAUX itself.

The DAUX integrity check is similar to the UDP checksum described in IETF RFC 768. The checksum is the 16-bit ones complement of the ones' complement sum of the octets that comprise the DAUX subheader, excluding the integrity check itself, padded with zero octets at the end (if necessary) to make a multiple of two octets. Octet ordering is as transmitted. If the computed checksum is zero, it is transmitted as all ones. An all-zero transmitted checksum value means that the creator of the DPDU generated no checksum.

Transmission order of the integrity check is MSB, i.e., with the first octet reflecting bit operations on odd-numbered octets, with the octet count starting at 1, in the DAUX subheader (octets 1, 3, 5, etc.) and the second octet from even-numbered octets in the DAUX subheader (octets 2, 4, 6, etc.).

NOTE 2 The use of a secret D-subnet security key for advertisements enables those advertisements to be trusted after the DLE has joined the D-subnet. Advertisements are usable for periodic surveys of neighboring routers, or for periodic time updates by DLEs with low-duty cycles.

The advertisement provides only the DL16Address of the advertising router. However, an EUI64Address is needed for subsequent exchange of DPDUs with that router. As described in 9.1.14.2, the responding DLE shall acquire the EUI64Address from the advertising DLE.

9.3.5.2.5 Configuring advertisements

NOTE The join process, including solicitation and advertisements and use of the information conveyed in advertisements, is described in 7.4.

The timing of advertisements is determined by the structure of the advertising DLE's superframes and links. Any link may include an advertisement flag, which indicates that the advertisement is included in the DAUX.

An index value in the attribute dlmo.AdvSuperframe (see Table 141) selects a superframe in dlmo.Superframe that shall be used as a reference to build the advertisement. The reference superframe is configured by the system manager by establishing a superframe, which may be idle, and referring to its index in the dlmo.AdvSuperframe attribute. The reference superframe shall not use features that cannot be represented in the join superframe information field in Table 124.

A zero value in dlmo.AdvSuperframe is the default, and indicates that the advertisement has not been configured.

Link information is placed in the dlmo.AdvJoinInfo attribute, in exactly the format in which it is transmitted in the advertisement in the position corresponding to Table 128. In this way, the new DLEs JoinTx, Join Rx, and AdvRx links are specified.

The system manager configures superframes in the advertising router that match those specified in the advertisement DPDU. At the time of JoinTx links, the advertising router shall be configured with links to receive join DPDUs. At the time of JoinRx links, the advertising router shall be configured with links where JoinResponse=1 (see Table 182).

9.3.5.3 Solicitation auxiliary subheader

9.3.5.3.1 General

NOTE The join process, including solicitation and advertisements and use of the information conveyed in advertisements, is described in 7.4.

A solicitation is a request for an advertisement to be transmitted by an active scanning host in range, on the same channel as the solicitation itself (see 9.1.13.3).

Attributes within a solicitation DAUX can be grouped logically as:

- solicitation header; and
- D-subnet ID.

The solicitation does not have a reliable sense of time, nor does it necessarily have a secret security key. Therefore, to allow the receiver of the solicitation to decode its DMIC, a solicitation's DMIC shall be built using a security key of K_global and a nominal TAI time of zero. This allows for consistent processing, and provides a strong integrity check for the DPDU. No additional integrity check is included in the solicitation's DAUX.

A solicitation's MHR (IEEE MAC header) shall not provide a source or destination address, and it shall not specify a D-subnet. See 9.1.5.

9.3.5.3.2 Solicitation fields

Table 133 specifies the solicitation header in the solicitation DAUX.

Table 133 – Solicitation header subfields

Subfield name	Subfield encoding
DauxType	Type: Unsigned3 1= solicitation DAUX
DauxSubnetInclude (indicates whether to transmit DauxSubnetID in solicitation)	Type: Unsigned1
Reserved	Type: Unsigned4=0

The solicitation header is 1 octet. As shown in Table 133, elements include:

- DauxType. Set to 1 to indicate a solicitation DAUX.
- DauxSubnetInclude. Indicates whether to transmit the DauxSubnetID field in the solicitation. If DauxSubnetInclude=0, the DauxSubnetID field is not transmitted, and the receiver (active scanning host) shall use the default value of DauxSubnetID=0 in filtering.

Table 134 illustrates the structure of the solicitation header.

Table 134 – Solicitation header structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DauxType=1			DauxSubnetInclude	Reserved=0			

Table 135 specifies the other fields in the solicitation DAUX.

Table 135 – Solicitation DAUX fields

Field name	Field encoding
DauxSubnetID (specifies the SubnetID)	Type: Unsigned16 (LSB)

DauxSubnetID transmits a D-subnet ID that can be used as a filter by the receiver, based on the receiver's dlmo.SolicFilter attribute. When DauxSubNetInclude=0, DauxSubnetID defaults to 0x0000 and is not transmitted.

Table 136 summarizes the structure of the solicitation DAUX.

Table 136 – Solicitation DAUX structure

Number of octets	bits							
	7	6	5	4	3	2	1	0
1	Solicitation header (see Table 134)							
0, 2	DauxSubnetID (LSB)							

9.3.5.3.3 Configuring solicitations

Due to regulatory and safety requirements, some applications cannot tolerate DLEs that transmit DPDUs while they are idle or in transit. Therefore, the default in this standard does not include solicitations in its configuration. The intent is that DLEs will be configured with solicitations, as appropriate, when they are provisioned or subsequently in the lifecycle.

The timing of solicitations is determined by the structure of the DLE's superframes and links. Transmission of a solicitation is triggered by a dlmo.Link[ ].Discovery field set to a value of 3.

When a solicitation is transmitted, the contents of dlmo.SolicTemplate shall be copied verbatim into the DAUX subheader. If the size of SolicTemplate is zero, this shall be interpreted as a configuration error and the link shall be ignored.

To support regulatory and safety requirements, solicitations can be enabled and disabled on a timed basis by the system manager through the attributes dlmo.RadioSilence, dlmo.RadioSleep, and dlmo.Superframe[ ].IdleTimer.

9.3.5.4 Activate link auxiliary subheader

9.3.5.4.1 General

The activate link DAUX provides a mechanism that enables a transaction initiator to activate idle timeslots for a short period of time in order to efficiently forward a backlog of messages that have accumulated in a transaction initiator's message queue. These idle links, when so configured by the system manager, are activated by the router in response to a burst of messages flowing through a DL toward a particular neighbor.

The system manager configures:

- An idle transmit link on the transmission side, addressed to a particular neighbor or group of neighbors, with a particular link index and schedule.
- An idle receiver link on the reception side, with the same link index and schedule.
- A set of parameters in the neighbor table, indicating
  - the link index (LinkBacklogIndex),
  - the size of the backlog that should trigger link activation (LinkBacklog), and
  - the duration of link activation (LinkBacklogDur).

  See 9.4.3.4.2 for the definition of these parameters.

When the transaction initiator detects that there are LinkBacklog Data DPDUs on its message queue that can be forwarded to the Data DPDU's destination address, the transaction initiator should use the activate link auxiliary subheader to activate the idle receive link through the activate link auxiliary subheader. When the transaction originator receives an ACK/NAK DPDU for the Data DPDU, that implies that the message has been processed and that the receive side of the idle link has been activated. The transaction initiator should then activate the transmit side of the idle link for the designated number of communication opportunities.

The activated transmit link might be addressed to a group of neighbors, however, the receive side of the activated link occurs on only one neighbor. Therefore, only Data DPDUs addressed to that neighbor should be considered as candidates for the activated link.

The activate link DAUX provides a link index and a number of communication opportunities that are used to activate an idle link by the receiver of the Data DPDU. It has the result of immediately activating an idle link for reception, for the number of communication opportunities indicated by DauxActivateDur. The transaction initiator of the activate link message is, in essence, informing the receiver that queued messages will be following that will be sent during communication opportunities (timeslots) associated with a particular receive link.

Activation of idle links is triggered on the transaction initiator side by multiple queued Data DPDUs that can be routed to the receiver. See 9.4.3.4.2 for a description of the transmit side of the activate link message (LinkBacklogIndex, LinkBacklogDur).

9.3.5.4.2 Fields

Table 137 summarizes the activate link DAUX.

Table 137 – Activate link DAUX fields

Field name	Field encoding
DauxType	Type: Unsigned3 2=Link activation DAUX
Reserved (octet alignment)	Type: Unsigned5=0
DauxLink_ID (identifier for link)	Type: ExtDLUint
DauxActivateDur (number of communication opportunities (timeslots) to activate the link, link occurrences)	Type: Unsigned8

The link is activated for the number of occurrences of the link, whether the link is used or not. For example, the link occurrence is counted even if it is not used because of a higher priority link in the same timeslot. The link activation period begins with the next full timeslot after the link activation DAUX is received. See 9.4.3.4.2.

Table 138 illustrates the structure of the activate link DAUX field.

Table 138 – Activate link DAUX structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DauxType			Reserved=0				
1 or 2octets	DauxLinkID							
1t	DauxActivateDur							

9.3.5.5 Signal quality auxiliary subheader

9.3.5.5.1 General

The signal quality DAUX reports the quality of the received signal in an ACK/NAK DPDU, to support collection of round-trip signal quality diagnostics. Two octets are reported, one for signal strength (RSSI) and one for signal quality (RSQI). RSSI and RSQI are described in 9.1.15.2.

9.3.5.5.2 Fields

Table 139 summarizes the report received signal quality DAUX.

Table 139 – Report received signal quality DAUX fields

Field name	Field encoding
DauxType	Type: Unsigned3 3=Signal quality DAUX
Reserved (octet alignment)	Type: Bit5=0
DauxRSSI (RSSI)	Type: Integer8
DauxRSQI (RSQI)	Type: Unsigned8

Table 140 illustrates the structure of the report received signal quality DAUX.

Table 140 – Report received signal quality DAUX structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	DauxType			Reserved=0				
1	DauxRSSI							
1	DauxRSQI							

9.4 DL management information base

9.4.1 General

For information on the general handling of standard management objects in the DL, see 9.1.11.

9.4.2 DL management object attributes

9.4.2.1 General

Table 141 summarizes the DL management object (DLMO) attributes. OctetStrings with a size of zero are referred to as null.

Table 141 – DLMO attributes

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ActScanHostFract	1	DLE’s behavior as an active scanning host	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	See 9.4.2.2
AdvJoinInfo	2	Join information to be placed in advertisement	Type: OctetString Classification: Static Accessibility: Read/write Default value: Null	See 9.4.2.3
AdvSuperframe	3	Superframe reference for advertisement	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 0 Valid range: 0..32 767	See 9.4.2.3
SubnetID	4	Identifier of the D-subnet that the DLE has joined or is attempting to join	Type: Unsigned16 Classification: Dynamic Accessibility: Read only Default value: 0	See 9.4.2.4
SolicTemplate	5	Template of DAUX subheader used for solicitations	Type: OctetString Classification: Static Accessibility: Read/write Default value: Null	See 9.4.2.5
AdvFilter	6	Filter used on incoming advertisements during neighbor discovery	Type: OctetString See Table 142 Classification: Static Accessibility: Read/write Default value: See 9.4.2.20	See 9.4.2.20
SolicFilter	7	Filter used on incoming solicitations	Type: OctetString See Table 142 Classification: Static Accessibility: Read/write Default value: See 9.4.2.20	See 9.4.2.20
TaiTime	8	TAI time for DLE	Type: TAINetworkTime Classification: Static Accessibility: Read only	Units: 2^{-16} s See 9.4.2.6
TaiAdjust	9	Adjust TaiTime at an instant that is scheduled by the system manager	Type: OctetString See Table 143 Classification: Dynamic Accessibility: Read/write Default value: Null	See 9.4.2.21

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
MaxBackoffExp	10	Maximum backoff exponent for retries	Type: Unsigned8	See 9.4.2.7
			Classification: Static	
			Accessibility: Read/write	
			Default value: 5	
			Valid range: 3..8	
MaxDsduSize	11	Maximum octets that can be accommodated in a single DSDU	Type: Unsigned8	See 9.4.2.8
			Classification: Static	
			Accessibility: Read/write	
			Default value: 96	
			Valid range: 76..96	
MaxLifetime	12	Maximum Data DPDU lifetime	Type: Unsigned16	Units: 0,25 s See 9.4.2.9
			Classification: Static	
			Accessibility: Read/write	
			Default value: 120 (30 s)	
			Valid range: 8..1 920 (2 s..480 s)	
NackBackoffDur	13	Duration of backoff after receiving a NAK	Type: Unsigned16	Units: 0,25 s See 9.4.2.10
			Classification: Static	
			Accessibility: Read/write	
			Default value: 60 (15 s)	
			Valid range: 8..1 920 (2 s..480 s)	
LinkPriorityXmit	14	Default priority for transmit links	Type: Unsigned8	See 9.4.2.11
			Classification: Static	
			Accessibility: Read/write	
			Default value: 8	
			Valid range: 0..15	
LinkPriorityRcv	15	Default priority for receive links	Type: Unsigned8	See 9.4.2.11
			Classification: Static	
			Accessibility: Read/write	
			Default value: 0	
			Valid range: 0..15	
EnergyDesign	16	DLE's energy capacity as designed	Type: OctetString	See 9.4.2.22
			Classification: Constant	
			Accessibility: Read only	
			Default value: See 9.4.2.22	
EnergyLeft	17	Remaining energy for DLE	Type: Integer16	See 9.1.17
			Classification: Dynamic	
			Accessibility: Read only	

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
DeviceCapability	18	Device capabilities	Type: OctetString See Table 147	See 9.4.2.23
			Classification: Constant	
			Accessibility: Read only	
			Default value: See 9.4.2.23	
IdleChannels	19	Radio channels that shall be idle	Type: Unsigned16	See 9.4.2.12
			Classification: Static	
			Accessibility: Read/write	
			Default value: 0	
ClockExpire	20	Clock expiration deadline.	Type: Unsigned16(MSB)	Units 1s See 9.4.2.13
			Classification: Static	
			Accessibility: Read/write	
			Default value: See description	
ClockStale	21	DL clock source timeout	Type: Unsigned16	Units: 1 s See 9.4.2.14
			Classification: Static	
			Accessibility: Read/write	
			Default value: See description	
			Valid range: 5..300	
RadioSilence	22	Radio silence timeout	Type: Unsigned32	Units: 1s See 9.4.2.15
			Classification: Static	
			Accessibility: Read/write	
			Default value: 600	
			Valid range: Limited to 1..600 for radio silence profile; otherwise 0..2^{32}-1	
RadioSleep	23	Radio sleep period. Note: DLE's radio will be disabled when this attribute is set	Type: Unsigned32	Units: 1s See 9.4.2.17
			Classification: Dynamic	
			Accessibility: Read/write	
			Default value: 0	
RadioTransmitPower	24	Radios maximum transmit power level	Type: Integer8	Units: dBm See 9.4.2.18
			Classification: Static	
			Accessibility: Read/write	
			Default value: See text	
			Valid range: -20..36	

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
CountryCode	25	Information about the device's regulatory environment	Type: Unsigned16	See 9.4.2.19, 9.1.15.6 and Annex V
			Classification: Static	
			Accessibility: Read/write	
			Default value: 0x3C00	
Candidates	26	A list of candidate neighbors discovered by the DLE	Type: OctetString See Table 151	See 9.4.2.24
			Classification: Dynamic	
			Accessibility: Read/write	
			Default value: Null	
DiscoveryAlert	27	Control of NeighborDiscovery alert	Type: OctetString	See 9.4.2.24
			Classification: Dynamic	
			Accessibility: Read/write	
			Default value: See 9.4.2.25	
			Valid range: See 9.4.2.24	
SmoothFactors	28	Smoothing factors for diagnostics	Type: OctetString See Table 153	See 9.4.2.25
			Classification: Static	
			Accessibility: Read/write	
			Default value: See Table 153	
			Valid range: See Table 153	
QueuePriority	29	Queue buffer capacity for specified priority level	Type: OctetString See Table 155	See 9.4.2.26
			Classification: Static	
			Accessibility: Read/write	
			Default value: N=0	
Ch	30	Channel-hopping patterns	Type: OctetString (indexed) See Table 159	See 9.4.3.2
			Classification: Static	
			Accessibility: Read/write	
			Default value: See 9.4.3.2	
			Valid range: See 9.4.3.2	
ChMeta	31	Metadata for Ch attribute	Type: Metadata_attribute	See 9.4.3.2 (Note 1)
			Classification: Static	
			Accessibility: Read only	

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
TsTemplate	32	Timeslot templates	Type: OctetString (indexed) See Table 161 and Table 163	See 9.4.3.3
			Classification: Static	
			Accessibility: Read/write	
			Default value: See 9.4.3.3	
			Valid range: See 9.4.3.3	
TsTemplateMeta	33	Metadata for TsTemplate attribute	Type: Metadata_attribute	See 9.4.3.3 (Note 1)
			Classification: Static	
			Accessibility: Read only	
Neighbor	34	Neighbors	Type: OctetString (indexed) See Table 168	See 9.4.3.4
			Classification: Static	
			Accessibility: Read/write	
			Default value: Empty	
			Valid range: See 9.4.3.4	
NeighborDiagReset	35	Used to update DiagLevel field within Neighbor attribute	Type: OctetString (indexed) See Table 172	See 9.4.3.4.3
			Classification: Static	
			Accessibility: Read/write	
			Valid range: See 9.4.3.4.3	
NeighborMeta	36	Metadata for Neighbor attribute	Type: Metadata_attribute	See 9.4.3.4 (Note 1)
			Classification: Static	
			Accessibility: Read only	
Superframe	37	Superframes; structures and activation	Type: OctetString (indexed) See Table 175	See 9.4.3.5
			Classification: Dynamic	
			Accessibility: Read/write	
			Default value: Empty	
			Valid range: See 9.4.3.5	
SuperframeIdle	38	Used to update idle fields within Superframe attribute	Type: OctetString (indexed) See Table 177	See 9.4.3.5.3
			Classification: Dynamic	
			Accessibility: Read/write	
SuperframeMeta	39	Metadata for Superframe attribute	Type: Metadata_attribute	See 9.4.3.5 (Note 1)
			Classification: Static	
			Accessibility: Read only	

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Graph	40	Graphs	Type: OctetString (indexed) See Table 178	See 9.4.3.6
			Classification: Static	
			Accessibility: Read/write	
			Default value: Empty	
GraphMeta	41	Metadata for Graph attribute	Type: Metadata_attribute	See 9.4.3.6 (Note 1)
			Classification: Static	
			Accessibility: Read only	
Link	42	Links	Type: OctetString (indexed) See Table 180	See 9.4.3.7
			Classification: Static	
			Accessibility: Read/write	
			Default value: Empty	
LinkMeta	43	Metadata for Link attribute	Type: Metadata_attribute	See 9.4.3.7 (Note 1)
			Classification: Static	
			Accessibility: Read only	
Route	44	Routes	Type: OctetString (indexed) See Table 185	See 9.4.3.8
			Classification: Static	
			Accessibility: Read/write	
			Default value: Empty	
RouteMeta	45	Metadata for Route attribute	Type: Metadata_attribute	See 9.4.3.8 (Note 1)
			Classification: Static	
			Accessibility: Read only	
NeighborDiag	46	Neighbor link diagnostics	Type: OctetString (indexed) See Table 187	See 9.4.3.9
			Classification: Dynamic	
			Accessibility: Read only	
			Default value: Empty	
DiagMeta	47	Metadata for NeighborDiag attribute	Type: Metadata_attribute	See 9.4.3.9 (Note 1)
			Classification: Static	
			Accessibility: Read only	
ChannelDiag	48	Per-channel diagnostics for spectrum management	Type: OctetString	See 9.4.2.27
			Classification: Dynamic	
			Accessibility: Read only	
			Default value: See 9.4.2.27	
AlertPolicy	49	Report diagnostics if connectivity problems are detected between regular HRCO reports	Type: OctetString	See 9.6.1
			Classification: Static	
			Accessibility: Read/write	
			Default value: See 9.6.1	

Table 141 (*continued*)

Standard object type name: DL management object (DLMO)				
Standard object type identifier: 124				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
DLTimeout	50	DLE may reasonably reset to provisioned state if it doesn't receive a time update in this time interval	Type: Unsigned16	See 9.4.2.15
			Classification: Static	
			Accessibility: Read/write	
			Default value: See description	
			Valid range: > 0	
NOTE 1 Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for the table.				

9.4.2.2 dlmo.ActScanHostFract

dlmo.ActScanHostFract configures the DLE's behavior as an active scanning host, as specified in 9.1.13.3. The setting indicates the fraction of time that the DLE should respond when it receives an active scanning solicitation. The default of 0 indicates that the DLE is not configured as an active scanning host.

9.4.2.3 dlmo.AdvJoinInfo and dlmo.AdvSuperframe

dlmo.AdvJoinInfo and dlmo.AdvSuperframe configure the contents of an advertisement's DAUX subheader. Their meaning is described in 9.3.5.2.5.

9.4.2.4 dlmo.SubnetID

dlmo.SubnetID is the identifier for the single D-subnet that the DLE is currently using or attempting to join. The DL management SAPs handle only one active D-subnet at a time. If a given device is participating in multiple D-subnets concurrently, this may be modeled as multiple instances of the DLE. dlmo.SubnetID=0 shall never be used as a D-subnet ID; its use indicates that the DLE is not participating in a D-subnet. dlmo.SubnetID=1 shall be used exclusively to identify provisioning D-subnets. The system manager doesn't set the DLE's SubnetID directly; rather, it is set by the DLE itself in the process of discovering and joining the D-subnet. See 9.1.10.2.

NOTE In the IEEE 802.15.4:2011 design, SubnetID is usable as a filter for incoming MPDUs. As discussed in 9.1.10.2, a DMIC provides an additional, stronger and more reliable filter once the DLE has joined the D-subnet.

9.4.2.5 dlmo.SolicTemplate

dlmo.SolicTemplate is a template for the DAUX subheader in a solicitation. When a solicitation is transmitted, the exact data in this OctetString (without a prepended explicit size) shall be used as the DAUX subheader. It is null (zero size) by default. See 9.3.5.3.

9.4.2.6 dlmo.TaiTime

dlmo.TaiTime, when read by the DMAP, is reported as the DLE's best estimate of DL time at that instant. See 12.22.4.2 for encoding of TAINetworkTime.

NOTE The dlmo.TaiTime attribute is described as a read-only attribute, where time is acquired by the DLE from its neighbors and provided as a service to other layers through the DMAP. This style of specification is not intended to preclude implementations, such as on DLEs that are clock masters, where time is provided to the DLE from an alternative source.

9.4.2.7 dlmo.MaxBackoffExp

dlmo.MaxBackoffExp is the maximum backoff exponent for retries; see 9.1.8.2 for a discussion of exponential backoff.

9.4.2.8 dlmo.MaxDsduSize

dlmo.MaxDsduSize is the maximum number of octets that can be accommodated in a single DSDU. This is used by the NL to make fragmentation decisions. Its default value of 96 allows for the following constraints:

- A single EUI64Address in the MHR. See 9.3.3.2.
- A one-octet Crypto Key Identifier and a slow-channel-hopping-offset in the DMXHR. See 9.3.3.4.
- A single compressed route in DROUT (i.e. no source routing beyond the single hop case). See 9.3.3.6.
- No DAUX, so that a fragmented DSDU cannot be combined with an advertisement, with the exception of the link activation DAUX when 16-bit addressing is used.
- A DMIC-32, not DMIC-64 or DMIC-128.

NOTE MaxDsduSize was calculated as follows: 15 octets for the MHR (see 9.3.3.2); 1 octet for the DHR (see 9.3.3.3); 3 octets for the DMXHR (see 9.3.3.4); 0 octets for the DAUX (see 9.3.3.5); 2 octets for the DROUT (see 9.3.3.6); 4 octets for DADDR (see 9.3.3.7); 4 octets for the DMIC; and 2 octets for the FCS. This total of 31 octets is subtracted from the PhSDU capacity of 127 octets, to arrive at a MaxDsduSize of 96 octets.

The system manager shall reduce the value of dlmo.MaxDsduSize as needed if additional constraints apply to a particular configuration.

9.4.2.9 dlmo.MaxLifetime

dlmo.MaxLifetime is the maximum duration, in units of 0,25 s, for which a Data DPDU is to be held in the message queue of a single DLE before it shall be discarded. dlmo.MaxLifetime can be overridden by dlmo.Graph[ ].MaxLifetime (see 9.4.3.6).

9.4.2.10 dlmo.NackBackoffDur

dlmo.NackBackoffDur is the duration of the backoff, in units of 0,25 s, after receiving a NAK (see 9.1.9.4.4).

9.4.2.11 dlmo.LinkPriorityXmit and dlmo.LinkPriorityRcv

dlmo.LinkPriorityXmit and dlmo.LinkPriorityRcv are the default priorities to be used when selecting links. If no priority is specified in dlmo.Link[ ].Priority, use these priorities. For T/R links, use dlmo.LinkPriorityRcv as the priority for the receive side of the link. Link priorities are functionally described in 9.1.8.5.

9.4.2.12 dlmo.IdleChannels

dlmo.IdleChannels provides a list of channels that shall be idle, as a quick way for the system manager to block the usage of certain channels on a particular DLE without requiring a coordinated change of channel-hopping schedules. A link occurring on any of the channels designated as idle (value 1) by dlmo.ActiveChannels shall be treated as idle. Values of 1 in dlmo.IdleChannels shall not cause hop sequences to be shortened, but rather leaves the hop sequences intact and simply causes all links on designated channels to be treated as idle. (Shortening of the hop sequences themselves is accomplished through a different attribute, dlmo.Superframe[ ].ChMap.) Bit positions 0..15 correspond to channels 0..15. A bit value of 1 indicates that links using the channel shall be treated as idle. dlmo.IdleChannels is complimentary with dlmo.DeviceCapability.ChannelMap; in operation of the DLE, the two are logically combined as follows, resulting in a set of channels that are treated as active by the DLE:

ActiveChannels = ((NOT dlmo.IdleChannels) AND (dlmo.DeviceCapability.ChannelMap))

9.4.2.13 dlmo.ClockExpire

dlmo.ClockExpire is the maximum number of seconds that the DLE can safely operate without a clock update. The default is (1 000 s / DeviceCapability.ClockStability), which is intended to keep the DLE synchronized to within 1 ms during the join process and thereafter when participating in a D-subnet that provides only slotted-hopping. In other cases, the needed value scales linearly with the needed tighter or looser clock accuracy. See 9.1.9.2.2.

NOTE A device that requires use of slow-hopping is likely to have a longer ClockExpire duration than the above default.

9.4.2.14 dlmo.ClockStale

dlmo.ClockStale determines when the DLE should start accepting time updates from secondary DL clock sources. For example, if dlmo.ClockTimeout is set to the default of 45 s, then a DL clock recipient should not accept clock updates from a secondary DL clock source until at least 45 s has elapsed since it last received a clock update from a primary DL clock source. The default value is 0,5 × ClockExpire. See 9.1.9.2.3 for more information.

9.4.2.15 dlmo.ClockTimeout

dlmo.ClockTimeout is the maximum number of seconds that the DLE can reasonably operate in a D-subnet before resetting itself to the provisioned state. The default value is 2,0 × ClockExpire. See 9.1.9.2.2.

9.4.2.16 dlmo.RadioSilence

dlmo.RadioSilence designates when a DLE shall disable its transmitter after losing its D-subnet connection. See 9.1.15.4 for more information.

9.4.2.17 dlmo.RadioSleep

dlmo.RadioSleep is used to disable the DLE's radio for a period of time. See 9.1.15.4 for more information. Activation of this attribute shall be slightly delayed to allow for transmitting an application layer acknowledgment of the DMAP TPDU that causes the attribute to be set.

9.4.2.18 dlmo.RadioTransmitPower

dlmo.RadioTransmitPower is used to control the DLE's radio transmit power level, in dBm EIRP. It defaults to the device's maximum permitted transmit power level under the regulatory regime specified by dlmo.CountryCode (9.1.15.6), and is reported during the join process through dlmo.DeviceCapability.RadioTransmitPower. See 9.1.15.5.

9.4.2.19 dlmo.CountryCode

dlmo.CountryCode provides constraints on a device based on the applicable regulatory regime. When set during DLE provisioning, use of the supported content-locking functionality can constrain operation of the device until it is next provisioned (e.g., perhaps after repair and deployment to a different regulatory jurisdiction). See 9.1.15.6 and Annex V.

9.4.2.20 Subnet filters

A D-subnet filter attribute is a string of 4 octets that specifies how a DLE shall filter incoming advertisements or solicitations. AdvFilter is used to filter incoming advertisements, and SolicFilter is used to filter incoming solicitations.

AdvFilter and SolicFilter each include two fields, a 16-bit BitMask field and a 16-bit TargetID field. Table 142 shows the structure of each D-subnet filter.

Table 142 – D-subnet filter octets

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	BitMask							
2	TargetID							

Unlike most DLMO attributes, D-subnet filters use LSB octet ordering conventions. This reflects their use, which is to perform bit comparison operations with DPDU header elements that are transmitted in LSB order.

When a DLE receives an advertisement, it shall check the incoming DPDU's SubnetID. The advertisement shall be ignored unless:

(DPDU.SubnetID AND AdvFilter.BitMask) equals (AdvFilter.TargetID AND AdvFilter.BitMask)

AdvFilter.BitMask shall default to 0xFFFF, and AdvFilter.TargetID shall default to 0x0001, with the result that an unprovisioned DLE in the default state will filter all advertisements except those received from a provisioning D-subnet with SubnetID=1.

When a DLE receives a solicitation, it shall check the incoming DPDU's SubnetID. The solicitation shall be ignored unless:

(DPDU.DauxSubnetID AND SolicFilter.BitMask) ==
(SolicFilter.TargetID AND SolicFilter.BitMask)

SolicFilter.BitMask shall default to 0x0000, with the result that solicitations are not filtered by default.

9.4.2.21 Time adjustments

The dlmo.TaiAdjust attribute includes fields that are used to adjust dlmo.TaiTime at an instant that is scheduled by the system manager. This attribute is normally null, unless a time correction is pending. Its use is described in 9.1.9.3.6. The OctetString comprises a series of fields that are described in Table 143.

Table 143 – dlmo.TaiAdjust OctetString fields

Field name	Field encoding
TaiCorrection (indicates the magnitude and direction of a TAI clock correction)	Type: Integer32 Units: 2^{-15} s
TaiTimeToApply (indicates the time at which the correction shall be applied)	Type: TAITimeRounded Units: 1 s

NOTE The TaiCorrection unit of 2^{-15} s was chosen to match the 32 KiHz very-precise very-low-power "watch" crystals commonly used for the continuous clock hardware of WISN devices.

Table 144 illustrates the structure of the OctetString.

Table 144 – dlmo.TaiAdjust OctetString structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
4	TaiCorrection							
4	TaiTimeToApply							

9.4.2.22 DLE energy capacity

The dlmo.EnergyDesign attribute, as shown in Table 145, includes various elements that indicate the energy capacity of the device. The fields within this attribute are described in 9.1.17.

Table 145 – dlmo.EnergyDesign OctetString fields

Field name	Field encoding
EnergyLife (DLE energy life by design; positive for months, negative for days)	Type: Integer16 (constant)
ListenRate (DLE's energy capacity to operate its receiver, in seconds per hour)	Type: ExtDLUint (constant)
TransmitRate (DLE's energy capacity to transmit DPDUs, in DPDUs per minute	Type: ExtDLUint (constant)
AdvRate (DLE's energy capacity to transmit advertisements, in DPDUs per minute)	Type: ExtDLUint (constant)

Table 146 illustrates the structure of the OctetString.

Table 146 – dlmo.EnergyDesign OctetString structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	EnergyLife							
1..2	ListenRate							
1..2	TransmitRate							
1..2	AdvRate							

9.4.2.23 DLMO device capabilities

The dlmo.DeviceCapability attribute includes various elements that indicate the capabilities of the device. This is a read-only attribute, most of whose component values do not change during normal operation. (They may be changed due to remote system management, including by the download of new firmware.) dlmo.DeviceCapability shall be reported to the system manager as part of the join process.

The OctetString comprises a series of fields that are described in Table 147. Some of these fields, listed as static, do not change during operation and are reported only on joining. Other fields, listed as dynamic, may change during operation and are also available after joining through identically-named DLMO attributes.

Table 147 – dlmo.DeviceCapability OctetString fields

Field name	Field encoding
QueueCapacity (capacity of the queue that is available for forwarding operations)	Type: ExtDLUint (constant)
ClockStability (nominal clock stability of this device, as a multiple of $1{\times}10^{-6}$)	Type: Unsigned8 (constant)
ChannelMap (map of radio channels supported by the device	Type: Unsigned16 (constant)
DLE_roles (DLE roles supported by the DLE)	Type: BooleanArray8 (constant)
EnergyDesign (copy of attribute dlmo.EnergyDesign)	Type: OctetString (constant)
EnergyLeft (copy of attribute dlmo.EnergyLeft)	Type: Integer16
Ack_Turnaround (see Table 161) (Note 1)	Type: ExtDLUint (constant)
NeighborDiagCapacity (memory capacity for dlmo.NeighborDiag)	Type: ExtDLUint (constant)
RadioTransmitPower (copy of attribute RadioTransmitPower, see 9.1.15.5)	Type: Integer8
SupportedCCAmodes (bitmap description of CCA modes supported by the device)	Type: BooleanArray8 (constant)
ConstructionOptions (i.e., optional features supported by DLE)	Type: BooleanArray8 (constant)
NOTE 1 This is the greater of the time required to switch from transmit to receive or from receive to transmit, both of which occur (in different DLEs) at the end of a Data DPDU before the following ACK/NAK DPDU	

Table 148 illustrates the structure of the OctetString. Size of EnergyDesign includes a one-octet size field prepended within the unspecified-length OctetString.

Table 148 – dlmo.DeviceCapability OctetString structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1..2	QueueCapacity							
1	ClockStability							
2	ChannelMap							
1	DLERoles							
6..9	EnergyDesign (OctetString size, plus 5..8 octet content)							
2	EnergyLeft							
1..2	AckTurnaround							
1..2	NeighborDiagCapacity							
1	RadioTransmitPower							
1	SupportedCCAmodes							
1	ConstructionOptions							

Fields include:

- dlmo.DeviceCapability.QueueCapacity is the number of buffers in the queue that are available for forwarding operations. This capacity shall not include internal buffers that may be used for messages that flow through the NLE. This value shall be based on the worst case, wherein all DPDUs are of maximum size. In operation, the actual queue capacity may be larger than this reported value. See 9.1.8.5.
- dlmo.DeviceCapability.ClockStability is the nominal short-term clock stability of the device, as a multiple of $1{\times}10^{-6}$, in the absence of a time correction from the D-subnet. See 9.1.9.2.2.
- dlmo.DeviceCapability.ChannelMap is a list of channels that the device can legally support in the device's regulatory domain (as determined by dlmo.CountryCode, 9.1.15.6), where

a value of zero indicates that the device is not permitted to use the channel. Bit positions 0..15 correspond to channels 0..15 of this standard, which in turn correspond to 2,4 GHz DSSS channels 11..26 of IEEE 802.15.4:2011. If the DLE is configured with links that refer to such blocked channels, the DLE shall treat those links as idle. See 9.1.7.2.3, 9.1.15.6, 9.4.2.19 and Annex V.

- dlmo.DeviceCapability.DLERoles enumerates the DL role profiles supported by the DLE, where a value of TRUE indicates that the DLE supports the DL role profile. See 9.1.16 for a discussion of DLE roles.
  - Index 0 indicates whether the DLE supports the I/O role profile.
  - Index 1 indicates whether the DLE supports the router role profile.
  - Index 2 indicates whether the DLE supports the backbone router role profile.
  - Index 3 indicates whether the DLE supports the radio silence role profile. See 9.1.15.4.
  - Indices 4..7 are reserved and shall be set to FALSE.
- dlmo.DeviceCapability.EnergyDesign and EnergyLeft are described in 9.1.17 and 9.4.2.22.
- dlmo.DeviceCapability.DAckTurnaround indicates the time needed by the DLE to process a received data DPDU and respond with an ACK or NAK DPDU, in units of 2^{-15} s. All DLEs shall be capable of using the default timeslot templates (see Table 165, Table 166, and Table 167).

  The DAckTurnaround value is an upper bound on the externally-measured time interval required by the device to respond to a signal from its associated PHY that DPDU reception has completed and to initiate PHY transmission of any immediately following ACK/NAK DPDU. This measurement involves noting the time when the last symbol of a PhPDU corresponding to the initial DPDU of a transaction is presented to the receiving device and the first PhPDU signaling from that device is detected, where the transaction template used is a unicast receive template with the ACK/NAK DPDU timing referenced to the end of the just-received Data DPDU.
- dlmo.DeviceCapability.NeighborDiagCapacity indicates the capacity, in octets, of the NeighborDiag attribute. Only octets shown in Table 187 are included in NeighborDiagCapacity. The system manager indirectly creates OctetStrings in NeighborDiag by setting DiagLevel fields in the Neighbor attribute. The system manager should not configure a DLE to fill NeighborDiag in excess of its stated capacity, and a DLE may fail to accumulate data if the system manager exceeds this stated capacity. A value of 0x7FFF indicates that the capacity is sufficient to collect all available diagnostics for each dlmo.Neighbor.
- dlmo.DeviceCapability.RadioTransmitPower is the DLE's maximum supported power level, in dBm EIRP, that the DLE can legally support in the DLE's regulatory domain, as determined by dlmo.CountryCode. See 9.1.15.5, 9.1.15.6, 9.4.2.19 and Annex V.
- dlmo.DeviceCapability.SupportedCCAmodes is a list of CCA modes that the device supports (see 9.1.9.4.3):
  - Index 0 (Bit0) indicates whether CCA Mode 1 is supported.
  - Index 1 (Bit1) indicates whether CCA Mode 2 is supported.
  - Index 2 (Bit2) indicates whether CCA Mode 3 is supported.
  - Index 3 (Bit3) indicates whether CCA Mode 4 is supported.
  - Indices 4..7 (Bits4..7) are reserved and shall be set to FALSE.
- dlmo.DeviceCapability.ConstructionOptions indicates optional features that the device supports by construction:
  - Index 0 (Bit0) indicates whether group codes are supported in dlmo.Neighbor.
  - Index 1 (Bit1) indicates whether graph extensions are supported in dlmo.Neighbor.
  - Index 2 (Bit2) indicates whether the device is capable of receiving duocast or N-cast ACK/NAK DPDUs.

– Index 3 (Bit3) indicates whether the device is capable of supporting dlmo.Superframe.SfType=1, which may be needed in some regions for regulatory compliance.
– Index 4 (Bit4) indicates whether the device is capable of supporting the dlmo.Graph.Queue field which, when set to a non-zero value, reserves queue buffer capacity.

  Such queue capacity should not be so reserved unless dlmo.DeviceCapability.QueueCapacity exceeds the minimum requirement for a DLE's role profile (see Table B.8).
– Indices 5..7 (Bits5..7) are reserved and shall be set to FALSE.

9.4.2.24 Candidate neighbors

The dlmo.Candidates attribute is used to provide the system manager with a list of candidate neighbors. The DLE autonomously populates this attribute as it receives advertisements from a number of candidate neighbors. This attribute is then forwarded to the system manager so that routing decisions can be made. The system manager may reset dlmo.Candidates.N=0, thus signaling to the DLE to clear its history of received advertisements and resume the neighbor discovery process.

The attribute dlmo.DiscoveryAlert (Table 149) provides the system manager with control over neighbor discovery and reporting. The system manager sets dlmo.DiscoveryAlert, and later receives a copy of the dlmo.Candidates attribute through the dlmo.NeighborDiscovery alert. Alternatively, the system manager can read the dlmo.Candidates attribute on its own schedule, or arrange to report it periodically through the HRCO.

Table 149 – dlmo.DiscoveryAlert fields

Field name	Field encoding
Descriptor	Type: Alert report descriptor (see Table 269) Default: [FALSE, 0]
Duration	Type: Unsigned16 Units: 1 s Default: 60

Table 150 illustrates the structure of DiscoveryAlert.

Table 150 – dlmo.DiscoveryAlert structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Alert report disabled							
1	Alert report priority							
2	Duration							

When dlmo.DiscoveryAlert is enabled (Descriptor.Disabled=FALSE), the DLE shall report the contents of dlmo.Candidates.Duration seconds later using the dlmo.NeighborDiscovery alert. Once the DLE completes the alert, the DLE resets dlmo.DiscoveryAlert to zero.

When the NeighborDiscovery alert is triggered by the state of Duration (dlmo.DiscoveryAlert.Duration), the DLE shall automatically set Duration to zero, thereby resulting in a single NeighborDiscovery alert each time Duration is set to a non-zero value.

dlmo.DiscoveryAlert shall be enabled and default to Duration=60 when the DLE enters the joined state. This default indicates that the DLE shall, when it enters the joined state, accumulate information from advertisements in dlmo.Candidates for a period of 60 s, and then report the results using the dlmo.NeighborDiscovery alert. See 9.1.14.6. The system manager may override this default by writing to dlmo.DiscoveryAlert in conjunction with the join response.

Advertisements from neighboring routers include links that can be used to communicate with that router. When DiscoveryAlert is enabled, the DLE may use these links to interrogate, with a Data DPDU carrying a null DSDU, each candidate router, on multiple channels, and receive signal quality metric in the DAUX. This information enables the DEL to build a more accurate picture of signal quality in dlmo.Candidates.

When dlmo.DiscoveryAlert is disabled, the DLE should nonetheless passively build a dlmo.Candidates list as a background process after it joins the D-subnet, based on whatever advertisements it happens to receive in the course of operating the DLE’s state machine.

If there is a dlmo.DL_Connectivity alert and DiscoveryAlert is enabled, the DLE shall also send a dlmo.NeighborDiscovery alert at the same time.

When dlmo.DiscoveryAlert.Duration is set to 0, the DLE shall not send dlmo.NeighborDiscovery alerts on a timed basis. The DLE should continue to maintain the Candidates attribute so that it can be read as needed by the system manager by reading the dlmo.Candidates attribute directly or by configuring the DLE to report it periodically through the HRCO.

The process of scanning for advertisements is described in 9.1.13.

dlmo.Candidates comprises a series of fields that are described in Table 151. In essence, the <Candidate>,<RSQI> tuple is repeated N times, providing an indication of signal quality to multiple neighbors.

dlmo.Candidates may reasonably exclude current entries in the dlmo.Neighbor table, when the same information for those neighbors is available through the neighbor diagnostics.

Table 151 – dlmo.Candidates OctetString fields

Field name	Field encoding
N (count of discovered neighbors)	Type: Unsigned8
$Neighbor_1$ (16-bit address of first candidate)	Type: ExtDLUint
$RSSI_1$ (radio signal strength of first candidate)	Type: Integer8
$RSQI_1$ (radio signal quality of first candidate)	Type: Unsigned8
…	…
$Neighbor_N$ (16-bit address of Nth candidate)	Type: ExtDLUint
$RSSI_N$ (radio signal strength of Nth candidate)	Type: Integer8
$RSQI_N$ (radio signal quality of Nth candidate)	Type: Unsigned8

Table 152 illustrates the structure of Candidates.

Table 152 – dlmo.Candidates structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	N							
1..2	Neighbor_1							
1	RSSI_1							
1	RSQI_1							
…	…							
1..2	Neighbor_N							
1	RSSI_N							
1	RSQI_N							

Fields include:

- dlmo.Candidates.N is the number of neighbors that have been discovered. A DLE shall support at least five (5) candidate entries. If many neighbors are discovered, the DLE may report only the best candidates based on the quality of the radio link.
- dlmo.Candidates.Neighbor_N is the 16-bit address of each candidate neighbor in the D-subnet.
- dlmo.RSSI_N indicates the strength of the radio signal in dBm from each candidate neighbor, based on received advertisements and possibly other DPDUs. See 9.1.15.2 for description of RSSI, including the fixed dBm bias within the reported measurement values.
- dlmo.RSQI_N indicates the quality of the radio signal from each candidate neighbor, based on received advertisements and possibly other considerations. A higher number indicates a better radio signal. See 9.1.15.2. for description of RSQI.

9.4.2.25 Smoothing factors

The dlmo.SmoothFactors provides the smoothing factors for the dlmo.NeighborDiag attribute. The use of these factors is described in 9.1.15.3.

Three fields in dlmo.NeighborDiag involve exponential smoothing: RSSI, RSQI, and ClockBias. The smoothing factor α (alpha) for each of these values is individually configurable. Alpha is expressed as an integer percentage in the range of 0..100. RSSI and RSQI default to 10%, so the values tend to reflect recent history. ClockBias defaults to 1%, since that value is intended to show clock bias over an extended period of time.

Each smoothing factor involves three values: x_AlphaHigh, x_AlphaLow, and x_Threshold, for differing x. If the new data is below the threshold, use x_AlphaLow as the smoothing factor; otherwise use x_AlphaHigh.

The fields in dlmo.SmoothFactors are described in Table 153.

Table 153 – dlmo.SmoothFactors OctetString fields

Field name	Field encoding	Default
RSSI_Threshold (threshold for RSSI)	Type: Integer16	0
RSSI_AlphaLow (AlphaLow for RSSI)	Type: Unsigned8	10
RSSI_AlphaHigh (AlphaHigh for RSSI)	Type: Unsigned8	10
RSQI_Threshold (threshold for RSQI)	Type: Integer16	0
RSQI_AlphaLow (AlphaLow for RSQI)	Type: Unsigned8	10
RSQI_AlphaHigh (AlphaHigh for RSQI)	Type: Unsigned8	10
ClockBias_Threshold (threshold for ClockBias)	Type: Integer16	0
ClockBias_AlphaLow (AlphaLow for ClockBias)	Type: Unsigned8	1
ClockBias_AlphaHigh (AlphaHigh for ClockBias)	Type: Unsigned8	1

Table 154 illustrates the structure of SmoothFactors.

Table 154 – dlmo.SmoothFactors structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	RSSI_Threshold							
1	RSSI_AlphaLow							
1	RSSI_AlphaHigh							
2	RSQI_Threshold							
1	RSQI_AlphaLow							
1	RSQI_AlphaHigh							
2	ClockBias_Threshold							
1	ClockBias_AlphaLow							
1	ClockBias_AlphaHigh							

9.4.2.26 dlmo.QueuePriority

9.4.2.26.1 General

dlmo.QueuePriority is an attribute that enables the system manager to specify the nominal buffer capacity in the DLE's forwarding queue for specific priority levels. As described in 9.1.8.5, the system manager may configure a DLE's nominal buffer capacity to limit the number of buffers that can be used to forward low-priority Data DPDUs. For example, the system manager may configure dlmo.QueuePriority so that no more than 3 buffers shall be used to forward Data DPDUs with priority ≤2.

The nominal capacity of the forwarding queue can be found in dlmo.DeviceCapability.QueueCapacity (see 9.4.2.23).

9.4.2.26.2 Semantics

Table 155 specifies the fields for dlmo.QueuePriority.

Table 155 – dlmo.QueuePriority fields

Field name	Field encoding
N (count of priorities specified, 0..15, default N=0)	Type Unsigned8
$Priority_1$ (first priority)	Type: Unsigned8
$QMax_1$ (first buffer capacity)	Type: Unsigned8
…	
$Priority_N$ (Nth priority)	Type: Unsigned8
$QMax_N$ (Nth buffer capacity)	Type: Unsigned8

For example, if Priority is 2 and QMax is 3, then no more than 3 queue buffers shall be used to forward Data DPDUs with priority ≤ 2. This count shall not include Data DPDUs that are using queue capacity that was set aside for Data DPDUs being forwarded along a particular graph, based on dlmo.Graph[ ].Queue. Priority shall be enumerated in increasing order, so that $Priority_X$ shall be less than $Priority_{X+1}$. Similarly, $QMax_X$ shall be less than $QMax_{X+1}$.

The count $QMax_X$ sets the maximum available slots available for low-priority messages, not reserved slots for low-priority messages. In effect, $QMax_X$ ensures that the remainder of the queue is available for DPDUs of priority 1+PriorityX.

As described in 9.2.2, the DE indicator in a Data DPDU header indicates whether a Data DPDU on the queue is eligible to be discarded in favor of an incoming DPDU of higher priority.

The default, where N=0, indicates that the system manager has not configured a limit on the number of forwarding buffers that can be used for low priority DPDUs.

Table 156 illustrates the structure of dlmo.QueuePriority.

Table 156 – dlmo.QueuePriority structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	N							
2 (opt)	$Priority_1$							
3 (opt)	$QMax_1$							
…	…							
2N (opt)	$Priority_N$							
2N+1 (opt)	$QMax_N$							

9.4.2.27 dlmo.ChannelDiag

9.4.2.27.1 General

dlmo.ChannelDiag is a read-only dynamic attribute that reports DPDU transmit failure rates on each radio channel supported by this standard. This enables the system manager to be aware of consistent connectivity problems on a per-channel basis, as a diagnostic for spectrum management in a D-subnet.

Two diagnostics are reported for each channel, each indicating a different type of failure. $NoACK_N$ indicates the percentage of time that unicast DPDU transmissions on channel N did not result in reception of an ACK/NAK DPDU. $CCABackoff_N$ indicates the percentage of time that the device aborted a transaction-initiating transmission on channel N due to CCA.

Under some situations, CCA backoff is part of normal D-subnet operation and is not indicative of poor channel quality. In particular, contention for shared links will lead to CCA backoff. Therefore, the DLE may be selective in its counting of CCA backoff. It is recommended that CCA backoff should not normally be counted when it occurs in shared links within slotted-channel-hopping superframes. Shared links are described in 9.1.8.2

ChannelDiag values are 8-bit signed integers.

- A value of 0 indicates that no transmission has been attempted on the channel.
- Positive values in the range of 1..101 indicate the percentage failure rate plus one. For example, a $CCABackoff_6$ value of 26 indicates that 25% of transaction-initiating transmissions on channel 6 were aborted due to CCA.
- Negative values in the range of -1 to -101 indicate the percentage failure rate as a negative number minus one. Failure rates are reported as negative numbers if they are based on 5 or fewer attempted transmissions. For example, a $NoACK_8$ value of -34 indicates that 33% of unicast transmissions on a particular channel did not receive an acknowledgment, and that 5 or fewer transmissions have been attempted on that channel.

The DLE may selectively skip transmission links in anticipation of a failed transmission, as described in 9.1.7.2.4. Such skipped links should be treated as equivalent to NAK for the applicable channel.

Each time ChannelDiag is read by the system manager or reported periodically through the HRCO, its underlying accumulators shall be reset to zero.

9.4.2.27.2 Semantics

Table 157 specifies the fields for dlmo.ChannelDiag.

Table 157 – dlmo.ChannelDiag fields

Field name	Field encoding
Count (number of attempted unicast transmissions for all channels)	Type: Unsigned16
$NoACK_0$ (percentage of time transmissions on channel 0 did not receive an ACK DPDU)	Type: Integer8
$CCABackoff_0$ (percentage of time transmissions on channel 0 aborted due to CCA)	Type: Integer8
…	…
$NoACK_{15}$ (percentage of time transmissions on channel 15 did not receive an ACK DPDU)	Type: Integer8
$CCABackoff_{15}$ (percentage of time transmissions on channel 15 aborted due to CCA)	Type: Integer8

Table 158 illustrates the structure of dlmo.ChannelDiag.

Table 158 – dlmo.ChannelDiag structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Count							
1	$NoACK_0$							
1	$CCABackoff_0$							
…	…							
1	$NoACK_{15}$							
1	$CCABackoff_{15}$							

9.4.3 DLMO attributes (indexed OctetStrings)

9.4.3.1 General

Indexed OctetString management object attributes are structured collections of data, similar in concept to SQL tables. For example, the DLE maintains a list of neighbors in an indexed OctetString attribute called dlmo.Neighbor, where each neighbor can be visualized as a row in a table, with the structure of each row as shown in Table 168 and Table 169. A 15-bit index for each row, in this case corresponding to the 16-bit address of the neighbor, is provided for each indexed OctetString attribute in the DLMO.

For consistency of processing, all indexed OctetString attributes in the DL include an index in the first field that is type ExtDLUint. However, in the case of dlmo.Ch, dlmo.TsTemplate, and dlmo.Superframe, the index is limited to a range of 1..127, thus guaranteeing that the index can be represented in a single octet. References to these structures can also be compressed to a single octet.

Indexed OctetString index of zero shall not be used in the DLMO, except to indicate a null entry.

Figure 90 illustrates the relationship among some of the indexed OctetString DLMO attributes. Referential relationships are shown with arrows. For example, dlmo.Link refers to dlmo.TsTemplate, so an arrow is shown pointing in the direction of the reference. This roughly corresponds to a one-to-many relationship, where one template can be referenced by multiple links.

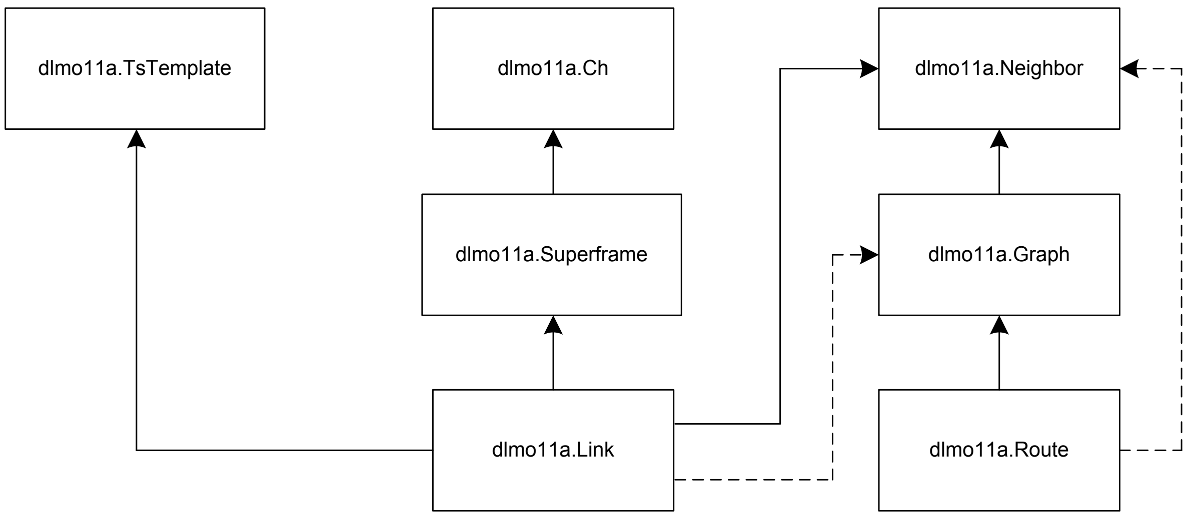


Figure 90 – Relationship among DLMO indexed attributes

As shown in Figure 90, dlmo.TsTemplate is referenced by dlmo.Link. Timeslot templates specify transaction structure and timing when a link is used.

dlmo.Link, dlmo.Superframe, and dlmo.Ch are related. Superframes select a channel-hopping pattern and a cyclic schedule. Links describe the various actions that are taken during each superframe cycle. Each link refers to one superframe.

For unicast (and duocast) transactions, dlmo.Link refers to dlmo.Neighbor. A single reference may encompass a group of neighbors. If a link is reserved for use of a certain graph, or gives preferential access to a certain graph, then dlmo.Link will also refer to dlmo.Graph. Since it is possible to configure a link without a reference to a graph, this reference is shown as a dotted line in Figure 90.

dlmo.Route and dlmo.Graph are related in that routes usually refer to graphs. dlmo.Graph and dlmo.Neighbor are related in that graphs refer to neighbors. dlmo.Route and dlmo.Neighbor are loosely related (indicated by a dotted line in Figure 90) in that source routing may implicitly refer to a neighbor.

An additional relationship, not shown in Figure 90, exists between dlmo.Neighbor and dlmo.NeighborDiag. Both are indexed by the 16-bit address of the DLE's neighbors, but with different purposes.

- dlmo.Neighbor: The system manager provides this static indexed OctetString attribute to enable the DLE to communicate with its immediate neighbors.
- dlmo.NeighborDiag: The system manager configures this dynamic indexed OctetString attribute to enable the DLE to collect and periodically report diagnostics for its immediate neighbors.

9.4.3.2 dlmo.Ch

9.4.3.2.1 General

dlmo.Ch is an indexed OctetString collection that contains available channel-hopping patterns.

Channel-hopping patterns 1 through 5 are reserved as standard defaults, as described in 9.1.7.2.5. Additional channel-hopping patterns may be added.

Each DLE can store multiple channel-hopping patterns, with a unique index for each pattern. Advertisements assume that channel-hopping patterns for the join process are configured in the DLE when the advertisement is received. Thus any channel-hopping pattern referenced in an advertisement shall match one of the defaults.

The system manager inserts, updates, or deletes channel-hopping patterns by sending the DMAP a channel-hopping pattern, along with a unique index and (if selected) a TAI cutover time.

For a given channel-hopping pattern, the standard provides a mapping between the DL channel numbers and the more general IEEE 802.15.4:2011 MAC channel numbers. As applied to the 2,4 GHz DSSS IEEE 802.15.4:2011 radio, channel numbers shall be limited to the range of 0..15, corresponding to IEEE 802.15.4:2011 channel numbers 11..26 (channel page 0), in the same order. Channel-hopping patterns for this radio shall not exceed a size of 16.

NOTE dlmo.Ch data types are limited to radios with 16 or fewer channels. Future radios with more channels would involve providing support for a less compressed representation.

Five default channel-hopping patterns are reserved and defined by this standard. Default channel-hopping patterns are enumerated and described in 9.1.7.2.5.

9.4.3.2.2 Semantics

Table 159 specifies the fields for dlmo.Ch. An index, used to identify each channel-hopping pattern, is consistent with DL conventions for indexed OctetStrings.

Table 159 – dlmo.Ch fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..127
Size	Type: Unsigned8 Valid range: 1..16
Seq (channel-hopping pattern, with Size entries)	Type: SEQUENCE OF Unsigned4 (SIZE (Size))

Table 160 illustrates the structure of dlmo.Ch.

Table 160 – dlmo.Ch structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Index (1..127)							
1	Size							
(Size+1)/2	Seq_1				Seq_0			
	Seq_3				Seq_2			
	…				…			
	0x0000 when Size is an odd number; Seq_{Size-1} when Size is an even number				$Seq_{2(n-2)}$			

9.4.3.3 dlmo.TsTemplate

9.4.3.3.1 General

dlmo.TsTemplate is an indexed OctetString collection that contains timeslot templates. Timeslot templates describe D-transaction timing.

Timeslot templates 1, 2, and 3 shall be reserved as standard defaults, as enumerated in Table 165, Table 166, and Table 167. Additional timeslot templates may be added by the system manager.

The system manager inserts, updates, or deletes timeslot templates by sending the DMAP a template, along with a unique index and (if selected) a TAI cutover time.

Template time offsets are specified in units of 2^{-20} s, which is approximately 1 μs. Timeslot duration is set by the superframes that use the template.

Template types include receive and transaction initiator templates. Both template types include acknowledgments for unicast transactions. The same templates can also be used for broadcast links, such as solicitations, that don't need acknowledgments and don't use the ACK/NAK DPDU timings.

Templates can be defined on three levels:

- Default templates are defined in the standard. These are the timeslot templates needed for joining (Table 165, Table 166, and Table 167). Template index=1 describes a generic receive transaction, and is used for receiving join responses. Template index=2 describes a generic transmit transaction, and is used for transmitting join requests. Template index=3 describes a transaction that operates its receiver for an entire timeslot, intended for operations such as scanning for advertisements or receiving loosely-timed slow-channel-hopping DPDUs. These generic templates shall be used during provisioning and joining, and may also be used for other purposes.

- Provisioned templates may be added during the provisioning process, with a lifetime that lasts until the DLE has joined the D-subnet. See 9.1.14.4.
- Subnet-specific templates may be provided to the DLE after the join process is completed.

Data DPDU transmission timing is based on the transaction initiator's internal clock. ACK/NAK DPDU timing is specified as a time offset from a reference point that is indicated in the template. Usually, the configured time range for a transaction initiator is narrower than the time range for the transaction receivers, to account for guard times.

Timeslot templates shall always provide a reception window of at least 192 μs, which implies that DLEs shall be capable of controlling transmission timing with an accuracy of at least ±96 μs, i.e., ±6 PHY symbol periods.

By convention, timeslot template timing is specified based on the start and end times of DPDUs. PhPDU timing, dependent on the details of the physical layer that contains the DPDU, can be inferred from DPDU times. DPDU start time, as specified in timeslot templates, uses a convention that the DPDU begins at the instant just before the first octet in the DPDU header is transmitted. This convention applies to Data DPDU transmissions as well as ACK/NAK DPDUs.

NOTE In actual implementations based on IEEE 802.15.4:2011 (2,4 GHz), PhL timing signals sometimes are triggered when an SFD (start frame delimiter) is completely transmitted or received. In such cases, the start time of the DPDU is 1 octet, or 32 μs, after that reference point.

ACK/NAK DPDU response time is commonly specified in relation to the end of the immediately preceding and triggering Data DPDU, with the ACK/NAK DPDU starting approximately 1 ms thereafter.

Alternatively, ACK/NAK DPDUs may be timed in relation to the scheduled end of the timeslot. By placing ACK/NAK DPDUs at a fixed offset within the timeslot, it is possible to meet regulatory requirements that would prohibit transmission of an ACK/NAK DPDU after a very short Data DPDU, where the sender of the ACK/NAK DPDU had also transmitted near the end of the prior timeslot.

Such timeslot-end-relative placement also supports routers that operate on multiple channels at the same time, sharing a common antenna, or devices whose antennas are in close proximity to each other. In both cases, without such scheduled alignment, Data DPDUs and/or ACK/NAK DPDUs transmitted on one channel might unintentionally jam time-overlapping reception whose reception is being attempted on another channel.

When ACK/NAK DPDU times are defined as offsets from the end of the timeslot, the time offsets, which are unsigned integers, shall be interpreted as referring to a time prior to the end of the timeslot.

Transaction receiver templates specify:

- The time range when the first octet of a Data DPDU can be received, indicating a time range to enable and disable the radio's receiver. A time range that exceeds the timeslot's duration indicates that the range extends only to the scheduled end of the timeslot.
- ACK/NAK DPDU delay time range.

  A transaction responder for a unicast transaction should respond with an ACK/NAK DPDU as early as possible within this range, with the exception of intentionally staggered n -cast ACK/NAK DPDUs. ACK/NAK DPDU delay time may be specified in reference to the end of the received DPDU or as a backward offset from the scheduled end of the timeslot.
- Whether it is acceptable to operate past the timeslot boundary once reception of a DPDU begins, essentially extending timeslot duration.

Overrun of a slot boundary can be used to accommodate slow-channel-hopping, where transaction-originating DLEs may have a time-skewed sense of the slot boundaries. Support of such transactions originated by such DLEs involves allowing reception even when the transaction overruns a slot boundary.

Transaction initiator templates specify:

- Time range to begin transmission. A compliant DLE may begin its transmission at any time within the time range, based on its internal DLE clock. The time range is based on the start time of the Data DPDU. The CCA operation, if any, and the PhPDU's SHR and PHR precede that event and therefore may occur prior to the earliest permitted time to begin transmission.
- ACK/NAK DPDU delay time range. A transaction responder usually will respond as early as possible within this range, per its timeslot template but subject to regulatory constraints on the minimal required delay since the last prior transmission by the same device. ACK/NAK DPDU delay time may be specified in relation to the end of reception of the Data DPDU or the scheduled end of the timeslot.
- Indication of the CCA mode to be used before transmission to check for competing or ongoing transmissions.
- Indication of whether the DLE should periodically continue operating its receiver for the entire ACK/NAK DPDU delay time range, even after receiving an ACK/NAK DPDU (intended for duocast coverage testing; see 9.1.9.4.7).

9.4.3.3.2 Semantics

There are two variations of dlmo.TsTemplate, one for a transaction receiver template and another for a transaction initiator template. The variations are distinguished by a 2-bit type that is the first element. Type=0 indicates a transaction receiver template, and type=1 indicates a transaction initiator template. Types 2 and 3 are reserved for future use.

Table 161 specifies the fields for the transaction receiver template.

Table 161 – Transaction receiver template fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..127
Type (indicates that this is a transaction receiver template)	Type: Unsigned2: Named values: 0: transaction receiver template 1: see Table 163; other choices are reserved
AckRef (indicates reference for ACK/NAK DPDU time range)	Type: Unsigned1: Named values: 0: offset from end of incoming DPDU; 1: negative offset from end of timeslot
RespectBoundary (specifies whether or not slot boundaries shall be respected, i.e., whether a D-transaction may extend past a slot boundary)	Type: Boolean1: FALSE: slot boundaries do not need to be respected; TRUE: slot boundaries shall be respected
Reserved (octet alignment)	Type: Unsigned4=0
WakeupDPDU (earliest time when DPDU reception can be expected to begin, indicating when to enable radio's receiver; offset from timeslot's scheduled start time)	Type: Unsigned16 Units: 2^{-20} s
TimeoutDPDU (latest time when DPDU reception can be expected to begin, indicating when to disable receiver if no incoming DPDU is detected; offset from timeslot's scheduled start time)	Type: Unsigned16 Units: 2^{-20} s
AckEarliest (start of ACK/NAK DPDU delay time range, with start of that DPDU (PhSDU) being the time reference. Semantics depend on AckRef: if AckRef=1, subtract value from scheduled end time of timeslot to determine AckEarliest)	Type: Unsigned16 Units: 2^{-20} s
AckLatest (end of ACK/NAK DPDU delay time range, with start of that DPDU (PhSDU) being the time reference. Semantics depend on AckRef: if AckRef=1, subtract value from scheduled end time of timeslot to determine AckLatest)	Type: Unsigned16 Units: 2^{-20} s

Table 162 specifies the transaction receiver template structure.

Table 162 – Transaction receiver template structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	* Index (range 1..127)							
1	Type=0		AckRef	RespectBoundary	Reserved=0			
2	WakeupDPDU							
2	TimeoutDPDU							
2	AckEarliest							
2	AckLatest							

Table 163 specifies the fields for the transaction initiator template.

Table 163 – Transaction initiator template fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..127
Type (indicates that this is a transaction initiator template)	Type: Unsigned2: Named values: 0: see Table 161; 1: transaction initiator template; remaining elements are reserved
AckRef (indicates reference for ACK/NAK DPDU delay time range)	Type: Unsigned1: Named values: 0: positive offset from end of transmitted/received DPDU 1: negative offset from end of timeslot
CCAmode (indicates whether to check CCA before transmission)	Type: unsigned2 (see 9.1.9.4.3): Named values: 0: CCA mode 4; 1: CCA mode 1; 2: CCA mode 2; 3: CCA mode 3.
KeepListening (indicates whether the DLE should periodically continue operating its receiver until the end of the timeslot, even after reception of an ACK/NAK DPDU; see 9.1.9.4.7)	Type: Boolean1
Reserved (octet alignment)	Type: Unsigned2=0
XmitEarliest (earliest time to start DPDU transmission; offset from timeslot's scheduled start time)	Type: Unsigned16 Units: 2^{-20} s
XmitLatest (latest time to start DPDU transmission; offset from timeslot's scheduled start time)	Type: Unsigned16 Units: 2^{-20} s
WakeupAck (earliest time when reception of an ACK/NAK DPDU can be expected to begin; enable receiver early enough to receive an ACK/NAK DPDU beginning at this time. Semantics depend on ACKref; if AckRef=1, subtract value from scheduled end of timeslot to determine WakeupAck.)	Type: Unsigned16 Units: 2^{-20} s
TimeoutAck (latest time when reception of an ACK/NAK DPDU can be expected to begin. DLE may disable receiver if ACK/NAK DPDU reception has not started by this time. Semantics depend on ACKref; if AckRef=1, subtract value from scheduled end of timeslot to determine TimeoutAck.)	Type: Unsigned16 Units: 2^{-20} s
NOTE An AckEarliest value of 402 (384 μs) accommodates the IEEE 802.15.4:2011 SIFS requirement of 6 octets, plus 6 octets for a PhPDU's SHR and PHR prior to the start of the ACK/NAK DPDU's PhSDU.	

Table 164 specifies the transaction initiator template structure.

Table 164 – Transaction initiator template structure

Number of octets	bits							
	7	6	5	4	3	2	1	0
1	* Index (range 1..127)							
1	Type		AckRef	CheckCCAmode		KeepListening	Reserved=0	
2	XmitEarliest							
2	XmitLatest							
2	WakeupAck							
2	TimeoutAck							

Table 165, Table 166, and Table 167 specify the values for the three default DLE timeslot templates. These read-only timeslot templates use indexes 1, 2, and 3, and shall be used for links that are specified in advertisements. Their structure is general purpose, and they may be referenced by other links as well.

The DLE shall be capable of transmitting and receiving ACK/NAK DPDUs in the 1 032 μs ± 100 μs timing specified in these default templates, or more slowly if so specified in an alternative timeslot template. The attribute dlmo.DeviceCapability.DAckTurnaround informs the system manager if a device is capable of handling ACK/NAK DPDUs more quickly.

NOTE These default templates are required for the initial device provisioning and join prcesses. There is no requirement for any non-join use in an operational system.

Table 165 – Default transaction responder template, used during join process

Field	Default value	Explanation
* Index	1	—
Type	0	—
AckRef	0	—
RespectBoundary	1	—
WakeupDPDU	1 271	1 212 μs
TimeoutDPDU	3 578	3 412 μs
AckEarliest	977	932 μs
AckLatest	1 187	1 132 μs

Table 166 – Default transaction initiator template, used during join process

Field	Default value	Explanation
* Index	2	—
Type	1	—
AckRef	0	—
CCAmode	1	—
KeepListening	0	—
XmitEarliest	2 319	2 212 μs
XmitLatest	2 529	2 412 μs
WakeupAck	977	932 μs
TimeoutAck	1 187	1 132 μs

Table 167 – Default transaction responder template, used during join process

Field	Default value	Explanation
* Index	3	—
Type	0	—
AckRef	0	—
RespectBoundary	0	If DPDU reception commences within the timeslot boundaries, complete processing of transaction
WakeupDPDU	0	Start of timeslot; allowing for timeslots that are contiguous. In the first timeslot of a contiguous series, a device may insert setup time at the start of the first timeslot not to exceed 1 271 μs
TimeoutDPDU	0xFFFF	End of timeslot
AckEarliest	977	Same as default transaction responder template
AckLatest	1 187	Same as default transaction responder template

9.4.3.3.3 Default template timings

Default timeslot templates are intended to match the timeslot structure of IEC 62591.

The default data DPDU transmission time is based on an offset of 2 312 μs from the start of the timeslot to the start of the data DPDU. The default transaction initiator template accounts for ±100 μs of transaction initiator jitter, resulting in a default range of 2 312 μs ± 100 μs. The default transaction receiver template accounts for the same ± 100 μs of transmit jitter, plus clock drift of ±1 000 μs, resulting in a receive range for the data DPDU of 2 312 μs ± 1 100 μs.

The default ACK/NAK DPDU transmission time is based on an offset of 1 032 μs from the end of the data DPDU to the start of the ACK/NAK's DPDU. The default slot transaction templates allow for ± 100 μs of transmit jitter, for a default template of 1 032 μs ± 100 μs. Clock drift is considered inconsequential in the short time span from the end of a data DPDU to the start of an immediately-following ACK/NAK DPDU.

In the IEEE 802.15.4:2011 radio used in this standard, the physical layer header is 6 octets, or 192 μs, in duration. Thus, radio transmission or reception begins 192 μs earlier than the times specified in the templates. CCA, if required, precedes the radio startup.

Templates do not account for the internals of radio operation, leaving that as an internal device matter. For example, the value of 932 μs for WakeupACK means that the physical layer header is allowed to begin transmission 192 μs sooner, or as early as 932-192=740 μs following the end of the DPDU. The receiver of the ACK/NAK DPDU, also with a nominal WakeupAck of 932 μs, therefore needs its radio to be running at 740 μs following the end of the DPDU and ready to start receiving the ACK/NAK DPDU at that time. If the receiver needs additional time to account for radio startup and receiver jitter, then the radio needs to be enabled sufficiently in advance of 740 μs to ensure that it can receive the full physical layer header.

9.4.3.3.4 Considerations for required minimum inter-transmission gap

Some regulatory regimes require that there be a minimum time period after one transmission ceases before the device is again permitted to transmit. It is the responsibility of each DLE to note the time that the most recent transmission ceased, to use that information to determine the earlierst moment that the device may again transmit, and to refrain from activating its transmitter before that moment.

NOTE This can be achieved by recording separately the ending time of the most recent transmission, adding a claimed-operating-mode-dependent Tx-gap-time constant, then comparing that resultant time with the projected time of the next transmission to determine whether the transmission is permitted under the applicable regulations. See V.4.

While the system manager can configure the various dlmo.TsTemplates to provide behavior in accordance with these regulatory requirements, it is the responsibility of the individual device to ensure that the requirements are observed no matter the configuration. dlmo.CountryCode (9.1.15.6, 9.4.2.19) provides the configuration information necessary to determine which regulatory aspects are in force.

9.4.3.4 dlmo.Neighbor

9.4.3.4.1 General

dlmo.Neighbor is an indexed OctetString collection that contains information about immediate unicast neighbors. dlmo.Neighbor entries are referenced by graphs and links.

The system manager inserts, updates, or deletes dlmo.Neighbor entries by sending the DMAP neighbor entries, along with a unique index and (if selected) a TAI cutover time. The neighbor’s 16-bit address is used as an index.

The neighbors in the dlmo.Neighbor attribute are set by the system manager, not by the DLE itself. The DLE autonomously builds a list of candidate neighbors in the dlmo.Candidates attribute, as described in 9.4.2.24. This list is then forwarded to the system manager. The system manager considers the radio connectivity that is reported in dlmo.Candidates, but may also consider other criteria such as resource constraints, historical performance, or D-subnet topology.

When the DLE processes an advertisement during the join procedure, the DLE automatically adds the advertising router as an entry in the dlmo.Neighbor table, thereby enabling communication through the proxy. This entry persists after the DLE successfully joins the D-subnet, unless it is deleted or updated by the system manager. The system manager infers the existence and content of this entry based on the identity of the proxy that the DLE uses to join the D-subnet. See 9.3.5.2.1.

This standard follows the IETF RFC 4944 convention, whereby 16-bit unicast addresses are limited to the range of 0x 0xxx xxxx xxxx xxxx. See 9.3.3.6.

Diagnostic information related to neighbors can be found in the attribute dlmo.NeighborDiag (see 9.4.3.9).

9.4.3.4.2 Semantics

Table 168 specifies the fields for dlmo.Neighbor.

Table 168 – dlmo.Neighbor fields

Field name	Field encoding
* Index (DL16Address of the neighbor)	Type: ExtDLUint (used as an index) Valid range: 1..32 767
EUI64 (EUI64Address of the neighbor)	Type: EUI64Address
GroupCode1 (associate a group code with a set of neighbors; used by dlmo.Link)	Type: Unsigned8
GroupCode2 (associate a group code with a set of neighbors; used by dlmo.Link)	Type: Unsigned8
ClockSource (indicates whether neighbor is a DL clock source)	Type: Unsigned2: Named values: 0: not a clock source 1: secondary 2: preferred 3: reserved
ExtGrCnt (count of graphs virtually extended for this neighbor)	Type: Unsigned2
DiagLevel (selection of neighbor diagnostics to collect)	Type: BooleanArray2 Named indices: 0: collect link diagnostics 1: collect clock diagnostics
LinkBacklog (indicates that link information is provided to facilitate clearing message queue backlog to the neighbor)	Type: Unsigned1; Named values: 0: no extra link information 1: extra link information is provided
Reserved (octet alignment)	Type: Unsigned1=0
ExtendGraph	Type: SEQUENCE OF Octet2 (SIZE ExtGrCnt) -- octet pairs Valid range: See Table 170
LinkBacklogIndex (Activate this link to clear queue backlog)	Type: ExtDLUint Valid range: 1..127 Null and not transmitted if LinkBacklog=0 Link index if LinkBacklog=1
LinkBacklogDur (duration of link activation)	Type: Unsigned8 Units: Link occurrences Null and not transmitted if LinkBacklog=0 1..255 if LinkBacklog=1
LinkBacklogActivate (link activation criterion)	Type: Unsigned8 Units: DPDUs on queue Null and not transmitted if LinkBacklog=0 1..255 if LinkBacklog=1

Table 169 illustrates the structure of dlmo.Neighbor.

Table 169 – dlmo.Neighbor structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1–2	* Index							
8	EUI64							
1	GroupCode1							
1	GroupCode2							
1	ClockSource		ExtGrCnt		DiagLevel		LinkBacklog	Reserved
2 × ExtGrCnt	$ExtendGraph_1$							
	…							
	$ExtendGraph_{ExtGrCnt}$							
0..1	LinkBacklogIndex							
0..1	LinkBacklogDur							
0..1	LinkBacklogActivate							

Fields include:

- EUI64. In order to communicate with a neighbor, the DSC needs the EUI64Address of that neighbor. This information is stored in the Neighbor table. It is populated by the system manager, with one exception. During the join process and provisioning process, where the neighbor entry is created automatically from the advertisement, the EUI64Address shall be acquired by the DLE through the ACK/NAK DPDU as described in 9.1.10.1.
- GroupCode1, GroupCode2. Links with a matching group code may be used to address this neighbor. The scope of the group code is within a single DLE. When a link has a group NeighborType=2, the link designates a group code instead of a neighbor, and the link applies to any queued DPDU where the neighbor has a matching group code. This enables a single transmit link to be shared by a group of neighbors. A value of zero indicates that no group code applies. Support for group codes is mandatory in routers but is a construction option in I/O devices. The presence or absence of this capability is reported to the system manager when the DLE joins the D-subnet through the field dlmo.DeviceCapability.ConstructionOptions (see 9.4.2.23).
- ClockSource. If this indicator is >0, then the neighbor shall be a DL clock source for this DLE. A value of 1 indicates a secondary DL clock source, and a value of 2 indicates a preferred DL clock source. See 9.1.9.2.3.
- ExtGrCnt and ExtendGraph. See 9.1.6.3 for a discussion of graph extensions. See Table 170 for the fields in each graph extension entry in ExtendGraph. If the neighbor's address matches the destination address encoded in the DADDR subheader, and the Graph_ID designated in ExtendGraph matches the DPDU's DL route, extend the designated graph to that neighbor for that DPDU. For each neighbor, up to three such graphs may be extended. Support for ExtendGraph is a device construction option, and the presence or absence of this capability is reported to the system manager when the DLE joins the D-subnet through the field dlmo.DeviceCapability.ConstructionOptions (see 9.4.2.23).

  For each ExtendGraph entry, include:

  – Graph ID is the 12-bit graph ID that is being extended. See 9.4.3.6.

  – LastHop. If this indicator is 1, the DLE shall only use links to the neighbor for applicable DPDUs. In this case, the DLE shall treat the extended graph index as the single forwarding alternative.

  – PreferredBranch. If this indicator is 1, the DLE should treat this graph extension as the preferred branch for applicable DPDUs. (See PreferredBranch field in 9.4.3.6.2).

Table 170 specifies the fields for each element in the ExtendGraph sequence.

Table 170 – ExtendGraph fields

Field name	Field encoding
Graph_ID (*Index of dlmo.Graph attribute)	Type: Unsigned12
LastHop (indicates whether the neighbor shall be the last hop)	Type: Boolean1
PreferredBranch (indicates whether to treat the neighbor as the preferred branch)	Type: Boolean1
Reserved (octet alignment)	Type: Unsigned2=0

Graphs are extended implicitly whether ExtendGraph is designated or not. The optional explicit ExtendGraph feature is intended to support optimizations that seek to control the graph ID of this final hop, and/or designate it as the last hop or preferred branch.

- DiagLevel. If this indicator has any non-zero bits, then the DLE shall collect link diagnostics for this neighbor in the read-only attribute dlmo.NeighborDiag. If Bit0=1, summary diagnostics shall be accumulated. If Bit1=1, clock diagnostics shall be accumulated. See 9.4.3.9.
- LinkBacklog, LinkBacklogIndex, LinkBacklogDur, and LinkBacklogActivate provide an index to a link that may be activated through the DAUX subheader (type 2). See 9.3.5.4 for a general description of link activation. The DLE, when transmitting a DPDU to this neighbor, may detect a backlog of applicable DPDUs on its message queue, and therefore signal the neighbor to activate the link Index=LinkBacklogIndex to receive DPDUs for the next LinkBacklogDur occurrences of the link (see 9.3.5.4). LinkBacklogActivate indicates the size of the applicable DPDU backlog on the queue, excluding the DPDU being transmitted, that should trigger sending the link activation DAUX, unless the link is already activated.

The system manager, when it configures LinkBacklogIndex, LinkBacklogDur, and LinkBacklogActivate, should also configure a transmit link with the same index, so that a receive link on the neighbor has the same index as a corresponding transmit link in the DLE that originates the link activation message, with both being activated for the number of occurrences indicated by LinkBacklogDur. DPDUs queued to this neighbor should be given high priority for that link index during the period defined by LinkBacklogDur. When counting candidate DPDUs on the message queue, the DLE should account for all queued DPDUs that can be addressed to the neighbor.

The transaction initiator (the DLE that initiates activation of the idle links) should activate its own idle link for a count of LinkBacklogDur transmission opportunities (link occurrences). The receiver (the DLE on which the receive idle link has been activated) should activate its idle link for LinkBacklogDur reception opportunities (link occurrences). Transmission and reception opportunities should be counted even if a higher priority link is actually used in a particular timeslot. Generally, the idle link should be configured with a high priority.

Table 171 specifies the ExtGraph structure.

Table 171 – ExtGraph structure

Number of octets	bits							
	7	6	5	4	3	2	1	0
1	Graph_ID (bits 11..4)							
1	Graph_ID (bits 3..0)				LastHop	PreferredBranch	Reserved=0	

9.4.3.4.3 dlmo.NeighborDiagReset

dlmo.NeighborDiagReset provides read/write access to the field of the dlmo.Neighbor attribute that is used to set the neighbor diagnostic level. It is conceptually similar to a SQL view of dlmo.Neighbor. It can be used to read and write a specific field within dlmo.Neighbor, but it shall not be used to add or delete entries.

Table 172 specifies the fields within a dlmo.NeighborDiagReset OctetString.

Table 172 – dlmo.NeighborDiagReset fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..32 767
Reserved (octet alignment)	Unsigned4 = 0
DiagLevel (selection of neighbor diagnostics to collect)	Type: BooleanArray3; Named indices: 0: collect summary information 1: collect clock information
Reserved (octet alignment)	Type: Unsigned2 = 0

Table 173 illustrates the structure of dlmo.NeighborDiagReset.

Table 173 – dlmo.NeighborDiagReset structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1..2	* Index							
1	Reserved=0				DiagLevel		Reserved=0	

Fields are exactly as specified for the identically named fields in dlmo.Neighbor attribute, and the instantaneous values of these fields are the same as in dlmo.Neighbor.

9.4.3.5 dlmo.Superframe

9.4.3.5.1 General

dlmo.Superframe is an indexed OctetString collection that contains superframes. The superframe structure enables the system manager to connect a set of links (dlmo.Link) to a repeating schedule of timeslots (dlmo.Superframe) of fixed duration. The superframe also designates a baseline channel-hopping schedule for those links.

The system manager inserts, updates, or deletes dlmo.Superframe entries by sending the DMAP a superframe, along with a unique index and, if selected, a TAI cutover time.

Each superframe describes a schedule that is specified in reference to TAI time zero. Derivation of current timeslot state from the superframe definition is described in 9.4.3.5.3.

A superframe may be configured with randomized timings, intended exclusively for links that transmit or receive solicitations and/or advertisements. Since a randomized superframe schedule is not synchronized to its neighbors, such a superframe shall not be used to transmit payload in DSDUs.

9.4.3.5.2 Semantics

Table 174 specifies the fields for dlmo.Superframe.

Table 174 – dlmo.Superframe fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..127
TsDur (duration of timeslots within superframe; timeslots are realigned with TAI time reference every 250 ms)	Type: Unsigned16 Units: 2^{-20} s
ChIndex (selects channel-hopping pattern from dlmo.Ch)	Type: ExtDLUint
ChBirth (absolute slot number where channel channel-hopping pattern nominally started)	Type: Unsigned8
SfType (type of superframe)	Type: Unsigned2; Named values: 0: baseline 1: hop on link only 2: randomize slow-hop duration 3: randomize superframe period
Priority (priority to select among multiple available links)	Type: Unsigned4
ChMapOv (indicates whether to override ChMap default)	Type: Boolean1; Valid range: FALSE: ChMapOv not transmitted, so defaults to 0x7FFF; TRUE: ChMapOv transmitted and used
IdleUsed	Type: Boolean1; Valid range: FALSE: IdleTimer not transmitted, so defaults to -1; TRUE: IdleTimer transmitted and used
SfPeriod (base number of timeslots in each superframe cycle)	Type: Unsigned16 Valid range: >0
SfBirth (absolute slot number where the first superframe cycle nominally started)	Type: Unsigned16
ChRate (indicates the number of timeslots per hop)	Type: ExtDLUint Valid range: 0 = invalid 1 = slotted-channel-hopping >1 = slow-channel-hopping a)
ChMap (channel map used to eliminate certain channels from the channel-hopping pattern, to limit the frequency spectrum in use)	Type: Unsigned16 or null
IdleTimer (idle/wakeup timer for superframe)	Type: Integer32 or null See text
RndSlots (indicates extent of randomization, in number of slots)	Type: Unsigned8 or null

a) In some regulatory regimes, as specified via dlmo.CountryCode (9.4.2.19), the maximum value of ChRate is constrained to be no greater than (400 ms / TsDur).

Table 175 illustrates the structure of dlmo.Superframe.

Table 175 – dlmo.Superframe structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	* Index							
2	TsDur							
1	ChIndex (range 1..127)							
1	ChBirth							
1	SfType		Priority				ChMapOv	IdleUsed
2	SfPeriod							
2	SfBirth							
1..2	ChRate							
0 or 2	ChMap							
0 or 4	IdleTimer							
0 or 1	RndSlots							

Fields include:

- Timeslot duration (dlmo.Superframe[ ].TsDur). All timeslots within a superframe have the same duration, and a DLE is not required to handle multiple timeslot durations at the same time. Timeslots shall be realigned to the TAI clock every 250 ms. See 9.1.9.1 for information on timeslot alignment.
- Channel-hopping pattern identifier (dlmo.Superframe[ ].ChIndex). Select an available pattern from dlmo.Ch (see 9.4.3.2.2).
- Channel-hopping birthday (dlmo.Superframe[ ].ChBirth). Specifies the starting point of the channel-hopping pattern, as a timeslot offset from TAI=0. Calculation of current position in hop sequence is described in 9.4.3.5.3.
- Superframe type (dlmo.Superframe[ ].SfType) indicates the type of superframe. Handling of each superframe type is described in 9.4.3.5.3.
- Superframe priority (dlmo.Superframe[ ].Priority) indicates the priority of the superframe. A higher Priority value will give that superframe link a higher priority. See 9.1.8.5 (pseudocode) for additional information on priority levels.
- Channel map override default (dlmo.Superframe[ ].ChMapOv). Indicates whether to override ChMap default of 0x7FFF. If ChMapOv=1, change the default based on ChMap.
- Superframe period (dlmo.Superframe[ ].SfPeriod) indicates the number of timeslots in each base cycle of the superframe. With 10 ms timeslots, a 16-bit superframe period supports superframes up to about 10 minutes long.
- Superframe birthday (dlmo.Superframe[ ].SfBirth) indicates the nominal starting point for the “first” superframe that began its first cycle soon after TAI=0. It provides the slot offset from absolute timeslot 0, which occurred at nominal time TAI=0. SfBirth shall equal ChBirth when SfType= 1. See 9.4.3.5.3.
- Channel-hopping rate (dlmo.Superframe[ ].ChRate) indicates the number of timeslots per hop. A channel-hopping rate of 1 indicates slotted-channel-hopping; a channel-hopping rate greater than 1 indicates some degree of slow-channel-hopping. ChRate shall =1 when SfType= 1.
- Channel map (dlmo.Superframe[ ].ChMap) is used to eliminate certain channels from the channel-hopping pattern, thus shortening the channel-hopping pattern in use. Bit positions 0..15 correspond to channels 0..15, where a 0 bit in any position indicates that the corresponding channel shall not be used by the superframe, with the channel-hopping sequence shortened accordingly. This attribute shall not be transmitted and defaults to 0x7FFF if ChMapOv=0 (i.e., optional channel 15 is excluded by default).

- Idle superframe timer (dlmo.Superframe[ ].IdleUsed, dlmo.Superframe[ ].IdleTimer) provides the system manager with control over when a superframe is activated or idle, where an idle superframe treats all of its links as idle. The system manager may set IdleTimer through the dlmo.Superframe attribute or the dlmo.SuperframeIdle attribute. IdleTimer is not transmitted and defaults to a value of -1 if IdleUsed=0.
- When IdleTimer is set to a positive number, the superframe shall be active and IdleTimer shall be decremented by the DLE each TAI second until it reaches a value of 0. When the value of IdleTimer is 0, the superframe shall be idle.
- When IdleTimer is set to a negative number that is less than -1, the superframe shall be idle and IdleTimer shall be incremented by the DLE each TAI second until it reaches a value of -1. When the value of IdleTimer is -1, the superframe shall be active.

Randomization of superframes is controlled by dlmo.Superframe[ ].RndSlots. RndSlots is meaningless and shall not be transmitted if dlmo.Superframe[ ].SfType<2. Randomized superframes are intended exclusively to enable randomized D-subnet discovery processes. For example, a DLE in the provisioned state may be configured with a randomized superframe that is used to search for advertisements from a target D-subnet. Such randomization can be used to guarantee that the scan's sleep cycle is not synchronized with the advertisement schedule of the target D-subnet, thereby ensuring that an advertisement is eventually received. Only receive links, dedicated advertisements, and solicitations should be configured for use with a superframe where RndSlots>0. All other links for such randomized superframes shall be treated as idle.

– When SfType=2, a randomized number of timeslots in the range of 0 to RndSlots shall be added to the end of each superframe cycle.

– When SfType=3, a randomized number of timeslots in the range of 0 to RndSlots shall be added to the duration of each slow-channel-hopping period. If slotted-channel-hopping is used, each hop shall be treated as a slow-channel-hopping period extended by a randomized number of timeslots.

9.4.3.5.3 Superframe current timeslot state

The current superframe timeslot state shall be derived from the superframe fields as described herein. This description is not intended to constrain implementations, but only results. All features described herein shall be supported by all DLEs that comply with this standard, unless specifically designated as a construction option.

These derivations of the current timeslot state use fields in dlmo.Superframe[ ], which are based on the state at TAI time of zero or soon thereafter. Implementations may reasonably use these formulas to establish a starting state when the superframe is initialized, and then update that state incrementally going forward. However, the incremental update approach will not work when there is a change in any field used in the base calculation, or in fields in other attributes (dlmo.Ch[ ] in general, and dlmo.Link[ ] for SfType=1) that are used in the base calculation. When those fields are changed, the current state needs to be derived again.

NOTE The notion of an absolute timeslot is used here as a variable to calculate the current timeslot state. Other standards use an absolute timeslot to identify the timeslot. This standard uses an absolute timeslot only as an intermediate value in a calculation; it is not referenced elsewhere in this standard.

Each TAI quarter-second period has a fixed number of timeslots that can be described by the formula:

$$\text{SlotsPer0\_25s} = \text{floor}((2^{18}\ \text{s}) / \text{TsDur})$$

where floor(x) is the largest integer not greater than x.

An absolute slot number (SlotNumAbs) can be derived from the scheduled start time of the current timeslot (ScheduledTaiTime), simplified by the fact that ScheduledTaiTime is required to be re-aligned to TAI time every quarter-second. The slot offset from TAI=0 can be derived accordingly:

Tai0_25sStart = floor(ScheduledTaiTime / (0,25 s)) × (0,25 s)

SlotWithin0_25s = (ScheduledTaiTime – Tai0_25sStart) / TsDur

SlotNumAbs = ((Tai0_25sStart / (0,25 s)) × SlotsPer0_25s) + SlotWithin0_25s

SfType=0 designates the baseline case, where all superframe cycles include a fixed number of timeslots and the channel-hopping schedule also has a fixed cycle.

The superframe provides a fixed superframe period (SfPeriod) which is the number of timeslots in each superframe cycle. It also provides an absolute slot number (SfBirth), following TAI=0, as a reference starting time for the first superframe. The superframe offset of the current timeslot is:

SfOffset = (SlotNumAbs – SfBirth) mod SfPeriod

where (x mod y) equals (x - (floor(x / y) × y)) for positive y.

The channel-hopping pattern nominally begins at absolute slot number ChBirth. The number of elements in the channel-hopping schedule (ChCount) can be determined from the size of the channel-hopping pattern selected by ChIndex, and by subtracting the number of entries that are removed from the sequence as indicated by ChMap.

The number of timeslots in a cycle of the channel-hopping pattern depends on whether slow-channel-hopping or slotted-channel-hopping is used, as indicated by ChRate.

ChCycle = ChCount × ChRate

The timeslot offset into that channel-hopping cycle is:

ChOffset = (SlotNumAbs – ChBirth) mod ChCycle

SfType=1 designates a variant of slotted-channel-hopping, where channel-hopping occurs only when there is a link.

Device support for SfType=1 is a construction option, as reported to the system manager through the attribute dlmo.DeviceCapability.ConstructionOptions (bit 5). SfType=1 shall not be combined with slow-channel-hopping, i.e., ChRate=1.

The superframe offset is as described in SfType=0.

The number of channel hops per superframe cycle (ChPerSuperframe) is determined by counting the number of timeslots in each superframe cycle that are referenced by at least one link. This is not a simple count of links, because some links may refer to multiple timeslots, and some timeslots may be referenced by multiple links.

Since the number of channel hops is a multiple of the number of superframes, the next step is to calculate the number of superframe cycles that have been completed since TAI=0.

CurrentSfStartAbs = (SlotNumAbs – SfOffset)

SfCyclesSinceBirth = (CurrentSfStartAbs – SfBirth) / SfPeriod

For SfType=1, the superframe cycle and channel-hopping cycles are required to start at the same time (SfBirth=ChBirth). The channel offset at the start of the current superframe is a function of the number of mapped channels (see 9.4.2.12). It is determined by the formula:

$\text{ChOffset}_{\text{StartSf}}$ = (SfCyclesSinceBirth × ChPerSuperframe) mod NumMappedChannels

where NumMappedChannels = $\sum \mathrm{dlmo.Superframe[\,].ChMap}_k$ for the specific superframe.

Starting from $\text{ChOffset}_{\text{StartSf}}$, the current channel-hopping-offset can be determined by stepping through the superframe from the start of the superframe cycle to the current timeslot.

The channel-hopping-offset within each superframe cycle cannot be reduced to a simple linear formula since the links are not necessarily spread evenly through the cycle.

SfType=2 extends each slow-channel-hopping interval by a randomized number of timeslots in the range of 0 to dlmo.Superframe[ ].RndSlots. The initial starting point of the channel-hopping sequence may also be randomized when SfType=2, and superframe timing shall be a described for SfType=0.

SfType=3 extends each superframe cycle by a randomized number of timeslots in the range of 0 to dlmo.Superframe[ ].RndSlots. The initial starting point of the superframe cycle should also be randomized when SfType=3, and the channel-hopping sequence shall be as described for SfType=0.

9.4.3.5.4 Slow-channel-hopping

Slow-channel-hopping is defined as a superframe where dlmo.Superframe[ ].ChRate>1, resulting in a set of contiguous links on the same channel. The channel-hopping rate, ChRate, may be configured as equal to the superframe period, SfPeriod, and in that case each channel-hopping period may reasonably be configured as a range of links using dlmo.Link[ ].Schedule=2.

The receive side of a slow-channel-hopping configuration should use the default transaction receiver template for scanning as per Table 167, or a similar template. This template, when applied to contiguous timeslots on the same channel, should run its receiver continuously, and may run a transaction to completion even if that transaction runs across the edge of a timeslot. A receive link using that template may repeat frequently or continuously within a superframe, usually with a low priority to give precedence to slotted-channel-hopping operations.

A set of slow-channel-hopping receive links on a given channel, using the default transaction receiver template for scanning, may be temporarily interrupted by higher-priority transactions, for example, as shown graphically in Figure 66. In the absence of such transactions, the receiver should run continuously across the timeslot boundaries in such configurations.

The transmit side of a slow-channel-hopping configuration should use a template appropriate for a transmit transaction on the D-subnet, such as the default transaction initiator template as per Table 166. The transmit link configuration should be configured to account for clock drift. For example, in a D-subnet with 10 ms timeslots, a particular device in a slow-channel-hopping configuration might be expected to experience clock drift of up to 15 ms between clock updates. In that example, the first and last two timeslots in each slow-channel-hopping period should not be designated as transmit links, thereby incorporating 20 ms guard times into the configuration.

The transmit side in a slow-channel-hopping configuration may designate specific timeslots for transmission, or alternatively it may designate a range of timeslots. When a range of timeslots is designated, the channel-hopping rate should match the superframe period, and the transmit link should be designated as a range (dlmo.Link[ ].Schedule=2), shown graphically in Figure 72. In that configuration, the DLE should treat each range as a single transmit opportunity, and select the transmit link within the range on a randomized basis.

9.4.3.5.5 dlmo.SuperframeIdle

dlmo.SuperframeIdle provides read/write access to only the fields of the dlmo.Superframe attribute that relate to the idle superframe. It is conceptually similar to an SQL view of dlmo.Superframe. It can be used to read and write specific fields within superframe indexed OctetStrings, but cannot be used to add or delete entries.

Table 176 specifies the fields within a dlmo.SuperframeIdle OctetString.

Table 176 – dlmo.SuperframeIdle fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index)
Reserved (octet alignment)	Type: Unsigned7=0
IdleUsed (indicates whether the superframe is idle when the IdleTimer is zero)	Type: Boolean1
IdleTimer (idle/wakeup timer for superframe)	Type: Integer32 or null

Table 177 illustrates the structure of dlmo.SuperframeIdle.

Table 177 – dlmo.SuperframeIdle structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	* Index							
1	Reserved=0							IdleUsed
0 or 4	IdleTimer							

Fields are exactly as specified for identically named fields in the dlmo.Superframe attribute, and the instantaneous values of these fields are identical to those in dlmo.Superframe.

9.4.3.6 dlmo.Graph

9.4.3.6.1 General

dlmo.Graph is an indexed OctetString collection that contains graphs. On a particular DLE, a graph is simply a list of neighbors that can be used for the next hop when a graph is specified in the DROUT subheader.

The system manager inserts, updates, or deletes dlmo.Graph entries by sending the DMAP a graph, along with a unique index and (if selected) a TAI cutover time.

Graph ID = 0 shall not be used, because this value is reserved as an indicator in the DROUT subheader, as described in 9.3.3.6.

Graph IDs are limited to a range of 12-bit values, with a range of $1..2^{12}$. In source routing, these 12-bit graph IDs are encoded as 0x 1010 gggg gggg gggg.

As described in 9.1.6.3, immediate neighbors are implicitly treated as covered by a graph, whether the neighbor is listed in the graph structure or not. When the DPDU's destination address is in a DLE's neighbor table, the graph is automatically extended to cover that neighbor. Thus, even though the structure of dlmo.Graph can support only a few neighbors, the graph can handle many more through such graph extensions.

It is advisable for Graph IDs to be unique within the scope of a D-subnet. However, duplicated graph IDs are not prohibited. When two graphs with the same graph ID intersect in a single DLE, they unite to become a single graph.

9.4.3.6.2 Semantics

Table 178 specifies the fields for dlmo.Graph.

Table 178 – dlmo.Graph

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index) Valid range: 1..4095
PreferredBranch (indicates whether to treat the first listed neighbor as the preferred branch)	Type: Boolean1
NeighborCount	Type: Unsigned3 Valid range: 0..4
Queue (allocates buffers in the message queue for DPDUs that are being forwarded using this graph)	Type: Unsigned4
MaxLifetime	Type: ExtDLUint
Neighbors (index into dlmo.Neighbor; usually two or three neighbors in list for next-hop diversity)	Type: SEQUENCE OF ExtDLUint (SIZE(NeighborCount))

Table 179 illustrates the structure of dlmo.Graph in the case where each ExtDLUint requires one octet.

Table 179 – dlmo.Graph structure

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1..2	* Index							
1	PreferredBranch	NeighborCount			Queue			
1..2	MaxLifetime							
1..2	Neighbors$_0$							
1..2	Neighbors$_1$							
…	…							
1..2	NeighborsNeighborCount-1							

Elements include:

- dlmo.Graph[ ].PreferredBranch. If this indicator is 1, treat the first listed neighbor as the preferred branch, and the DLE should wait until there is an opportunity to try at least one transmission along the preferred branch before attempting other alternatives. If this indicator is 0, do not give such preferential treatment to the first listed neighbor.
- dlmo.Queue allows the system manager to reserve up to 15 buffers of the message queue for DPDUs that are following the graph.
- dlmo.Graph[ ].MaxLifetime (units ¼ s). If this element is non-zero, the value of dlmo.MaxLifetime shall be overridden for all DPDUs being forwarded following this graph.
- List of neighbors (commonly two neighbors for next-hop link diversity).

9.4.3.7 dlmo.Link

9.4.3.7.1 General

dlmo.Link is an indexed OctetString collection that contains links. Each link refers to exactly one dlmo.Superframe entry.

The system manager inserts, updates, or deletes links by sending the DMAP a link, along with a unique index and (if selected) a TAI cutover time.

When a neighbor is referenced in a transmit link, DPDUs that refer to that neighbor are considered as candidates for the link. DPDUs that refer to the neighbor through the first entry in the DROUT subheader, either directly by address or indirectly through a graph, shall be considered as candidates for the link. In addition, DPDUs that designate the neighbor as the destination address in the DADDR subheader shall also be considered as candidates for the link. The exception is that certain links are designated exclusively for DPDUs following specific graphs, and only DPDUs with matching GraphIDs shall be considered as candidates for such links. If multiple DPDUs on the message queue are candidates for a given link, the DPDU is selected by priority as described in 9.1.8.5.

9.4.3.7.2 Semantics

Table 180 specifies the fields for dlmo.Link.

Table 180 – dlmo.Link fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index)
SuperframeIndex (reference to dlmo.Superframe entry)	Type: ExtDLUint
Type (see Table 182 and associated text)	Type: Unsigned8
Template1 (dlmo.TsTemplate reference to primary template)	Type: ExtDLUint
Template2 (dlmo.TsTemplate reference to secondary template)	Type: ExtDLUint or null
NeighborType	Type: Unsigned2 Valid range: 0..2
GraphType	Type: Unsigned2 Valid range: 0..2
SchedType	Type: Unsigned2
ChType	Type: Unsigned1
PriorityType	Type: Unsigned1
Neighbor (identify neighbor or group)	Type: ExtDLUint or null
GraphID (12-bit identity of graph with exclusive or prioritized access to link)	Type: ExtDLUint or null
Schedule (link schedule; see Table 184)	Type: See Table 184
ChOffset (select channel based on offset from superframes hop pattern)	Type: Unsigned8 or null
Priority (link priority)	Type: Unsigned8 or null Valid range: 0x 0000 xxxx

Table 181 illustrates the structure of dlmo.Link. ExtDLUint fields are shown as single octets.

Table 181 – dlmo.Link structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1..2	* Index							
1	SuperframeIndex (range 1..127)							
1	Type							
1	Template1 (range 1..127)							
0 or 1	Template2 (range 1..127)							
1	NeighborType		GraphType		SchedType		ChType	PriorityType
0..2	Neighbor							
0..2	GraphID							
1..4	Schedule (see Table 184)							
0 or 1	ChOffset							
0 or 1	Priority							

Elements include:

- dlmo.SuperframeIndex. Indicates the superframe reference for the link.
- dlmo.Link[ ].Type. Indicates how the link is configured for transmission and/or reception, and/or neighbor discovery. See Table 182.
- dlmo.Link[ ].Template1. Primary timeslot template. See 9.4.3.3 for a discussion of templates.
- dlmo.Link[ ].Template2. Secondary timeslot template, for transmit/receive (TR) slots only, in combination with other link selections. Use Template2 as the transaction receiver template, if there is no DPDU in the queue for the primary template. Template 2 is transmitted and meaningful only for TRx links, that is, links where Link[ ].Type bits 6 and 7 both have a value of 1. See 9.1.8.5.
- dlmo.Link[ ].NeighborType, and dlmo.Link[ ].Neighbor. A neighbor is designated for transmit links, and may be designated for duocast/N-cast receive links. See 9.4.3.4 for a discussion of neighbors. When a neighbor is designated in a link, it may reference either a dlmo.Neighbor index or a group (dlmo.Neighbor[ ].GroupCode).
  - If dlmo.Link[ ].NeighborType=0, dlmo.Link[ ].Neighbor is null, and not transmitted.
  - If dlmo.Link[ ].NeighborType=1, dlmo.Link[ ].Neighbor designates an index into the dlmo.Neighbor attribute.
  - If dlmo.Link[ ].NeighborType=2, dlmo.Link[ ].Neighbor designates a group.
- dlmo.Link[ ].GraphType, dlmo.Link[ ].GraphID. DPDUs following a particular graph may be given exclusive or priority access to certain transmit links. These fields, when so configured, limit link access to certain graphs, thereby connecting the link to a particular communication flow through the D-subnet. When GraphType is left blank, the transmit link is available to any DPDU that is being routed through the link's designated neighbor. When GraphType is used, a particular graph is given exclusive or priority access to the link.
  - If GraphType=0, the GraphID element is null and is not transmitted.
  - If GraphType=1, the link is designated for exclusive use by a particular graph. Access to the link shall be limited to DPDUs following that graph.
  - If GraphType=2, DPDUs with a matching graph ID are given priority access. DPDUs following that graph should be given priority over other DPDUs when the link is used.

- dlmo.Link[ ].SchedType, dlmo.Link[ ].Schedule. Indicates the timeslot position(s) of the link within each superframe cycle. The schedule may designate a fixed offset, a repeating set with a fixed interval, a range, or a bitmap, as follows:

  0: offset only;

  1: offset and interval;

  2: range;

  3: bitmap.

- dlmo.Link[ ].ChType, dlmo.Link[ ].ChOffset. Indicates how the link's channel is selected.
  - If dlmo.Link[ ].ChType=0, dlmo.Link[ ].ChOffset is null and not transmitted. Simply follow the superframe's baseline channel-hopping pattern.
  - If dlmo.Link[ ].ChType=1, add dlmo.Link[ ].ChOffset to the superframe's current dlmo.Superframe[ ].ChOffset, modulo the effective channel-hopping pattern size (after accounting for excluded channels) and select the channel accordingly.
- dlmo.Link[ ].PriorityType, dlmo.Link[ ].Priority. Indicates how the link's priority is set. Link priorities are functionally described in 9.1.8.5.
  - If dlmo.Link[ ].PriorityType=FALSE, dlmo.Link[ ].Priority is null and not transmitted. For transmit links, use priority dlmo.LinkPriorityXmit. For receive links, use priority dlmo.LinkPriorityRcv.
  - If dlmo.Link[ ].PriorityType=TRUE, use the dlmo.Link[ ].Priority. If the link is both a transmit link and a receive link, use dlmo.Link[ ].Priority for transmissions, and dlmo.LinkPriorityRcv for reception.

Table 182 illustrates the structure of the field dlmo.Link[ ].Type.

Table 182 – dlmo.Link[ ].Type structure

Octet	Bits						
	7	6	5	4	3	2	1	0
1	Transmit	Receive	Exponential backoff	Idle	Discovery: Named values: 0: none 1: advertisement 2: reserved 3: solicitation		JoinResponse: link used to send join response to neighbor	SelectiveAllowed

The link types are defined as follows:

- Bit 7: Transmit (T=TRUE). Indicates transmission of payload.
- Bit 6: Receive (R=TRUE).
- Bit 5: Exponential backoff (B=TRUE). Indicates whether the transaction originator should apply exponential backoff rules for retries (shared), versus using the timeslot without regard to exponential backoff (not shared). See 9.1.8.2.
- Bit 4: Idle (I=TRUE). When TRUE, the link shall be idle unless temporarily activated in conjunction with transmission of an activate DAUX subheader; see 9.3.5.4. When FALSE, link activation does not apply to the link. A timeslot designated as idle may include a neighbor reference. Even without a neighbor reference, it needs transmit and receive Booleans set as needed to refer to the timeslot template it will need when activated.
- Bits 3..2 Discovery (DD): Advertisement or solicitation configuration. DD='01' specifies a single advertisement transmitted with timing as defined in the timeslot template. If DD='01', the link should be used to transmit an advertisement or solicitation, even if there is no higher-order payload that needs to be sent. DD='11' distinguishes a solicitation from an advertisement. DD='10" shall be ignored.

  NOTE DD='10' was used by ISA100.11a:2011 for an alternate form of advertisement, and is thus not available for future assignment. It's use is not supported by this standard.

- Bit 1: JoinResponse (J=TRUE). When a router proxies a join request for an immediate neighbor, it will eventually receive a message from the system manager to forward to the DLE that is joining. The router forwards these messages using links that are flagged with JoinResponse=TRUE. A join response on the message queue can be identified by a DPDU header with a destination EUI64Address. A timeslot designated as supporting a join response may include a neighbor reference, thereby enabling the link to also be used for regular D-subnet traffic. Even without a neighbor reference, it needs the Transmit boolean to be set.
- Bit 0: SelectiveAllowed (S=TRUE). The DLE may, without system manager direction, autonomously and selectively treat transmit links as idle if they occur on certain radio channels with a history of poor connectivity. This is a form of selective channel utilization, and is described in 9.1.7.2.4. The DLE may skip links occurring on channels that it autonomously deems problematic due to a history of poor connectivity, potentially with the granularity of a specific channel used for communication with a specific neighbor. In this manner, the DLE can save energy and reduce unnecessary interference with other users of the spectrum. However, the DLE shall not skip links in this fashion when the link is flagged with SelectiveAllowed=0.

Table 183 shows allowed combinations of bits in the representation of dlmo.Link[ ].Type. Bits shown as X indicate that a value of FALSE (0) or TRUE (1) is allowed. For example, several combinations involving Transmit=TRUE show an X for Exponential backoff, indicating that such links might be configured as shared or not.

Table 183 – Allowed dlmo.Link[ ].Type combinations

Combination TRBI DDJS	Description
1X10 001X	Join response
10XX 000X	Transmit, no advertisement
11XX 000X	Transmit/receive, no advertisement
10X0 010X	Transmit, advertisement
0000 0100	Dedicated advertisement
0000 1100	Solicitation
010X 0000	Receive

A transmit/receive link combination is essentially a compressed representation of a transmit link and a separate receive link. If at least one outbound DPDU on the queue matches the link, it shall be treated as a transmit link using the primary timeslot template. Otherwise, it shall be treated as a receive link using the secondary timeslot template, with priority dlmo.LinkPriorityRcv.

It is possible to configure channel-hopping patterns, superframe periods and link intervals so that the result is that only certain radio channels in the channel-hopping sequence are actually used. For example, if a link repeats every 20th timeslot, and the channel-hopping pattern includes 15 channels, then only three channels will actually be used by the link. To avoid such scenarios, the link interval and the channel-hopping pattern may be configured to have a greatest common divisor equal to one (1).

Table 184 specifies the different types of schedules for a given link. Links in a superframe are indexed based on the timeslot offset in each cycle, with the first timeslot in each cycle having an offset of zero.

Table 184 – Values for dlmo.Link[ ].Schedule

Value for dlmo.Link[ ].Schedule	Element encoding	Description
0=offset only	ExtDLUint (offset)	Link occurs once at a fixed timeslot position in each superframe cycle
1=offset and interval	ExtDLUint, ExtDLUint (offset, interval)	Link occurs multiple times in each cycle, first at the given offset and then repeating at an interval until the end of the cycle. Values are specified in number of timeslots
2=range	ExtDLUint, ExtDLUint (first, last)	Link occurs at a range of slots in each superframe cycle, starting with the offset given by the first value and continuing until the offset given by the second value
3=bitmap	BooleanArray32	Bitmap covers the first 32 timeslots in each superframe cycle. Link occurs in timeslots with a corresponding TRUE value in the array. Following LSB conventions, the array is transmitted in DMAP messages with indices 7..0 transmitted first and indices 31..24 transmitted last

9.4.3.8 dlmo.Route

9.4.3.8.1 General

dlmo.Route is an indexed OctetString collection that contains routes. dlmo.Route describes available routes for DPDUs. When a DSDU comes down the protocol stack from the NL, it receives a final destination address, along with a contract ID and a priority class. The DLE maps the contract ID and destination address into a route, based on table lookups. The priority class from the NL is simply copied to the DPDU header without being considered in route selection.

The system manager inserts, updates, or deletes routes by sending the DMAP a route, along with a unique index and (if selected) a TAI cutover time.

For a description of route selection, see 9.1.6.5.

9.4.3.8.2 Semantics

Table 185 specifies the fields for dlmo.Route.

Table 185 – dlmo.Route fields

Field name	Field encoding
* Index	Type: ExtDLUint (used as an index)
Size (size (number of entries) of Route attribute)	Type: Unsigned4 Valid range: 1..15
Alternative	Type: Unsigned2
Reserved (octet alignment)	Type: Unsigned2=0
ForwardLimit (initialization value for the forwarding limit in DPDUs that use this route)	Type: Unsigned8
Route (series of routing destinations; if entry high-order bit is 0, specifies a unicast address; if entry high-order bits are 0x 1010, specifies a graph)	Type: SEQUENCE OF Unsigned16 (SIZE (size))
Selector (see text)	Type: Unsigned16 or null
SrcAddr (see text)	Type: ExtDLUint or null

Table 186 illustrates the structure of dlmo.Route.

Table 186 – dlmo.Route structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1..2	* Index							
1	Size				Alternative		Reserved=0	
1	ForwardLimit							
2	Route$_0$							
…	…							
2	Route$_{Size-1}$							
0 or 2	Selector							
0..2	SrcAddr							

The attribute dlmo.Route[ ].Selector depends on the setting of dlmo.Route[ ].Alternative:

- When dlmo.Route[ ].Alternative=0, select this route if dlmo.Route[ ].Selector matches ContractID and dlmo.Route[ ].SrcAddr matches the SrcAddr (source address) field in the DADDR subheader. This alternative shall not be used unless the DLE is a backbone router. If dlmo.Route[ ].Alternative<>0, dlmo.Route[ ].SrcAddr is null and shall not be transmitted.
- When dlmo.Route[ ].Alternative=1, select this route if dlmo.Route[ ].Selector matches ContractID.
- When dlmo.Route[ ].Alternative=2, select this route if dlmo.Route[ ].Selector matches destination address.
- When dlmo.Route[ ].Alternative=3, use this route as the default. dlmo.Route[ ].Selector is null and shall not be transmitted.

dlmo.Route[ ].Alternative shall be applied in order, with lower numbered Alternatives given precedence over higher numbered alternatives. There should be no more than one dlmo.Route entry per ContractID/SrcAddr combination (Alternative=0), no more than one entry per ContractID (Alternative=1), no more than one entry per destination address (Alternative=2), and no more than one default (Alternative=3). If there are duplicates, the matching entry with the lowest index shall be selected.

9.4.3.9 dlmo.NeighborDiag

9.4.3.9.1 General

dlmo.NeighborDiag is an indexed OctetString collection that contains diagnostics for a set of neighbors. The attribute is read-only, with rows created as needed by the DLE.

Each NeighborDiag entry comprises an array of one or two OctetStrings, with each entry corresponding to a different neighbor.

NeighborDiag entries are instantiated by the system manager, by setting dlmo.Neighbor[ ].DiagLevel bits to non-zero values. If and only if Bit0=1, then summary diagnostics shall be collected for the neighbor, consolidated across all channels. If and only if it1=1, then detailed clock diagnostics shall be collected for the neighbor, consolidated across all radio channels.

NOTE Individual channel diagnostics are collected through the attribute dlmo.ChannelDiag.

Diagnostics include counters and levels, that are accumulated as described in 9.1.15.3. Generally, counters are incremented by one in the course of successful or unsuccessful transactions, while RSSI (signal strength) and RSQI (signal quality) are levels that are accumulated as exponential moving averages.

NeighborDiag is reported in three general ways:

- Through the HRCO, the system manager can configure the DLE to report NeighborDiag periodically, such as every 30 min. Following each such report, on a per-entry basis, NeighborDiag counts shall be reset to zero. Levels shall use the current value as a starting point for the next period.
- The system manager can read (poll) NeighborDiag as a read-only attribute on a per-entry basis. As in an HRCO report, counts shall be reset to zero when read.
- The DLE can additionally be configured, through the dlmo.AlertPolicy attribute, to report NeighborDiag information when diagnostic values exceed a threshold. Only the row triggering the alert is reported. No values are reset.

Generally in this standard, an indexed OctetString's metadata capacity is reported as the number of rows. Since rows in NeighborDiag can have substantially variable sizes, metadata for NeighborDiag (DiagMeta) shall be reported in memory capacity in octets for the OctetStrings, with the convention that each ExtDLUint field is assumed to consume two octets. A DLE shall have the capacity for summary diagnostics (dlmo.NeighborDiag[ ].Summary) for at least half of its neighbor capacity as indicated by dlmo.NeighborMeta.Capacity, or for at least two neighbors, whichever is greater.

9.4.3.9.2 Semantics

Each NeighborDiag entry includes three OctetStrings, one each for Summary and ClockDetail diagnostics. A zero-length OctetString indicates that the diagnostic is not being accumulated. Table 187 specifies the fields for dlmo.NeighborDiag.

Table 187 – dlmo.NeighborDiag fields

Field name	Field encoding
* Index	Type: ExtDLUint (neighbor address, used as an index)
Summary	Type: OctetString
ClockDetail	Type: OctetString

Table 188 specifies the fields within the diagnostic summary OctetString.

Table 188 – Diagnostic summary OctetString fields

Field name	Field encoding
RSSI (level)	Type: Integer8
RSQI (level)	Type: Unsigned8
RxDPDU (count)	Type: ExtDLUint
TxSuccessful (count)	Type: ExtDLUint
TxFailed (count)	Type: ExtDLUint
TxCCA_Backoff (count)	Type: ExtDLUint
TxNAK (count)	Type: ExtDLUint
ClockSigma (level)	Type: Integer16

Table 189 specifies the structure of the diagnostic summary OctetStrings.

Table 189 – Diagnostic summary OctetString structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	RSSI							
1	RSQI							
1 or 2	RxDPDU							
1 or 2	TxSuccessful							
1 or 2	TxFailed							
1 or 2	TxCCA_Backoff							
1 or 2	TxNAK							
2	ClockSigma							

Fields include:

- RSSI (signal strength): See 9.1.15.2 for discussion of RSSI units. RSSI is accumulated as an exponential moving average; see 9.1.15.3.
- RSQI (signal quality): See 9.3.5.5 and 9.1.15.2 for discussion of RSQI units. RSQI is accumulated as an exponential moving average; see 9.1.15.3.
- RxDPDU: Count of valid Data DPDUs received from neighbor, excluding DPDUs with null DSDU payloads (as well as AKC/NAK DPDUs).
- TxSuccessful: Count of successful unicast transmissions to the neighbor, where an ACK DPDU was received in response.
- TxFailed: Count of DPDU unicast transmissions, where an ACK/NAK DPDU was expected but not received in response.
- TxCCA_Backoff: Count of unicast transmissions that were aborted due to CCA. These aborted transmissions are not included in TxFailed.
- TxNAK: Count of NAK DPDUs received, not included in TxFailed.
- ClockSigma: A rough estimate, to within one standard deviation, of recent clock corrections, in units of 2^{-20} s. A one-sigma value accounts for approximately 68% of clock corrections. ClockSigma is reset to zero whenever counters are reset.

NOTE 1 See 9.1.15.3 for the behavior of counters.

If the DLE autonomously treats a transmit link as idle, as described in 9.1.7.2.4, such skipped links shall not be counted in the neighbor diagnostics. However, such skipped links are reflected in channel diagnostics, as described in 9.4.2.27.

Table 190 specifies the fields within the diagnostic Clock OctetString.

Table 190 – Diagnostic ClockDetail OctetString fields

Field name	Field encoding
ClockBias (level, signed)	Type: Integer16
ClockCount (count)	Type: ExtDLUint
ClockTimeout (count)	Type: ExtDLUint
ClockOutliers (count)	Type: ExtDLUint

Table 191 specifies the structure of the diagnostic ClockDetail OctetString, with ExtDLUint fields shown as a single octet.

Table 191 – Diagnostic ClockDetail OctetString structure

Octets	Bits							
	7	6	5	4	3	2	1	0
2	ClockBias							
1 or 2	ClockCount							
1 or 2	ClockTimeout							
1 or 2	ClockOutliers							

If the neighbor is a preferred DL clock source, as indicated by IncludeClock=1, it is recommended that the DLE be configured to accumulate the ClockDetail fields.

ClockSigma, ClockBias, and ClockOutliers all relate to clock corrections. If the neighbor is a DL clock source, these values relate to clock corrections received from the DL clock source. If the neighbor is not DL clock source, these values relate to clock corrections sent to the DLE's neighbor through the ACK/NAK DPDU.

Fields include:

- ClockBias: An exponential moving average (EMA) of clock correction, in units of 2^{-20} s, including sign, with a 1% smoothing factor. See 9.1.15.3 for a discussion of EMA.

  NOTE 2 If this value is significantly non-zero it indicates that the clock is biased relative to the remote clock source.
- ClockCount: Count of clock updates received from or transmitted to the neighbor.
- ClockTimeout: Count of clock timeout events.
- ClockOutliers: Estimated count of clock corrections in excess of three standard deviations as per ClockSigma.

9.5 DLE methods

9.5.1 Method for synchronized cutover of DLE attributes

A Scheduled_Write method, with MethodID=1, is provided to set an attribute at a specific TAI time. It exactly follows the template found in Table J.1.

9.5.2 Methods to access indexed OctetString attributes

Various methods in the DLE relate to writing, reading, and deleting indexed OctetString attributes. These methods are generally based on the templates provided in Annex J.

All indexed OctetString attributes in the DL are indexed by a single integer encoded as an ExtDLUint (see 9.3.2.2). Following the convention of the template methods, these indexes are duplicated in the input arguments. For example, the Write_Row method includes an index as an input argument even though the index is also carried within the OctetString that constitutes the new entry.

Table 192 specifies the Read_Row method.

Table 192 – Read_Row method

Method name	Method ID	Method description		
Read_Row	2	Method to read the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Unsigned16	The attribute ID in the DLMO to which this method is being applied
	2	Index	Unsigned16	The * Index field in the attribute to access a particular row
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Data_Value	OctetString	An octet string that contains the contents of the row. If the row is empty, the OctetString shall contain only the Index, encoded as ExtDLUint

Table 193 specifies the Write_Row method.

Table 193 – Write_Row method

Method name	Method ID	Method description		
Write_Row	3	Method to set / modify the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Unsigned16	The attribute ID in the DLMO to which this method is being applied, as determined by the ordinal index of the attribute in the DLMO definition
	2	Scheduled_TAI_ Time	Unsigned32	TAI time in seconds at which the value should be written to the row of the structured attribute. If the time is in the past, relative to the receiving device’s time sense, the write shall be performed immediately
	3	Index	Unsigned16	The * Index field in the attribute to access a particular row
	4	Data_Value	OctetString	An octet string that contains the new contents of the row. If the DLMO row is unpopulated, a new row is created containing the OctetString if memory is available. If the DLMO row already exists, its contents are replaced with the OctetString. If the OctetString is null then the row shall be deleted

Method name	Method ID	Method description		
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

Table 194 specifies the Write_Row_Now method. It is identical to the Write_Row method, without the Scheduled_TAI_Time argument. It has the effect of writing an indexed OctetString row immediately on receipt.

Table 194 – Write_Row_Now method

Method name	Method ID	Method description		
Write_Row_Now	4	Method to set / modify the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Unsigned16	The attribute ID in the DLMO to which this method is being applied, as determined by the ordinal index of the attribute in the DLMO definition
	2	Index	Unsigned16	The * Index field in the attribute to access a particular row
	3	Data_Value	OctetString	An octet string that contains the new contents of the row. If the DLMO row is unpopulated, a new row is created containing the OctetString if memory is available. If the DLMO row already exists, its contents are replaced with the OctetString. If the OctetString is null then the row shall be deleted
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

9.6 DL alerts

9.6.1 DL_Connectivity alert

DLE performance diagnostics are accumulated in the attributes dlmo.NeighborDiag for per-neighbor diagnostics, and dlmo.ChannelDiag for per-channel diagnostics. Normally the system manager configures the HRCO to report these diagnostics periodically, and the DLE automatically resets the diagnostic counters whenever these attributes are so reported. Between such reports, diagnostics may indicate a problem that needs to be reported to the system manager immediately. The DL_Connectivity alert provides the mechanism for the DLE to report such issues, and the dlmo.AlertPolicy attribute enables the system manager to set thresholds for such reporting.

The attribute dlmo.AlertPolicy enables/disables the DL_Connectivity alert and provides thresholds to control whether alerts are reported. dlmo.AlertPolicy is an OctetString containing fields as shown in Table 195.

Table 195 – dlmo.AlertPolicy fields

Field name	Field encoding
Descriptor (enables or disabled the DL_Connectivity alert)	Type Alert report descriptor Default: Disabled=TRUE Default: Priority=0
NeiMinUnicast (minimum number of unicast transactions needed for a neighbor report)	Type: ExtDLUint
NeiErrThresh (report neighbor diagnostic if the percentage error rate reaches this threshold)	Type: Unsigned8
ChanMinUnicast (minimum number of unicast transactions on a channel needed as a pre-condition for triggering an alert)	Type: ExtDLUint
NoAckThresh (report ChannelDiag if a NoAck value is greater than this threshold)	Type: Unsigned8
CCABackoffThresh (report ChannelDiag if a CCABackoff value is greater than this threshold)	Type: Unsigned8

Table 196 specifies the structure of the dlmo.AlertPolicy OctetString.

Table 196 – dlmo.AlertPolicy OctetString structure

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Descriptor							
1 or 2	NeiMinUnicast							
1	NeiErrThresh							
1 or 2	ChanMinUnicast							
1	NoAckThresh							
1	CCABackoffThresh							

Fields include:

- dlmo.AlertPolicy.Descriptor determines whether or not the DL_Connectivity alert is enabled. By default, DL_Connectivity alert is disabled until the system manager enables it by populating this attribute with appropriate thresholds. When Disabled=TRUE, all other dlmo.AlertPolicy fields are meaningless and ignored.
- dlmo.AlertPolicy.NeiMinUnicast sets a minimum number of attempted unicast transactions before an error rate is considered significant. The count of attempted unicast transactions for a neighbor is the sum of the dlmo.NeighborDiag fields

    TxSuccessful+TxFailed+TxCCA_Backoff+TxNAK.

  If this sum is less than NeighborTxMinReport, do not send a DL_Connectivity alert for the neighbor.
- dlmo.AlertPolicy.NeiErrThresh sets the threshold for reporting a DL_Connectivity alert for a neighbor. The percentage error rate is calculated as:

    (TxFailed + TxCCA_Backoff + TxNAK) x 100 /
    (TxSuccessful + TxFailed + TxCCA_Backoff + TxNAK)

  If this value is greater than NeiErrThresh, the diagnostics for the neighbor should be reported using the DL_Connectivity alert unless there is an insufficient number of unicast transactions to the neighbor or the same alert has been recently reported.

- dlmo.AlertPolicy.ChanMinUnicast is similar to NeiMinUnicast. Counters underlying dlmo.ChannelDiag are not exposed, but a count of attempted unicast transactions is implicit in the reported ratios.
- dlmo.AlertPolicy.NoAckThresh and dlmo.AlertPolicy.CCABackoffThresh provide thresholds for reporting. Since the values reported by dlmo.ChannelDiag are ratios, the reported values are simply compared to the thresholds. If the value exceeds the thresholds, dlmo.ChannelDiag should be reported through the DL_Connectivity alert unless the ChanMinUnicast requirement is not met or the same alert has been recently reported.

The system manager may respond to the DL connectivity alert by collecting diagnostics to more fully characterize the situation. Alternatively, particularly if a modified topology is easily achieved, the system manager may simply reconfigure the D-subnet topology.

Table 197 illustrates the structure of the DL_Connectivity alert.

Table 197 – DL_Connectivity alert

Standard object type name: DL management object (DLMO)					
Standard object type identifier: 124					
Description of the alert: Poor neighbor connectivity					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority**	**Value data type**	**Description of value included with alert**
Event	Comm	0 = DL_Connectivity	Medium	Type: DL16Address	See Table 187

The format of the OctetString transmitted with the DL_Connectivity alert is shown in Table 198. It is simply the attribute number for either dlmo.ChannelDiag (48) or dlmo.NeighborDiag (46), followed by an OctetString containing the diagnostic data from that attribute. In the case of ChannelDiag, the entire attribute is transmitted. In the case of NeighborDiag, only the row that triggered the alert is transmitted, with the neighbor address specified within the row identifying the neighbor.

Table 198 – DL_Connectivity alert OctetString

Octets	Bits							
	7	6	5	4	3	2	1	0
1	AttributeNumber (Unsigned8)							
N	Attribute (OctetString)							

9.6.2 NeighborDiscovery alert

As described in 9.4.2.24, the NeighborDiscovery alert provides a mechanism for the DLE to report the contents of the OctetString in dlmo.Candidates attribute.

Table 199 illustrates the structure of the NeighborDiscovery alert.

Table 199 – NeighborDiscovery alert

Standard object type name: DL management object (DLMO)					
Standard object type identifier: 124					
Description of the alert: Neighbor discovery alert					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority**	**Value data type**	**Description of value included with alert**
Event	Comm	1 = NeighborDiscovery	Medium	Type: OctetString	An exact copy of the OctetString in dlmo.Candidates; see 9.4.2.24

10 Network layer

10.1 General

Clause 10 provides an overview of NL functionality. It also describes conceptual services that the NL offers to the layer above it (transport), the NL management object (NLMO), and the structure of NPDUs.

NOTE NPDU header formats have been designed for compatibility with RFC 6282.

The NL follows the big endian convention; multi-octet fields are documented and transmitted with the high-order octet first (since they are treated as a series of octets by the lower layer). Within an octet, bits are documented starting from the high-order bit (bit 7) on the left and continuing to the low-order bit (bit 0) on the right.

Parts of Clause 10 present notional implementation aspects as if they were subject to conformance testing. Where such aspects are not externally observable, any such specifications are strictly hypothetical. Only observable, testable aspects of Clause 10 are normative.

10.2 NL functionality overview

10.2.1 General

The NL in this standard performs the following functions:

- Addressing: An NLE determines the appropriate address information for an NPDU.
- Address translation: This standard uses primarily two types of addresses, short DL16Addresses, and long IPv6Addresses. The short DL16Addresses are used within a D-subnet to conserve energy and bandwidth. ALEs, TLEs and NLEs on backbone networks use long IPv6Addresses. The NLE is responsible for translation between the various types of addresses, e.g., when an NPDU moves from a D-subnet to a backbone network (or vice versa).
- NPDU formats: This standard allows for more than one NPDU format to accommodate conservation of energy and bandwidth (which favors short headers), a variety of network topologies, and internetworking with backbone networks. The NLE selects an appropriate format for the NPDU based on such considerations as addressing, routing, level of service, etc.
- Fragmentation and reassembly: NPDU fragmentation and reassembly occurs within the NLE. An NPDU of a size of more than the maximum DSDU size is fragmented by the sending NLE at the point of ingress into a D-subnet. Reassembly is performed by the receiving NLE at the point of egress from the D-subnet.

- Routing: This standard performs routing at two levels: within the backbone network and within the mesh D-subnet. Responsibility for routing at the NL and DL protocol layers is the responsibility of the respective layer entities.

10.2.2 Addressing

ALEs and TLEs in this standard use IPv6Addresses. Each NLE shall have an IPv6Address. If the NLE does not have an IPv6Address prior to the network join process, the NLE shall be assigned such an IPv6Address by the system manager during the joining process. The NL uses these IPv6Addresses, but does not associate any further meaning to them.

Each NLE compliant with this standard shall also have an IPv6Address that is autoconfigured by the NLE as part of the initialization of its protocol stack. This IPv6Address is referred to as the NLE's link-local address and is derived from the associated DLE's EUI64Address. The format of this IPv6Address is that of a link-local unicast address, as defined in IETF RFC 4291, 2.5.6. Table 200 illustrates this address structure.

Table 200 – Link-local address structure

10 bits	54 bits	64 bits
11 1111 1010	0	EUI64Address

When DPDUs are transmitted over a D-subnet, conveyance of IPv6Addresses consumes valuable bandwidth and device energy resources. Thus this standard defines 16-bit D-aliases for IPv6Addresses so that the short D-aliases are used over the D-subnet. For each D-subnet, a unique DL16Address shall be assigned to each DLE within that D-subnet, as well as to each DLE outside the D-subnet with which a DLE within the D-subnet has a contract. This allows short D-addresses to be used in the D-subnet to represent all origin and destination NLEs.

The scope of a DL16Address is the D-subnet within which it has been defined. Thus, a particular device may have one D-address in the D-subnet to which it belongs and a different D-address in a foreign D-subnet. When a DL16Address is used, it is carried in the DPDU's header.

During the joining process, an NLE might not yet have an IPv6Address and its associated DLE might not have a DL16Address. In this case, TPDUs between the joining device and the advertising D-router shall use the link-local IPv6Addresses when needed (e.g., for the TPDU pseudo-header in join TPDUs). The joining device and the advertising router shall be identified as such by using their EUI64Addresses in the DPDU headers that convey the join messaging.

The system manager assigns the DL16Address and IPv6Address of each DLE and NLE, respectively, that operate in a WISN conforming to this standard. These addresses are assigned during the join process. The NLE specified by this standard supports only unicast addressing.

NOTE 1 Multicast addressing is a subject for future standardization.

NOTE 2 Backbone and plant network technologies are outside the scope of this standard. Therefore this standard does not specify the representation of IPv6Addresses on a particular backbone or plant network.

10.2.3 Address translation

Since this standard employs DL16Addresses within a D-subnet, when a NPDU moves from a D-subnet to a backbone network (or vice versa), the NLE of the backbone router shall translate between the DL16Addresses and the IPv6Addresses. The same kind of translation shall be performed by the NLE of a D-subnet endpoint.

All devices in this standard shall maintain an address translation table (ATT), as shown in Table 201.

Table 201 – Address translation table (ATT)

D-address (DL16Address)	N-address (IPv6Address)
N1_16	N1_128
N2_16	N2_128
GW_16	GW_128
BBR_16	BBR_128
SM_16	SM_128

The address translation table is initialized during the join process with the DL16Address and the IPv6Address of the system manager. This information is part of the non-security component of join response, received from the system manager as described in 6.3.9.2.

The address translation table shall be updated by the source NLE whenever a communication session is established with a new destination NLE. Communication sessions are described in 6.3.11.2.5.2. The DL16Address and the IPv6Address of the destination NLE and its associated DLE are stored in the address translation table upon the successful completion of the session establishment process. The process of session establishment is described in 7.5. If a session is terminated for whatever reason, any entry associated with the destination device shall be deleted.

An NLE maintains entries in its ATT for other NLEs with which it communicates; these other NLEs may either belong to the same D-subnet as the first NLE or have a DL16Address in the same D-subnet.

Within a particular D-subnet, an NLE (whether local or remote) shall have only one DL16Address. Thus, the ATT can be used for both forward and reverse lookup by the NLE:

- IPv6Address determined through ATT_lookup of a DL16Address;
- DL16Address determined through ATT_lookup of an IPv6Address.

It is possible to package multiple NLEs or DLEs in a single physical device to support multi-homing. Although such operation is not specified by the standard, it is not prohibited.

An address with no entry in the ATT shall be translated with the help of the system manager. For each NLE joining the network, the system manager shall maintain the IPv6Address of the NLE and the associated DL16Address or D-alias for each D-subnet in which the NLE has such an alias. Hence, the local ATT at an NLE shall be updated through the system manager.

The ATT is an integral part of the NLMO and can be directly updated by the system manager by using the NLMO manipulation methods described in Table 210.

If a lookup in the ATT yields no results, then the lookup function notifies the NLMO. The NLMO issues a read primitive to the directory service object (DSO) in the system manager to obtain the appropriate translation. The lookup function return a value of null if the system manager also has no mapping for a particular address or the system manager is not available. Any new information from the system manager is stored in the ATT table.

This process is illustrated in Figure 91.

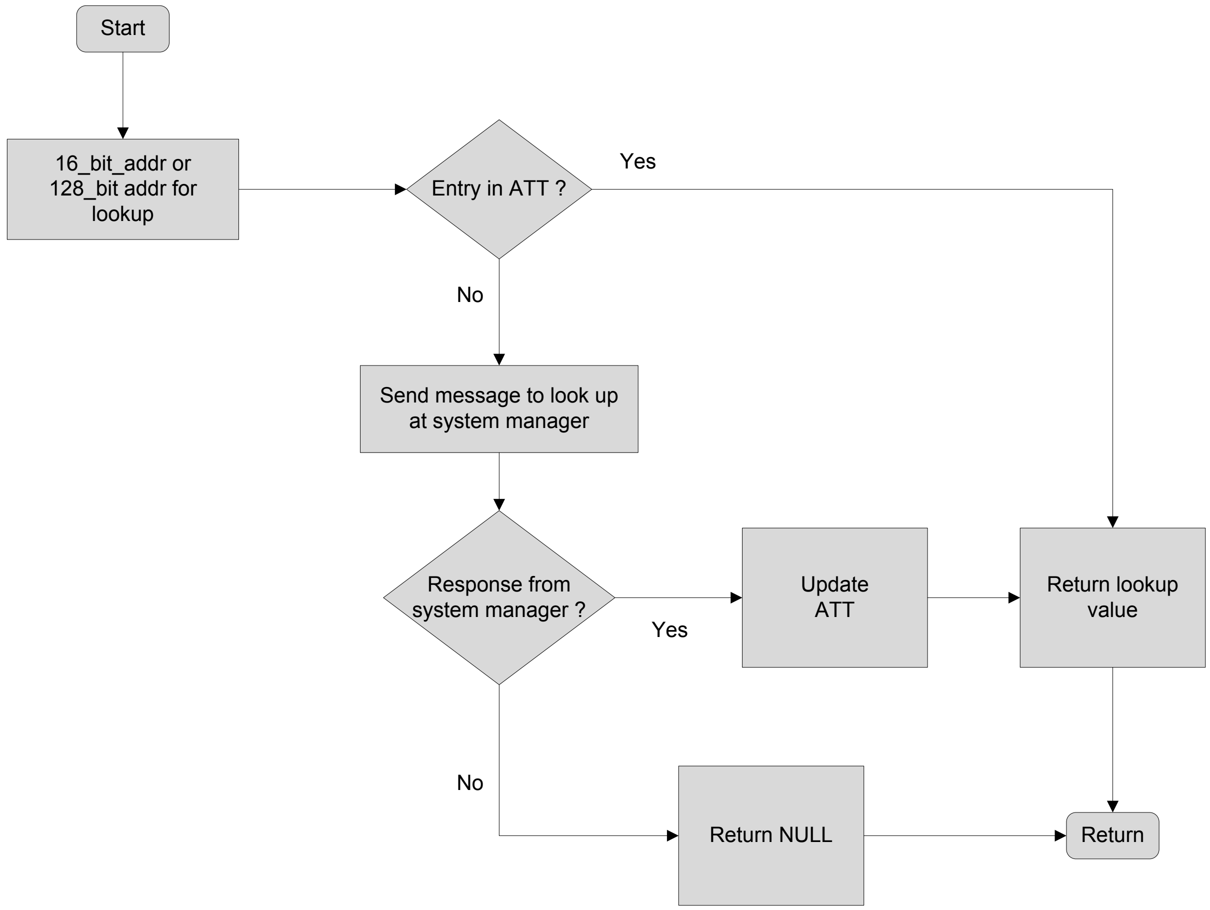


Figure 91 – Address translation process

10.2.4 Network protocol data unit headers

Three formats are used for NPDU headers. The value of the header's first octet provides the means to distinguish between these formats:

- Basic header: This format is intended for NPDUs traversing a single D-subnet and shall be used only over that D-subnet. It is expected to be the most common format in use because its use minimizes the overhead associated with the transmission of headers. The basic header is just an abbreviation for a specific fixed value of the 6LoWPAN compressed header; this value indicates that the source and destination addresses are elided, instead being conveyed in the DPDU header, as described in 10.5.2.
- Contract-enabled header: This format also is used only over a single D-subnet, when the originating device needs to include more information in the NPDU, such as a contractID. This additional information allows backbone routers to select appropriate resources (e.g., graphID, priority) for the routing of the NPDU, as described in 10.5.3.
- Full header: This is a full IPv6 header, suitable for use over the backbone. NPDUs containing a basic or contract-enabled header shall be expanded into the full header format before routing over the backbone. In return, backbone routers convert full headers into basic or contract-enabled headers for transmission over a D-subnet, as described in 10.5.4.

10.2.5 Fragmentation and reassembly

If the entire NPDU is smaller than the maximum DSDU size, the NPDU shall not be fragmented and the network header shall not contain a fragmentation header. If the NPDU exceeds the maximum DSDU size, the NPDU shall be fragmented into fragmented NPDUs that do not exceed the D-subnet's maximum DSDU size. Fragmentation shall be performed by the NLE at the point of ingress into a D-subnet. Reassembly shall be performed by the NLE at the point of egress from a D-subnet.

NOTE Origination by a TLE in a D-subnet-connected device constitutes “a point of ingress” into the D-subnet, and similarly delivery to a TLE in a D-subnet-connected device constitues “a point of egress” from the D-subnet.

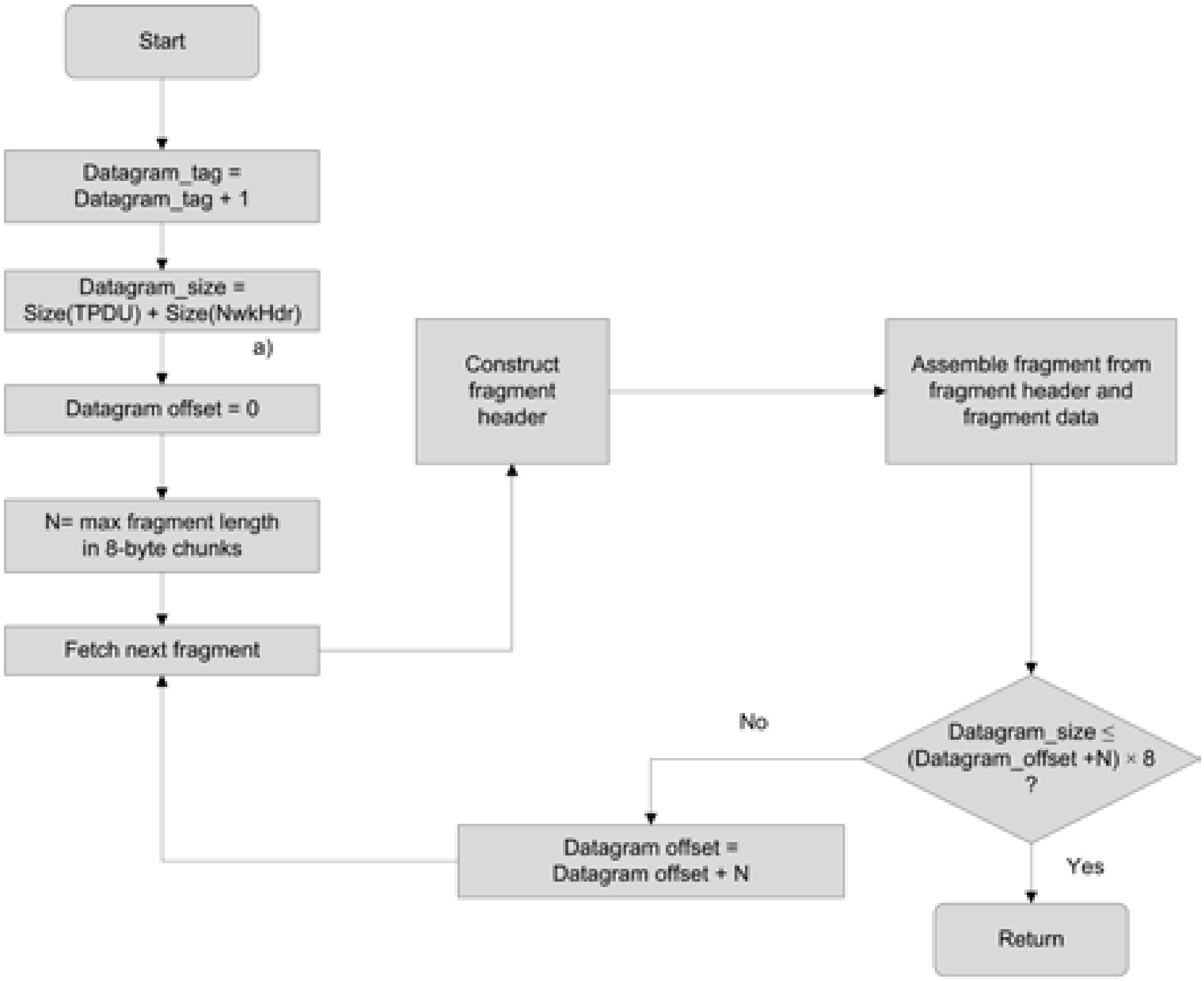
The first fragment shall contain the first fragment header as defined in Table 219. The second and subsequent fragments (up to and including the last fragment) shall contain a fragmentation header as defined in Table 220. The offset of this fragment, referred to as the datagram offset, shall be expressed in units of eight octets, so that each fragment other than the last consists of a multiple of eight octets.

The Datagram_size field shall be present in every fragment, to simplify the reassembly tasks when fragments arrive out of order at their reassembling NLE. The inclusion of the Datagram_size in every fragment allows the receiver to allocate the appropriate amount of buffer space when the first fragment is delayed.

All fragments (first and subsequent fragments) shall have a Datagram_tag field in their header. The value of this field shall be assigned by the device performing the fragmentation and shall be the same for all fragments of the NPDU, so that the reassembling device can recognize that the fragments belong to the same NPDU. To the extent possible, the NLE performing the fragmentation shall assign a different Datagram_tag value to each distinct NPDU that it fragments. To achieve this, each NLE shall have a counter that is initialized to a uniform-random value and is incremented for each NPDU that undergoes fragmentation; the value of this counter shall be placed in the Datagram_tag field of each fragment of the NPDU.

In the extremely rare case that two NPDUs from the same source to the same destination are fragmented by different intermediate routers that coincidentally pick the exact same Datagram_tag, the reassembling device may not be able to disambiguate fragments. In this case, the GraphID may be used to disambiguate further; however, this is not specified as mandatory in this standard. TPDUs reassembled in error from multiple sources will be dropped due to checksum errors and retransmitted. Intermediate routers that fragment NPDUs may also coordinate their fragmentation state machines in order to avoid scenarios in which the reassembling device might not be able to disambiguate fragments.

Figure 92 illustrates the fragmentation process.



a) This is the size of the NwkHdr excluding any contained fragmentation subheader.

Figure 92 – Fragmentation process

To identify all fragments that belong to the same NPDU, the reassembling NLE shall use:

- the source IPv6Address;
- the destination IPv6Address;
- the datagram_tag; and
- the datagram_size.

Otherwise the NLE shall begin reconstructing the original unfragmented NPDU, whose size is Datagram_size, using the Datagram_offset field to determine the relative location of the individual fragments within the original unfragmented NPDU.

When a NLE first receives a fragment with a given Datagram_tag that requires reconstruction, it starts a reassembly timer. If this timer expires before the entire NPDU has been reassembled, the received fragments shall be discarded. The reassembly timeout shall be set to a value defined in nlmo.Frag_Reassembly_Timeout (attribute identifier 11 in Table 206). If a fragment that partially overlaps another fragment is received, and it differs in either the size or Datagram_offset of the overlapped fragment, the fragment(s) already accumulated in the reassembly buffer shall be discarded.

The text just before Figure 92 provides one example of how such inconsistent fragmentation can arise.

A new reassembly commences with a fragment containing a tag for which no fragments are pending. This may lead to buffers being allocated when some fragments arrive after the timeout of the reassembly process that had been previously initiated for the same tag (in essence, attempting to reassemble the NPDU a second or later time). That repeated reassembly usually will fail to complete, causing the new buffers to eventually be flushed due to a nlmo.Frag_Reassembly_Timeout.

The reassembly process is completed when the NPDU is fully reassembled or the timer expires. If the NPDU exceeds the size indicated by nlmo.Max_NSDU_size, the reassembly process may be aborted and the NPDU may be discarded. The device may send a dropped PDU/PDU error alert with value 7 indicating that it is out of memory. Dropped PDU/PDU error alerts are shown in Table 211.

The NPDU reassembly process is shown in Figure 93.

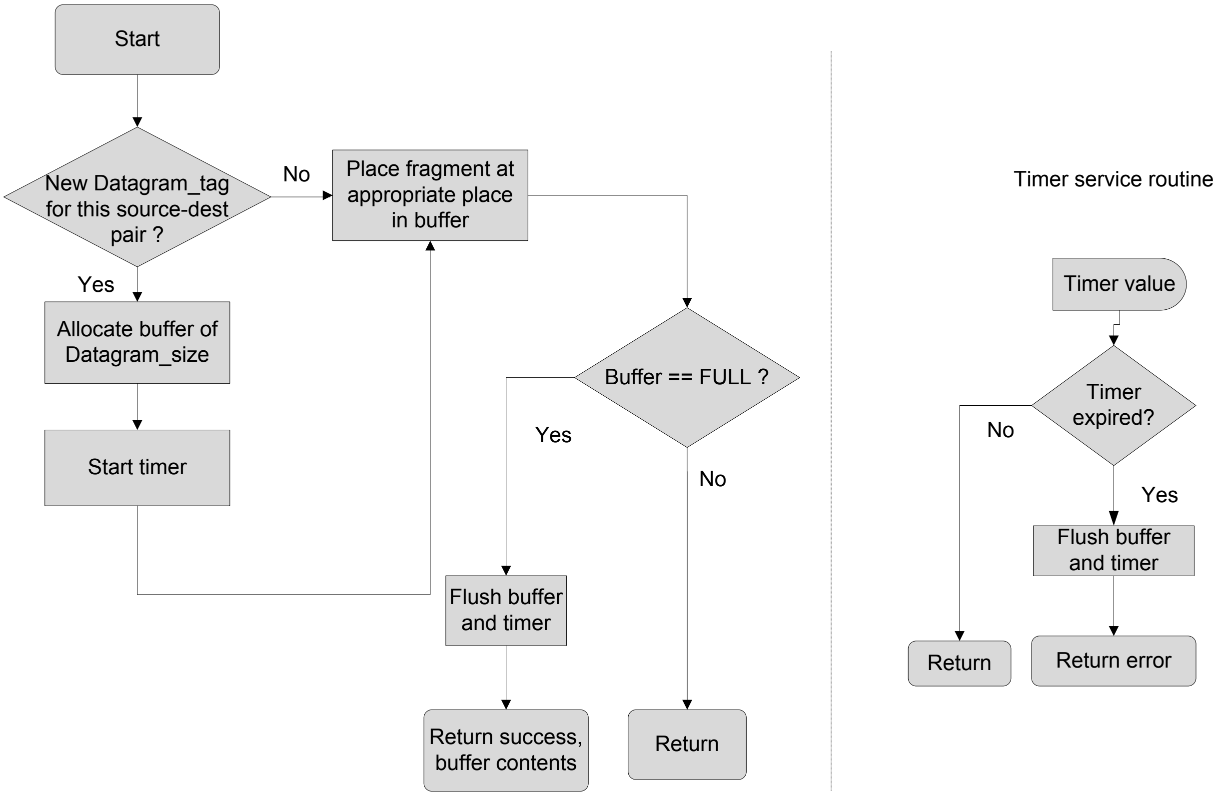


Figure 93 – Reassembly process

10.2.6 Routing

10.2.6.1 General

Routing within a network compliant with this standard happens at two levels:

- One level comprises the endpoints and the backbone devices, if any; the NL is responsible for routing PDUs at this level. This level does not handle routing over the DL links; traversal of a D-subnet appears as a single hop to an NLE.
- The second level of routing is within a D-subnet. This level is the responsibility of the DL (a layer 2 mesh implementation).

The routing between D-subnets and backbone networks is the responsibility of the NL, whose NPDUs conform to the IETF IPv6 and 6LoWPAN standards. This standard specifies minimum requirements for routing, along with notional management services for adding, deleting, and maintaining routes.

10.2.6.2 Routing tables

The NLE in devices compliant with this standard shall maintain a routing table (RT) to keep track of the next hop for a given destination. This table shall be maintained using IPv6Addresses, since such addresses are unique across an entire network compliant with this standard (including all D-subnets). An example of a routing table is provided in Table 202. The routing table may be updated at the source device whenever a communication session is established with a destination device.

Table 202 – Example of a routing table

DestinationAddress	NextHop	NWK_Hop_Limit	OutgoingInterface [a)]
N1	BBR1	2	Backbone
N2	BBR1	2	Backbone
GW	GW	2	Backbone
N3	N3	1	D-subnet
N4	N4	1	D-subnet
N5	N5	1	D-subnet
…	…	…	…

[a)] This field is set to D-subnet for all destinations in routers and I/O devices.

NOTE In this standard, the route table and all NL management objects are specified to support only one active D-subnet at a time. All DL16Addresses are unique within the scope of that single D-subnet. This is not intended to prevent a device from participating in multiple D-subnets simultaneously. Multiple D-subnets are represented by multiple NLEs.

DLEs that are not backbone-capable only route DPDUs within the D-subnet. Routing within the D-subnet is the responsibility of the DL (a layer 2 mesh implementation). Hence DLEs that operate in the D-subnet but are not backbone capable may maintain a routing table but are not required to do so. This is also reflected in Table B.18 that normatively presents the minimum routing table sizes that need to be supported by devices that meet various role profiles. NL routing tables provide layer independence and allow potential route-over implementations, where routing within the D-subnet is achieved through NL routing.

The routing table shall also be used by the backbone routers to decide whether to route a PDU over the backbone or over the D-subnet of this standard. The OutgoingInterface field indicates whether the PDU shall be sent over the backbone or over the D-subnet.

NextHop indicates the next device whose NLE shall process the NPDU destined for the DestinationAddress. Any device reachable through the DL mesh has NextHop equal to the destination address and the NWK_Hop_Limit field set to 1. From the perspective of the NLE, any device that is reachable through the DL mesh is a single network hop away.

10.2.6.3 Processing of a network service data unit received from a TLE

When an NSDU is passed to an NLE by a TLE, the NLE determines the final destination for that NSDU based on the ContractID. The contract table (see Table 207) is used to obtain the destination address. Devices with both a backbone and a DL interface compliant with this standard shall look up the destination address in the routing table to determine which network interface to use. All non-backbone DLEs shall always use their DL interface.

The NLE shall use the ContractTable to obtain the two-bit priority for the contractID in the N-Data.request. This contract priority shall be combined with the two bits of message priority (also passed in the N-Data.request) to obtain a 4-bit NPDU priority that is passed down to the DLE; the two most significant bits shall be the contract priority, and the two least significant bits shall be the message priority. The Discard Eligible (DE) field from the N-Data.request is also passed down to the DLE. If the OutgoingInterface for the destination address is the backbone then the 4-bit priority and DE eligible bits shall be included in the TrafficClass field of the IPv6 header.

The NLE shall use the ContractTable to check if the ContractID needs to be included in the NPDU. Including the ContractID in the NPDU allows intermediate backbone routers to make appropriate routing choices (level of service, graphID, etc.) on the backbone or a different D-subnet. When routing over the DL interface, if ContractID need not be included, then a basic NPDU header should be constructed; otherwise, a contract-enabled NPDU header should be constructed.

The NLE shall also determine whether fragmentation is needed for the NPDU and shall perform the fragmentation process if necessary. Fragmentation shall be required only for NPDUs routed over a D-subnet; dlmo.MaxDSDUSize shall indicate the maximum payload that can be carried over the D-subnet. If the DSDU size is greater than this value, then fragmentation is necessary.

BBR caching mechanisms and inter-BBR forwarding and reassembly protocols can provide the necessary functionality to permit NSDU fragments that arrive (from the D-subnet) at differing BBRs to be reassembled and forwarded by one of those BBRs.

NOTE A future edition of this standard may specify such an inter-BBR mechanism and protocol.

Unless the configuring system manager knows that the selected BBRs have the necessary capability, NPDUs requiring fragmentation shall not use D-subnet routes that terminate in more than one BBR, because the non-initial NPDUs resulting from 6LoWPAN fragmentation do not carry sufficient information for them to be routed directly to their intended final destination on the backbone subnet before reassembly has occurred.

Figure 94 illustrates the processing of an NSDU received from a TLE.

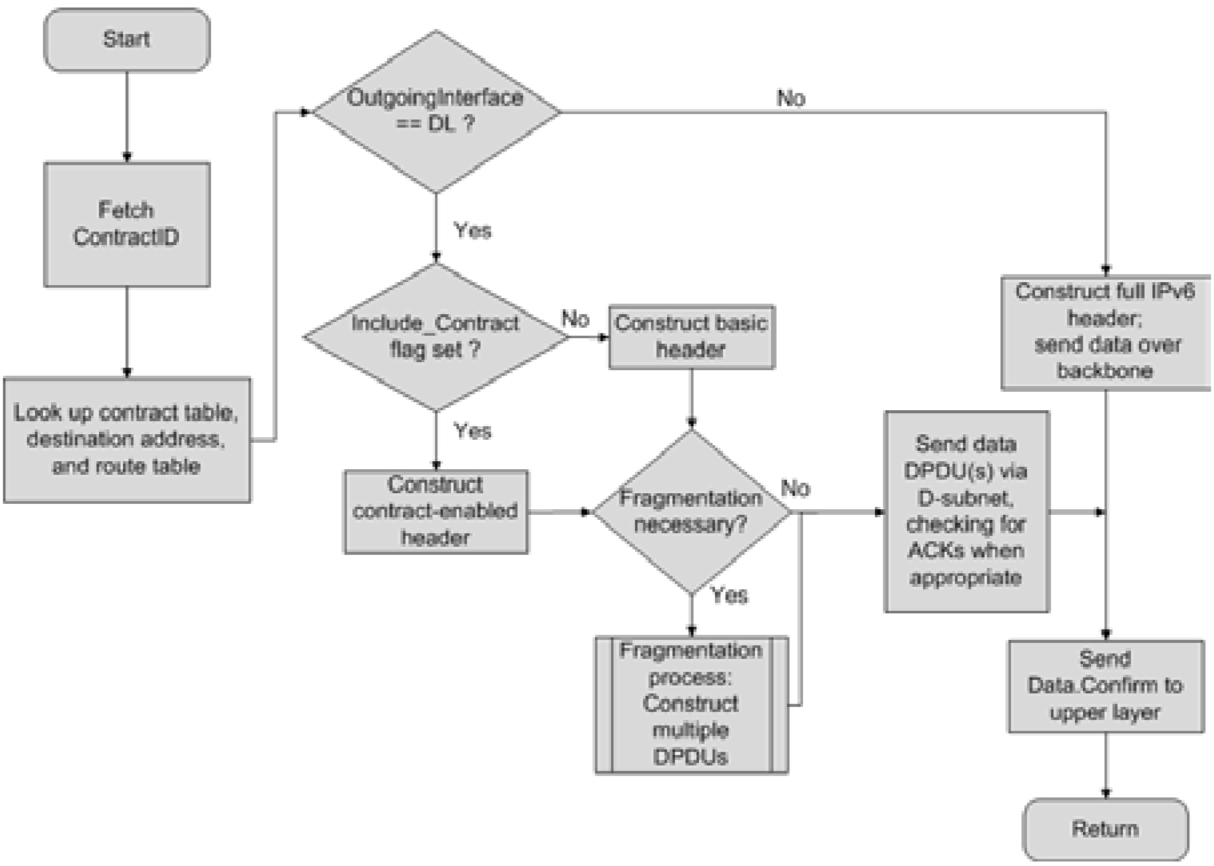


Figure 94 – Processing of a NSDU received from a TLE

10.2.6.4 Processing of a received NPDU

A received NPDU (i.e., a packet), whether received from the DL or the backbone interface, shall first be checked to determine if the final destination is a TLE of the current device. If so, the NSDU that is conveyed as the payload of the NPDU (after any required NPDU defragmentation) shall be passed to the collocated TLE. If the final destination is not the current device, then the device shall route the NPDU appropriately (via either the backbone or the associated DLE). The overall decision process is shown in Figure 95. Not all packets received from the DLE will have a corresponding DL16Address entry in the ATT. Some devices operating on the backbone may not have an assigned DL16Address, but only an IPv6Address. In such a case the NPDU will be delivered by the local DLE after being forwarded from the sending remote DLE over a default route. In that case the backbone-capable device will directly look up the route associated with the destination IPv6Address.

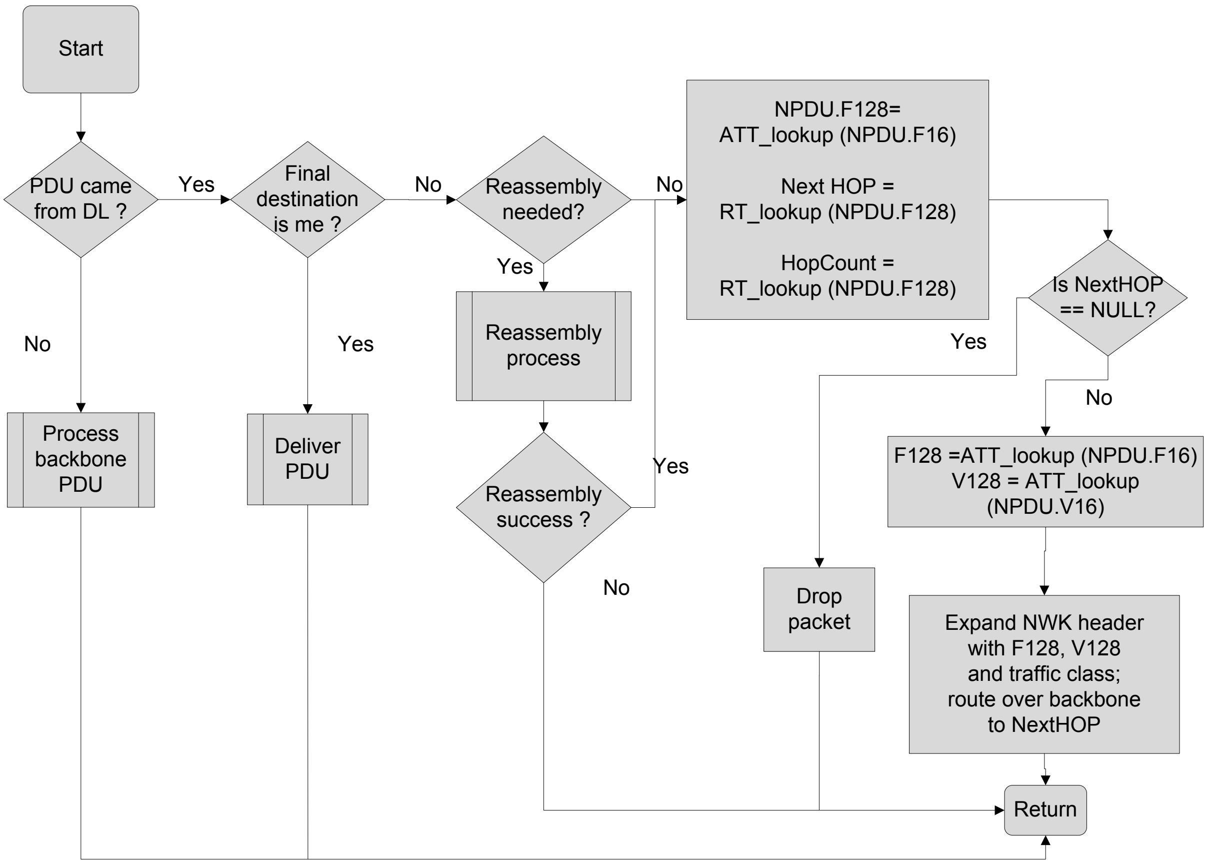


Figure 95 – Processing of a received NPDU

The DLE's notional DD-DATA.indication and DD-DATA.request services convey a LastHop (LH) parameter. When this LH parameter is set, it indicates that the PDU entered the D-subnet through a backbone router, and therefore is prohibited from exiting the D-subnet through a backbone router, thus avoiding circular routes within the NL. This restriction enables the NLE to elide the Hop Limit field from a compressed NPDU that uses the basic header format while still preventing circular routing.

When the NPDU is received from the DL at a device other than the destination, if the LastHop (LH) parameter is set in the DD-DATA.indication, the NPDU has reached the current device in error and shall be discarded. If the NPDU is received from the D-subnet and not discarded (see Figure 95), the intermediate router shall first fully expand the NPDU's network header. As part of this expansion, the explicit congestion notification (ECN) value provided by the DD-DATA.indication shall be included in the appropriate field of the expanded header.

After any header expansion, the receiving DLE shall check to see whether reassembly (due to prior fragmentation) is needed for this NPDU. Once any needed reassembly completes, the NPDU shall be prepared for routing over the backbone. The DL16Address of the origin (very first V) and destination (final destination F) in the DD-DATA.indication shall be translated into IPv6Addresses. Then the routing table shall be used to determine the next NL hop for reaching the final destination. The NPDU shall be presented to the NLE of the backbone interface for routing on the backbone network. This standard does not specify how the backbone handles and routes the NPDU. The backbone has the responsibility to deliver the NPDU to the NLE of the NextHop.

This standard always uses ECN. When congestion notification is carried in a DPDU header, if the ECN bits are non-zero in the NPDU header, they shall be set to zero in that header to indicate that the notification is carried in the DPDU header. A backbone router NLE that receives a potentially-reassembled NPDU from its associated DLE shall use the ECN information carried in the received DPDU header to fill in the ECN bits in the expanded NPDU header. NPDUs originating from backbone devices shall have the ECN bits set to indicate that explicit congestion notification is used.

If an NPDU is received from the backbone, it will have an expanded header, and the final destination and very first (originator) addresses will already be expressed as IPv6Addresses. If the NPDU needs to be routed over a D-subnet, the DL16Addresses in that D-subnet of the very first (originator) DLE and final destination DLE shall be obtained from the ATT and passed to the DLE in the DD-DATA.request. The NLE shall check if the ContractID and priority are included in the FlowLabel and TrafficClass fields, respectively, of the expanded NPDU header. If so, the ContractID and priority shall also be passed to the DLE to allow the selection of appropriate DL routing mechanisms (GraphID, etc.).

The presence or absence of congestion is determined from the ECN field of the NPDU received from the backbone, which is passed to the local DLE in a DD-DATA.request. When passing an NPDU with a basic header to a local DLE, then the LastHop (LH) parameter shall be set in the DD-DATA.request to indicate that the NPDU has entered a D-subnet from which it is not allowed to exit. If the NPDU size exceeds dlmo.MaxDsduSize for this D-subnet, the NPDU shall be fragmented before conveyance as DSDUs. This process is depicted as a flowchart in Figure 96.

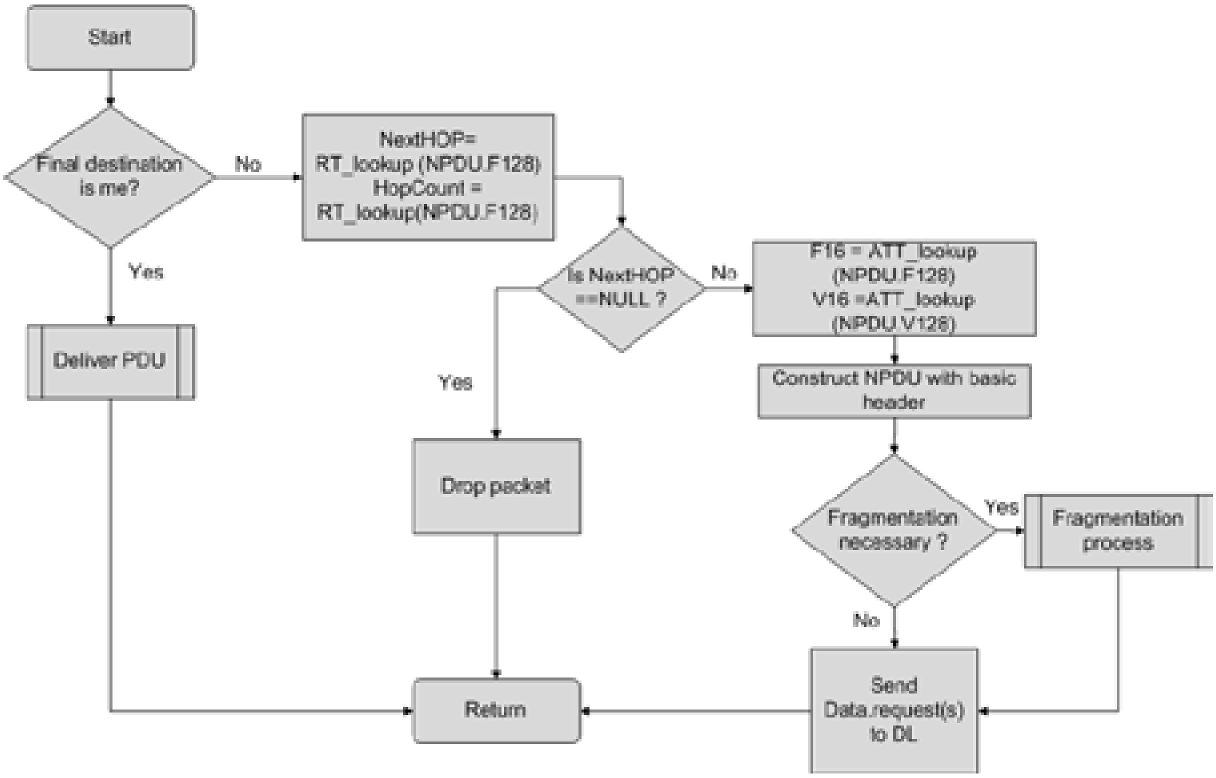


Figure 96 – Processing of a NPDU received by a NLE from the backbone

If the receiving NLE is the intended final destination, then that NLE shall process the NPDU and shall pass the conveyed NSDU up to an associated local TLE, along with an indication of whether congestion was encountered, as conveyed by the NPDU’s ECN bits. The NLE shall first check if it has received a fragment; if so, it shall perform the reassembly process (see Figure 93). The NPDU’s source IPv6Address shall be translated to an IPv6Address, if necessary, and the NSDU shall be passed to the associated TLE. Figure 97 depicts the flowchart for this processing.

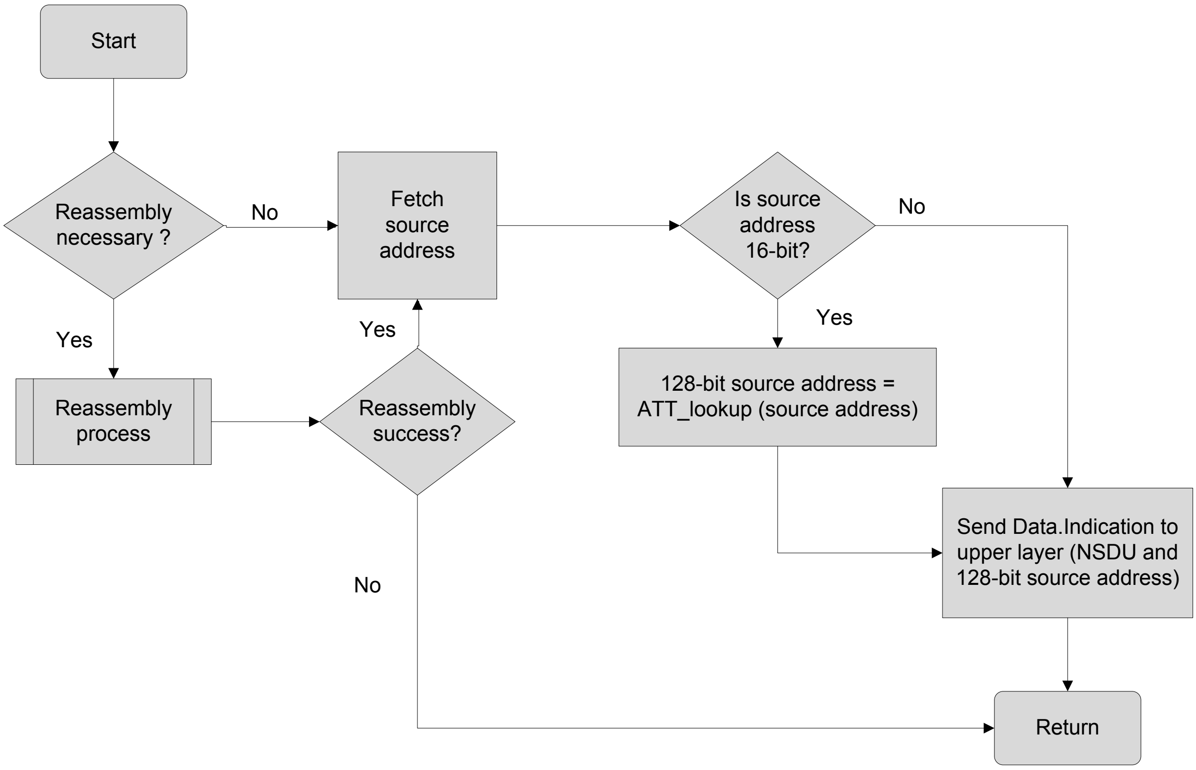


Figure 97 – Delivery of a received NPDU at its final destination NLE

10.2.7 Routing examples

10.2.7.1 Routing from a field device direct to a field-connected gateway

Figure 98 illustrates routing from a field device to a gateway with no backbone routing.

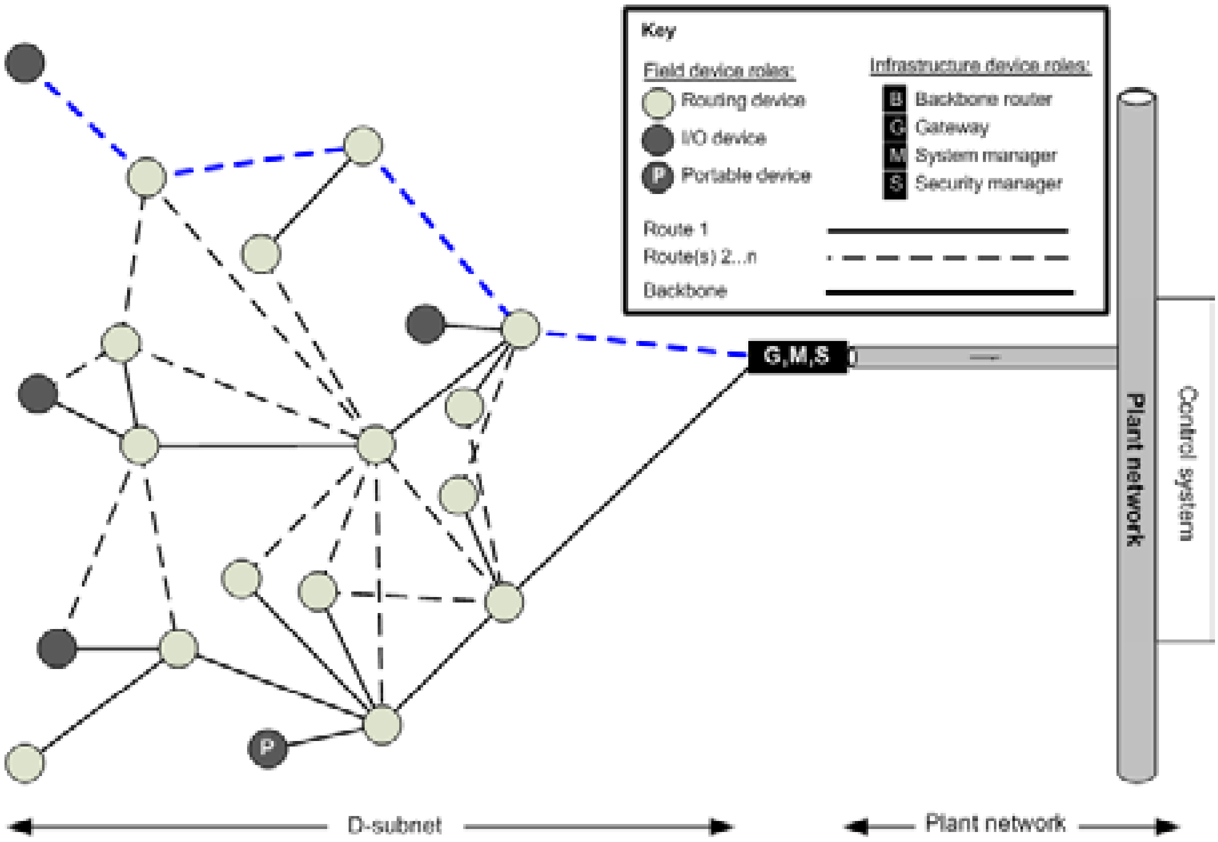


Figure 98 – Routing from a field device direct to a field-connected gateway without backbone routing

Figure 99 depicts the flow through the communication protocol suites as the PDU moves from an I/O device to the gateway. It is assumed that the NPDU needs no fragmentation.

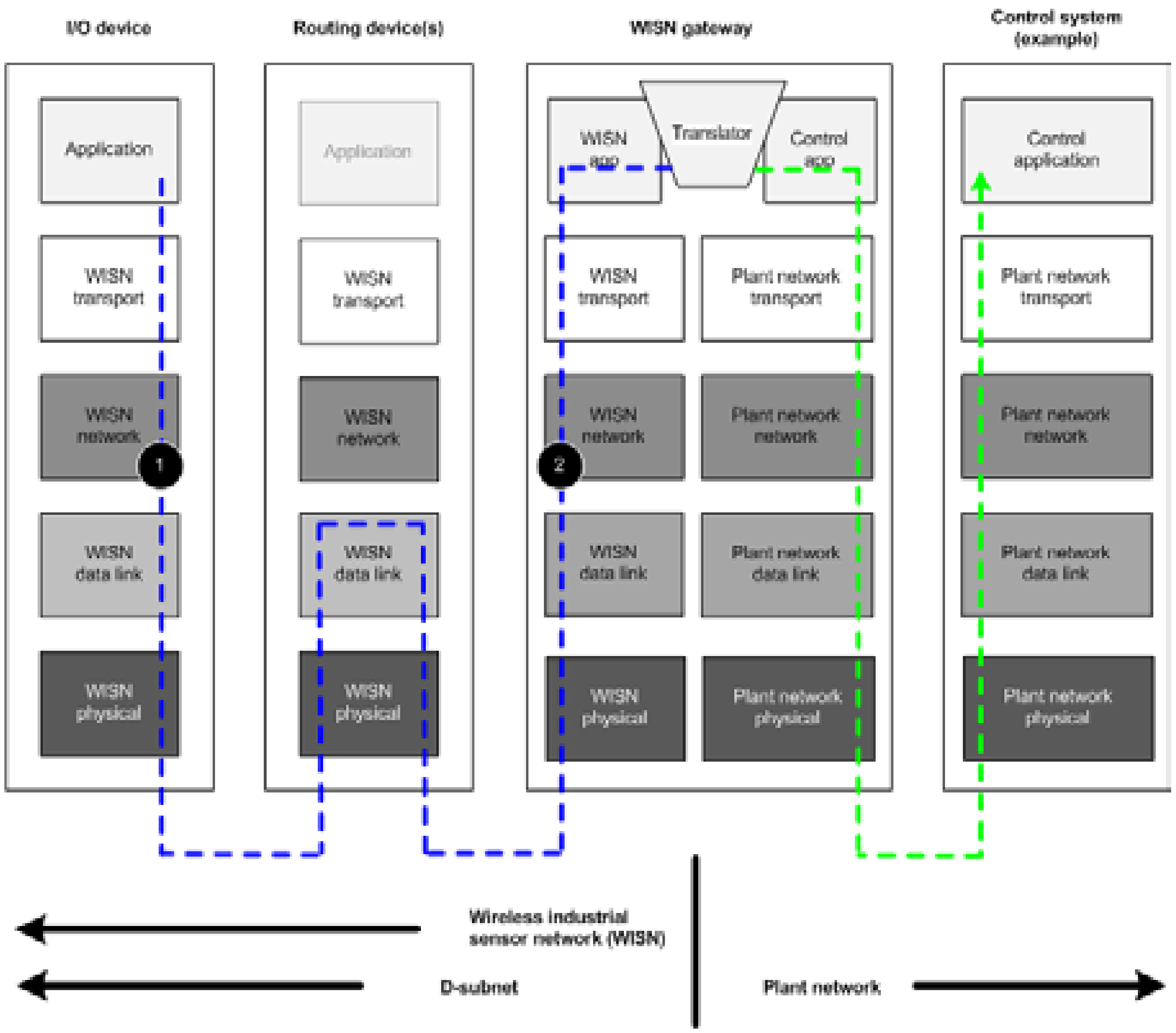


Figure 99 – Protocol suite diagram for routing from a field device direct to a field-connected gateway without backbone routing

In Figure 99, the gateway is shown to have a field medium; hence no backbone network is involved in this example. The operations of the NLEs at the devices that the NPDU traverses (numbered in order) are as follows:

a) The NLE in the originating field device uses a basic network header; the DL16Address of the gateway and the DL16Address of the device itself are obtained from the ATT and passed to the DLE as a DSDU for conveyance to the gateway.

b) The NLE in the gateway receives the NPDU, checks that the NPDU is intended for the gateway, translates the DL16Address of the originating device (provided by the DD-DATA.indication) into an IPv6Address, and then passes the NSDU to the TLE.

10.2.7.2 Routing from a field device to a gateway via a backbone router

Figure 100 illustrates the routing of a PDU from a field device to a gateway via a backbone router.

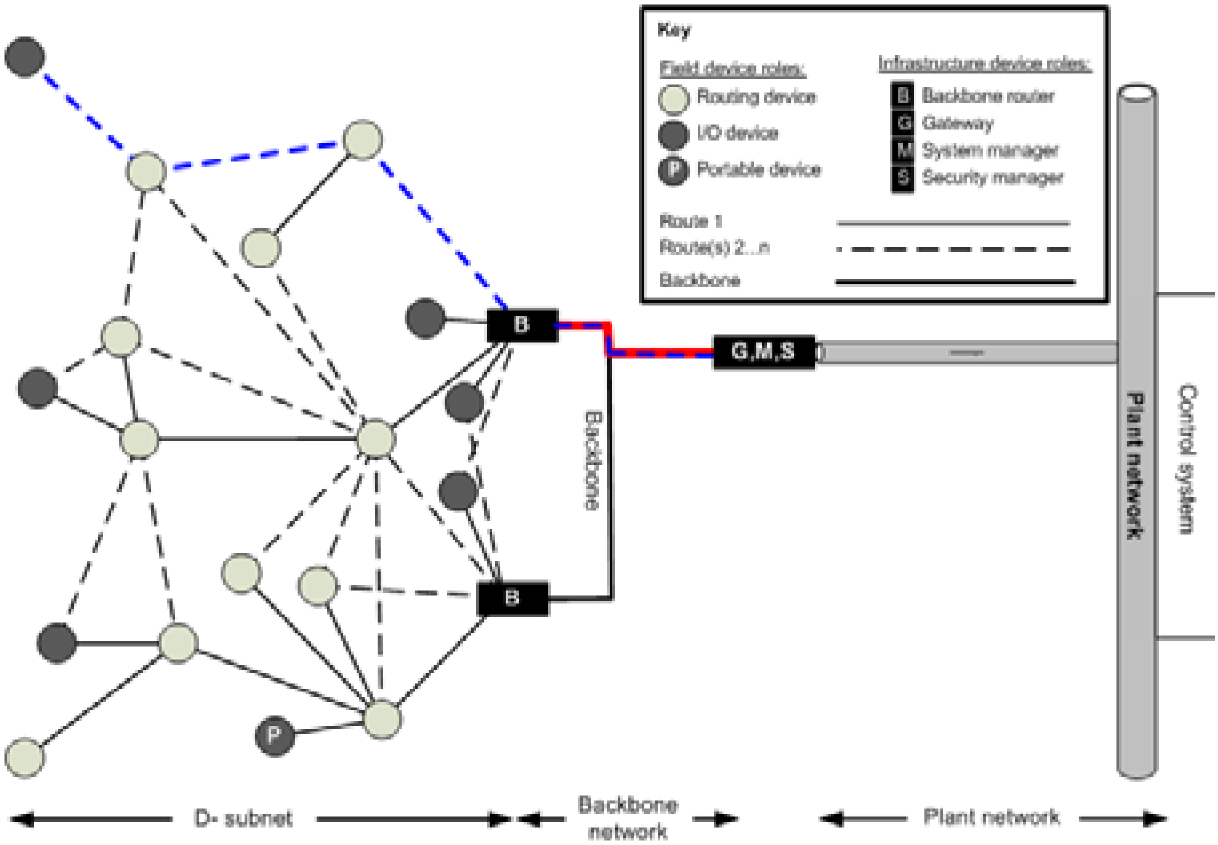


Figure 100 – Routing a NPDU from a field device to a gateway via a backbone router

Figure 101 depicts the flow of an NPDU from a field device to a gateway resident on the backbone network.

The NPDU is first routed to a backbone router over the D-subnet, and from there to the gateway over the backbone. The operation of the NLEs at the devices that the NPDU traverses (listed in order) is as follows:

a) The NLE of the I/O device passes to its local DLE its own DL16Address as the source address and the DL16Address of the gateway as the final destination address. If the ContractTable indicates that the ContractID needs to be included in the NPDU, the contract-enabled header is used; otherwise, the basic header is used if the compression used by the transport allows it (see 10.5.2.1). If the size of the NPDU is larger than maxDSDU size, the NPDU is fragmented. The complete NPDU (or the set of fragment NPDUs) is then passed as DSDU(s) to the associated DLE.

b) The DLE conveys the DSDU to the backbone router. If fragmented in a), the set of fragments is reassembled as the NPDU at the backbone router. The NLE at the backbone router receives the NPDU and determines that the NPDU is not intended for the backbone router, since the final destination address in the DD-Data.indication is the DL16Address of the gateway. The backbone router translates this DL16Address into the IPv6Address of the desired gateway, uses its routing table to determine the next-hop address to reach the gateway and creates a full header (format shown in Table 216).

c) The reconstituted NPDU with the expanded network header is presented to the backbone interface. The backbone interface routes the NPDU towards its final destination. In this example, the next hop is the final destination (the gateway).

d) The NPDU arrives at the NLE of the gateway over the backbone. The NLE at the gateway determines that the final destination address is equal to the address of the gateway itself and passes the NSDU to its TLE.

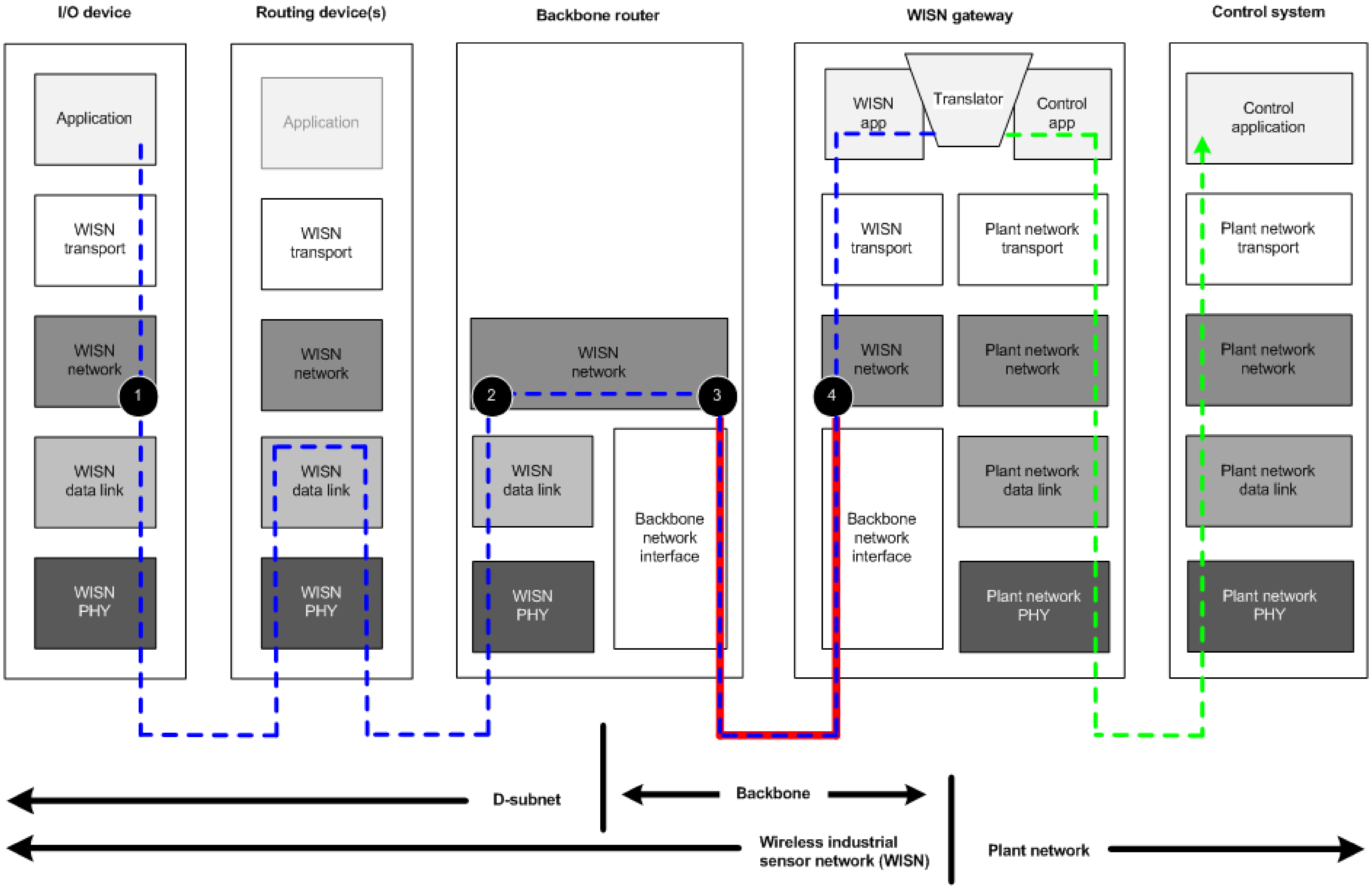


Figure 101 – Protocol suite diagram for routing an APDU from a field device to a gateway via a backbone router

10.2.7.3 Routing from a field device to another field device on a different D-subnet

Figure 102 illustrates routing from an I/O device on one D-subnet to another I/O device on a different D-subnet.

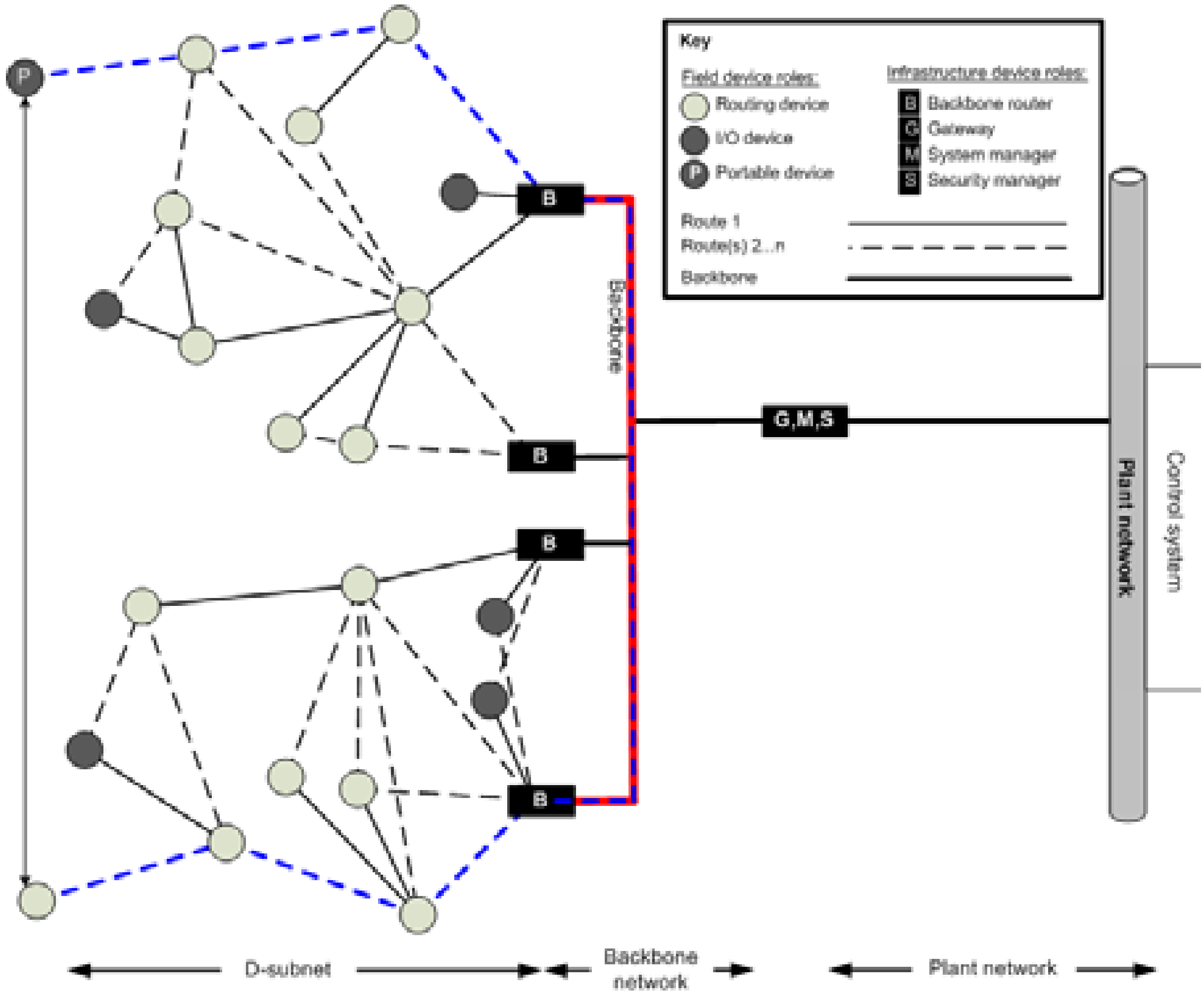


Figure 102 – Routing from a field device on one D-subnet to another field device on a different D-subnet

Figure 103 shows the flow of a NPDU between two field devices on different D-subnets (see 10.2.7.3). It is assumed that the NPDU needs no fragmentation.

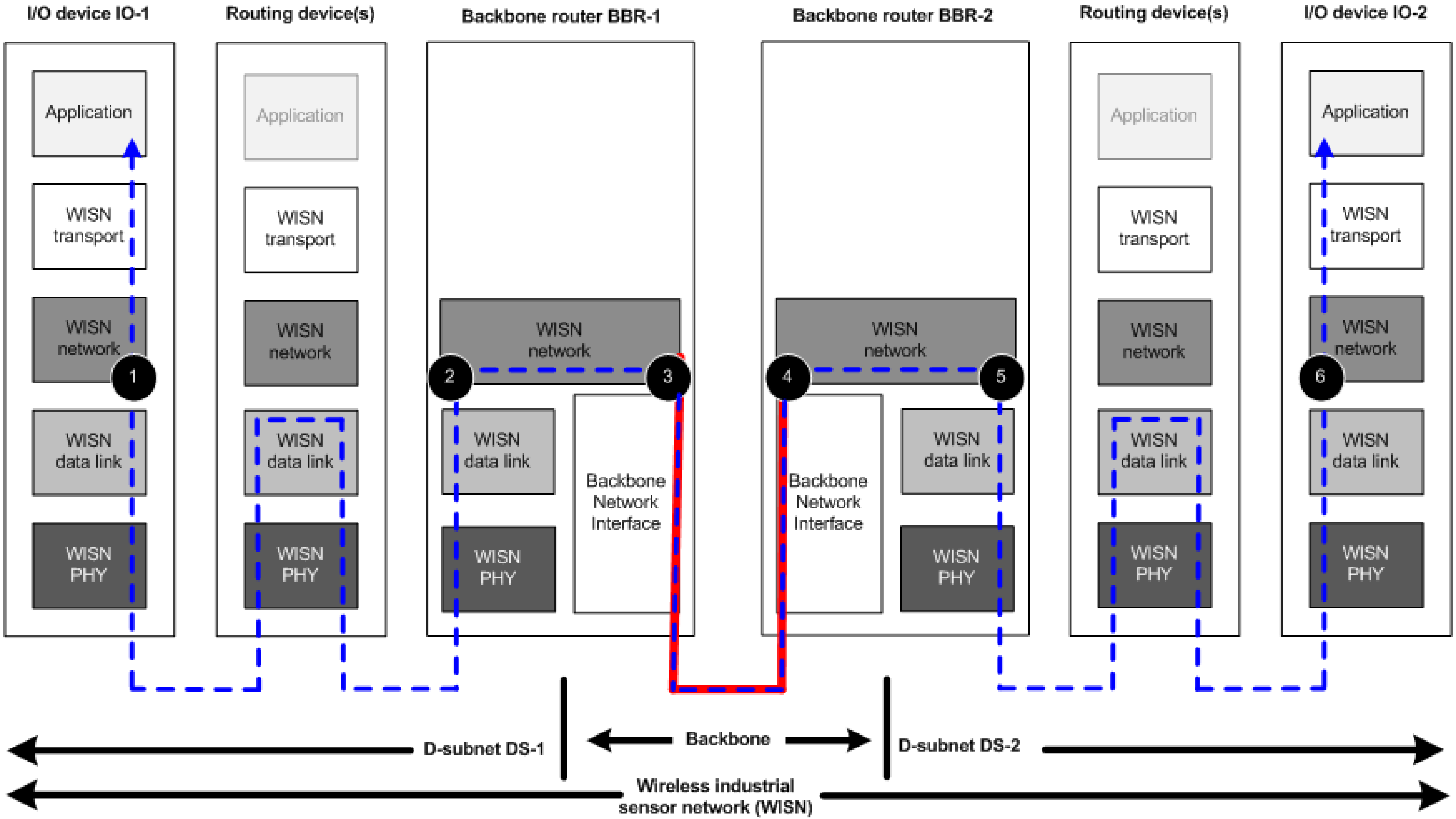


Figure 103 – Protocol suite diagram for routing from an I/O device on one D-subnet to another I/O device on a different D-subnet

The NPDU is routed over the backbone from one D-subnet to the other. The operations performed by the NLEs of the devices as the NPDU moves from the originating I/O device (I/O-1) located in D-subnet DL-1 to the destination I/O device (I/O-2) located in D-subnet DL-2 are as follows:

a) The NLE at I/O-1 creates the NPDU using the contract-enabled network header. The NLE passes the NPDU to the associated DLE as a DSDU, along with the DL16Address of I/O-1 within DL-1 as the source address, and the DL16Address of I/O-2 in DL-1 as the final destination address, via the notional DD-DATA.request. The ContractID is placed in the FlowLabel field of the contract-enabled header.

b) The resulting DPDU(s) is/are routed over DL-1 and arrive(s) at the DLE of BBR-1, i.e., the backbone router in DL-1. The DPDU payloads are used to regenerate the NPDU, which is checked to see if it is destined for BBR-1 itself. Since it is not destined for BBR-1, the DL16Addresses of I/O-1 and I/O-2 in the notional DD-DATA.indication are translated into their IPv6Addresses.

c) The expanded header (in the format defined in Table 216) is created and presented to the backbone interface. The next hop for this NPDU over the backbone is determined by looking up the IPv6Address of the final destination in the RT. The RT returns the IPv6Address of BBR-2 (the backbone router of DL-2) as the next hop, since BBR-2 is the backbone router serving I/O-2. The ContractID is placed in the FlowLabel field of the expanded header. The priority of the PDU is placed in the TrafficClass field.

d) The NLE at BBR-2 receives the NPDU over the backbone. The NPDU indicates that the final destination is the IPv6Address of I/O-2. This NPDU is then prepared for routing over DL-2 to reach I/O-2.

e) The NLE at BBR-2 creates a basic NPDU header. In the subsequent notional DD-DATA.request, the DL16Address of I/O-1 in DL-2 is the originator address and the DL16Address of I/O-2 in DL-2 is the final destination. The ContractID, extracted from the FlowLabel field of the expanded header, and the priority, extracted from the TrafficClass, are also passed down to the DL to enable selection of the appropriate routing resources (GraphID, etc).

f) The NPDU arrives at the NLE of I/O-2. Since the final destination indicated in the notional DD-DATA.indication is the DL16Address of I/O-2 in DL-2, the NLE translates the addresses into an IPv6Address and passes the NSDU to the TL of I/O-2.

10.2.7.4 Example of routing over an Ethernet backbone network

Figure 104 is an example of an implementation of the protocol suite diagram illustrated in Figure 103. In this network, a field device communicates with a control system that is aware of the native protocol of this standard; the backbone network in this example is an Ethernet network carrying IPv6 traffic.

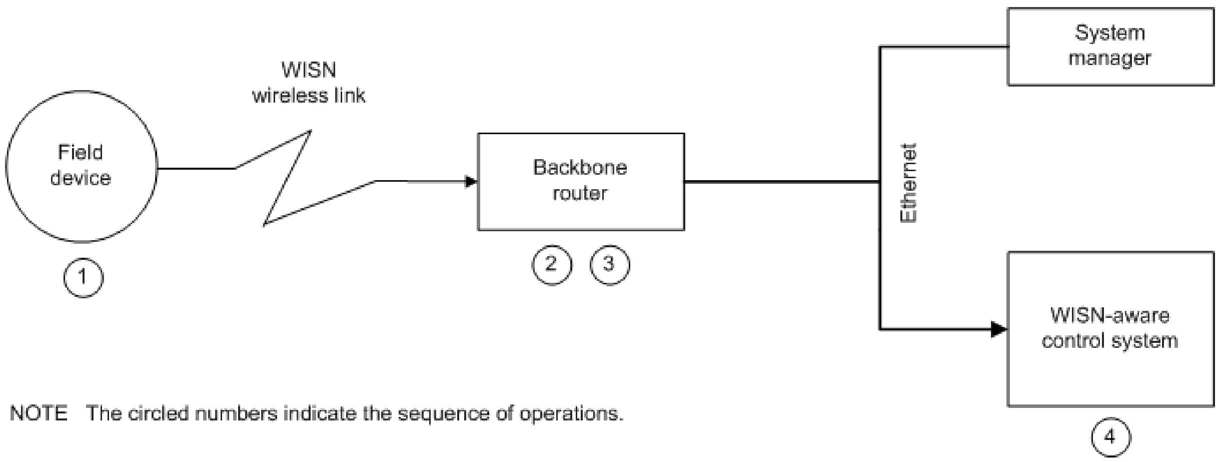


Figure 104 – Example of routing over an Ethernet backbone network

The numbered circles in Figure 104 indicate steps in the routing process and where they occur:

1) The NLE at the field device creates a NPDU and hands it to the associated DLE for transmission to the next NL hop (backbone router). The final destination address of the DPDU is the DL16Address for the native protocol-aware control system. The DL mesh delivers the NPDU to the NLE of the backbone router.
2) The backbone router receives the NPDU, replaces the DL16Addresses with the corresponding IPv6Addresses and expands the NPDU into a full IPv6 NPDU.
3) The backbone router sends the IPv6 NPDU over the Ethernet interface.
4) The NLE at the control system receives the IPv6 NPDU from its Ethernet interface, performs NL processing and passes the NSDU to the TLE.

10.2.7.5 Example of routing over a backbone network

Figure 105 is a variant of Figure 104 that substitutes a generic fieldbus for the Ethernet backbone network. In this variant the IPv6 NPDU is encapsulated for transport over the fieldbus network via one or more fieldbus PDUs.

NOTE Other variants of this fieldbus backbone scenario are possible.

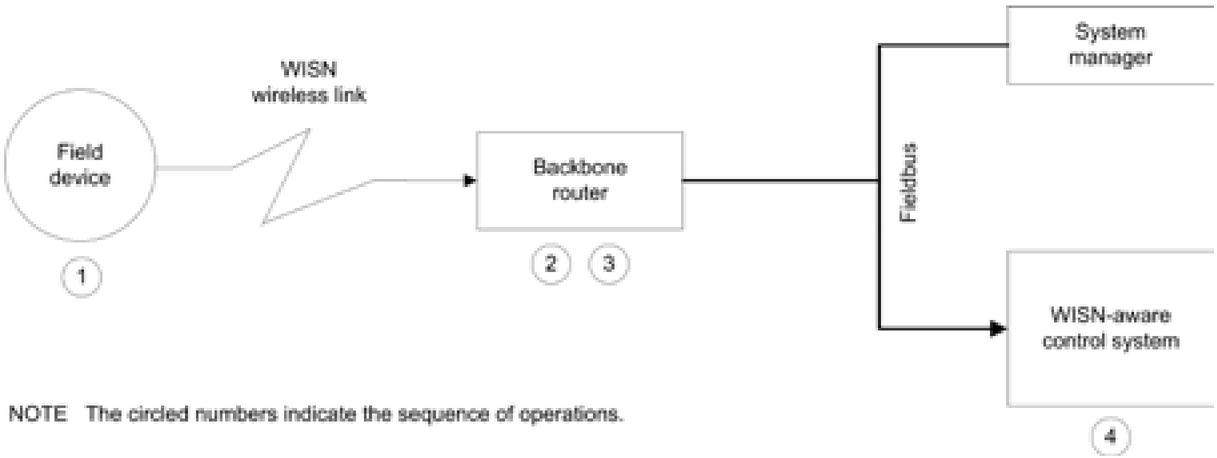


Figure 105 – Example of routing over a fieldbus backbone network

The numbered circles in Figure 105 indicate steps in the routing process and where they occur:

1) The NLE at the field device creates a NPDU and hands it to the associated DLE for transmission to the next NL hop (backbone router). The final destination address of the DPDU is the DL16Address for the native protocol-aware control system. The DL mesh delivers the NPDU to the NLE of the backbone router.
2) The backbone router receives the NPDU and translates the DL16Address into the IPv6Address and expands the NPDU into a full IPv6 NPDU.
3) The backbone router encapsulates the entire NPDU in one or more fieldbus PDUs addressed to the control system.
4) The control system (gateway) receives the fieldbus PDU(s), extracts the NPDU, performs NL processing, and delivers the NSDU to the associated TLE.

10.3 NLE data services

10.3.1 General

The TLE uses the NLE's NDSAP interface to send and receive data. This interface is internal to a compliant device and is therefore notional and not testable. The internal NSAPs of the device are depicted in Figure 16.

All interfaces between the NLE and its NME or the adjacent TLE and DLE are internal interfaces within the device, and thus are unobservable. Therefore they are not subject to standardization. Thus all of this description is notional.

10.3.2 N-DATA.request

10.3.2.1 General

N-DATA.request is used by a TLE to request the NLE to transmit a TSDU.

10.3.2.2 Semantics

The semantics of the N-DATA.request primitive is as follows:

```
N-DATA.request   (
                 DestAddress,
                 ContractID,
                 Priority,
                 DE,
                 NSDU,
                 NSDUSize,
                 NSDUHandle,
                 ECN
                 )
```

Table 203 specifies the elements for the N-DATA.request primitive.

Table 203 – N-DATA.request elements

Standard data type name: N-DATA.request		
Element name	**Element identifier**	**Element scalar type**
DestAddress (the IPv6Address of the destination NLE for the NSDU)	1	Type: Device address
ContractID (the contract ID associated with the resources to be used for transmitting this NPDU; this ID is passed through directly to the DLE)	2	Type: Unsigned16 Named value: 0 indicates no contract
Priority (priority of the message within the contract) [a]	3	Type: Unsigned2
DE (indicates whether the PDU is eligible for discard)	4	Type: Unsigned1
NSDU (the set of octets forming the NSDU to be transmitted by the NL, including the transport headers)	5	Type: OctetString
NSDUSize (the number of octets in the NSDU to be transmitted)	6	Type: Unsigned16
NDSUHandle (the handle associated with the NSDU to be transmitted)	7	Type: Unsigned16
ECN (explicit congestion notification)	8	Type: Unsigned2
a) The NLE shall combine this priority with the 2-bit contract priority to encode the D-priority as a 4 bit field before passing it to the DLE.		

10.3.2.3 Appropriate usage

The TLE invokes N-DATA.request to request that the NLE transmit an NSDU.

10.3.2.4 Effect on receipt

On receipt of an N-DATA.request, the NLE constructs the NL headers of the NPDU, first eliding the first octet of the LoWPAN_NHC of the NSDU if the basic header is used, then fragmenting the NPDU (if necessary), and conveying it to the DLE for transmission over the local D-subnet. If ContractID is zero, the NLE treats the NSDU as a join-related PDU with an associated destination EUI64Address, for which the required destination IPv6Address is derived from the link-local address passed as the DestAddress parameter.

10.3.3 N-DATA.confirm

10.3.3.1 General

N-DATA.confirm is used to report the result of an N-DATA.request.

10.3.3.2 Semantics

The semantics of the N-DATA.confirm primitive is as follows:

```
N-Data.confirm   (
                 NSDUHandle,
                 status
                 )
```

Table 204 specifies the elements for the N-DATA.confirm primitive.

Table 204 – N-DATA.confirm elements

Element name	Element identifier	Element scalar type
NSDUHandle (the handle of the NSDU whose status is being reported)	1	Type: Unsigned16
Status (the result of the N-DATA.request primitive that conveyed the NSDU)	2	Type: Unsigned Named value: 0: success

10.3.3.3 When generated

The NLE generates N-DATA.confirm as a delayed response to an N-DATA.request. N-DATA.confirm returns a status to the TLE that indicates either SUCCESS or FAILURE.

10.3.3.4 Appropriate usage

N-DATA.confirm notifies the TLE of the result of its request to transmit an NSDU.

10.3.4 N-DATA.indication

10.3.4.1 General

N-DATA.indication is used by an NLE to deliver a received TSDU to an associated TLE.

10.3.4.2 Semantics

The semantics of the N-DATA.indication primitive is as follows:

```
N-Data.indication (
                 SrcAddress,
                 DestAddress,
                 NSDU,
                 NSDUSize,
                 ECN,
                 Priority
                 )
```

Table 205 specifies the elements for the N-DATA.indication primitive.

Table 205 – N-DATA.indication elements

Element name	Element identifier	Element scalar type
SrcAddress (the IPv6Address of the source of the NSDU)	1	Type: Device address
DestAddress (the IPv6Address of the destination of the NSDU, e.g, the device's own IPv6Address.)	2	Type: Device address
NSDU (the received NSDU, including associated TL headers)	3	Type: Sequene of octets
NSDUSize (the number of octets in the received NSDU)	4	Type : Unsigned16
ECN (explicit congestion notification information from the received NSDU)	5	Type: BitArray4
Priority (4-bit NPDU priority as received)	6	Type: Unsigned4

10.3.4.3 Appropriate usage

The NLE invokes N-DATA.indication to notify the associated TLE of a received NSDU and associated conveyance information. If the received NPDU contained a basic header, then, before passing the NSDU to the TLE, the NLE restores (prepends) the first octet of the NSDU as LoWPAN_NHC (= 0x 1111 0111), which had been elided when the basic header was constructed. If the source D-address received from the underlying DLE is an EUI64Address, a link-local IPv6Address is constructed from the EUI64Address and passed as the SrcAddress parameter of the N-DATA.indication. The DestAddress parameter of the N-DATA.indication is the link-local IPv6Address of the device when used for join-related APDUs, or the globally-assigned IPv6Address for post-join operation.

10.3.4.4 Effect on receipt

On receipt of an N-DATA.indication, the TLE is able to process the reported NSDU.

10.4 NL management object

10.4.1 NL management information base

Table 206 specifies the attributes of the NL management object (NLMO).

Table 206 – NLMO attributes

Standard object type name: NL management object (NLMO())				
Standard object type identifier: 123				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute type**	**Description of behavior of attribute**
Backbone_Capable	1	A Boolean flag indicating whether the device is backbone capable	Type: Boolean1 Classification: Static Accessibility: Read only	Fixed value based on device capabilities and implementation details. Backbone capability may be ignored by a system manager
DL_Capable	2	A Boolean flag indicating whether the device is capable of communicate over the wireless Type A medium of this standard	Type: Boolean1 Classification: Static Accessibility: Read only	Fixed value based on device capabilities and implementation details. DL interface capability may be ignored by a system manager
DL16Address	3	The DL16Address of the device	Type: DL16Address Classification: Static Accessibility: Read/write Valid range: 1.. 2^{15}-1	A fixed value as assigned by the system manager at join.[a] This attribute is a duplicate of the corresponding attributes in the DMO and DLMO
Long_Address	4	The IPv6Address of the device	Type: IPv6Address Classification: Static Accessibility: Read/write Valid range: all with high-order bit reset	A fixed value as assigned by the system manager at join.[a] This attribute is a duplicate of the corresponding attributes in the DMO and DLMO
Route_Table	5	The routing table that includes information to route a NPDU.	Type: Array of NLRouteTbl structures (see NLRouteTbl) Classification: Static Accessibility: Read/write Valid range: See Table 208	The routing table consists of a destination address, next network hop for that destination and the number of network hops needed. The routing table structure, its corresponding alerts and methods are discussed in 10.2.6.2. The size of the routing table present in devices that only operate within the D-subnet and are not backbone capable is presented in Table B.18
Enable_Default_Route	6	A Boolean value set to indicate if a default routing is enabled	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: Disabled	Enables a default route

Table 206 *(continued)*

Standard object type name: NL management object (NLMO()				
Standard object type identifier: 123				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute type**	**Description of behavior of attribute**
Default_Route_Entry	7	Destination address associated with the default route	Type : IPv6Address Classification: Static Accessibility: Read/write	Can be used to look up the default route in the route table. Packets that include a destination address with no corresponding entry in the routing table shall be forwarded using the default route. Field devices may use the default route to send packets to devices operating on the backbone that have not been assigned 16-bit addresses, but only IPv6Addresses
Contract_Table	8	Includes information to construct the network header for a particular PDU flow	Type: Array of NLContractTbl structures (see NLContractTbl) Classification: Static Accessibility: Read/write Valid range: See Table 207	The contract table consists of the destination address, priority for a particular flow. The table also includes information indicating if a ContractID needs to be carried in the NPDU or not. The scope of a contract ID is local to a device, however, the combination of a source address (128-bit) and ContractID is globally unique. The contract table structure and its corresponding alerts and methods are discussed below
Address_Translation_Table	9	Includes the IPv6Address and the DL16Address alias for the IPv6Addresses	Type: Array of NLATTbl structures (see NLATTbl) Classification: Static Accessibility: Read/write Valid range: See Table 209	The address translation table in a device is indexed via the IPv6Address and stores the corresponding DL16Address alias within the D-subnet to which the device belongs
Max_NSDU_size	10	Maximum service data unit size supported by the NL of the device	Type: Unsigned16 Classification: Constant Accessibility: Read only Default value: 70 Valid range: 70..1 280	Fixed value based on device memory capabilities and implementation details. The value 1 280 comes from IETF RFC 4944. See 6.2.8 on how this value can necessitate fragmentation at the AL. NSDUs that exceed the maximum value may be rejected by the device

Table 206 *(continued)*

Standard object type name: NL management object (NLMO()				
Standard object type identifier: 123				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute type**	**Description of behavior of attribute**
Frag_Reassembly_Timeout	11	Amount of time (in seconds) a reassembly buffer needs to be held open for fragments to come in before discarding the NPDU	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 60 Valid range: 1..600	The default value is 60 s, but the system manager can change this value
Frag_Datagram_Tag	12	Current tag number for fragmentation at the device	Type: Unsigned16 Classification: Dynamic Accessibility: Read only Default value: Uniform random	A new tag number is used for every NPDU that needs to be fragmented. The Tag number is incremented by one for each NPDU to be fragmented. Value wraps back to zero after 65 535
NLRouteTblMeta	13	Metadata for Route Table attribute (Attribute 5)	Type: Metadata_attribute Classification: Static Accessibility: Read only	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table. The size of the routing table present in devices that only operate within the D-subnet and are not backbone capable is presented in Table B.18
NLContractTblMeta	14	Metadata for Contract Table attribute (Attribute 8)	Type: Metadata_attribute Classification: Static Accessibility: Read only	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table
NLATTblMeta	15	Metadata for Address Translation Table attribute (attribute 9)	Type: Metadata_attribute Classification: Static Accessibility: Read only	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table
DroppedNPDUAlertDescriptor	16	Describes how a dropped NPDU alert is reported on the network	Type : Alert report descriptor Classification: Static Accessibility: Read/write Default value: [TRUE, 7]	—
a) The attribute shall remain unchanged during normal operation; only a system manager may change it. Assignment of a new DL16Address shall trigger a Join_Command = 3, restartAsProvisioned, following which the device shall re-join the wireless network.				

10.4.2 Structured management information bases

The NLMO defines three structured management information bases (SMIBs) as tables. They are the contract table, the route table (RT), and the address translation table (ATT).

Table 207 specifies the elements for the contract table. Devices that are not backbone capable may elide the Source_Address field.

Table 207 – Contract table structure

Standard data type name: NLContractTbl		
Standard data type code: 441		
Element name	**Element identifier**	**Element scalar type**
Contract_ID*	1	Type: Unsigned16 Classification: Static Accessibility: Read/write
Source_Address*	2	Type: IPv6Address Classification: Static Accessibility: Read/write Default value : 0
Destination_Address	3	Type: IPv6Address Classification: Static Accessibility: Read/write Default value : 0
Contract_Priority	4	Type: Unsigned2 Classification: Static Accessibility: Read/write Default value : 00
Include_Contract_Flag	5	Type: Boolean1 Classification: Static Accessibility: Read/write Default value : FALSE
NOTE * indicates an index field.		

Table 208 specifies the elements for the route table.

Table 208 – Route table elements

Standard data type name: NLRouteTbl		
Standard data type code: 442		
Element name	**Element identifier**	**Element scalar type**
Destination_Address*	1	Type: IPv6Address Classification: Static Accessibility: Read/write
Next_Hop	2	Type: IPv6Address Classification: Static Accessibility: Read/write

Standard data type name: NLRouteTbl		
Standard data type code: 442		
Element name	**Element identifier**	**Element scalar type**
NWK_HopLimit	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 64
Outgoing_Interface	4	Type: Unsigned1 Classification: Static Accessibility: Read/write Default value: 0 Named values: 0: DL; 1: Backbone
NOTE * indicates an index field.		

Table 209 specifies the elements for the address translation table.

Table 209 – Address translation table structure

Standard data type name: NLATTbl		
Standard data type code: 443		
Element name	**Element identifier**	**Element scalar type**
Long_Address*	1	Type: IPv6Address Classification: Static Accessibility: Read/write
Short_Address	2	Type: DL16Address Classification: Static Accessibility: Read/write
NOTE * indicates an index field.		

10.4.3 NL management object methods

Standard methods such as read and write can be used for scalar or structured MIBs in their entirety. These methods are used to manipulate tables. They allow access to a particular row of a structured MIB based on a unique index field. All devices shall be capable of manipulating the NLMO attributes immediately. Devices support delayed manipulation of these attributes using cutover methods but are not required to do so.

It is assumed that the tables have a unique index field, which may either be a single element or the concatenation of multiple elements. The index field is assumed to be the (concatenation of the) first (few) element(s) of the table. For example, the contract table index field is the concatenation of the Contract_ID and Source_Address.

These methods do not specify the interface between the NME and NLMO, as this interface is internal to a particular device. The management entity may keep local copies of the MIBs.

Table 210 describes the methods for manipulation of structured MIBs. These methods are based on the Read_Row, Write_Row, and Delete_Row templates defined in Annex J.

Table 210 – NLMO structured MIB manipulation methods

Standard object type name: NLMO		
Standard object type identifier: 123		
Method name	**Method ID**	**Method description**
Set_row_RT	1	Method to set (either add or edit) the value of a single row of the route table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Get_row_RT	2	Method to get the value of a single row of the route table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Delete_row_RT	3	Method to delete a single row of the contract table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Set_row_ContractTable	4	Method to set (either write or edit) the value of a single row of the contract table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Get_row_ContractTable	5	Method to get the value of a single row of the contract table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Delete_row_ContractTable	6	Method to delete the value of a single row of the contract table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Set_row_ATT	7	Method to set (either add or edit) the value of a single row of the Address Translation table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)
Get_row_ATT	8	Method to get the value of a single row of the Address Translation table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)
Delete_row_ATT	9	Method to delete a single row of the Address Translation table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)

Table 211 describes the alert to indicate a dropped PDU/PDU error.

Table 211 – Alert to indicate dropped PDU/PDU error

Standard object type name(s): NLMO					
Standard object type identifier: 123					
Description of the alert: Alert to indicate dropped PDU /PDU error					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type** **(Enumerated: based on alert category)**	**Alert priority** **(Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	1 = Comm. diagnostic	0 = NL_Dropped_PDU	7 =Medium	Type: OctetString	The value is an octet string consisting of at least two octets. The first octet is an Unsigned8 that specifies the size of the value field included with the alert in octets. The second octet is an Unsigned8 that conveys the diagnostic class. Named values: 0: reserved; 1: Destination unreachable; 2: Fragmentation error; 3: Reassembly timeout; 4: Hop limit reached; 5: Header errors; 6: No route, next hop unreachable; 7: Out of memory; 8: NPDU size too large; 9..255: reserved. The remaining octets include the NPDU header for the dropped PDU.

10.5 NPDU formats

10.5.1 General

Each NPDU shall consist of two basic components:

- A network header possibly comprising addressing, class of service (CoS), and fragmentation fields.
- A network payload of variable size containing the data that needs to be transmitted.

This standard shall allow three different NPDU header formats:

– the basic header;
– the contract-enabled header; and
– the full header.

Devices compliant with this standard shall use dispatch prefixes (the least significant 3 bits of the first octet of the NPDU) to distinguish between these header formats (see Figure 106). The prefix 000 shall indicate that the NPDU header format is the basic header. The basic header prefix conforms to the NALP dispatch of 6LoWPAN.

The prefix 011 shall indicate the contract-enabled header. The contract-enabled header conforms to the LOWPAN_IPHC dispatch of 6LoWPAN.

The prefix 010 shall indicate the full header format. Resource constrained devices that operate in the wireless D-subnet and are backbone capable may construct the full header but are not required to do so. If a packet that is constructed using the full header is received by a device that is not backbone capable, the device may discard the NPDU and send a dropped PDU/PDU error alert with value 5 indicating a header error.

Finally, a prefix of 110 or 111 shall indicate that the PDU is a fragment of a larger NPDU that needs to be reassembled.

This standard primarily uses 16-bit addresses for very first (originator) and final destination in the DPDUs transmitted over the IEEE 802.15.4:2011 wireless network. Therefore, the network header is recommended to be either the basic or contract-enabled header format for the NPDUs transmitted over the IEEE 802.15.4:2011 wireless links. It is not recommended, although not prohibited, to use the full header format for transmission of NPDUs over the field medium.

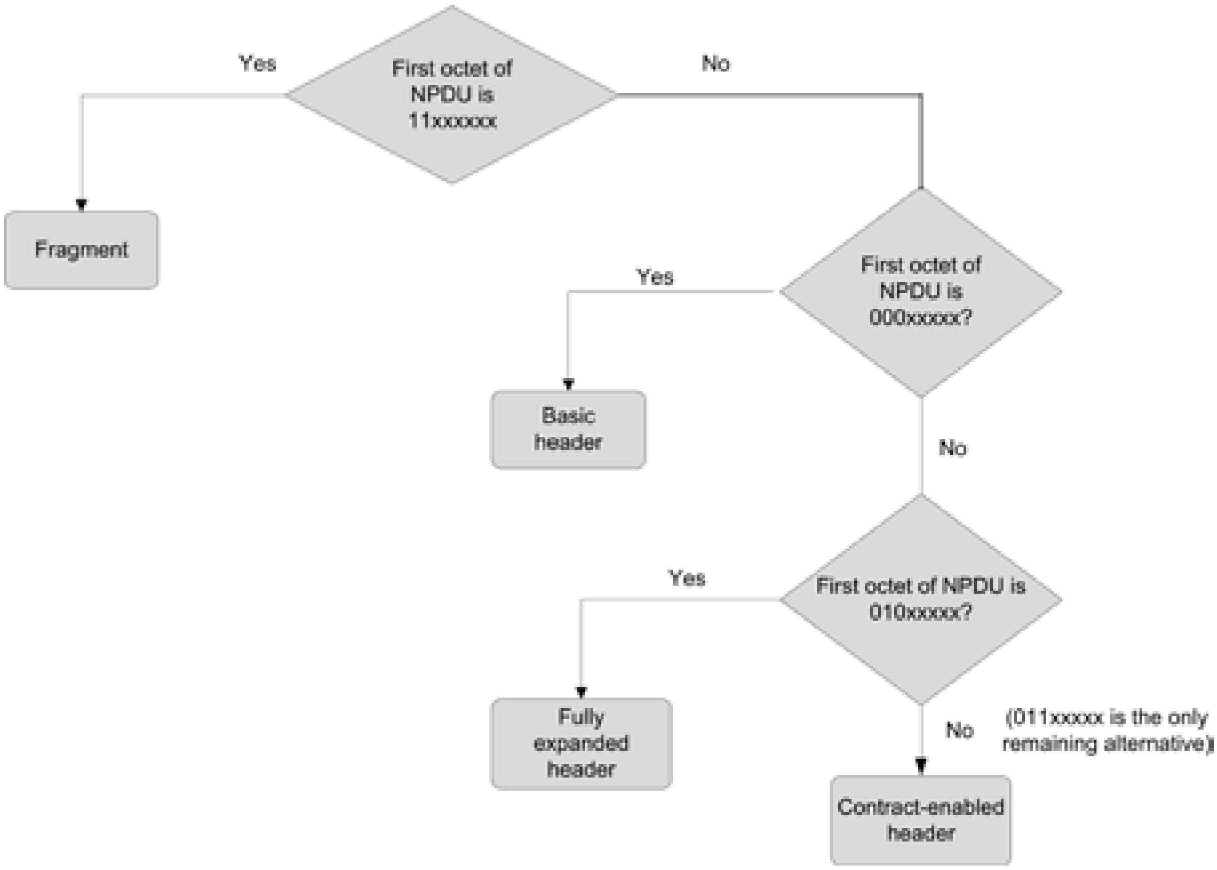


Figure 106 – Distinguishing between NPDU header formats

The dispatch octet bit patterns are shown in Table 212.

Table 212 – Common header patterns

Dispatch pattern	Header type
000xxxxx	NALP (Not A 6LoWPAN NPDU) – Basic header
010xxxxx	IPv6 (uncompressed IPv6 header) – Full header
011xxxxx	LoWPAN_IPHC (compressed IPv6 header) – Contract-enabled header
11xxxxxx	LoWPAN fragment

The NL headers shall follow the formats defined herein.

10.5.2 Basic header format for NL

10.5.2.1 Intended usage

The DL of this standard employs a link level mesh. In the most common case, a PDU will traverse a single D-subnet, so the basic header is optimized to minimize the NPDU overhead. The route that needs to be taken by the PDU is known to the device of ingress into the D-subnet; this device of ingress makes all the necessary DL routing decisions. The ContractID is not transmitted in the basic network header.

The basic header for the NL shall be used only if the user datagram protocol (UDP) header is fully compressed (i.e., the source and destination port numbers are compressed to four (4) bits each and the UDP checksum is elided). The NL can determine whether the UDP header is fully compressed by looking at the LOWPAN_NHC octet, which is always the first octet of the NSDU passed to the NL by the TL. Since the basic header is used only in case of fully compressed UDP header (i.e., fixed and known value of UDP LOWPAN_NHC) the UDP LOWPAN_NHC octet shall be elided by the NL of origin and restored by the destination NL.

10.5.2.2 Format

Table 213 describes the basic network header format.

Table 213 – Basic NL header format

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Dispatch							
(variable)	Network payload							

The basic NL header shall consist of a single Dispatch field as follows:

Dispatch: The dispatch field indicates the NL header format. For the basic header, the dispatch field shall have the value 0x 0000 0001.

10.5.2.3 Relation to 6LoWPAN

The 6LoWPAN format allows all fields of the full IPv6 header to be elided. The dispatch and encoding octets to achieve this are 0x0111 1110 0111 0111, which indicate (when parsed as 011.11.1.10.0.1.11.0.1.11):

- Dispatch = 011
- TF = 11 (both traffic class and flow label elided)
- NH = 1 (next header field elided)
- HLIM = 10 (HopLimit = 64)

- CID = 0 (no context identifier extension)
- SAC = 1 (stateful source address compression; the ATT provides the context)
- SAM = 11 (source address fully elided)
- M = 0 (no multicast)
- DAC = 1 (stateful destination address compression; the ATT provides the context)
- DAM = 11 (destination address fully elided)

The 6LoWPAN format also allows the UDP header to be compressed so that the source and destination port numbers are four (4) bits each and the checksum is elided. In this case, the UDP LOWPAN_NHC field has the value 0x 1111 0111 , which indicates (when parsed as 11110.1.11):

– protocol = 11 110 (UDP)
– checksum compression = 1 (checksum elided)
– port compression = 11 (source and destination ports are compressed to four (4) bits each and their implied prefix is 0xF0B)

The basic header is essentially a single-octet abbreviation for the three octets (two octets of fully compressed IP header and one octet of fully compressed UDP header) noted above. Since it is an abbreviation, it is fully compatible with the 6LoWPAN format. A device receiving a basic header NPDU can expand the basic header dispatch octet to the three octets noted above and obtain a 6LoWPAN-compliant PDU.

10.5.3 Contract-enabled network header format

10.5.3.1 Intended usage

Like the basic header, the contract-enabled network header is intended for use within D-subnets. The very first (originator) device of an NPDU may use the contract-enabled header instead of the basic header if it is desirable for devices other than the very first (originator) device to be aware of the NPDU stream to which the NPDU belongs. For example, an intermediate router on the backbone may need to select:

- appropriate backbone resources upon egress from the originating D-subnet; or
- appropriate DL resources (graph ID, priority, etc.) upon ingress into a destination D-subnet.

The contract includes a flag to indicate to the originating NL whether the network header requires inclusion of the ContractID.

The contract-enabled header shall also be used if the UDP LOWPAN_NHC does not indicate full compression of the UDP header. Join process messages between the new device and the advertising router will always fall under this category since they do not elide the UDP checksum.

10.5.3.2 Format

Table 214 describes the contract-enabled header format.

Table 214 – Contract-enabled NL header format

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2	LOWPAN_IPHC dispatch			LOWPAN_IPHC encoding (bits 8..12)				
	LOWPAN_IPHC encoding (bits 0..7)							
0 or 3	reserved				FlowLabel (bits16..19)			
	Flow Label (bits 8..15)							
	Flow Label (bits 0..7)							
0..1	HopLimit							
(variable)	Network payload							

Fields include:

- LOWPAN_IPHC dispatch: This field shall indicate that the header format is contract-enabled and that LOWPAN_IPHC header compression encoding bits follow. The LOWPAN_IPHC dispatch field shall be 011.
- LOWPAN_IPHC encoding: This field is 13 bits long; its value shall be encoded as 0x0 11HH 0111 0111 when octets 3..5 are present to carry the contract ID, or as 0x1 11HH 0111 0111 when octets 3..5 are elided. In either case, HH shall have the value 00 if HopLimit is carried inline, 01 if the hop limit is 1, 10 if the hop limit is 64 and 11 if the hop limit is 255.
- FlowLabel: The lower order 16 bits of the FlowLabel shall be set to ContractID. The higher order 4 bits shall be all zeros. This field shall only be present if octets 3 through 5 are present, as indicated by LOWPAN_IPHC encoding.
- HopLimit: This field shall indicate the number of layer-3 hops permitted before the NPDU is discarded. The HopLimit field shall be set to a value indicated by the device's routing table (RT; see 10.2.6.2). The default value for HopLimit field shall be 64. Devices that only operate within the D-subnet and are not backbone capable shall set the HopLimit to 1. From the perspective of the NL, any device reachable through the DL mesh is a single network hop away.

For join process messages, LOWPAN_IPHC encoding shall have the value 0x1 1101 0111 0111 to indicate that octets 3 through 5 are elided (no contract ID) and that the hop limit is 1 (HopLimit field elided).

10.5.3.3 Relation to 6LoWPAN

Table 215 shows the 6LoWPAN_IPHC encoding format.

Table 215 – 6LoWPAN_IPHC encoding format

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	0	1	1	TF		NH	HLIM	
1	CID	SAC	SAM		M	DAC	DAM	
…	Non-compressed fields							

The encoding for the contract-enabled NL header is derived by using the following values for the 6LoWPAN_IPHC encoding fields:

- TF = 01 or 11. In a 6LoWPAN format, TF = 01 implies a 3-octet inline field is carried in the non-compressed fields. This inline field consists of 2 bits of ECN, followed by a 2-bit pad, followed by 20 bits of flow label. Since in this standard the ECN is always enabled and the congestion indication is carried by the lower layer, both the ECN and the 2-bit pad shall be all zeros. In a 6LoWPAN format, TF = 11 implies this 3-octet field is elided.

- NextHeaderNH = 1 to indicate that the next header can be inferred from the prefix of the transport header. In this standard, the next header is always UDP.
- HLIM = HH. These bits are used by this standard to indicate the hop limit compression scheme as intended in 6LoWPAN.
- CID = 0 to indicate no additional 8-bit context identifier extension is used.
- SAC = 1 to indicate stateful compression for the source address.
- SAM = 11 to indicate that all 128 bits of the source address are elided (since, in this standard, the 16-bit D-alias is carried by the lower layer and is indexed in the ATT, which provides the translation context).
- M = 0 to indicate the destination address is not multicast.
- DAC = 1 to indicate stateful compression for the destination address.
- DAM = 11 to indicate that all 128 bits of the source address are elided (since, in this standard, the 16-bit D-alias is carried by the lower layer and is indexed in the ATT, which provides the translation context).

10.5.4 Full header (IPv6) format

10.5.4.1 Intended usage

The full header format is used primarily over the backbone network. A field device may also use the full header format instead of the basic or contract-enabled format but it is not recommended.

When the NPDU reaches an intermediate backbone router over the DL, the backbone router shall fully expand the header to include all fields as defined in the IPv6 header. It is necessary to expand the NL addresses of the very first (originator) device and the final destination device to 128 bits to disambiguate their DL16Addresses when routing outside the D-subnet.

10.5.4.2 Format

Table 216 describes the IPv6 header format.

Table 216 – IPv6 NL header format

Number of octets	bits							
	7	6	5	4	3	2	1	0
1	Version				TrafficClass (bits 7..4)			
3	TrafficClass (bits 3..0)				FlowLabel (bits 19..16)			
	FlowLabel (bits 15..8)							
	FlowLabel (bits 7..0)							
2	PayloadSize (bits 15..8)							
	PayloadSize (bits 7..0)							
1	NextHeader							
1	HopLimit							
16	Source address							
16	Destination address							
(variable)	Network payload							

Fields include:

- Version: 4-bit IP version number shall be set to 6.
- TrafficClass: The higher order four bits of this 8-bit field shall be used to carry the 4-bit priority of the NPDU over the backbone. The fifth bit shall be 0 and the sixth bit shall be used to carry the Discard Eligible (DE) bit of the NPDU over the backbone. The two lowest-order bits shall carry the ECN.
- FlowLabel: The value of this field shall be as defined in the contract-enabled header format (see 10.5.3.2). This field shall be set to all zeros if the Contract ID is not carried as part of the flow.
- PayloadSize: This 16-bit unsigned integer shall contain the size of the IPv6 payload, i.e., the rest of the NPDU following this header, in octets.
- NextHeader: This 8-bit field shall bet set to 0x 0001 0001 (17 decimal) identifying the following header as UDP.
- HopLimit: The value of this 8-bit field shall be set to the number of NL (layer 3) hops needed to get to the destination. If the NPDU is received over a D-subnet with the basic header, this field shall be updated upon egress from the D-subnet by a backbone router according to the routing table of the router. This field is decremented by 1 by the NL of each device if the NL forwards the NPDU. The NPDU shall be discarded if HopLimit is decremented to zero.
- Source address: This field shall contain the IPv6Address of the originator of the NPDU.
- Destination address: This field shall contain the IPv6Address of the final destination (intended recipient) of the NPDU.

10.5.4.3 Relation to 6LoWPAN

It is not recommended to use the full NL header format in an NPDU being transmitted over the D-subnet of this standard. However, the standard does not preclude the use of the full header over D-subnets. If used over the D-subnet, the NPDU shall contain the IPv6 dispatch as shown in Table 217.

Table 217 – Full NL header in the DL

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	IPv6 dispatch							
4	Version				TrafficClass (bits 7..4)			
	TrafficClass (bits 3..0)				FlowLabel (bits 19..16)			
	FlowLabel (bits 15..8)							
	FlowLabel (bits 7..0)							
2	PayloadSize (bits 15..8)							
	PayloadSize (bits 7..0)							
1	NextHeader							
1	HopLimit							
16	Source address							
16	Destination address							
(variable)	Network payload							

10.5.5 Fragmentation header format

10.5.5.1 Intended usage

If an NPDU with size greater than the dlmo.maxDSDUSize needs to be transmitted over the DL, the NPDU shall be fragmented. When an NPDU needs to be fragmented, the fragmentation header shall be inserted.

10.5.5.2 Format

The fragmentation header contains a 5-bit fragmentation type, followed by a 27-bit header for the first fragment and a 35-bit header for subsequent fragments. Fields of the basic, contract-enabled, or full headers from the dispatch octet onwards shall be placed in the first fragment only. Table 218 shows the NL header format for fragmented NPDUs.

Table 218 – NL header format for fragmented NPDUs

Number of octets	Bits							
	7	6	5	4	3	2	1	0
4..5	Fragmentation type					Fragmentation header		
	Other fields of fragmentation header (see Table 219 and Table 220)							
(variable)	Basic / contract-enabled / full header (for first fragment only)							
(variable)	Network payload							

Fields include:

- Fragmentation type: This 5-bit field shall be set to 0x1 1000 for the first fragment and to 0x1 1100 for the second and subsequent fragments.
- Fragmentation header:
  – First fragment: The first fragment shall contain the first fragment header as defined in Table 219 and the following text.

Table 219 – Format of first fragment header

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Fragmentation type					Datagram_size (bits 10..8)		
	1	1	0	0	0			
	Datagram_size (bits 7..0)							
2	Datagram_tag							

  – Subsequent fragments: The second and subsequent fragments (up to and including the last) shall contain a fragmentation header as defined in Table 220 and the following text.

Table 220 – Format of second and subsequent fragment headers

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2	Fragmentation type					Datagram_size (bits 10..8)		
	1	1	1	0	0			
	Datagram_size (bits 7..0)							
2	Datagram_tag							
1	Datagram_offset							

All fragment headers contain:

– Datagram_size: This 11-bit field encodes the size of the entire NSDU before fragmentation plus the size of the basic, contract-enabled or full header. The value of Datagram_size shall be the same for all fragments of the NSDU.

– Datagram_tag: This provides the identification for the reassembling NLE. The originating NLE shall increment Datagram_tag for successive, fragmented NPDUs. The incremented value of Datagram_tag is left-truncated to 16 bits (i.e., modulo 2^{16}) for inclusion in the NPDU. To minimize its predictability to an attacker, the initial value for Datagram_tag shall be selected from a uniform-random distribution. The value of Datagram_tag shall be the same for all fragments of an NSDU.

Second and subsequent fragment headers also contain:

– Datagram_offset: This field shall be present only in the second and subsequent fragments and shall specify the offset, in units of 8 octets, of the fragment from the beginning of the NPDU payload. (The first fragment of the NSDU has an offset of zero; the implicit value of Datagram_offset in the first fragment is zero.) This field is 8 bits long.

10.5.5.3 Relation to 6LoWPAN

The fragmentation header formats in this standard are based entirely on the IETF RFC 4944 format, with no changes. As in IETF RFC 4944, the Datagram_size field is carried in all fragments, and the Datagram_offset is expressed in 8-octet units. Fragmentation and reassembly can occur at intermediate devices and are not necessarily end-to-end.

11 Transport layer

11.1 General

The reference model in this standard is composed of five protocol layers. In this model, transport is the fourth layer, immediately subordinate to the AL. A TLE responds to service requests from a superior ALE at a TSAP and issues service requests to an inferior NLE at an NSAP.

The TL is responsible for end-to-end communication and operates only at the communication endpoints (as opposed to the routing devices).

The TL provides connectionless services, usually with cryptographic-based security:

- The connectionless service extends UDP over IPv6, as defined in IETF RFC 768 and IETF RFC 2460, by providing an alternative, more secure TPDU integrity check with cryptographic authentication and, when so configured, privacy.
- Security is handled in a manner similar to that of the DL, but from end-to-end rather than hop-by-hop.
- The connectionless service also extends 6LoWPAN as defined by IETF RFC 6282. When security is activated, it is possible to compress the UDP header to one octet by eliding the UDP checksum and relying on the larger, appended transport message integrity code (TMIC) to provide end-to-end integrity.

11.2 TLE reference model

The TLE reference model is shown in Figure 107.

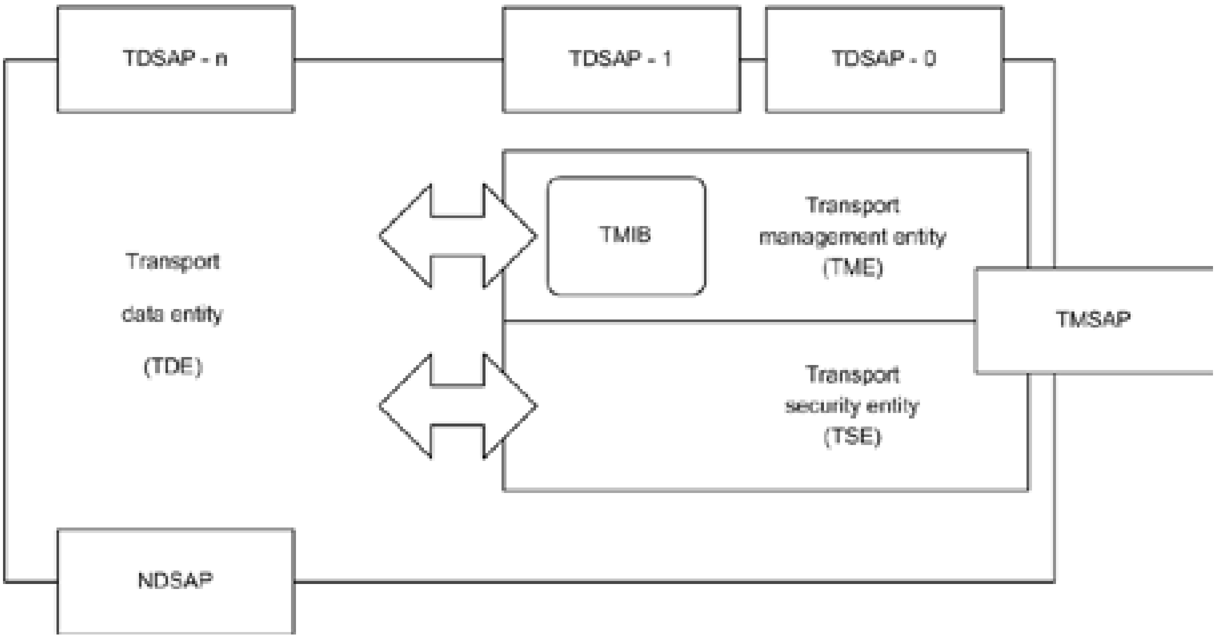


Figure 107 – TLE reference model

The TLE conceptually includes the transport management entity (TME), the transport data entity (TDE) and the transport security component (TSC).

The TDE has a dedicated TDSAP for the DMAP, TDE-0; a TDSAP that is reserved for SMAP, TDE-1; and a TDSAP for each UAP, TDE-2 to TDE-n.

The TME configures and monitors the actions of the TLE. The transport management information base (TMIB) (see 11.6.2.5.2) maintains abstract information for the TME, that is accessed at the TMSAP as part of the TLMO. The TSC provides the TLE's security functions, based on tables of security information that are maintained and monitored via the TMSAP. The TDE uses the TSC to perform security operations on TPDUs.

11.3 Transport security entity

11.3.1 General

The TSC is conceptually a compartmented service within the TLE. TL security can protect the integrity of the conveyed transport service data unit (TSDU), the transport header and the transport endpoint IPv6Addresses. When active, it also provides protection against excessive TPDU delay and TPDU replay, and can encrypt the TSDU for confidentiality.

11.3.2 Securing the TL

The TSC and DSC share functionality. The TSC is responsible for:

- determining which security level shall be applied to a given secure session based on policies;
- on receipt of a TPDU (as part or all of an NSDU),
  - – discarding non-conforming TPDUs;
  - – discarding TPDUs that fail, depending on T-association configuration, either
    - the traditional UDP integrity check that protects against non-malicious errors, or
    - cryptographic authentication checks that protect against both accidental and deliberate TPDU modification, excessive TPDU delay and TPDU replay;
  - – decrypting a protected TSDU conveyed by the TPDU;
- when preparing a TPDU for transmission via a NLE,
  - – encrypting TSDUs that are to have confidentiality protection during TL conveyance; and
  - – depending on the session configuration, either
    - including the traditional, computed UDP integrity checksum to protect against non-malicious errors, or
    - including cryptographic authentication material to protect against both accidental and deliberate TPDU modification, excessive TPDU delay and TPDU replay.

The functionality of the TSC is defined in 7.3.3.

Similar to DL security, TL security uses a cryptographic block cipher in a generic authenticated encryption block cipher mode called the Counter with CBC-MAC (CCM*), as defined by IEEE 802.15.4:2011, Annex B. The default block cipher is AES-128, but other block ciphers can be used where required by national regulation.

CCM* enables authentication of a message while encrypting only a part of that message, leaving the rest of the message (usually a header) in the clear. When this feature is enabled for a particular session, the UDP checksum is not used and is elided in the compressed form of the TPDU, as specified in 11.4.3.4. It is always present in the expanded form of the TPDU, to provide compliance with IETF RFC 2460 (UDP over IPv6).

11.4 Transport data entity

11.4.1 General

The TDE provides connectionless services based on the User Datagram Protocol (UDP, IETF RFC 768) over IPv6 (IETF RFC 2460), with a compression as defined in IETF RFC 6282.

The main steps in TPDU construction are, in order:

a) A TSDU is received from a local ALE through a TDSAP.

   NOTE This TSDU can convey a single APDU or multiple concatenated APDUs.

b) The TSDU is protected and timestamped as described in 7.3.3.2.1. The resulting UDP payload comprises a TL security header, the TSDU, and, when cryptographic security is configured, a TMIC. The TSDU is encrypted for confidentiality if so indicated by the TL security header. The presence of the TMIC, and its size, are conveyed to receiving device(s) in the TL security header.

c) A UDP header is prepended to the UDP payload. The UDP header may be compressed. Compression involves eliding the UDP checksum from the UDP header when the UDP payload contains an alternative integrity check. An integrity check within the UDP payload may be a TMIC as indicated in the TL security header and/or an integrity check embedded in a tunneled payload.

11.4.2 UDP over IPv6

UDP (IETF RFC 768) provides a connectionless mode of computer communication in the environment of an interconnected set of computer networks.

UDP is essentially transparent to application operation, leaving both control and responsibility for proper network operation to the AL. Proven, standard-based approaches exist for this purpose. Relevant issues are further documented in 11.4.4.

IETF RFC 768 assumes that IPv4 is used as the underlying protocol and defines the computation of a UDP checksum, used for error detection, that covers a UDP pseudo-header of information from the IPv4 network header, the UDP header, and the UDP payload. If that computation yields a result of zero, it is changed to 0xFFFF for placement in the UDP header and the value of zero is reserved to indicate that there is no checksum computation.

IETF RFC 2460 specifies the changes to IETF RFC 768 to adapt UDP for IPv6. The changes to UDP are minor and relate to the computation of the UDP checksum, which becomes mandatory and includes a modified UDP pseudo-header adapted for IPv6, as shown in Figure 108. Per IETF RFC 2460, IPv6 receivers discard UDP packets containing a checksum of 0x0000 and log this event as an error.

Source address	Destination address	Upper-layer packet length	0 (zero)	Next header

Figure 108 – UDP pseudo-header for IPv6

In the pseudo-header, the value in the field named ‘Next header’ is set to 17 to indicate UDP and the value in the field named ‘Upper-layer packet size’ is the same as the Length field in the UDP header and accounts for the size of the whole TPDU, including the UDP header.

A different pseudo-header, the TSS pseudo-header is used in TL security processing, as described in 7.3.3.2.1.

NOTE See IETF RFC 768, IETF RFC 2460, 4.5.2.1 and 7.3.3.2.1. to understand the purposes of these pseudo-headers.

11.4.3 UDP header transmission and compression

11.4.3.1 General

The TL supports uncompressed UDP for both transmission and reception of a TPDU. A device that implements the TL of this standard shall support uncompressed UDP, but should compress the UDP header for transmission over the wireless network.

Table 221 describes the UDP header encoding.

Table 221 – UDP header encoding

Number of octets	Bits (presented in IEC convention, which is different from IETF convention)						
	7	6	5	4	3	2	1..0
2	Source port: that maps one to one with the source device TDAP						
2	Destination port: that maps one to one with the destination device TDAP						
2	Size: the size in octets of this user datagram, including this header and the APDU						
2	Checksum: the 16-bit one's complement of the one's-complement sum of a) the UDP pseudo-header of Figure 108, derived from the IP header; b) the UDP header; and c) the possibly-encrypted TSDU, padded with zero octets at the end (if necessary) to make a multiple of two octets. The checksum field of the UDP TPDU is first set to zero during the checksum computation.						

The TL also complies with 6LoWPAN, which specifies a method for compressing UDP using a Next Header Compression (LoWPAN_NHC) format and mandates the use of UDP compression over the local D-subnet. A device that implements the TL of this standard shall thus support all the combinations offered by the LOWPAN_NHC for compression and expansion of the UDP header.

The maximum size of a TSDU depends on a variety of factors, such as the buffering capacity at each session endpoint TLE, the selected TL security level, and network characteristics. The system manager configures TSDU maximum size on a per-contract basis to account for these and any other relevant considerations, through the value of Assigned_Max_TSDU_Size, as described in Table 30.

11.4.3.2 Compressing and restoring UDP port numbers

Compression for UDP port numbers is described in IETF RFC 6282:2011, 4.3. Those port numbers that are optimized by the compression process should be assigned to the ports (AEs) with highest frequency of use. Optimum compression is obtained when both the source and destination port numbers start at a base number P and are expressed as P+short_port, where:

- P is the base port, $61\,616_{10}$, that is, 0xF0B0;
- short_port is a positive integer that is ≤ 15 (i.e., 0x0F).

In such a case, the pair of source and destination ports is compressed to a single octet that specifies the source and destination values as short_ports, which reduces the size of the TPDU field required to convey the UDP port numbers from four octets to one octet.

When it is not possible to have both ports within the 0xF0B0..0xF0BF range, it is still possible to reduce the TPDU size by one octet if either the source or the destination port is in the 0xF000..0xF0FF range; in that case the corresponding high-order octet of 0xF0 is elided. It is thus recommended that server applications listen to a port in the 0xF000..0xF0FF range. For example, a typical field device has a DMAP at 0xF0B0 (encoded as short_port 0) and may have its single UAP at 0xF0B2 (encoded as short_port 2), since 0xF0B1 (encoded as short_port 1) is reserved for the local SMAP.

11.4.3.3 Eliding and restoring the UDP Length field

If the UDP header is not compressed, then the NSDU size is equal to the TPDU size that is placed in the UDP header.

If the UDP header is compressed, the UDP Size field is always elided in transmission, and is inferred upon reception from the NSDU size passed by the receiving NLE. In that case, the TPDU size is equal to the NSDU size plus te computable difference between the sizes of a full UDP header and the actual UDP header as compressed.

Even if none of the other fields in the UDP header are compressed or elided, compressing the UDP Size field reduces the TPDU size by one octet and enables the use of the basic header at the NL.

For example, in the optimal case described in Table 223, the compressed UDP header requires 2 octets, thus saving 6 octets compared with the fully-expanded UDP header. The TDPU size for the expanded UDP header is thus NSDU_Size + 6.

11.4.3.4 Eliding and restoring the UDP checksum

The TL complies with the 6LoWPAN rules and operations defined in IETF RFC 6282:2011, 4.3.2, as follows:

- The authorization to elide the UDP checksum might come from the ALE or the TSC. The TSC hints at the presence or absence of this checksum by specifying whether the TMIC is present or not.
- An ALE may elide the UDP checksum in the following cases:
  - Tunneling: The source ALE is tunneling a PDU (of unspecified type) that possesses its own integrity mechanism that provides at least as much protection as the 16-bit UDP checksum.
  - Application MIC: The source ALE applies an end-to-end message integrity check of at least 16 bits (e.g., as part of a key exchange).
- If the local NLE is a backbone router, then the router shall recompute the elided checksum based on the received packet as specified in IETF RFC 2460 and shall place the result of that computation in the reformed UDP header before transmission over an IPv6 backbone network.
- If the NLE is the destination of the packet and is aware of the presence of the TMIC in the TPDU, then it may omit the UDP checksum operation completely (neither recompute nor check the checksum).
- A backbone router that forwards an IPv6 packet into D-subnet uses the security control field in TPDU security header to determine whether a TMIC is present or not. When performing the UDP compression, the backbone router should elide the UDP checksum if the TMIC is present but it shall not elide the UDP checksum if the TMIC is not present. Conversely, the fact that the checksum is not compressed is not an indication that there is no TMIC in the TPDU.

6LoWPAN permits UDP checksum compression only when the alternative protection within the TPDU provides integrity protection at least as great as the UDP checksum, including end-to-end protection for all the fields that the UDP checksum would protect. To meet that requirement, this standard defines a TSC pseudo-header that is included in the TMIC computation, as described in 7.3.3.2.1. Compared to the standard UDP pseudo-header of IETF RFC 2460, the TSC pseudo-header includes the UDP ports but does not include the payload size, since that is implicitly protected by the TMIC. The structure of the TSC pseudo-header is defined in Table 47.

In the case of a TPDU being prepared for transmission, the source and destination IPv6Addresses are obtained from the contract information; the contract ID itself is passed by the ALE as a parameter associated with the TSDU. The Next header field of the pseudo-header is set to 17 (UDP). The source port is obtained from the TL context associated with the TSAP, whereas the destination port is another parameter associated with the TSDU. The TSC pseudo-header has been modeled as if it were being constructed by the TDE and passed together with the TSDU to the TSC for security processing.

The TSC uses the TSC pseudo-header in the TMIC computation by prepending it to the TSDU, but the TSC pseudo-header is not encrypted when encryption is to be performed on the TSDU. Once the cryptographic process is completed, the TSC shall remove the TSS pseudo-header from the processed TSDU, add its own headers and trailers, and return the protected UDP payload to the TDE. The TDE shall complete the TPDU by prepending, and usually compressing, the UDP header to form the NSDU that is passed to the NL.

In the case of a received TPDU, the source and destination IPv6Addresses and the priority bits are passed by the NL as parameters associated with the NSDU, and the ports are obtained from the UDP header in the TPDU once the UDP header is expanded. The TDE shall fill the pseudo-header, remove the UDP header from the NSDU, and pass the resulting protected UDP payload together with the pseudo-header and the priority bits to the TSC for security processing.

Since an NPDU might be received out of sequence, perhaps due to its priority settings (see Table 205), the TSC may use that priority information to partition its look-aside cache, potentially optimizing its ability, within limited memory resources, to validate TPDUs that incurred substantial transit delay. The priority information is conveyed with the NSDU in the N-DATA.indication received by the TLE from a local NLE; the TLE forwards that information and the TPDU to the TSC.

The receiving TSC strips the TL security headers and trailers from the protected TSDU, prepends the TSC pseudo-header, UDP ports, and security headers (see Figure 41), and then performs the security verifications. If an integrity check is applied, the TLE can verify whether any of the critical parts of the NL and TL information was modified during end-to-end transport.

If the UDP upper-layer checksum field is elided, it is up to the receiving device on the wireless D-subnet to recompute the UDP checksum as part of the process of reversing the 6LoWPAN compression, if necessary, before further forwarding of the conveying NPDU to a destination outside the wireless D-subnet. This compression reversal step is needed in particular by a middlebox, such as a BBR border router, that forwards the uncompressed NPDU onto a different network.

NOTE It is useless for the addressed destination TLE to reconstitute the uncompressed TPDU.

If the UDP checksum is not elided, then the UDP checksum shall be computed in transmission after the security operation is complete and the security fields are preset for the computation. The UDP payload that is used for the UDP checksum shall include all data from the UDP header to the end of the TPDU, including the security fields, application data, and MIC fields, whether the application data is encrypted or not. Upon reception, it shall be verified by the TDE prior to TSC operation unless it is known by the TDE that the TSC verifies the MIC, in which case the checksum may be ignored.

11.4.4 TSAPs and UDP ports

A device autoconfigures one link-local IPv6Address based upon its EUI64Address. After the join process is complete, a device also has at least one global IPv6Address. This standard permits, but does not require, a device to support more than one global IPv6Address. A UDP port shall be associated with no more than one UAP for a given IPv6Address.

The port is used as the local port for all transmissions from/to that UAP using that IPv6Address. Further multiplexing between UAP objects is the responsibility of the ALE, done within TSDUs, and thus is transparent to the TLE. As a result, only a limited number of ports are actually used in the system, and there is no dynamic port allocation after the (re-)initialization phase of the applications.

Each UAP shall be responsible for knowing its UDP port number and registering it with the TLE. At that time, a TLSAP shall be mapped on a one-to-one basis with the UAP and the local port for the given IPv6Address. An application process address shall consist of a device address concatenated with its TLSAP identifier.

11.4.5 Good network citizenship

(N)-SDUs are exchanged with the upper and lower layers through (N)-DATA primitives. An implementation can report transient congestion in a lower layer via a specific completion code in an (N)-DATA.confirm primitive that is conveyed all the way up the protocol suite. Upon receiving a completion code indicating transient congestion, an ALE should either delay retransmission of the (N)-SDU by an exponential backoff amount or simply drop the (N)-SDU.

Since the TL is based on UDP, it is desirable that the UAPs support TCP-Friendly Flow Control or TCP-Friendly Rate Control (TFRC, IETF RFC 5348), depending on the nature of the flow, in order to protect future uses of the network and to share the backbone network fairly.

This is usually achieved by implementing a blocking protocol at the AL that enforces a round-trip time (RTT) between TPDUs or uses RTT to compute a loss recovery timer. In the case of periodic samples, this may be achieved either by reserving bandwidth or by keeping the sampling period above the initial RTT value for TCP of 3 s (see IETF RFC 6298). Higher sample rates may be used in some applications.

11.5 TPDU encoding

11.5.1 General

A TPDU exchanged between the two TLEs of a communication shall be composed of a UDP header in its uncompressed form, a security information header, the TSDU, and a TMIC, as shown in Figure 109.

Uncompressed UDP header	Security header	Application payload	MIC

Figure 109 – TPDU structure

The NSDU data passed by the sending TLE to a local NLE via an NSAP includes any UDP header compression that has been applied to the TPDU. If compression was applied, the 6LoWPAN_NHC-for-UDP octet is the first octet of the NSDU.

The pair of IPv6Addresses, the transport type (UDP=17) and the size of the NSDU are not present in the NSDU, so are passed as associated parameters. Whether 6LoWPAN_NHC compression took place is also passed as a parameter.

11.5.2 Header compression – User datagram protocol encoding

When the UDP header is not compressed, the NSDU is identical to the TPDU and the UDP header shall be present in full without a prefixed 6LoWPAN_NHC-for-UDP encoding octet.

If any of the UDP header fields is compressed, a 6LoWPAN_NHC-for-UDP encoding octet shall be prepended that describes the compression. The NSDU shall be formed by replacing the UDP header at the beginning of the TPDU with the 6LoWPAN_NHC-for-UDP encoding octet followed by the compressed data. The 6LoWPAN_NHC-for-UDP encoding octet is structured as in Table 222.

Table 222 – UDP 6LoWPAN_NHC-for-UDP encoding octet

Number of octets	Bits (presented in IEC convention, which is different from IETF convention)							
	7	6	5	4	3	2	1	0
1	1	1	1	1	0	C	P	

The C (checksum compression) and P (ports compression) fields are defined as follows:

C field (1 bit):

0: The TPDU's 16-bit checksum is conveyed in the TPDU.

1: The TPDU's 16-bit checksum is elided from the TPDU.

P field (2 bits):

00: The TPDU conveys the source and destination ports as 16-bit values.

01: The TPDU conveys the source port as a 16-bit value while only the low-order 8 bits of the destination port, which is in the range 0xF000..0xF0FF, is conveyed in the TPDU.

10: The TPDU conveys the destination port as a 16-bit value while only the low-order 8 bits of the source port, which is in the range 0xF000..0xF0FF, are conveyed in the TPDU.

11: The TPDU conveys only the low-order 4 bits of the source and destination ports, which are in the range 0xF0B0..0xF0BF.

Compressed fields appear in the same order as they do in the UDP header format specified in IETF RFC 768.

When the highest degree of compression is achieved, only the compressed short_port numbers are carried after the 6LoWPAN_NHC-for-UDP encoding octet as shown in Table 223.

Table 223 – Optimal UDP header encoding

Number of octets	Bits (presented in IEC convention, which is different from IETF convention)						
	7	6	5	4	3	2	1..0
1	1	1	1	1	0	1 (checksum is elided)	11 (source and destination ports are compressed)
2	source short_port				destination short_port		

The expanded values for the compressed source port and destination ports shall be calculated using the formula:

UDP_port_number = P + short_port

where short_port is the 4-bit compressed port number and P is the value 0xF0B0 ($61\,616_{10}$). In other words, short_port conveys the low-order 4 bits of the UDP port number.

At the time of the join process, the field device does not have the relevant cryptographic material to compute a TMIC. Because this situation requires that the TMIC be omitted, the device shall compute the UDP checksum as prescribed by IETF RFC 768 and IETF RFC 2460 and shall use the link-local IPv6Addresses to compute and send the checksum and transmit. In this case, if the UDP ports can be compressed, the encoded UDP header is formatted as represented in Table 224.

Table 224 – UDP header encoding with checksum and compressed port numbers

Number of octets	Bits						
	7	6	5	4	3	2	1..0
1	1	1	1	1	0	0 (checksum is present)	11 (source and destination ports are compressed)
2	source short_port				destination short_port		
3	UDP checksum						
4							

11.5.3 TPDU security header

For information on the TPDU security header, encoding, and decoding, see 7.3.3.6.

11.6 TL model

11.6.1 General

A TLE provides two interfaces:

- A TDE TDSAP for each UAP and one for the DMAP.
- A TME TMSAP for the DMAP.

These interfaces are illustrated in the protocol reference model of this standard, shown in Figure 16.

All interfaces between the TLE and adjacent layer entities or management entites are internal interfaces within the device, and thus are unobservable. Therefore they are not subject to standardization.

Upper layers use the TDSAP to exchange data communicated via the TLE. There is a separate TDSAP instance for each UAP, plus an instance for the DMAP.

The TDSAP includes a multiplexer function that adapts the address namespace of the ALE to the native address namespace of the TLE. The DMAP uses the TMSAP to configure the TLE and monitor its operation.

The TLE communicates with a local NLE using an NDSAP.

11.6.2 Data services

11.6.2.1 General

For illustration purposes, an example set of primitives is provided in this standard. The TL offers an unconnected service based on the UDP model.

A TLE uses the interface supplied by the TDSAP to transmit and receive protocol data units with the ALE.

The TDSAP transfers the TSDU, along with control and status information parameters.

11.6.2.2 T-DATA.request

11.6.2.2.1 General

T-DATA.request instructs the TL to transmit a protocol data unit.

11.6.2.2.2 Semantics of the service primitive

The semantics of T-DATA.request are as follows:

```
T-DATA.request   (
                 ContractId
                 TSDU_size
                 TSDU
                 Priority
                 DE
                 TDSAP
                 \Destination_Port)
```

Table 225 specifies the elements for T-DATA.request.

Table 225 – T-DATA.request elements

Element name	Element identifier	Element scalar type
ContractId (identifies the contracted TL resources associated with the protocol data unit)	1	Unsigned16 Named values: 0: no contract
TSDU_size (size in octets of the protocol data unit passed from the ASL)	2	Unsigned16
TSDU (the TSDU to be transmitted)	3	OctetString
N-priority (identifies the priority within the NL of this TSDU)	4	Unsigned2
DE (identifies the Discard Eligibility of this TSDU)	5	Unsigned1
TDSAP (ID of the TDSAP that grants UDP service over SourcePort)	6	Unsigned16
Destination_Port (UDP destination (remote) port for the TSDU)	7	Unsigned16

The TLE maintains a table indexed by TDSAP that contains (implicitly or explicitly) the local IPv6Address and (explicitly) the local port for transmission. That table is accessed to obtain the source IPv6Address and port for transmission over a given TDSAP. The destination IPv6Address is obtained from the Destination_Address field in the contract information, indexed by the contract ID.

The TLE does not retain state information related to the remote port; thus, information that is passed by the UAP is placed in the TPDU without checking.

The N-priority parameter communicates the N-priority that is to be used by the NL. The N-priority settings are not used in the TL, but they are passed on through the NL to the DL, where they are used to select the priority class in the DL header.

11.6.2.2.3 Appropriate usage

An ALE invokes the T-DATA.request primitive to pass a TSDU to a local TLE for transmission on the network.

11.6.2.2.4 Effect on receipt

On receipt of the T-DATA.request primitive, the TLE looks up the T-security level in the KeyDescriptor to determine the processing required by the local TSC, constructs the TPDU header, forms the TPDU, and generates the N-DATA.request to a local NLE at an NSAP.

11.6.2.3 T-DATA.confirm

11.6.2.3.1 General

T-DATA.confirm is used by the TLE to respond to a T-DATA.request. The confirmation is immediate and tells the requesting ALE either that the request was successful or that an error was detected. Error conditions include such issues as an unrecognized contract ID, a TSDU size that is incorrect or excessive, or an internal transient error such as network congestion beyond the capacity of the TSC and its agents to cope.

11.6.2.3.2 Semantics of the service primitive

The semantics of T-DATA.confirm are as follows:

```
T-DATA.confirm    (
                                    ContractId
                                    TDSAP
                                    status
                                    )
```

Table 226 specifies the elements for T-DATA.confirm.

Table 226 – T-DATA.confirm elements

Element name	Element identifier	Element scalar type
ContractId (identifies the contracted TL resources associated with the TPDU)	1	Type: Unsigned16 Named values: 0: no contract
TDSAP (ID of the TDSAP that grants UDP service over SourcePort)	2	Type: Unsigned16
Status (the result of the T-DATA.request primitive)	3	Type: Unsigned Valid range: (see Table 227)

Table 227 provides the status codes for T-DATA.confirm.

Table 227 – T-DATA.confirm status codes

Name	Description
SUCCESS	TSDU accepted
TRANSIENT_FAILURE	TSDU rejected, but can be retried after a short period of time
FAILURE	Generic failure: TSDU rejected without explicit reason
INVALID_CONTRACT	Specific failure: Unrecognized contract ID; TSDU rejected
INVALID_LENGTH	Specific failure: TSDU is larger than Assigned_Max_TSDU_Size; rejected
PORT_ERROR	Specific failure: SourcePort is not registered for TDSAP; TSDU rejected
SAP_ERROR	Specific failure: Unknown TDSAP; TSDU rejected

11.6.2.3.3 When generated

A TLE generates T-DATA.confirm in response to a local T-DATA.request. The T-DATA.confirm returns synchronously either a status of success or the appropriate error code.

11.6.2.3.4 Appropriate usage

The T-DATA.confirm notifies the ALE of the result of its request to transmit an TSDU.

11.6.2.4 T-DATA.indication

11.6.2.4.1 General

T-DATA.indication is used to transfer a TSDU to the ALE. It is generated when a TPDU has been successfully received from a local NLE and processed by the TLE.

A received TPDU that fails T-security processing or that specifies an unregistered destination port is discarded on receipt. Such errors are logged in the TLE's PIB.

11.6.2.4.2 Semantics of the service primitive

The semantics of T-DATA.indication are as follows:

```
T-DATA.indication (
            SrcAddr
            SrcPort
            TSDU_size
            TSDU
            ECN
            TDSAP
            transportTime
            )
```

Table 228 specifies the elements for T-DATA.indication.

Table 228 – T-DATA.indication elements

Element name	Element identifier	Element scalar type
SourceNetworkAddress (IP address of the remote end)	1	Type: IPv6Address
SourcePort (UDP source port in incoming TPDU)	2	Type: Unsigned16
TSDU_size (size in octets of the accompanying TSDU)	3	Type: Unsigned16
TSDU (the received higher-layer content of the TPDU)	4	Type: OctetStringN
ECN (explicit congestion notification bits)	5	Type: Unsigned2
TDSAP (ID of the TDSAP that grants UDP service and matches a local port)	6	Type: Unsigned8
TransportTime (end-to-end transit time from originating to receiving TSC)	7	Type:Unsigned16
NOTE TransportTime is related to tpduMaxAge.		

A TLE maintains a table that contains the TDSAP, which is indexed by (implicitly or explicitly) the local address and (explicitly) the local port for reception. That table is accessed to find the TDSAP used to pass the TSDU to the UAP.

11.6.2.4.3 Appropriate usage

A TLE invokes T-DATA.indication to notify the addressed ALE of a received TSDU.

11.6.2.4.4 Effect on receipt

On receipt of T-DATA.indication, the ALE processes the received TSDU.

11.6.2.5 Management services

11.6.2.5.1 General

The TLE's management service is controlled by the TL management object (TLMO) in the DMAP. The TLMO controls the TLE functionalities by:

- measuring TL latency and making the related adaptations to comply with latency requirements dynamically; and
- collecting operational parameters.

The TLMO uses its TMSAP interface to configure and control the operation of the TLE. The TLE's TME provides a TMIB that maintains the state information necessary to implement the TMSAP functionality.

11.6.2.5.2 Attributes

Table 229 specifies the attributes of the TLMO.

Table 229 – TLMO attributes

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved	1	Reserved by the standard	—	—
MaxNbOfPorts	2	Number of active ports	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: Device-dependent Valid range: 2..255	The minimum value covers a typical device with a single DMAP and a single UAP TDSAPs
TPDUin	3	Counter reporting the number of received TPDUs	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each TPDU received from a remote TLE
TPDUinRejected	4	Counter reporting the number of rejected TPDUs	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each data unit received from a remote TLE that was discarded (e.g., for security reasons). Note: there is no such counter within the DSMO
TSDUout	5	Counter reporting the number of TSDUs passed to a local ALE	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each TPDU received from a remote TLE that resulted in the conveyance of a contained TSDU to a local ALE
TSDUin	6	Counter reporting the number of received TSDUs	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each TSDU received from a local ALE
TSDUinRejected	7	Counter reporting the number of rejected TSDUs	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each TSDU received from a local ALE that is rejected
TPDUout	8	Counter reporting the number of TPDU passed to the NL	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Default value: 0	Incremented with each TSDU received from a local ALE that is conveyed to and accepted by a local NLE

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
IllegalUseOfPortAlertDescriptor	9	Used to change the priority of IllegalUseOfPort alert; this alert can also be turned on or turned off	Type: Alert report descriptor	—
			Classification: Static	
			Accessibility: Read/write	
			Default value: [TRUE, 8] -- medium	
TPDUonUnregisteredPortAlertDescriptor	10	Used to change the priority of TPDUonUnregisteredPort alert; this alert can also be turned on or turned off	Type: Alert report descriptor	—
			Classification: Static	
			Accessibility: Read/write	
			Default value: [TRUE, 4] -- low	

For each attribute, the TLMO provides read and write methods available to the DMAP. Those methods are implemented by requesting TME services across the TMSAP.

11.6.2.5.3 Methods

In addition to the read and write service for the attributes, additional TLMO methods provide access to TME services across the TMSAP.

Table 230 describes the reset method.

Table 230 – TL management object methods – Reset

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Method name	**Method ID**	**Method Description**		
Reset	1	Reset the transport to initial states		
	Input Argument			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Forced	Boolean	Forced means that all data are updated without any interaction with other entities. TSAP-related tables are emptied, related memory is freed and all counters are set to 0
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Status	Unsigned8	0 = success, > 0 failure

Table 231 describes the halt method.

Table 231 – TL management object methods – Halt

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Method name	Method ID	Method Description		
Halt	2	Halts a port and places it back in its initial state. Similar to a reset, but scoped to one UDP port only. All traffic is interrupted on that port and the TSAP needs to be reopened for that port to become operational again. The TSAP-related table entries for the port are emptied and related memory is freed		
	Input Arguments			
	Argument number	Argument name	Argument type (data type and size)	Argument description
	1	DeviceAddress	IPv6Address	This device IPv6Address
	2	PortNumber	Unsigned16	The port to halt
	Output Arguments			
	Argument number	Argument name	Argument type (data type and size)	Argument description
	1	Status	Unsigned8	Named values: 0: success, 1: generic failure, 2: bad port number

Table 232 describes the PortRangeInfo method.

Table 232 – TL management object methods – PortRangeInfo

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Method name	Method ID	Method Description		
PortRangeInfo	3	Reports the UDP ports range information		
	Input Arguments			
	Argument number	Argument name	Argument type (data type and size)	Argument description
	1	DeviceAddress	IPv6Address	This device IPv6Address
	Output Arguments			
	Argument number	Argument name	Argument type (data type and size)	Argument description
	1	Status	Unsigned8	0 = success, > 0 failure
	2	NbActivePorts	Unsigned16	Number of active ports
	3	FirstActivePort	Unsigned16	First active port
	4	LastActivePort	Unsigned16	Last active port

Table 233 describes the GetPortInfo method.

Table 233 – TL management object methods – GetPortInfo

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Method name	**Method ID**	**Method Description**		
GetPortInfo	4	Reports the UDP port information for a given UDP port or the first active UDP port		
	Input Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	DeviceAddress	IPv6Address	This device IPv6Address
	2	PortNumber	Unsigned16	The port whose info is requested (0 = lowest)
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Status	Unsigned8	0 = success, > 0 failure
	2	PortNumber	Unsigned16	This port number
	3	UID	Unsigned32	Owner application ID
	4	Compressed	Boolean	Compression applies to this port
	5	TPDUsInOK	Unsigned32	Number of TPDUs accepted over the port
	6	TPDUsInKO	Unsigned32	Number of TPDUs rejected over the port
	7	TPDUsOutOK	Unsigned32	Number of TPDUs sent over the port
	8	TPDUsOutKO	Unsigned32	Number of TPDU transmission failures

Table 234 describes the GetNextPortInfo method.

Table 234 – TL management object methods – GetNextPortInfo

Standard object type name: TL management object (TLMO)				
Standard object type identifier: 122				
Method name	**Method ID**	**Method Description**		
GetNextPortInfo	5	Reports the UDP port information for a given UDP port or the first active UDP port		
	Input Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	DeviceAddress	IPv6Address	This device IPv6Address
	2	PortNumber	Unsigned16	The previous port from which info is requested
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Status	Unsigned8	0 = success, > 0 failure
	2	PortNumber	Unsigned16	The port for which info is reported
	3	UID	Unsigned32	Owner application ID
	4	Compressed	Boolean	Whether compression applies to this port
	5	TPDUsInOK	Unsigned32	Number of TPDUs accepted over the port
	6	TPDUsInKO	Unsigned32	Number of TPDUs rejected over the port
	7	TPDUsOutOK	Unsigned32	Number of TPDUs sent over the port
	8	TPDUsOutKO	Unsigned32	Number of TPDU transmission failures

11.6.2.5.4 Alerts

Table 235 describes the alert to indicate illegal use of a port by an application.

Table 235 – TL management object alert types – Illegal use of port

Standard object type name(s): TL management object (TLMO)					
Standard object type identifier: 122					
Description of the Alert: Alert to indicate illegal use of a port by an application					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	1= Communication diagnostic	0 =IllegalUseOfPort	8 = Medium	Type: Unsigned16	16-bit port number

Table 236 describes the alert to notify of a received TPDU that addresses an unregistered port.

Table 236 – TL management object alert types – TPDU received on unregistered port

Standard object type name(s): TL management object (TLMO)					
Standard object type identifier: 122					
Description of the Alert: Alert to notify of a TPDU received on unregistered port					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	1 = Communication diagnostic	1=TPDUonUnregisteredPort	4 = Low	Type: OctetString	First 40 octets of the TPDU

Table 237 describes the alert to notify of a received TPDU that does not meet local security policies.

Table 237 – TL management object alert types – TPDU does not match security policies

Standard object type name(s): TL management object (TLMO)					
Standard object type identifier:					
Description of the Alert: Alert to notify of a TPDU that does not match security policies					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	1 = Communication diagnostic	2=TPDUoutOfSecurityPolicies	2 = Journal	Type: OctetString	First 40 octets of the TPDU

12 Application layer

12.1 General

The application layer (AL) defines software objects to model real-world objects, and also defines the communication services necessary to enable object-to-object communication between distributed applications in an open, interoperable application environment compliant with, and based on, this standard. This standard does not define the operation of the distributed applications themselves; that is, neither the local operation of the application itself, such as the manner in which an application acquires the values of the object attributes it supports for access, nor direction regarding how and when an application applies the models and/or the services defined herein, are addressed by this standard.

NOTE 1 For example, a real-world analog input is modeled as an AnalogInput object. The AnalogInput object often communicates its process variable to a correspondent party by using the AL-provided publish service.

The AL supports wireless devices in the field, as well as gateways that integrate a wireless network compliant with this standard and its devices with a host control system.[8]

The application model in this standard is specifically designed to satisfy the constraints of wireless communication environments.

NOTE 2 An object oriented AL approach is used for the following key reasons:

Command-based protocols are able to be designed to conform to the object model defined by this standard by describing the commands as separate object methods. That is, a command-based application is able to be modeled using the object model in this standard.

The object model supports well accepted architectural principles of logical information separation. For example, management information is logically separate from operating data. Operational information for independent variables is logically separate. In order to maintain separation of information, the protocol is required to identify the corresponding object. This adds a one-octet overhead to identify the object, which was deemed to be a more than reasonable approach for architectural separation of information.

12.2 Energy considerations

The need to extend battery life makes energy-efficient messaging extremely important. The use of battery power or energy scavenging/harvesting techniques for connected field devices requires additional considerations in communication layer design, compared to the approach taken for wired devices. Not only does every communicating layer need to consider device resource availability, but it also needs to consider energy consumption minimization (within architecturally appropriate constraints of course) in order to extend battery life or to operate within the scavenging/harvesting budget.

Since energy is consumed by message processing, as well as by the basic control operation of the device, it is necessary to balance communications efficiency with employing proven and well-accepted architectural principles of information separation as well as message processing efficiency. The native application model in this standard is defined to meet these needs.

12.3 Legacy control system considerations

Wireless networks compliant with this standard may be connected to legacy control systems.

NOTE 1 A model that integrates with existing systems makes possible the reuse of existing and proven tools and interfaces, and also reduces overall development and test time, resulting in earlier production of robust implementations.

---

8 See 5.2.6.5 for a more complete discussion of the roles supported by this standard.

An application process in a native device communicates over the network using only ASL-defined services and payloads. An application process in a non-native device requires communication of constructs that are not defined by the ASL service payloads. This communication from the non-native device over the network defined by this standard is accomplished by using the subset of standard services defined in this standard that support communication of payloads that are not explicitly and entirely defined by this standard.

The native AL defines special objects and services to support non-native protocol tunneling to meet system integration needs.

The set of defined application objects that support non-native defined payloads are the tunnel object and the interface object. The set of standard-defined application services that support payloads beyond those defined within this standard are the tunnel service, which is used for aperiodic communication, and the publish service using non-native mode, which is used for periodic communications. For example, the payload of the tunnel service for aperiodic messaging in order to encapsulate data constructed by a legacy system is not defined within WISN.

NOTE 2 One way to achieve a synthesis with existing systems is to create an energy- and resource-optimized version of a wire-oriented existing legacy approach by mapping the legacy model to the model in this standard and directly employing the native AL. Such mappings usually are defined by individual protocol consortia, such as the HART Communication Foundation (HCF), Fieldbus Foundation (FF), PROFIBUS Nutzerorganisation (PNO), ODVA, which supports managed protocols such as CIP (Common Industrial Protocol), and others.[9] Another way to achieve synthesis is to use a protocol tunnel through the native AL. For both approaches, energy and resource implications are important considerations.

NOTE 3 Whichever method is chosen, the higher-level system still communicates with wireless devices within a network compliant with this standard.

NOTE 4 Electronic device description files often are used to meet this requirement. For this standard, a device description language (DDL) or an extended device description language (EDDL) describe native devices.

NOTE 5 See Annex R for further details regarding host system interface.

12.4 Introduction to object-oriented modeling

12.4.1 General

An object model is a protocol-, platform-, and language-neutral means of describing and distinguishing components (system elements) that have a unique identity. Objects separate the world into meaningful and manageable pieces. Not only do object definitions promote modularity, but they also promote component reusability. An object can represent anything that has a state and a behavior; objects expose attributes to represent state, and provide methods that operate on the object's state to effect particular behaviors. For example, device-specific methods that may be supported by a DMAP may include device-specific self-test methods or device reset methods.

12.4.2 Object-to-object communication concept

From the user's point of view, AL communication occurs from one object in an application process to another object in an application process. Concepts of polymorphism allow the same communication techniques to be applied to this standard's industry-independent user application objects and industry-dependent objects, as well as to this standard's management objects.

In keeping with this principle, the application model defines both an object model and a communication interaction model (service and protocol). The application model also supports multiple application processes within a device, each of which may contain multiple standard

9 HART, FF-H1, PROFIBUS, and ODVA are the trademarks of various trade organizations. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

objects. This enables this standard to meet specific market needs, such as for process industries or factory automation, as well as to enable support for both single-processor and multi-processor device architectures.

The application object may, for example use the services of the AL to:

- read the value of an attribute of a remote object;
- write the value of an attribute of a remote object;
- request execution of an object-specific method of a remote object;
- report an alert related to a remote object;
- acknowledge an alert reported by a remote object;
- publish data to a remote object by using scheduled communication bandwidth;
- tunnel a non-WISN-native application message to a remote object.

Coding for these services can be found in 12.23.1.4.

12.4.3 AL structure

The AL is divided into two sublayers, the upper AL (UAL) and the application sublayer (ASL), as shown in Figure 16. There is a one-to-one relationship between an ASLDE-SAP and a TLDE-SAP.

The UAL contains the application processes for the device. These processes may be represented as a UAP or as a management process (MP), for example the DMAP or other logical management application such as a security management application.

UAPs may be used, for example, to:

- handle input and/or output hardware;
- distribute communications to a set of co-resident UAPs within a device (proxy function);
- support tunneling of a non-native (e.g., control system legacy) protocol compatible with the network environment of this standard; and/or
- perform a computational function.

A UAP may perform an individual function or any combination of functions. How a UAP accomplishes these functions internally is beyond the scope of this standard. The AL is concerned with application-specific message content, the externally visible behavior of the standard objects contained within the UAP, and the logical interfaces to the ASL that represent UAP communication to and from the lower communication protocol suite of this standard. UAL processes may contain one or more objects that communicate with one another over the network described in this standard, using the standard services provided by the ASL.

NOTE How the ASL implements internal message routing or inter-application communication within a device (within a UAP, across UAPs in a common processor, across UAPs in different physical processors of the same device, or between a UAP and an MP) is a local matter and hence is outside the scope of this standard.

12.4.4 UAP structure

Figure 110 depicts the overall structure of a UAP as defined by this standard.

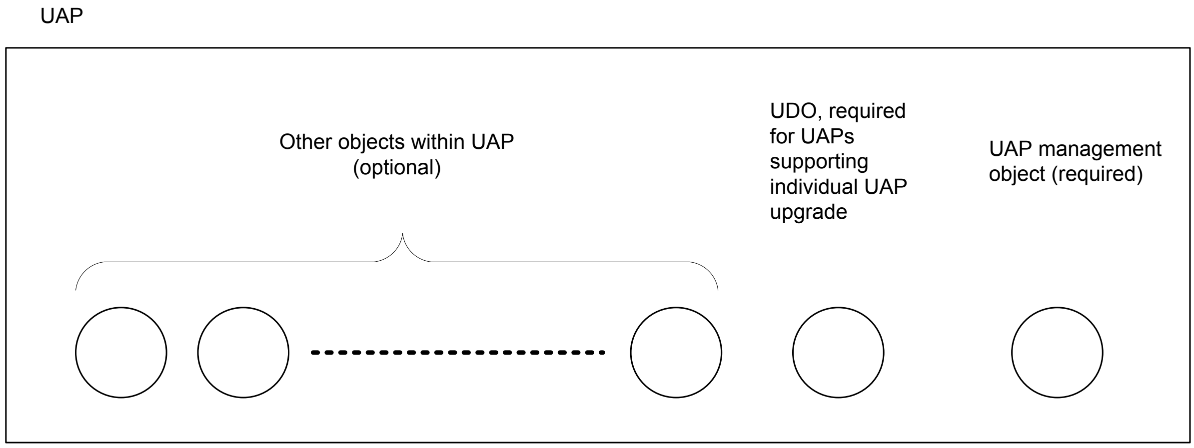


Figure 110 – User application objects in a UAP

Representation of the applications and their functions by standard object definitions allows uniform management and construction of distributed applications. The UAP management object identifies the UAP to the standard-compliant network and allows visibility of and/or control over certain operational aspects of the UAP as a whole. This UAP management object has a reserved object identifier of 1 (one). If a UAP supports individual upgrade, the UAP shall also contain a standard UploadDownload object (UDO) to support UAP upgrade. This UDO has a reserved object identifier of 2 (two). If a UAP does not support individual UAP upgrade, the UAP shall not contain an object instance with an object identifier of 2 (two). Additional objects also may be contained within a UAP in order to provide application-specific functionality to the UAP.

NOTE 1 The UAPMO is sufficient to represent the UAP to the communication system.

NOTE 2 Other objects are statically or dynamically instantiated within the UAP.

NOTE 3 It is outside the scope of this standard to define what happens to the data contained within a UAP when the UAP is upgraded.

The interaction model describes inter-object communication, including message classification and messaging formats. The ASL contains services that support object-oriented communication and routing to the appropriate destination object within a UAP, across the network. This interaction maps the ASL to the services provided by the lower communication protocol suite layer (see Table 281 and Table 282 indicating AL use of TL services and qualities of service). Between the UAL and the ASL is an ASL data entity SAP(ASLDE-n SAP).

ASL-specific management is also locally supported via a separate management SAP.

12.5 Object model

The AL defined by this standard takes advantage of object-oriented modeling concepts to support both native protocol and non-native (legacy) protocol tunneling within applications. Non-native protocol tunneling is achieved by a specialized UAP that includes one or more tunnel objects (TUN) and protocol translation facilities. This UAP consists of exactly one UAP management object to enable uniform system and network management of the UAP, plus one or more TUNs to send/receive encapsulated messages being tunneled to the UAP. Other native objects defined by this standard may also be used within a tunneling UAP. For example, tunneling may be used for wireless device-to-wireless device communication, as well as for wireless device-to-gateway communication.

NOTE 1 Addressing constructs other than these are outside the scope of this standard. Legacy protocol APDU (as opposed to native APDU) constructs are preserved by means of encapsulation when application tunneling as defined by this standard is used.

An object-oriented approach is used to encapsulate data (attributes) and functionality (methods and internal state) for re-use and consistency. Objects are individually addressable using an object identifier that is unique within the application. This unique identifier allows the AL to route messages to the appropriate object destination. Each message is interpreted and acted on by the destination object based on the message context and content. Object operation is described in terms of the network-visible operation of the destination object that is the target of the AL service.

This standard defines standard object identifiers.

NOTE 2 The complement of standard identifiers supported by a device is indicated by the version of this standard that is supported. The supported version is available from the DMAP. See Clause 6 for further details.

An object instance that is accessible within an application process is distinguished from another instance of the same object class in the same application process by its object identifier. Different object instances may also have different attribute values and/or the conditional attributes contained in its set of supported attributes.

As an example, an application process may contain two instances of the AnalogInput object. The object instances within the application process can be distinguished by their object identifier. Each object may support different values for scaling, and also may require a different complement of alarms to be reported. Hence, the instances may have a different complement of analog descriptor attributes supported by each instance.

The objects defined in the UAPs and MPs of this standard adhere to traditional object modeling concepts; specifically, these objects contain attributes, and appropriate object-specific methods, if applicable, are defined. The AL defines standard objects to provide interoperability (within its domain of application) via access to standard attributes and invocation of standard methods.

Standard objects can be classified into usage profiles to meet the needs of particular industries.

Standard management objects are expected to be always available. Application user objects may be statically instantiated, or they may be dynamically instantiated as the result of a download operation. Standard objects are extensible in the following ways:

- Industry-specific standard object types may be added.
- Vendor-specific object types may be added.
- Industry-specific attributes may be added to standard objects.
- Vendor-specific attributes may be added to standard objects.
- Industry-specific methods may be added to objects via industry-specific profiles.
- Vendor-specific methods may be added to objects.
- Industry-specific profiles of object types may be defined, such as a process industry profile, a factory automation industry profile, and other profiles.

12.6 Object attribute model

12.6.1 General

Objects defined by this standard support two kinds of attributes, object key attributes and named attributes.

In this standard, a resource is represented as an object, and identified by an object key attribute, which is a numeric value.

NOTE 1 Support of an object key attribute using an alphanumeric representation of the resource, and of corresponding directory services to locate the resource and to resolve the external alphanumeric name form to an internal numeric form, are a subject of future standardization.

An attribute indicates an accessible element representing a property or characteristic of a resource. In this standard, an attribute is represented by a unique numerical value that uniquely identifies the attribute relative to the containing object. The supported range of the valid values for an attribute identifier is 0..4 095.

This standard defines standard object attributes. Additional standard attributes may be defined in the future.

NOTE 2 The complement of standard attributes supported is established by the version of this standard that is supported. The version of the standard that is supported is available from the DMAP.

Exposing the resource elements as attributes of an object allows the state of the object to be determined and also allows the behavior of the object to be modified. A resource attribute is defined by:

- its influence on object behavior;
- the set of values it can take (as constrained by the object definition containing the attribute);
- the valid tests (for example, valid value set matching) that may be performed on it; and
- the specific set of error conditions that may cause an object-defined failure as a result of performing an attribute-oriented operation.

Attributes themselves do not have accessible properties or subtypes. Attribute values may be explicitly established by external means or by internal means (for example, derived by computation using the values of other attributes).

Attributes shall have a data type that is a standard scalar type defined by this standard. Attributes may have a data type that is a standard data structure defined by this standard. Up to two indices are available to address constructs of standard types defined by this standard. The valid range for an index is 0..32 767.

NOTE 3 For example, access to an individual element of a singly-dimensioned array of standard scalar types is supported. As another example, access to an individual element of a data structure contained in a singly-dimensioned array of such structures is supported.

For an attribute constructed as an array, the array size shall be fixed and all elements of an array shall be homogenously sized. For example, an array of octet strings shall all have the same size octet string for each element in the array. When it is necessary to indicate both the current and maximum dimension of an array, metadata information should be used. This metadata information is described by data type code 406 (Metadata_attribute data structure) which is defined in 6.2.6.3.

12.6.2 Attributes of standard objects

For each standard object, standard attributes are defined. Each standard attribute has a standard attribute identifier that is used to address the attribute.

Standard object attributes may be extended in the following ways:

- Industry-specific attributes may be added to standard objects.
- Vendor-specific attributes may be added to standard objects.

Extensions to standard attributes need to be coordinated to ensure that attribute identifiers remain unique within an object type. The mechanisms used by industries and vendors to extend the attributes of the standard object are outside the scope of this standard.

12.6.3 Attribute classification

Attributes are classified to provide guidance regarding their expected frequency of change. This information is useful, for example, to gateway devices that cache information. The frequency at which attribute values change is characterized as:

- constant;
- static;
- static-volatile;
- dynamic; or
- non-cacheable.

A constant attribute is unchanging throughout time. An example of a constant attribute is the serial number of a wireless device. Default values in this standard are all constant attributes. The values of these attributes shall be preserved when a device undergoes a warm restart / power-fail, when a device resets to factory defaults, or when a reset command to the relevant attribute or management object is received.

Constant information may be either:

- fixed information that never changes, such as information to indicate a manufacturer or serial number; and
- information that does not change during normal system operation, but that may change, for example, when a firmware download occurs.

A static attribute changes its value infrequently. Usually, static data is changed as the result of an external message, request, or event. Some static information may, for example, only be changed by a configuration tool. Operating ranges, units, communication end points, alarms, and constant input values are examples of static information. Attributes storing provisioning information, as well as configuration information provided to a device, are static attributes. The values of these attributes shall be preserved when a device undergoes a warm restart / power-fail or when a reset command to an un-related attribute or management object is received.

A static-volatile attribute changes its value infrequently. Usually, static-volatile data is changed as the result of an external message, request, or event. Some static-volatile information may, for example, only be changed by a configuration tool. The values of these attributes need not be preserved when a device undergoes a warm restart / power-fail. The values of these attributes shall be preserved when a reset command to an un-related attribute or management object is received.

A dynamic attribute may be changed spontaneously by the device containing the object and without external stimulation from the wireless network. Examples of dynamic attributes are frequently changing values such as process variables, calculations, and timers. Dynamic attributes may be treated differently by a gateway cache infrequently changing (static) values; this is entirely up to gateway internal implementation. A dynamic attribute is not required to survive when a device goes through a warm restart / power-fail or when a device resets to factory defaults. It can be reset when a reset command to the relevant attribute or management object is received.

Non-cacheable information is never buffered; for example, it may be used for critical information such as safety-related information (which use may be outside the explicit scope of this standard), and for values that change too often to make caching a valid technique. Whenever the value of a non-cacheable attribute is requested, it shall be retrieved from the end device that owns the object and attribute.

If device local caching is needed, it is the local responsibility of the application.

12.6.4 Attribute accessibility

Network accessible attributes may be:

- accessible to be read only; or
- accessible to be both read and written.

12.7 Method model

Methods represent the set of object type-specific interfaces (functions) that can be used to access an object instance. For example, the UploadDownload object supports a StartUpload method.

The standard object methods shall always be available, and shall not be enhanced beyond the definition given by this standard.

Methods shall not be defined if the equivalent result can be achieved using a standard (object type-independent) AL service, for example, the ASL-provided read service. Definition of a method may be warranted, for example, to replace a sequence of communication transactions in order to save energy. Definition of a method may also be warranted when synchronization issues may result if individual actions are used rather than an atomic transaction set.

Standard object methods may be added in future by this standard.

Time-based triggering of application process activities is not a communication subsystem responsibility. If such time-based triggering is necessary, either a parameter of a method or a dedicated attribute of an object may be employed. If coordination across objects is required, an application object may be defined with an attribute representing the coordinated action, acting as a proxy for the coordination.

12.8 Alert model

The term alert is used to describe an application message that advises or warns the recipient of the presence of an impending or existing situation of interest. The alert model describes alerts reported by application process resident objects and the mechanism to report them.

Two types of alerts are supported, alarms and events. Event is the term used to represent a stateless condition (that is, to indicate a situation has occurred). Events simply report that something happened. An alarm is a stateful condition of an existing situation, for example, that an alarm has transitioned to an abnormal state, or has returned to normal from an abnormal state. The alarm condition remains true until the alarm condition clears. Alert is the term used to describe the messaging of an event condition or alarm condition. Both alarms and events are reported through the alert reporting mechanism defined in this standard.

An alarm is characterized by a state, and alerts are used to report:

- the occurrence of a condition; and
- the return to normal of the previously reported condition.

Events and alarms supported by this standard fall into one of the following categories:

– device-related;
– communication-related;
– security-related; or
– control process-related.

Each alarm and event defined for an object shall have an associated attribute that describes how it is reported. This associated attribute shall include:

- whether it is enabled or disabled for reporting; and
- its priority (importance).

For all alarms, descriptive information shall also include, if the alert is an alarm, whether or not the condition is in or out of alarm.

An analog alarm occurs when a value meets or exceeds an established limit. For analog value alarms, descriptive information shall also include limit information, if any, relating to when the alarm condition is triggered.

12.9 Alarm state model

Table 238 and Figure 111 represent the alarm state model.

Table 238 – State table for alarm transitions

Transition	Current State	Event(s)	Action(s)	Next state
T1	Clear	Alarm is detected	Report alarm to ARMO in DMAP	In alarm
T2	In alarm	Clear is detected	Report alarm clear to ARMO in DMAP	Clear
T3	In alarm	Recovery requested	Report alarm to ARMO in DMAP	In alarm
T4	Clear	Recovery requested	Ignore	Clear
NOTE Recovery is usually requested by a remote device.				

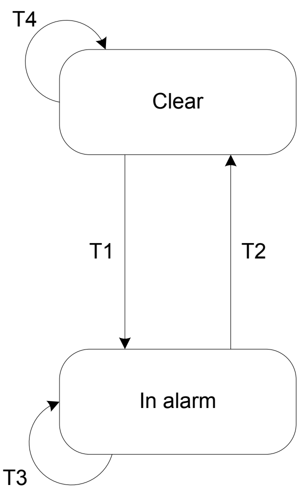


Figure 111 – Alarm state model

Alarm detection applies both to analog and discrete values. Examples of analog alarms include:

- analog limit alarms (for example, when a value exceeds a high or low threshold);
- analog deviation alarms (for example, the difference between a process variable and set point);
- a Boolean alarm (for example, when the state of the Boolean matches the discrete limit parameter); and
- diagnostics, such as those defined in the NAMUR 107 recommendation.

NOTE 1 Alarms that depend upon evaluation of a combination of device-, inter-object-, or intra-object-specific state conditions are considered a local matter and are thus outside the scope of this standard.

NOTE 2 Different levels of alarm conditions are indicated by different alarms. For example, for an analog input, a High alarm represents one level, and a High-High alarm represents a higher level.

12.10 Event state model

12.10.1 General

The state model for an event is a subset of the state model for an alarm.

12.10.2 State table and transitions

Table 239 and Figure 112 represent the event state table and transitions, respectively.

Table 239 – State table for event transitions

Transition	Current state	Event(s)	Action(s)	Next state
T1	Clear	Event condition is detected	Determine report characteristics (e.g., priority)	Detected
T2	Detected	Event condition is reported	Report event to ARMO in DMAP	Submitted
T3	Submitted	Event condition submission to ARMO as completed	Reset to prepare for next event report	Clear

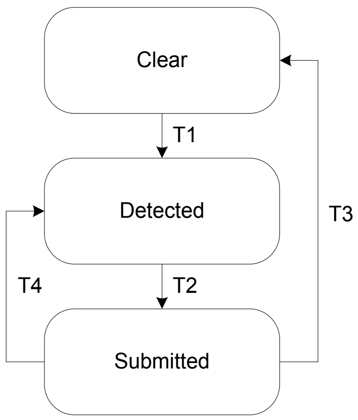


Figure 112 – Event model

12.11 Alert reporting

12.11.1 General

Alerts are reported promptly and accurately time-stamped using a queued unidirectional alert report communication. Queued unidirectional alert reporting involves the alert detecting device reporting the condition using a source/sink communication flow. A queued unidirectional alert acknowledgment is received in return.

NOTE 1 In a published message, status information sometimes indicates that an alert is available in the reporting device, accessible via a client/server read service. This method is sometimes used for factory automation; other factory automation systems publish a tag to a server that generates alarms by testing limit values in the server.

NOTE 2 Source/sink communication is used rather than client/server in anticipation of a future release of this standard that supports multicast alert reports.

12.11.2 Alert types

The alert reporting management object (ARMO) contained in the DMAP provides encapsulation of the alert report, handles timeouts and retries, and throttles alert reporting in a common manner for all the applications contained within the device.

There shall be only one ARMO in each device, in the DMAP. As described in 6.2.7.2.3, the DMAP may provide limited access to entities other than the SMAP in order to support services related to process-related alerts and device-related alerts. Alert acknowledgments shall be addressed to the ARMO.

Diagnostic alerts are specific to the device reporting them. For example, diagnostic alerts may indicate that:

- an error has occurred;
- a symptom has been detected that may indicate that an undiagnosed error occurred;
- a symptom has been detected that may indicate that an error may occur in the future;
- an error will occur if preventative action is not taken.

Diagnostic alerts may pertain to a device as a whole, to an individual component, or to a defined set of components of a device. Diagnostic alerts may be stateless or state-oriented. Diagnostic alerts may be specified by this standard, such as for communication-related alarms, or may be vendor-specific. Diagnostic alerts provide information that can later be examined to establish device and/or communication system behavior patterns.

Process alerts are specific to the process being controlled by the device reporting the process alarm. A process alarm indicates a situation in which the alarmed variable has exceeded established operational limits. For example, a process alert may be generated when a measurable control condition occurs that is outside of desired control system operation parameters.

Process alerts provide information that can later be examined to establish control system behavior patterns, such as:

– alerts that often occur in a particular sequence;
– alerts that often occur close in time;
– alerts that were active for significant periods of time;
– actions that are required to resolve an alert situation;
– assistance in determining optimal trip point and hysteresis settings; or
– information regarding control system performance in terms of alert prevention and resolution.

Process alerts pertain to a particular control object and attribute value of that object (e.g., the PV of an analog input object). Process alerts are usually state-oriented (i.e., alarms).

One octet is used in the coding of alert type information. For all objects, three standard ranges are identified for disjoint definitions of an object-specific alert type:

00..49: reserved for and defined by this standard;

51..100: reserved for future standard industry profiles;

101..255: vendor-assignable for vendor-specific alerts.

12.11.3 Alert report information

The APDU header indicates the application and object that initiated the communication. For alert reports, this would indicate the DMAP and the ARMO. The individual alert report information therefore also shall identify the application process and the object within it that is the detecting source of the alert. Additionally, alert reports shall include the following information:

- network time of detection;
- individual alert identifier (so that duplicates may be detected by the UAL process);

- alert class (alarm or event);
- alarm direction (transition into alarm, or not (i.e., either return-to-normal or an event));
- alert category (device diagnostic, communication-related, security-related, or process-related);
- alert priority (ranges are defined for high, medium, low, and journal-only alert priorities);
- alert type (object-specific, and within the specific alert category);

  NOTE See 6.2.7.2 for further information on communication alerts and 6.1.2 for further information on security alerts.

- associated-data size, in octets; and
- associated data for the alert condition.

The associated data for device diagnostics should be defined for compatibility with NAMUR Recommendation NE107; such diagnostics should indicate whether the device condition is abnormal, and if so its NAMUR class: failure, off-specification, diagnostic maintenance, or diagnostic check function.

12.11.4 Alarm state recovery

If a loss of communication with a wireless device occurs, process industries require that existing conditions be reliably recoverable by an alert receiving device, such as by determining the state when an alert receiving device starts up.

NOTE 1 It is possible for multiple alarm conditions to exist simultaneously within a process control device.

Recovery of alarm state may be requested from the ARMO. A single alarm recovery request triggers the re-reporting of all existing alarm conditions in the device of a given category. When recovering alarms, the alarm reporting device shall provide alerts to indicate when the recovery is commencing and when the recovery has completed.

NOTE 2 As event conditions are stateless, they are not recoverable.

12.12 Communication interaction model

12.12.1 General

Native communication in this standard supports both native protocol and encapsulation of legacy protocols via tunneling. The following types of communication flows are anticipated for compliant devices:

- queued unidirectional communication (e.g., alarm reporting or alarm acknowledgment);
- queued bidirectional communication (e.g., read, write, method invocation); and
- buffered unidirectional publication communication (e.g., publish).

The actual location of the buffers used to hold the data for scheduled unidirectional publication communication is a device local matter.

NOTE Buffered scheduled data publication (periodic, change of state, and application driven publication) all occur (as needed) within the scheduled phase. Communication contracts for periodic communication employ buffered unidirectional publication communication. Communication contracts for aperiodic communication employ a queued communication paradigm.

12.12.2 Buffered unidirectional publication communication

12.12.2.1 General

Buffered unidirectional communication is used when a publishing application is sending a message to a subscribing application. The buffer contains the message to be sent. On each buffered unidirectional publication contract, there is a parameter to indicate if the buffer is to always be transmitted (whether the content has been updated or not), or if the buffer is only to be transmitted if it has been updated since the prior transmission.

12.12.2.2 Buffer content always transmitted

In buffered unidirectional communication, if a publishing communication protocol suite receives another ASL publish service request for a particular communication contract before the previous message has been transmitted, the new request replaces the previous request. In the subscriber, if a new message is received before the previous one has been delivered to the application, the new message shall replace the previous undelivered message.

NOTE 1 In establishing a contract for periodic communication, the system manager ensures that there is adequate capacity within the intermediate devices along a route to support the periodic communication.

NOTE 2 It is anticipated that an application that receives an older publication after a newer one is able to choose to discard the older publication.

If a publishing communication protocol suite does not receive a new service request for the contract when it is time to transmit, the previous request shall be retransmitted. If the subscriber receives the same application service request in succession, the subscribing application shall treat this situation as receipt of a duplicate message. Application handling of a duplicate buffered message is left to the application, and is not defined by this standard.

12.12.2.3 Buffer content transmitted on change only

This mode of buffered communication supports a change of state communication mechanism.

In buffered unidirectional communication, if a publishing communication protocol suite receives another service request for a particular communication contract before the previous message has been transmitted, the new request replaces the previous request. In the subscriber, if a new message is received before the previous one has been delivered to the application, the new message shall replace the previous undelivered message.

NOTE In establishing a contract for periodic communication, the system manager ensures that there is adequate capacity within the intermediate devices along a route to support the periodic communication.

If a publishing communication protocol suite does not receive a new service request for the contract when it is time to transmit, the previous request shall not be retransmitted. If the subscriber receives the same application service request in succession, the subscribing application shall treat this situation as an error situation. Application handling of a duplicate buffered message is left to the application, and is not defined by this standard.

12.12.3 Queued unidirectional communication

Queued unidirectional communication supports queued distribution of unconfirmed (unidirectional) ASL communication services. To satisfy this type of communication need, the lower layers of the correspondents are expected to provide a queued data transfer service.

Application handling of a duplicate AlertReport shall result in sending another AlertAcknowledgment. Receipt of a duplicate AlertAcknowledgment shall be ignored. Application handling of duplicate queued unidirectional Tunnel messages is left to the application, and is not defined by this standard.

12.12.4 Queued bidirectional communication

12.12.4.1 General

Queued bidirectional communication supports queued distribution of confirmed (bidirectional) ASL communication services. To satisfy this type of communication need, lower layers of the correspondents are expected to provide a queued data transfer service.

This maximum number of simultaneously outstanding queued bidirectional (client/server) confirmed service requests permitted for a contract is indicated to the application process by the communication contract when the contract is granted. The default value for this maximum value shall be 1, i.e., the default indicates that contract supports only one outstanding request at any given time.

Application handling of a duplicate request is to send another response. Application handling of a duplicate response when a response is received that does not match a pending request identifier shall result in ignoring the response.

12.12.4.2 Retries and flow control

12.12.4.2.1 General

The AL defined by this standard is required to track what happens to the queued service client/server requests that it sends. This is necessary for two reasons, to ensure reliability of delivery and for flow control.

For delivery reliability, the application needs to be able to determine when it should re-send (retry) its message. There are two situations in which message retry may be necessary, first when the message request did not arrive at its final destination, and second when the message request arrived, but the application response did not make it back to the original requestor. Flow control is necessary to ensure that the destination device is not overwhelmed with messages it cannot handle, as well as to protect the network and optimize network throughput.

This standard supports both forward explicit congestion notification by the lower communication protocol suite and an application level echo back to a client of a four-part service requestor if congestion occurred on the path taken for the original service request.

To enable multiple outstanding requests simultaneously while still allowing an application to achieve both reliable delivery and flow control, each client request shall contain a unique identifier. Retransmission of a request (retries) shall use the same identifier. This identifier enables the application to implement a sliding window technique to control flow. The client shall start a service related time-out timer when it initiates a client service request. This timer shall be based on round trip times (RTT) for messages, and shall allow sufficient time for a message from an application in device X to reach the destination application in device Y, for the server application in device Y to issue a response, and for the response to travel back to the service requesting application in device X.

NOTE This method is commonly known in communications as communication using positive acknowledgment with retransmission.

Figure 113 represents an example of three simultaneously outstanding write request messages, with a single concatenated message that contains the responses for all the outstanding write requests. Concatenation is used in order to save messaging traffic.

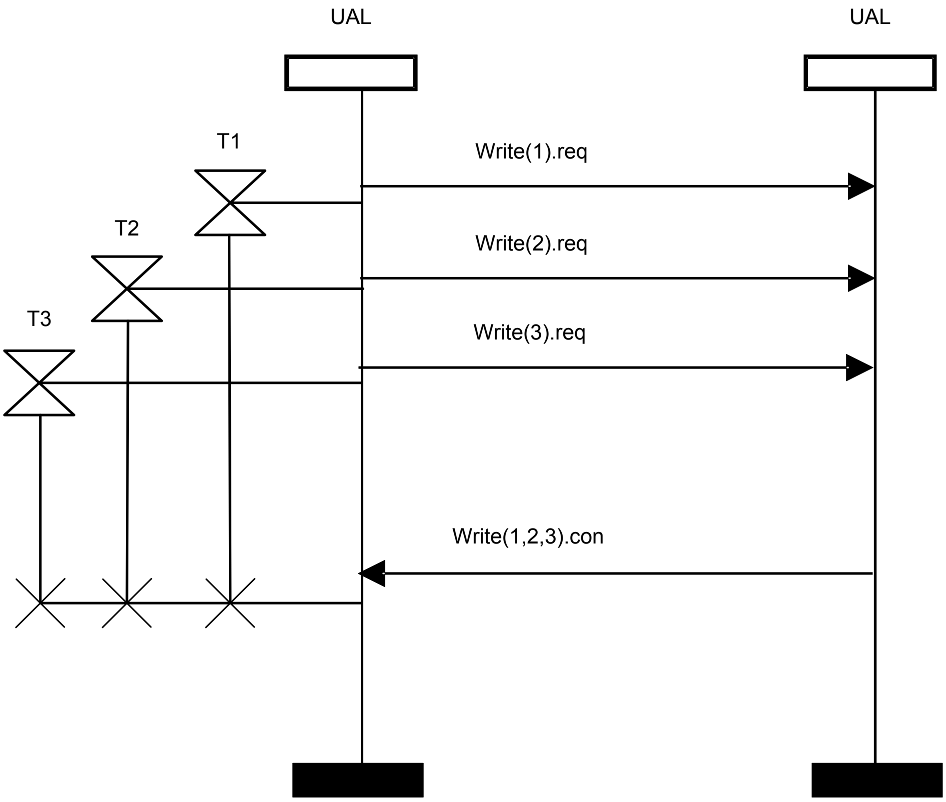


Figure 113 – A successful example of multiple outstanding requests, with response concatenation

12.12.4.2.2 Retries and timeout intervals

12.12.4.2.2.1 General

The method defined by IETF RFC 6298 shall be used for calculating an appropriate value for the retry timer-out interval (RTO). To compute the current RTO, a client shall maintain two state variables, $SRTT$ (smoothed round-trip time) and $RTTV$ (round-trip time variation), within the scope of a contract.

Until a round-trip time (RTT) measurement has been made for a segment sent between the client and server, the client should set RTO equal to 3 s.

When the first RTT measurement R is made, the client shall set:

$SRTT = R$

$RTTV = R/2$

$RTO = SRTT + 4 \times RTTV$

When a subsequent RTT is available, R is made. The client shall update the $RTTV$, $SRTT$, and RTO using the following calculations, where the recommended value for β is 0,25, and the recommended value for α is 0,125:

$RTTV = (1 - \beta) \times RTTV + \beta \times |SRTT - R|$

$SRTT = (1 - \alpha) \times SRTT + \alpha \times R$

$RTO = SRTT + 4 \times RTTV$

Whenever RTO is computed, the RTO shall be rounded based on the following rules:

If $RTO < 1$ s, set RTO to 1 s.

If RTO > 60 s, set RTO to 60 s.

Determination of timeout occurrence is a local matter. When a timeout has been determined to have occurred, exponential backoff shall be employed for consecutive timeouts by setting $RTO = RTO \times 2$ to send the retries. The maximum value of 60 s should be used to provide an upper bound to this doubling operation. Retries cease either when a response is received, or when the maximum retry limit is reached. The maximum retry permitted for a client request is indicated via an attribute of the UAP management object. The value selected for the maximum retry permitted is a local matter.

NOTE IETF RFC 6298 contains a recommendation regarding management of the TimeoutInterval timer.

12.12.4.2.2.2 Retries for unordered messages

Unordered messages are independent in that the order of responses may be received in a different order than the order in which the requests were sent. Accordingly, each request message times out and is retried independently.

Figure 114 is an example of how a timeout and retry of the second message in a sequence of three unordered messages, due to failure of the request to reach the server, is handled.

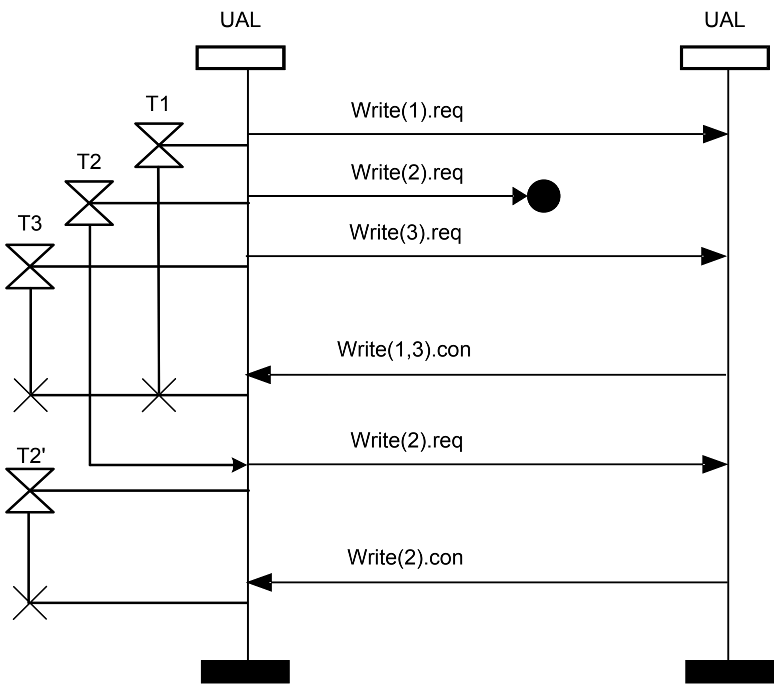


Figure 114 – An example of multiple outstanding unordered requests, with second write request initially unsuccessful

12.12.4.2.2.3 Retries for ordered messages

Ordered messages are order-dependent; that is, the order of responses may not be received in a different order than the order in which the requests were sent. Accordingly, if a later message receives a response before an earlier message, it indicates that the message for which no response was received shall be timed-out and retried.

Figure 115 is an example of how a timeout and retry of the second message in a sequence of three ordered messages, due to failure of the request to reach the server, is handled.

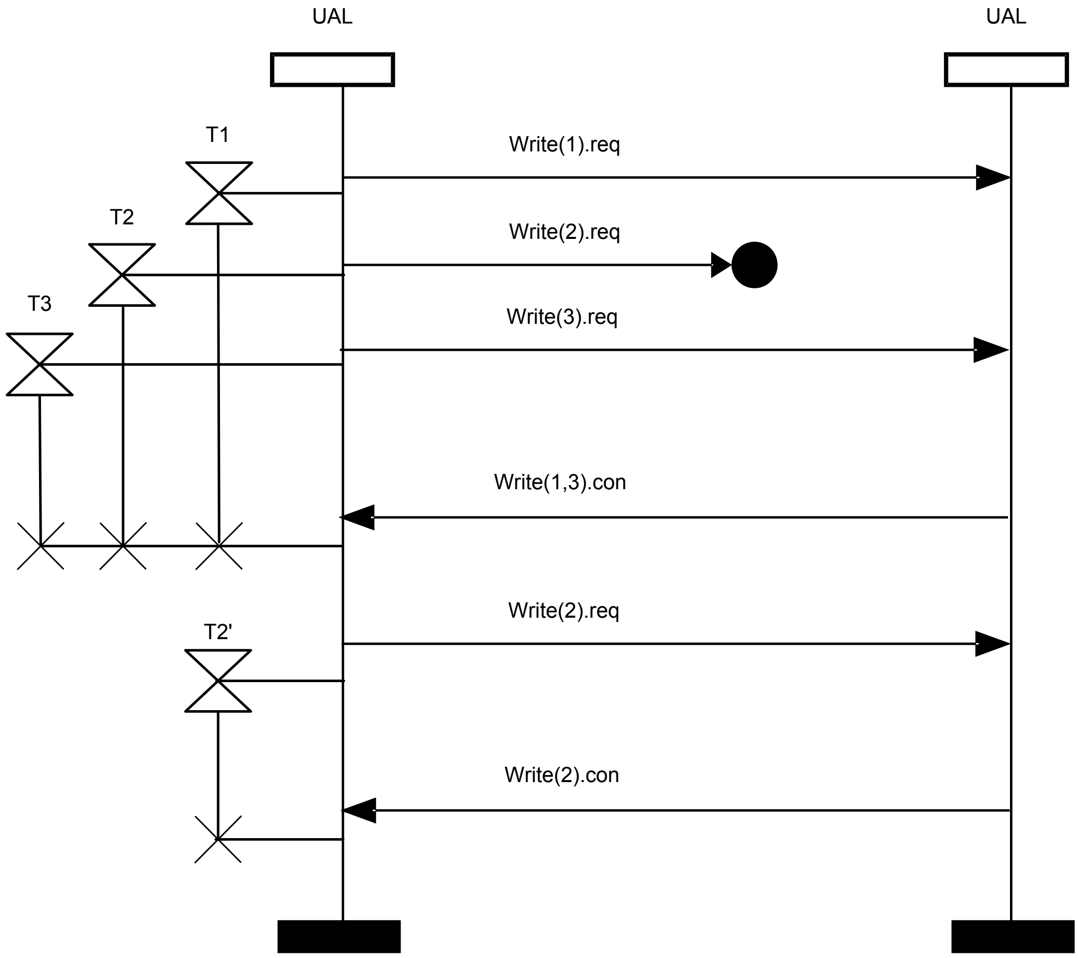


Figure 115 – An example of multiple outstanding ordered requests, with second write request initially unsuccessful

Ordered delivery only pertains to upload/download. The Max_Send_Window_Size for upload/download communication contracts shall be fixed at 1. As such, ordered message delivery is not supported by the lower layers of the protocol suite defined by this standard.

12.12.4.2.3 Flow control

Client/server communications are not application process rate controlled; rather, AL flow rate fairness is enforced by the application processes on a per-contract basis, in order to minimize the cost fairness of congestion on the network.

Max_Send_Window_Size (Table 26) for a contract is the maximum number of client requests that may be simultaneously awaiting a response within the scope of a contract. It is recommended (but not required) that clients use sequentially contiguous request identifiers. The value of Max_Send_Window_Size is established on a contract basis by the system manager. The OutstandingList represents the messages that have been sent, and that are currently awaiting a response.

The AvailableSendWindowSize represents the usable send window, that is, the set of client requests that may be sent, without violating the Max_Send_Window_Size, taking into consideration the number of messages contained in the OutstandingList. When the windows are empty, and CurrentSendWindowSize equals the Max_Send_Window_Size, the usable send window stretches from the last acknowledged client request for the next Max_Send_Window_Size number of requests, and represents the set of client requests that may be sent, without violating the Max_Send_Window_Size.

Applications shall initially set their value for their CurrentSendWindowSize to be one (1). RTT is then measured by the application for *n* complete transactions. If no timeout occurs during these transactions, and if the Max_Send_Window_Size has not been reached, the CurrentSendWindowSize shall be incremented by one (1). The value of CurrentSendWindowSize shall be equal the size of the CurrentSentWindowLimit +1.

NOTE This mechanism for increasing the window size is more conservative than that usually used to meet TCP congestion avoidance requirements

As a response is received, the corresponding request moves from the sent window to the response completed window, and the size of AvailableSendWindowSize increases by one if the Max_Send_Window_Size has not been reached. If a timeout occurs awaiting a response, message loss or congestion is indicated. If this occurs, then:

- No additional message shall be placed into the OutstandingList until after the OutstandingList has first become empty.
- The CurrentSendWindowLimit shall be set to one (1).
- The messages that were in the OutstandingList at time of collapse shall be retried in order, according to the retry policies defined above. Retries use exponential backoff if the first retry does not succeed. Retries shall continue until either a response is received or the maximum number of retries has been met. When either of these conditions occurs, the message handling is considered complete, and the message shall be removed from the OutstandingList.

Client requests may continue again, building up the CurrentSendWindowLimit to the Max_Send_Window_Size value using the procedure described above.

EXAMPLE In an example of how the windows are used in a situation when there are no retries:

Let Max_Send_Window_Size be a constant, equal to the maximum number of simultaneously outstanding requests permitted by the contract. This limit is established by the system manager.

Let CurrentSendWindowSize be the variable that represents the number of simultaneously outstanding requests that exist for the contract at point in time *t*. CurrentSendWindowSize is a non-negative integer less than or equal to Max_Send_Window_Size.

Let UsedSendWindowSize be the variable that represents the number of simultaneously outstanding requests that are still awaiting responses.

AvailableSendWindowSize = CurrentSendWindowSize – UsedSendWindowSize.

Assume: Max_Send_Window_Size = 3

T1: Initialization: CurrentSendWindowSize = 1; UsedSendWindowSize = 0; AvailableSendWindowSize = 1

T2: Message M_1 sent: CurrentSendWindowSize = 1; UsedSendWindowSize = 1; AvailableSendWindowSize = 0

T3: Message M_1 response received: CurrentSendWindowSize = 1; UsedSendWindowSize = 0; AvailableSendWindowSize = 1

T4: Message M_2 sent: CurrentSendWindowSize = 1; UsedSendWindowSize = 1; AvailableSendWindowSize = 0

T5: Message M_3 response received: Current SendWindowSize = 2; UsedSendWindowSize = 0; AvailableSendWindowSize = 2.

The CurrentSendWindowSize has been incremented by one, since:

a) it has been at size 1, and 2 transactions have completed successfully

b) CurrentSendWindowSize < Max_Send_Window_Size.

T6: Message M_4 sent: CurrentSendWindowSize = 2; UsedSendWindowSize = 1; AvailableSendWindowSize = 1

T7: Message M_5 sent: CurrentSendWindowSize = 2; UsedSendWindowSize = 2; AvailableSendWindowSize = 0

T8 Message M_4 and M_5 responses received : CurrentSendWindowSize = 2; UsedSendWindowSize = 0; AvailableSendWindowSize = 2

T9: Message M_6 sent: CurrentSendWindowSize = 2; UsedSendWindowSize = 1; AvailableSendWindowSize = 1

T10: Message M_7 sent: Current SendWindowSize = 2; UsedSendWindowSize = 2; AvailableSendWindowSize = 0

T11: Message M_6 response received : Current Send Window Size = 3; UsedSendWindowSize = 1; AvailableSendWindowSize = 2

The CurrentSendWindowSize has been incremented by one, since:

c) it has been at size 2, and 3 transactions have completed successfully.

d) CurrentSendWindowSize < Max_Send_Window_Size.

T12: Message M_8 sent: CurrentSend Window Size = 3; UsedSendWindowSize = 2; Available SendWindowSize = 1

T13: Message M_9 sent: CurrentSendWindowSize = 3; UsedSendWindowSize = 3; AvailableSendWindowSize = 0

T14: Messages M_7 response received: CurrentSendWindowSize = 3; UsedSendWindowSize = 2; AvailableSendWindowSize = 1

T15: Message M_{10} sent: CurrentSendWindowSize = 3; UsedSendWindowSize = 3; AvailableSendWindowSize = 0

Figure 116 depicts a situation wherein the current send window has not yet built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, one message in the outstanding list has been sent and is awaiting a response, and one message may be sent before the usable send window limit is reached.

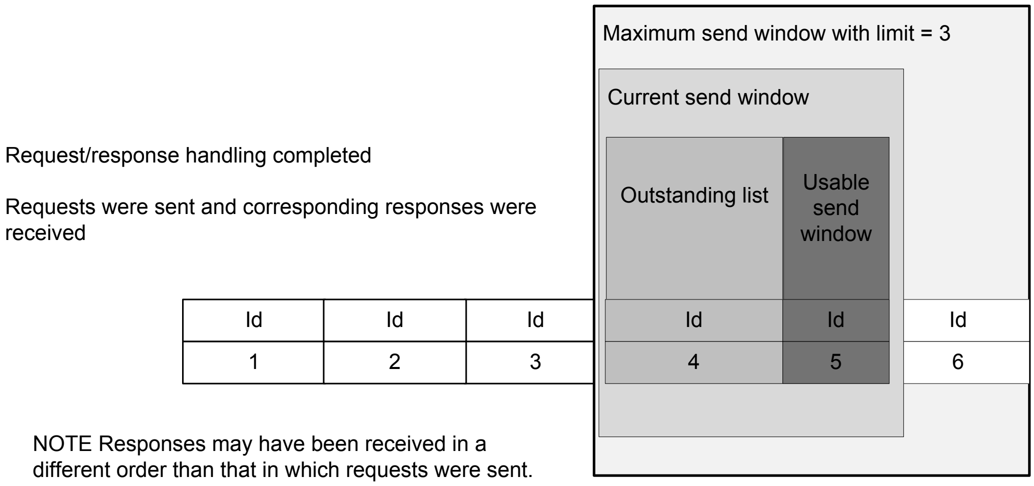


Figure 116 – Send window example 1, with current send window smaller than maximum send window

Figure 117 depicts a situation wherein the current send window has built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, one message in the outstanding list has been sent and is awaiting a response, and two messages may be sent before the usable send window limit is reached.

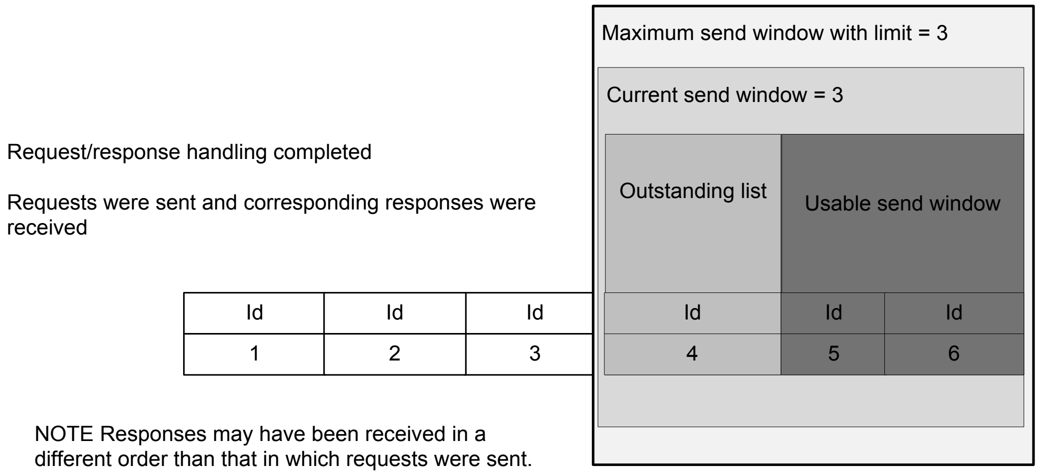


Figure 117 – Send window example 2, with current send window the same size as maximum send window, and non-zero usable send window width

Figure 118 depicts a situation wherein the current send window has built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, and three messages have been sent and are awaiting responses.

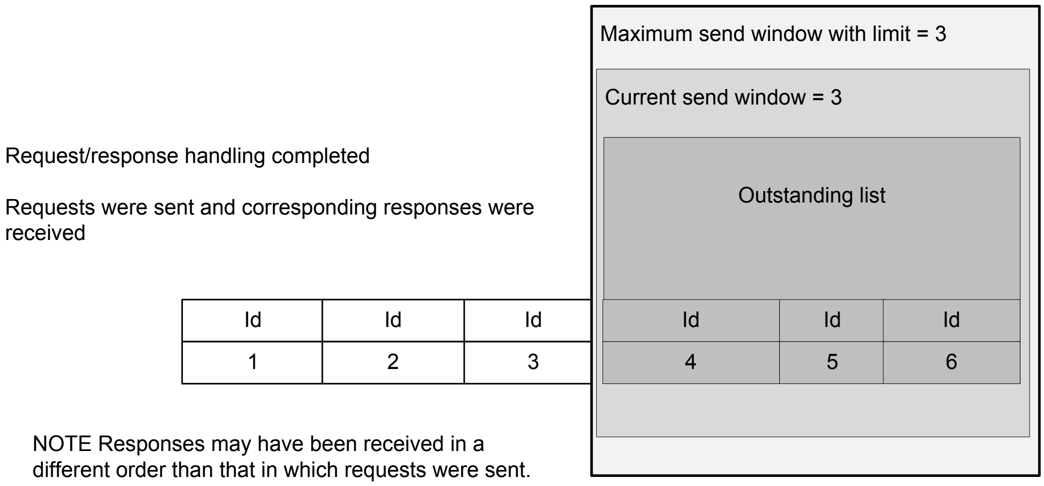


Figure 118 – Send window example 3, with current send window the same size as maximum send window, and usable send window width of zero

12.12.4.2.4 Probing for congestion

Some system configurations are more likely than others to incur message loss due to network congestion. In system configurations where congestion is more likely, an application may wish to regulate its AL service requests based on whether or not network congestion is present. To do this, an application may probe for congestion. To effect such a probe, the application may engage in a simple single message exchange.

NOTE 1 Probing is intended for diagnostic purposes only. A single message is used to ensure that the probes do not overload the network, and to ensure that the response to a probe is distinguishable.

The message request to use when probing shall be a non-concatenated read service. The read request and corresponding read response for the probe shall each fit within a single DL fragment. Any object attribute may be used as a probe; however, it is recommended (but not required) that the same object and attribute be used consistently for probing. For example, a standard attribute of the UAPMO may be used to probe UAPs, and a standard attribute of the

DMAP DMO for probing the DMAP since those objects are required to be present in the corresponding applications.

If the probe timeout does not expire prior to reception of the response, then the application should assume that there is no congestion. However, if the response does not return before the retry timeout interval passes, this indicates a higher probability that network congestion is present. In this situation, the application process shall self-regulate its communication activities by setting its CurrentWindowSize to 1. See 12.12.4.2.3. If the application desires to send another congestion probe message, it may do so using exponential backoff as described in 12.12.4.2.2.1, but shall use a temporally-distinguishable request identifier for each message probe.

NOTE 2 Such distinction makes it possible for an application to compute RTT specific to congestion probing, and to make congestion decisions accordingly based on the congesting probing RTT data.

For example, a UAP in device X may issue a read service request for the required standard state attribute of the required UAPMO contained within the destination application in device Y. This read request may be treated as an application process-initiated congestion probe.

In the specific case of a download or upload, an application may probe for congestion. In these situations, congesting probing shall be performed as follows:

- To probe prior to commencing a download operation, the client probe shall be a read request to the UploadDownload attribute MaxDownloadSize.
- To probe for congestion during a download operation, the client probe shall be a read service request for the UploadDownload objects LastBlockDownloaded attribute.
- To probe prior to commencing an upload operation, the client probe shall be a read request to the UploadDownload attribute MaxUploadSize.
- To probe for congestion during an upload operation, the client probe shall be a read service request to the UploadDownload objects LastBlockUploaded attribute.

12.12.5 Communication service contract

A UAP makes a contact request to the local DMAP via an UAPME-n SAP in order to establish an agreement for a communication service needed by the UAP. If the need can be met, the DMAP provides a service contract identifier to the UAP that represents the agreement. The contract identifier is passed from the UAP to the ASL when it makes an ASL service request. This service contract ID is then used by the lower communication protocol suite to identify the layer-specific characteristics of the contract that have been established into the lower communication protocol suite layers by the local DMAP as part of establishing the service contract. The communication of information required from the UAP to the DMAP in order to acquire a service contract identifier is a device-internal matter, and hence not specified by this standard.

All communication contracts have a base set of information. Additional required information depends on the type of communication relationships desired. For example, a publish/subscribe relationship for periodic communication requires specification of the desired phase and period.

This standard does not specify how to determine the information needed by the UAP to specify the characteristics of a contract. For example, such information may be configured, such as the periodicity and phase to use in scheduled communication, or such information may be determined by the vendor of the device that contains the UAP.

Contract requests may be negotiated down by the system manager. UAP policies regarding the handling of negotiated down contracts, as well as policies regarding the handling of a declined contract, are outside the scope of this standard. See 6.3.11 for further information about the information that needs to be specified to request a contract.

The publishing period is represented by a signed 16-bit integer value. A positive value indicates a publication period as a multiple of 1 s (e.g., a value of 5 indicates a publishing period of 5 s; a value of 3 600 indicates a publishing period of 1 hr). A negative value indicates publication on a fraction of 1 s (e.g., -4 indicates publish every ¼ s, -2 indicates publish every ½ s). A zero value indicates that no publishing should occur.

The periodicity selected should be based on the efficiency of the operation with this standard and the typical process practice.

DMAP knowledge of the destination TSAP port is not a requirement for creating a service contract, as contract establishment is concerned with resources for communication over the network conditions, whereas the destination port is used within the destination device, after the over-the-network communication has occurred.

NOTE Policies to retry establishment of a contract in the event of failure of contract establishment or revocation of a contract are behaviors of the device as a whole (as opposed to behaviors of a component within the device). Device-level behaviors are discussed in 6.3.11.2.4.2.

12.13 AL addressing

12.13.1 General

Certain information is required to address an object, an object's attribute, an element of an object's attribute (e.g., an element of a structure or an element of an array), or an object's method in native communications. Figure 119 represents the general addressing model for UAL process objects.

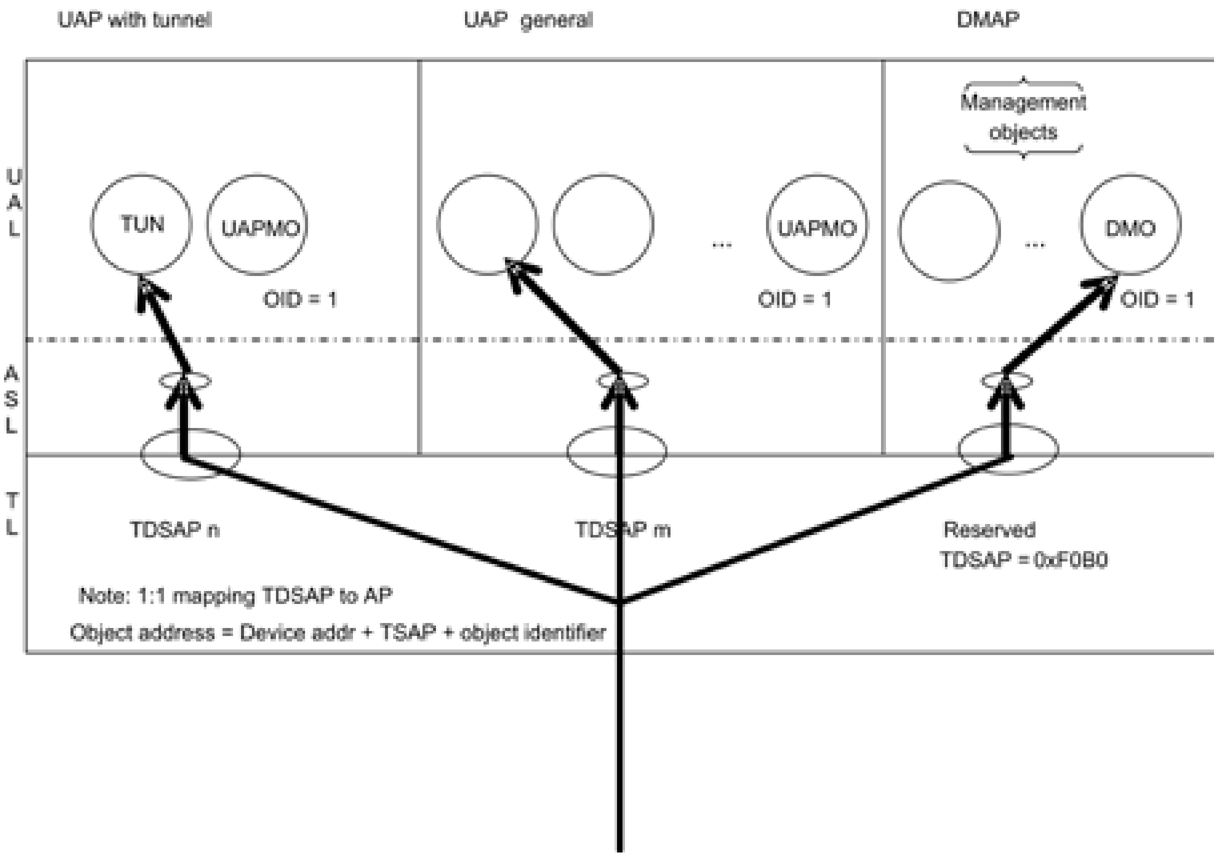


Figure 119 – General addressing model

12.13.2 Object addressing

An object is addressed in unicast communications by specifying:

- its containing device physical address;
- the TL TDSAP is used to communicate with the unique UAL process that contains the object (that is, the TDSAP maps 1:1 to an application process);
- a T-port number corresponds to a particular TDSAP; and
- the unique object identifier within the UAL process.

T-ports shall be assigned in consecutive order to TDSAPs, starting from the first available T-port. For example, TDSAP number 0 shall be correlated with the first T-port 0xF0B0. TDSAP number 1 shall be correlated with the second port, 0xF0B1, and so forth.

Particular TDSAPs, and their corresponding T-ports, are reserved by this standard so that they are well-known to all applications. Specifically, the DMAP in every device shall have the reserved transport port number 0xF0B0, which is associated with TDSAP number 0. The SMAP in a device shall have the reserved transport port number 0xF0B1, which is associated with TDSAP number 1. Devices that do not have an SMAP present shall not use T-port number 0xF0B1.

It is recommended that UAPs that are anticipated to have a large amount of messaging use the T-ports numbered 0xF0B2 through 0xF0BF, as they are represented in compressed form over the network, thus minimizing use of network resources, as well as RF congestion and device energy demand. See 11.4.4 for further details on TDSAPs and T-ports.

In order to minimize the encoding of application messages, it is recommended that object identifiers be allocated consecutively, starting at 1.

NOTE Object identifier 0 is reserved by this standard for the use of the application process management object contained within all application processes.

Multicast communication is not supported.

12.13.3 Object attribute addressing

An attribute of an object is addressed by specifying:

- the addressing of its containing object; and
- the unique attribute identifier within the object.

In order to minimize the encoding of application message attributes, it is recommended that attributes be allocated consecutively, starting at 1.

NOTE Attribute identifier 0 is reserved by this standard as a means of referring to an aggregate as a whole.

12.13.4 Object attribute addressing

12.13.4.1 General

Addressing for attributes is defined based on the type of attribute. This standard supports the following attribute types:

a) standard scalar types defined by this standard;
b) 1-origin singly-dimensioned homogeneous or heterogeneous arrays of elements of type a);
c) [1,1]-origin doubly-dimensioned arrays, where the first dimension indexes a homogeneous array of elements of type b).

Standard data structures defined by this standard are modeled and accessed as 1-origin singly-dimensioned heterogeneous arrays of elements. Thus access to the *k*'th member of a data structure, as enumerated in declaration order of its member elements, is provided by accessing it's *k*'th element as if the data structure were a 1-origin heterogeneous array.

NOTE In programmatic terms, this means that access to the k'th member of structure *s*, which programmatically might be referenced as $s.\text{memberName}_k$, is accessed as if it were *s*[*k*], where *k* is the 1-origin ordinal index of the member in the containing declaraion.

Elements of a doubly-dimensioned array that are themselves structures or arrays are accessible only by representing those individual elements as octet strings of uniform size.

12.13.4.2 Scalars

This standard supports access to attributes that are scalars of the following types:

- Boolean, mapped to Boolean8, or to Boolean1 when in a packed data structure;
- Integer, mapped to Integer8, Integer16, Integer32, Unsigned8, Unsigned16, Unsigned32, Unsigned64, Unsigned128, or to UnsignedN where N < 16 when in a packed data structure;
- Float, mapped to Float32 or Float64;
- VisibleString, mapped to VisibleStringN when N is fixed or determined by context;
- OctetString, mapped to OctetStringN when N is fixed or determined by context;
- BitString, mapped to BitStringN when N is fixed or determined by context;
- SymmetricKey, mapped to OctetString16 in this edition of this standard.

NOTE 1 Each BitString is represented as an integral number of octets, or as an appropriate number of adjacent bits when in a packed data structure;

NOTE 2 See 12.22.3 on data types for the scalar types supported by this standard.

NOTE 3 OctetString and BitString provide a means for transparent conveyance of information that is unintelligible to the conveying protocol layer.

12.13.4.3 Structured protocol addresses treated as scalars

The following are also considered scalars when used in the data structures of this standard, even though their own defining standards specify a substructure for the item:

- IPv6Address, mapped to Unsigned128 to support simple numeric comparison;

  NOTE 1 The substructure of this class of address is specified in IETF RFC 2460 and its related standards.

- EUI64Address, mapped to Unsigned64 to support simple numeric comparison;

  NOTE 2 The substructure of this class of address is specified by the IEEE's Guidelines for 64-bit Global Identifier (EUI-64™)

- DL16Address, mapped to Unsigned16 to support simple numeric comparison.

  In IEEE 802.15.4:2011, the value 0xFFFF is the broadcast DL16Address, while any value in the range 0x0000..0x7FFD may be assigned to a DLE as a unicast DL16Address. However, this standard reserves the value 0 to indicate an unassigned DL16Address, so for this standard the range of unicast DL16Addresses is 0x0001..0x7FFF.

  NOTE 3 IEEE 802.15.4:2011 reserves the value 0xFFFE. IETF RFC4944 (6LoWPAN over IEEE 802.15.4) specifies that the range 0x80FF..0x9FFF is reserved for D-subnet-local multicast. 9.1.6.4 specifies that the range 0xA000..0x0AFFF is reserved by this standard for graph numbers used in source routes.

12.13.4.4 Singly-dimensioned arrays and standard data structures

This standard supports access to standard data structures and to arrays of either scalar elements or standard data structures.

Supported access to an array, or to a standard data structure not contained within an array, is as follows:

- A singly-dimensioned array or standard structure a may be accessed in its entirety by specifying access to member zero (e.g., an “index” value of 0, $a[0]$).
- A single member of a singly-dimensioned array or standard structure a may be accessed by identifying the 1-origin index of the desired member k , as specified in 12.13.4.1.
- A scalar member k of a standard structure member b of a standard structure a (e.g., $a.b.k$, where a and b are standard structures and k is a scalar supported by this standard).
- A singly-dimensioned array element b of a standard structure a may be accessed in its entirety by specifying access to member zero (e.g., $a.b[0]$, where a is a standard structure and b is a singly-dimensioned array, and the array b is comprised either of scalars or standard structures as defined by this standard).
- A single element of a singly-dimensioned array b of standard structures that is a member of a standard structure a (e.g., $a.b[k]$, where a is a standard structure, b is a singly-dimensioned array comprised either of scalars or standard structures as defined by this standard, and k is the 1-origin index of the member of interest).

12.13.4.5 Singly-dimensioned arrays

This standard supports access to a singly-dimensioned array, whose individual members are either scalars or standard data structures as defined by this standard, as follows:

- A single element of a single dimension array, comprised of scalars (e.g., $a[k]$, where a specifies the array and k specifies the element in the array).
- A singly-dimensioned array of scalars or standard data structures may be accessed in its entirety (e.g., $a[0]$, where a specifies the array, and 0 specifies that access is to the entire array).
- An element of a singly-dimensioned array comprised of standard structures (e.g., $a[k][0]$, where a specifies the array, k specifies the array element that is the standard structure, and 0 specifies that access is to the entire member structure).
- A scalar member of a standard structure contained in a singly-dimensioned array (e.g., $a[k].j$, where a specifies the structure as defined by this standard, k specifies the array element that is the standard structure, and j specifies the member within that standard structure).
- A singly-dimensioned array contained as a member of a singly-dimensioned array (e.g., $a[k][0]$, where a is an array of standard structures, k specifies the element of the array, and 0 specifies that access is to the entire array).
- A member of a singly-dimensioned array of scalars or standard structures contained as a member of a singly-dimensioned array (e.g., $a[k][j]$, where a specifies the outer scope array, k specifies an element of that array that is itself an array, and j specifies the element of the inner scope array).

12.13.4.6 Doubly-dimensioned arrays

This standard supports access to a doubly-dimensioned array, consisting of a singly-dimensioned homogeneous array of singly-dimensioned homogeneous or heterogeneous arrays of scalars as defined by this standard, as follows:

a) a scalar element of a doubly-dimensioned array (e.g., $a[k][j]$);

b) a doubly-dimensioned array in its entirety (e.g., $a[0][0]$);

c) a row of a doubly-dimensioned array (e.g., $a[k][0]$); or

d) a column of a doubly-dimensioned array (e.g., $a[0][k]$).

NOTE Addressing form d) specifies a *slice* of the array, where the result is a singly-dimensioned array whose elements are the k'th member of each subarray. This slice mode of access enables selective access to any single member (element) of each component data structure in an array of identically-structured data structures.

12.13.5 Object method addressing

An object method is addressed by specifying

- the addressing of its containing object, and
- the object-unique index of the method identifier of the object.

12.14 Management objects

Standard management objects to manage the device as a whole are defined in this standard. These objects are defined in 6.2 and are accessed through a UAL-contained MP that may include, for example, a management object to support identification of the device, management objects for each layer of the communication protocol suite, and a management object to report alerts from the device.

NOTE Though each object tracks its own event and alarm conditions, the reporting of such conditions is specified by a single ARMO for the device as a whole. This object manages aspects including, but not limited to, the local alert reporting queue(s), the local timer(s) associated with retransmitting if an individual alert acknowledgment is not received, the local alert queue overflow handling, and requests for alarm recovery. See 6.2.7.2 for further details.

12.15 User objects

12.15.1 General

Standard UAP-containable objects are defined to enable interworkability across industries and segments. These objects may be industry-independent (that is, applicable across industries supported by this standard), or industry-dependent (that is, applicable to a particular industry supported by this standard, but not used across industries).

12.15.2 Industry-independent objects

12.15.2.1 General

The standard objects (UAPMO, ARO, UDO, Concentrator, Dispersion, Tunnel, and Interface) are applicable across industries supported by this standard.

12.15.2.2 UAP management object

12.15.2.2.1 General

There is exactly one addressable UAP management object (UAPMO) per UAP supported by the AL defined by this standard. The numeric object identifier of an object indicates a particular object instance. The numeric object identifier of the UAPMO in every UAP shall be fixed and shall have the value one (1). This object facilitates common management of application processes within a device. Attributes of this object are used to indicate such information as the version/revision of the application process and the logical status of the application process. For example, an attribute of the UAPMO indicates if the corresponding UAP is active or inactive.

NOTE 1 It is possible for a UAPMO to support management of a particular group (set) of objects within the UAP.

NOTE 2 Dynamic instantiation of UAPs is outside the scope of this standard.

12.15.2.2.2 Object attributes

A UAPMO has the attributes defined in Table 240.

Table 240 – UAP management object attributes

Standard object type name: UAP management object (UAPMO)				
Standard object type identifier: 1				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant	N/A
UAP_ID	Object key identifier	Associated TLDE SAP	Type: Local matter (as defined by local TL) Classification: Constant	Local TDLE-SAP
UAP_TL_Port	Object key identifier	Associated T-port	Type: Unsigned16 Classification: Constant	NOTE 1 The specification of the UAP to its local TL is a local matter. NOTE 2 Transport defines the hexadecimal value set 0xF0B0+n (where n may range from 0..15) to specify the most compressed representation (into 4 bits) for communication.
Reserved for future use	0	—	—	—
VersionRevision	1	VersionRevision of the UAP	Type: VisibleString Max size: 64 octets Classification: Constant Accessibility: Read only	Human readable identification associated with the UAP Management object. NOTE The UAP vendor determines content of this attribute.
State	2	Status of UAP	Type: Unsigned8 Classification: Static Accessibility: Read only Default value: 1 Named values: 0: inactive; 1: active; 2: failed	See Table 241.
Command	3	Command to change the state of the UAP	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0 Named values: 0: none; 1: stop; 2: start; 3: soft reset; 4: hard reset	The value 'none' shall not be indicated in a write request. Soft reset shall preserve configuration/commissioning data. Hard reset returns application to factory default settings.
MaxRetries	4	The maximum number of client request retries this application process will send in order to have a successful client/server	Type: Unsigned8 Classification: Static Accessibility: Read only Default value: 3	The number of retries sent for a particular message may vary by message based on application process determination of the importance of the message.

Standard object type name: UAP management object (UAPMO)				
Standard object type identifier: 1				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
		communication.	Valid range: 0..8 Classification: Static Accessibility: Read only	For example, some messages may not be retried at all, and others may be retried the maximum number of times.
Number of objects in the UAP including this UAPMO	8	Number of objects in the UAP including this UAPMO	Type: Unsigned8 Classification: Static Accessibility: Read only Default value: 1 Valid range: > 0	All UAPs are required to have a UAPMO, hence the default value is indicated as being 1 (one). The actual value of this attribute shall be the total number of objects contained in the UAP, including the UAPMO.
Array of UAP contained objects	9	Identification of the objects and type contained in this UAP	Type: Array of ObjectIDandType Classification: Static Accessibility: Read only	See Table 271.
Static_Revision_Level	10	Revision level of the static data associated with all management objects	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	Revision level is incremented each time a static attribute value of any object contained in this UAP is changed.
Reserved for future use by this standard	5..7 11..63	—	—	N octets of presently undefined content.

12.15.2.2.3 State table for UAP management object

Table 241 describes the state table for the UAP management object.

Table 241 – State table for UAP management object

Transition	Current state	Event(s)	Action(s)	Next state
T1	Inactive	Write(Command, Start)	Write.rsp(success)	Active
T2	Active	Write(Command,Stop)	Write.rsp(success)	Inactive
T3	Inactive	Write (Command,Stop)	Write.rsp(success)	Inactive
		Write(any Reset command)	Write.rsp(operationAccepted)	
T4	Active	Write(Command,Start)	Write.rsp(success)	Active
		Write(any other command)	Write.rsp(objectStateConflict)	
T5	Inactive	Write(Command,Start)	Write.rsp(failed) Note: Fails to start	Failed
T6	Active	Application problem	N/A	Failed
T7	Failed	Write(Any Reset command)	Write.rsp(operationAccepted)	Inactive
T8	Failed	Write(Any command other than Reset)	Write.rsp(objectStateConflict)	Failed

Figure 120 shows the UAP management object state diagram.

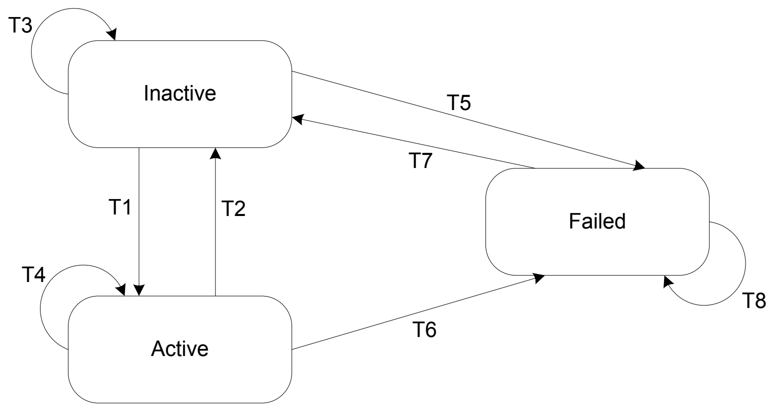


Figure 120 – UAP management object state diagram

12.15.2.2.4 Standard object methods

A UAP management object has methods as defined in Table 242.

Table 242 – UAP management object methods

Standard object type name: UAP Management Object		
Standard object type identifier: 1		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.15.2.3 Alert-receiving object

12.15.2.3.1 General

There may be up to four alert-receiving objects in a device, one per alert reporting category. These alert-receiving objects may receive more than one category of alert report. Categories of alert reports received by alert objects shall be unique; that is, if one alert-receiving object is receiving alerts of category X from the ASL, no other alert objects in the device may also receive alerts of category X from the ASL. These alert-receiving objects may be contained in the same or different processes (e.g., an alert-receiving object for security alerts may be contained in a one application process, while another for process alerts may be contained in another application process).

NOTE Further separation of alerts, or consolidation and re-reporting of alerts, if necessary, is an application process local matter, outside the scope of the AL specification.

12.15.2.3.2 Object attributes

An alert-receiving object may receive alerts from one or more alert-reporting sources. The object has the attributes defined in Table 243.

Table 243 – Alert-receiving object attributes

Standard object type name: Alert-receiving object				
Standard object type identifier: 2				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: > 0	N/A
Reserved for future use	0	—	—	—
Categories	1	BitString of alert categories indicating which object instance supports receiving	Type: BitString Classification: Static Accessibility: Read only Default value: 0 Named indices: 0: device alerts; 1: communication alerts; 2: security alerts; 3: process alerts; 4..7: reserved for future use by this standard	N/A
Errors	2	Count of reports received not for a category that the receiving object indicated was supported	Type: Unsigned16 Classification: Dynamic Accessibility: Read only Default value: 0	Wraps to 0 when maximum value is reached
Reserved for future use by this standard	3..63	—	—	N octets of presently undefined content

12.15.2.3.3 State table for AlertReport handling

Table 244 indicates the states for handling reception of an AlertReport.

Table 244 – State table for handling an AlertReport reception

Transition	Current State	Event(s)	Action(s)	Next State
T1	Ready	AlertReport.ind received	Note processing alert report from device X	Handle individual alert in report
T2	Handle individual alert in report	Check category	Valid category: Acknowledge alert report, and process it	Ready
			Invalid category: Increment the Alert-receiving object instances value of its Errors attribute	

Figure 121 shows the state diagram for alert reception.

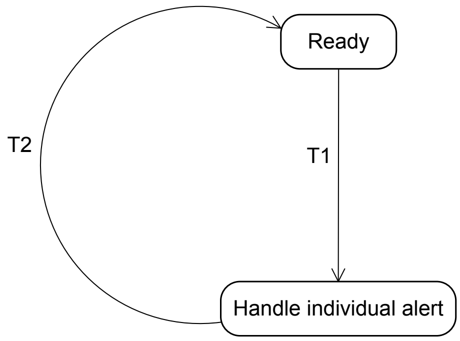


Figure 121 – Alert report reception state diagram

Figure 122 shows one example of alert reporting from multiple devices sources to multiple alert-receiving objects contained in a single UAP of a single sink device.

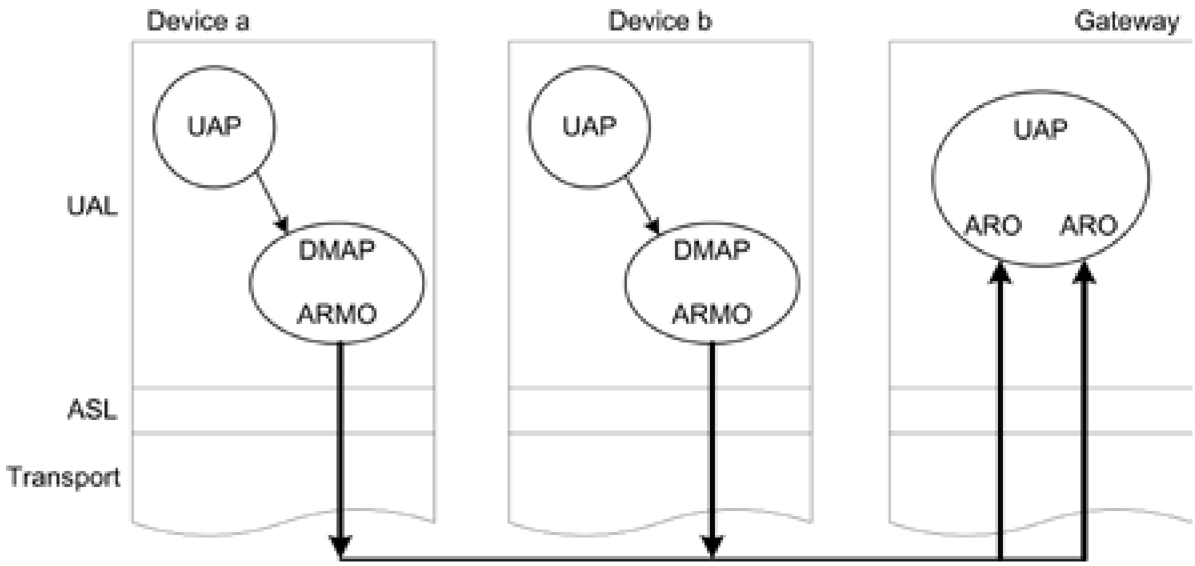


Figure 122 – Alert-reporting example

12.15.2.3.4 Standard object methods

An AlertReceiving object has the methods defined in Table 245.

Table 245 – AlertReceiving object methods

Standard object type name: AlertReceiving object		
Standard object type identifier: 2		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.15.2.4 UploadDownload object

12.15.2.4.1 General

An UploadDownload object is used for either uploading or downloading information to a device. The UploadDownload object may be used to support operations such as downloading a new version of operating firmware or downloading new UAP-contained code or UAP-required bulk data. The UploadDownload object maintains revision control information to indicate what was downloaded or what is available for upload (or both).

An UploadDownload object is likely to support upload or download for a single semantic set of information. An UploadDownload object shall support only one upload or download operation at a time.

A process may have zero or more UploadDownload object instances. Multiple UploadDownload object instances are required if more than one semantic set of information is needed to upload or download its required content.

NOTE 1 The local effect of an application process upload or download (e.g., the creation of new network-visible objects as a result of a download) is a local matter, outside the scope of this standard.

NOTE 2 UploadDownload objects are usable to support upload operations such as the upload of statistical or historical information from the device for analysis. An UploadDownload object is usable to update software/firmware in the target device.

NOTE 3 Support of multicast download is a subject of future standardization. To support multicast loads to a specific set of devices, a configuration tool is currently envisioned to be used to configure the multicast address/device/object relationships for the objects in the multicast set.

12.15.2.4.2 Object attributes

An UploadDownload object has the attributes defined in Table 246. Attributes are included in this object type in order to provide application-level communication timing guidance to the client that is communicating with the UploadDownload object.

NOTE Further guidance to the client, such as regarding tuning of communication timing (for example, related to network communication delays due to the topology of the messaging graph traversed, potential queuing delays, etc.), usable to tune client application behavior, is transparent to an application process, and hence the application itself is unable to provide complete guidance.

Table 246 – UploadDownload object attributes

Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: >0	N/A
Reserved for future use	0	—	—	—
OperationsSupported	1	Indicates if this object supports uploads, downloads, or both	Type: Unsigned8 Classification: Constant Accessibility: Read only Named values: 0: Defined size unicast upload only; 1: Defined size unicast download only; 2: Defined Size unicast upload and unicast download; 3..15: reserved for future use by this standard	N/A
Description	2	Human readable identification of associated content.	Type: VisibleString SIZE (0..64) Classification: Static Accessibility: Read only	
State	3	State of the UploadDownload Object instance	Type: Unsigned8 Classification: Dynamic Accessibility: Read only Default value: 0 Named values: 0: Idle; 1: Downloading; 2: Uploading; 3: Applying; 4: DLComplete; 5: ULComplete; 6: DLError; 7: ULError	See state table below
Command	4	Action command to this object	Type: Unsigned8 Classification: Non-cacheable Accessibility: Read/write Named values: 0: Reset; 1: Apply (used for Download only); 2..15: reserved for future use by this standard	See Table 254

Table 246 *(continued)*				
Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
MaxBlockSize	5	Maximum size of a block which can be accepted for a download, or provided for an upload	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 1 to (MaxNPDUsize - Max TL header size – max(sizeof (additional coding of AL UploadData service request), additional coding of sizeof(AL DownloadData service response)) Valid range: 0..maximum size for data in an APDU	Unit: octets The value shall not exceed the maximum amount of data that can be conveyed in a single APDU per the communication contract. Additionally, space in the APDU shall be left for service related encoding. Block sizes conveyed may be smaller than this value, but shall not be larger
MaxDownloadSize	6	Maximum size available for download as a whole.	Type: Unsigned32 Classification: Static Accessibility: Read only Default value: 0	Unit: octets
MaxUploadSize	7	Size available for Upload	Type: Unsigned32 Classification: Static Accessibility: Read only Default value: 0	Unit: octets
DownloadPrepTime	8	Time required, in seconds, to prepare for a download	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	Time required between sending the StartDownload response till the object can handle a DownloadData
DownloadActivationTime	9	Time in seconds for the object to apply newly downloaded content	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	N/A
UploadPrepTime	10	Time required, in seconds, to prepare for an upload	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	Time from sending the StartUpload response till the object can accept an UploadData
UploadProcessingTime	11	Typical time in seconds for this application object to process a request to upload a block	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	This information is intended to allow a client of an Upload operation to tune its upload related messaging to correspond to the operation of the particular UploadDownload object instance. For example, a client may use this time to help determine its timeout/retry policy, or to determine when to invoke a method on the object instance

Table 246 *(continued)*				
Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
DownloadProcessingTime	12	Typical time in seconds for this application object to process a downloaded block	Type: Unsigned16 Classification: Static Accessibility: Read only Default value: 0	This information may be used by a client of a Download operation to tune its download related messaging to correspond to the operation of the particular UploadDownload object instance. For example, a client may use this time to help determine its timeout/retry policy, or to determine when to invoke a method on the object instance
CutoverTime	13	Time (in seconds) specified to apply the download content	Type: TAINetworkTime Classification: Static Accessibility: Read Write Initial default value : 0	Downloaded content will be applied at the time specified by this attribute
LastBlockDownloaded	14	Number of last block successfully downloaded	Type: Unsigned16 Classification: Static Accessibility: Read only Default value : 0	Updated when an execute response to a DownloadData method is returned. Block number counting shall start at 1 (one). See 12.15.2.4.5.3
LastBlockUploaded	15	Number of last block successfully uploaded	Type: Unsigned16 Classification: Static Accessibility: Read only Default value : 0	Updated when an execute response to an UploadData method is returned. Block number counting shall start at 1 (one). See 12.15.2.4.5.3
ErrorCode	16	Upload or Download error	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 0 Named values: 0: noError; 1: timeout; 2: clientAbort; 18: InconsistentContent; 27: InsufficientDevice Resources; 3..17, 19..26, 28..63: reserved for future use by this standard; 64..255: manufacturerSpecific	Updated when there is an error in uploading or downloading to this object. The error is cleared when the object transitions out of the error state to the idle state. Use InconsistentContent to indicate that the device did not cutover as scheduled due to problem with download payload. Use InsufficientDevice Resources to indicate that the download could not be completed due to lack of memory or other resources

Table 246 ***(continued)***				
Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Reserved for future use by this standard	16..63	—	—	—
This standard does not prescribe product lifecycle management or versioning policies. Description may be used to indicate interchangeability of versions, or to identify features / fixes / software builds. The maximum value of any additional coding for the application coding for this standard is 9 octets. Implementers may wish to consult IETF RFC 2348 regarding recommendations for a maximum size of PDUs.				

12.15.2.4.3 Standard object methods

Initiation of an upload or download bulk data transfer requires first reaching an agreement between the corresponding application objects to participate in the data transfer.

Any additional coordination required to ensure the readiness of the responding device to accept an upload or download request is the responsibility of the UAL process that will be starting a bulk transfer operation.

Client/server messaging to access coordination information from an attribute or set of attributes of the UploadDownload object may be used to support this coordination activity. Specifically, a read request may be used in advance of starting a bulk transfer in order for a client to collect bulk transfer communication-related information specific to an UploadDownload object instance. Once agreement is reached, the client application controls when the data is provided (for a download) to the UploadDownload object, or requested (for an upload) from the UploadDownload object. When transfer is complete, the client indicates the transfer has ended. Additionally, the client may close the transfer if it determines that the entire data transfer should not be completed.

The serving application may request the data transfer be aborted if it determines that the data transfer cannot or should not be completed.

A serving application making a decision to abort based on lack of communication from the client should at least allow for the default standard retries and retry timing policy for the client/server communication policy in order to establish an appropriate timeout.

As with other application communications, it is required that transmission bandwidth be allocated by a communication service contract in order to support a bulk data transfer. Bandwidth for bulk data transfer is not considered dedicated bandwidth as used for periodic messaging, but rather is considered shared bandwidth as used for aperiodic messaging. Use of shared bandwidth among all users of shared bandwidth by a device is dependent on a combination of overall contract priority and message priority. Contract priority is defined by the system manager. Message priority is defined by the application process.

NOTE 1 Any required coordination or sequencing of multiple images to different UploadDownload objects is the responsibility of the host application process. Different uploadable or downloadable images necessitate separate UploadDownload object instances.

NOTE 2 The semantics and syntax of the content and use of uploaded or downloaded information are outside the scope of this standard. The resulting activity in the application process of the device providing upload data or accepting download data, other than updating the UploadDownload object itself, is a local matter, and hence is outside the scope of this standard.

NOTE 3 A proxy application within the device is one way for a single device to process the download multiple times.

Upload from or download to a single device uses a unicast protocol; that is, the upload or download content is sent from/to a single UploadDownload object within a single device.

NOTE 4 File content and/or naming conventions, if applicable to an upload or download are outside the scope of this standard.

An UploadDownload object has the methods defined in Table 247.

Table 247 – UploadDownload object methods

Standard object type name: UploadDownload object		
Standard object type identifier: 3		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
StartDownload	1	This method is used by a client to reach an agreement with an UploadDownload object to participate in a download for which the client will be providing the data, one block at a time
DownloadData	2	This method is used by a client to provide data to an UploadDownload object for an agreed download operation
EndDownload	3	This method is used by a client to terminate a download operation that either has completed successfully, or which the client wishes to abort
StartUpload	4	This method is used by a client to reach an agreement with an UploadDownload object to participate in an upload for which the client will be requesting the data, one block at a time
UploadData	5	This method is used by a client to request data from an UploadDownload object for an agreed upload operation
EndUpload	6	This method is used by a client to terminate an upload operation that either has completed successfully, or that the client wishes to abort
Reserved for future use by this standard	7..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use
The approach used for upload and download has roots in the experiences of multiple accepted standards, including but not necessarily limited to Fieldbus Foundation, Device Net International, IETF RFC 1350 and IETF RFC 2347. Attributes of the UploadDownload object provide application-level information to assist in timeout interval determination by a client, hence IETF RFC 2349 is not followed. Acknowledgment retry as proposed in IETF RFC 2347 is not adopted for the following reasons: • The use cases driving this standard, and the agreed set of technical requirements this standard was to meet, have the vast bulk of communication being publish/subscribe, with very limited use of upload/download. • The upload/download operations that have been defined are not time-critical. • The client application receives feedback from the server if the server is getting duplicates, and can elect to terminate the operation. • The server application is aware of when it is sending error messages back to the client, and is able to elect to abort the operation.		

12.15.2.4.4 StartDownload method

Table 248 describes the StartDownload method of the UploadDownload object.

Table 248 – UploadDownload object StartDownload method

Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Method name	**Method ID**	**Method description**		
StartDownload	1	A client uses the StartDownload method to indicate to an UploadDownload object instance that it desires to download the object.		
	Input arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	BlockSize	Unsigned16	The size of a block of data in octets that will be downloaded
	2	DownloadSize	Unsigned32	The total size of data to be downloaded in octets
	3	DownloadMode	Unsigned8	The desired mode of operation
	Output arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	None			

12.15.2.4.4.1 Method description

A client uses the StartDownload method to indicate to an UploadDownload object instance that it desires to download the object, and to specify the parameters of the download in the input argument list. The UploadDownload object may accept or reject the download, indicating one or the other outcome via the output argument list.

If an UploadDownload object accepts to participate in a download operation, it shall not accept another download operation, or an upload operation until the download operation in process has been terminated or the object has been reset.

12.15.2.4.4.2 Input arguments

Input arguments include:

- BlockSize, which indicates the size of a block of data in octets. All blocks shall have the same size, except that the last block of a download may contain a smaller positive number of octets.
- DownloadSize, which represents the total size of data to be downloaded, in octets.
- DownloadMode, which indicates the operational mode desired. The valid value for this argument indicates unicast and is represented by a value of zero.

12.15.2.4.4.3 Output arguments

There are no output arguments for this method.

12.15.2.4.4.4 Response codes

The following feedback codes are valid for this method:

- operationAccepted;
- invalidBlockSize;
- invalidDownloadSize;
- unexpectedMethodSequence;

- insufficientDeviceResources;
- deviceHardwareCondition; and
- those that are vendor-defined.

12.15.2.4.5 DownloadData method

12.15.2.4.5.1 General

Table 249 describes the DownloadData method of the UploadDownload object.

Table 249 – UploadDownload object DownloadData method

<table>
<tr><th colspan="5">Standard object type name: UploadDownload object</th></tr>
<tr><th colspan="5">Standard object type identifier: 3</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="9">DownloadData</td><td>2</td><td colspan="3">A client uses the DownloadData method to provide data to an UploadDownload object that has agreed to be downloaded.</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type</th><th>Argument description</th></tr>
<tr><td>1</td><td>BlockNumber</td><td>Unsigned16</td><td>BlockNumber being downloaded</td></tr>
<tr><td>2</td><td>Data</td><td>OctetString</td><td>The data for the block being downloaded. The maximum size of this string may vary, such as it may differ for different destination UploadDownload objects</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type</th><th>Argument description</th></tr>
<tr><td>1</td><td>CurrentBlockNumber</td><td>Unsigned16</td><td>This argument is present if the serviceFeedbackCode indicates either blockout of sequence or duplicate</td></tr>
</table>

12.15.2.4.5.2 Method description

StartDownload is used first to have the UploadDownload object agree to the download.

Data is sent one block at a time, sequentially from the lowest numbered block to the highest numbered block using the DownloadData method. Only one DownloadData method invocation may be outstanding at a time; for example, if DownloadData for block n has been invoked, DownloadData for block n+1 shall not be invoked until a successful response containing the output arguments for the download of block n has been received by the client.

The UploadDownload object may indicate that it needs to abort via output argument MethodStatus.

If a client of an upload or download operation is issuing multiple data transfer method invocations for the same block, it may be due to either a network-related problem (e.g., the request is not reaching the server) or a problem at the server device. In this situation, the client may employ the appropriate operation end method (EndDownload or EndUpload) to terminate the operation.

If a client of an upload or download receives multiple dataSequenceError responses, it may be due either to network-related problems (for example, loss of a method invocation response), or problems at the server device. In this situation, the client may employ the appropriate operation end method (EndDownload or EndUpload) to terminate the operation.

Correspondingly, if an UploadDownload object has sent multiple dataSequenceError responses, it may infer that there are either network-related problems or problems at the client device and may elect to abort the operation. If an UploadDownload object indicates operation abort, and this abort is lost over the network, the response sent to a subsequent data method (DownloadData or UploadData) or end method (EndDownload or EndUpload) indicates that the object is no longer participating in an upload or download operation with this client by sending a response indicating unexpectedMethodSequence.

12.15.2.4.5.3 Input arguments

Input arguments include:

BlockNumber, which is the number of the block for which data is provided, where the count of block numbers start at 1 (one); and

Data, which represents the data for the block being downloaded.

12.15.2.4.5.4 Output arguments

This current BlockNumber argument is present if the serviceFeedbackCode indicates either blockout of sequence or duplicate. The argument indicates the last BlockNumber received. The intent is to permit the client to resolve an out-of-sequence or duplicate block reception error without aborting the download operation.

12.15.2.4.5.5 Response codes

The following feedback codes are valid for this method:

- success;
- invalidBlockNumber;
- blockDataError (e.g., wrong block size; content problem);
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- operationAborted;
- dataSequenceError (e.g., duplicate);
- timingViolation; and
- those that are vendor-defined.

12.15.2.4.6 EndDownload method

12.15.2.4.6.1 General

Table 250 describes the EndDownload method of the UploadDownload object.

Table 250 – UploadDownload object EndDownload method

Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Method name	**Method ID (non-negative)**	**Method description**		
EndDownload	3	A client uses the EndDownload method to indicate that the download is terminating.		
	Input arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	Rationale	Unsigned8	This argument indicates the client’s reason for terminating the download operation
	Output arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	None			

12.15.2.4.6.2 Method description

A client uses the EndDownload method to indicate that the download operation is terminating. Termination may occur, for example, if the download has completed, or if the client has elected to terminate the download operation.

EndDownload may be sent from a client that is presently engaged in a download operation, as agreed by the StartDownload method.

12.15.2.4.6.3 Input arguments

The Rationale argument indicates the client’s reason for terminating the download operation. The value used shall be from the following set:

- 0: download completed successfully; or
- 1: client abort.

12.15.2.4.6.4 Output arguments

There are no output arguments for this method.

12.15.2.4.6.5 Response codes

The following feedback codes are valid for this method:

- success;
- operationIncomplete;
- unexpectedMethodSequence;
- timingViolation; and
- those that are vendor-defined.

12.15.2.4.7 StartUpload method

12.15.2.4.7.1 General

Table 251 describes the StartUpload method of the UploadDownload object.

Table 251 – UploadDownload object StartUpload method

Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Method name	**Method ID**	**Method description**		
StartUpload	4	A client uses the StartUpload method to indicate to an UploadDownload object instance that it desires to upload data from the object.		
	Input arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	DownloadMode	Unsigned8	The desired mode of operation
	Output arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	BlockSize	Unsigned16	The size of a block of data in octets
	2	UploadSize	Unsigned32	The total size of the data to be uploaded in octets

12.15.2.4.7.2 Method description

A client uses the StartUpload method to indicate to an UploadDownload object instance that it desires to upload data from the object. The UploadDownload object may accept or reject the upload, indicating the outcome via the output argument list.

If an UploadDownload object accepts to participate in an upload operation, it shall not accept another upload operation or a download operation until the upload operation in process has been terminated or the object has been reset.

12.15.2.4.7.3 Input arguments

Input arguments include:

DownloadMode, which specifies the desired mode of operation. The valid value for this argument indicates unicast and is represented by a value of zero.

12.15.2.4.7.4 Output arguments

Output arguments include:

- BlockSize, which is the size of a block of data in octets. All blocks shall have the same size, except that the last block of an upload may contain a smaller positive number of octets.
- UploadSize, which indicates the size of the data to be uploaded, in octets.

12.15.2.4.7.5 Response codes

The following feedback codes are valid for this method:

- success;
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- those that are vendor-defined.

12.15.2.4.8 UploadData method

12.15.2.4.8.1 General

Table 252 describes the UploadData method of the UploadDownload object.

Table 252 – UploadDownload object UploadData method

Standard object type name: UploadDownload object				
Standard object type identifier: 3				
Method name	**Method ID**	**Method description**		
UploadData	5	A client uses the UploadData method to acquire data from an UploadDownload object that has agreed to be uploaded.		
	Input arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	BlockNumber	Unsigned16	The number of the block for which data is requested
	Output arguments			
	Argument number	**Argument name**	**Argument type**	**Argument description**
	1	Data	OctetString	This argument contains the data for the requested block. This argument is present if and only if the serviceFeedbackCode indicates success. The maximum size of this may vary by UploadDownload object instance being uploaded

12.15.2.4.8.2 Method description

A client uses the UploadData method to acquire data from an UploadDownload object which has agreed to be uploaded.

The StartUpload is used first to have the UploadDownload object agree to the upload. Data is requested one block at a time, sequentially from the lowest numbered block to the highest numbered block. Only one UploadData method invocation may be outstanding at a time. For example, if UploadData for block n has been invoked, UploadData for block n+1 shall not be invoked until the corresponding successful response containing the output arguments has been received by the client.

The UploadDownload object may indicate that it needs to abort via an output argument.

12.15.2.4.8.3 Input arguments

The BlockNumber argument specifies the number of the block for which data is requested. Block number counting shall start at 1 (one).

12.15.2.4.8.4 Output arguments

The Data argument contains the data for the requested block. This argument is present if and only if the serviceFeedbackCode indicates success.

12.15.2.4.8.5 Service feedback codes

The following feedback codes are valid for this method:

- success;
- unexpectedMethodSequence;

- insufficientDeviceResources;
- deviceHardwareCondition;
- operationAborted;
- dataSequenceError (e.g., duplicate, invalid block number, unexpected block number);
- timingViolation; and
- those that are vendor-defined.

12.15.2.4.9 EndUpload method

12.15.2.4.9.1 General

Table 253 describes the EndUpload method of the UploadDownload object.

Table 253 – UploadDownload object EndUpload method

<table>
<tr><th colspan="5">Standard object type name: UploadDownload object</th></tr>
<tr><th colspan="5">Standard object type identifier: 3</th></tr>
<tr><th>Method name</th><th>Method ID (non-negative)</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="8">EndUpload</td><td>6</td><td colspan="3">A client uses the EndUpload method to indicate that the upload operation is terminating.</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type</th><th>Argument description</th></tr>
<tr><td>1</td><td>Rationale</td><td>Unsigned8</td><td>This argument indicates the client’s reason for terminating the upload operation</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type</th><th>Argument description</th></tr>
<tr><td colspan="4">None</td></tr>
</table>

12.15.2.4.9.2 Method description

A client uses the EndUpload method to indicate that the upload operation is terminating. Termination may occur for example if the upload has completed, or if the client has elected to terminate the upload operation.

EndUpload may be sent from a client that is presently engaged in an upload operation, as agreed by the StartUpload method.

12.15.2.4.9.3 Input arguments

The Rationale argument indicates the client’s reason for terminating the upload operation. The value used shall be from the following set:

- 0: upload completed successfully; or
- 1: client abort.

12.15.2.4.9.4 Output arguments

There are no output arguments for this method.

12.15.2.4.9.5 Service feedback codes

The following feedback codes are valid for this method:

- success;
- operationIncomplete;
- unexpectedMethodSequence ;
- timingViolation;
- those that are vendor-defined.

12.15.2.4.10 State table for download

Table 254 shows the download state table.

Table 254 – Download state table for unicast operation mode

Transition	Current State	Event(s)	Action(s)	Next State
T1	Idle	Execute.indicate(StartDownload)	Execute.response(success)	Downloading
T2	Downloading	Execute.indicate(DownloadData) Request for block is from same client object that started the download, and download data parameters are acceptable	Execute.response(success)	Downloading
		Execute.indicate(StartDownload) or Execute.indicate(any Upload Method)	Execute.response (objectStateConflict)	
		Execute.indicate(DownloadData) and request is from wrong client, or something is wrong with the download data parameters or timing	Execute.response(appropriate error) where the appropriate error may be, for example, invalidArgument, incompatibleMode, timingViolation, … NOTE It is a local matter for the UploadDownload object to determine if/when to abort the download.	
		Execute.indicate(EndDownload [Success]) and UploadDownload object does not agree download was completed successfully	Execute.response(incompatibleMode)	
		Write.indicate(StateCommand.Any value)	Write.response(objectStateConflict)	
T3	Downloading	Execute.indicate(EndDownload [Success])	Execute.response(Success)	DLComplete
T4	DLComplete	Write.indicate(StateCommand, Apply)	Write.response(operationAccepted)	Applying
T5	Applying	Application successful	None	Idle
T6	Downloading	Timeout waiting for subsequent method invocation	Update ErrorCode attribute of UploadDownload object	DLError
		Execute.indicate(EndDownload[Abort])	Update ErrorCode attribute of UploadDownload object. Execute.response (Success)	

Transition	Current State	Event(s)	Action(s)	Next State
T7	Idle	Execute.indicate(any Download method other than StartDownload)	Execute.response(objectStateConflict)	Idle
		Execute.indicate(StartDownload) and Request is unacceptable. For example, one or more input arguments are not agreeable	Execute.response(appropriate error) (e.g., invalidObjectID)	
		Write.indicate(StateCommand.Any value other than Reset)	Write.response(objectStateConflict)	
		Write.indicate(StateCommand.Reset)	Write.response(success)	
T8	DLComplete	Execute.indicate(any Download method or any Upload method)	Execute.response(objectStateConflict)	DL_Complete
T9	Applying	Execute.indicate(any Download method or any Upload method)	Execute.response(objectStateConflict)	Applying
		Write.indicate(StateCommand.Any value)	Write.response(objectStateConflict)	
T10	DLComplete	Write(StateCommand, Reset)	1. Discard download content; and 2. Write.req(success)	Idle
T11	DLError	Write(StateCommand, Reset)	1. Discard download content; 2. Clear ErrorCode attribute; and 3. Write.req(success)	Idle
T12	DLError	Any Upload or Download method	Execute.response (objectStateConflict)	DLError
		Any state command other than Write(StateCommand.Reset)	Write.req(objectStateConflict)	
T13	Applying	Application failure	Update ErrorCode attribute of UploadDownload object	DLError
T14	DLComplete	Timeout waiting for command to apply	Update ErrorCode attribute of UploadDownload object	DLError

Figure 123 shows the Upload/Download object download state diagram.

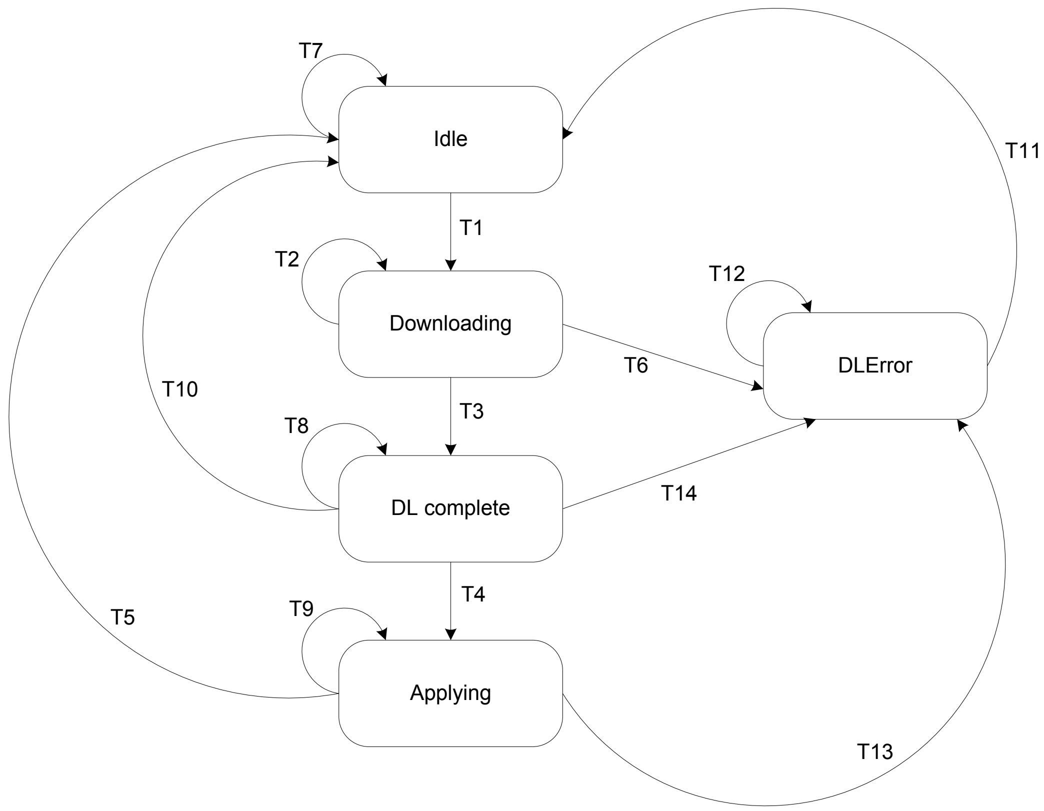


Figure 123 – Upload/Download object download state diagram

12.15.2.4.11 State table for upload

Table 255 shows the upload state table.

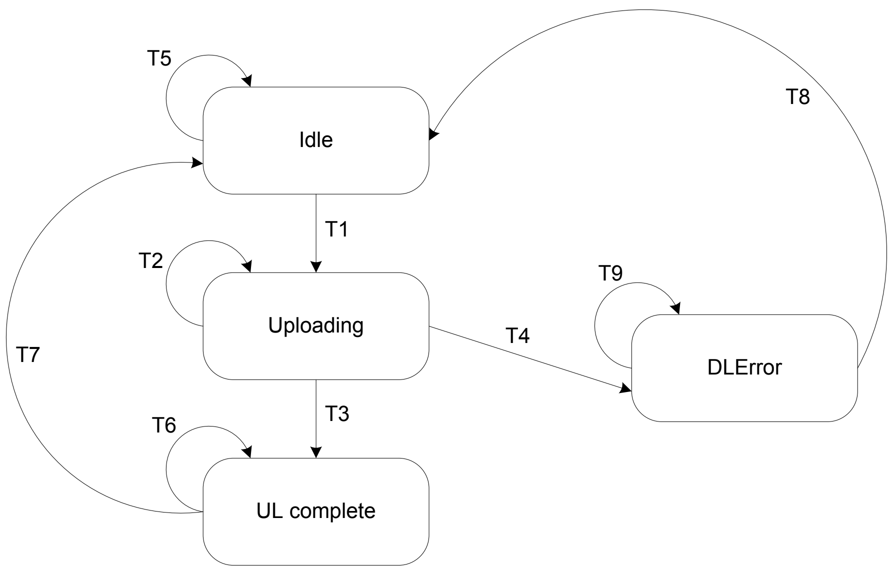


Figure 124 – Upload/Download object upload state diagram

Table 255 – Upload state table for unicast operation mode

Transition	Current State	Event(s)	Action(s)	Next State
T1	Idle	Execute.indicate(StartUpload)	Execute.response(Success)	Uploading
T2	Uploading	Execute.indicate(UploadData). Request for block is from same client object that started the upload, and upload data parameters are as acceptable	Execute.response(Success)	Uploading
		Execute.indicate (StartUpload) or any Download method	Execute.response (objectStateConflict)	
		Execute.indicate(UploadData) and request is from wrong client, or something is wrong with the upload data parameters or timing	Execute.response(appropriate error) where the appropriate error may be, for example, invalidArgument, incompatibleMode, timingViolation, … NOTE It is a local matter for the UploadDownload object to determine if/when to abort the upload if this occurs more than once consecutively.	
		Execute.indicate(EndUpload [Success]) and UploadDownload object does not agree upload was successful	Execute.response(incompatibleMode)	
		Write.indicate(StateCommand .Any value)	Write.response(objectStateConflict)	
T3	Uploading	Execute.indicate(EndUpload [Success])	Execute.response(Success)	ULComplete
T4	Uploading	Timeout waiting for subsequent method invocation	Update ErrorCode attribute of UploadDownload object	UL_Error
		Execute.indicate(EndUpload [Abort])	1. Update ErrorCode attribute of UploadDownload object; 2. Execute.response (Success)	
T5	Idle	Execute.indicate(Any Upload method other than StartUpload)	Execute.response(objectStateConflict)	Idle
		Execute.indicate(StartUpload) and Request is unacceptable. For example, one or more input arguments are not agreeable.	Execute.response(appropriate error) (e.g., invalidObjectID)	
		Write.indicate(StateCommand. any other than Reset)	Write.response(objectStateConflict)	
		Write.indicate(StateCommand .Reset)	Write.response(success)	
T6	ULComplete	Execute.indicate(any DownloadMethod or any UploadMethod)	Execute.response (objectStateConflict)	UL_Complete
T7	ULComplete	Write(StateCommand, Reset)	Write.req(success)	Idle
T8	ULError	Write(StateCommand, Reset)	1. Clear “ErrorCode attribute; and 2. Write.req(success)	Idle

Transition	Current State	Event(s)	Action(s)	Next State
T9	ULError	Any Upload or Download method	Execute.response (objectStateConflict)	ULError
		Write(StateCommand.other than Reset)	Write.req (error)	

Figure 124 shows the Upload/Download object's upload state diagram.

12.15.2.4.12 Client responsibilities for upload/download operations

In order to handle message delays in both requests and responses, and to avoid a congestion collapse due to a retransmission loop, only the first instance of a response indicating success shall cause the next data block to be sent via a DownloadData or requested via an UploadData method invocation by the client.

NOTE The intent is to avoid recreating historical situations such as occurred with the trivial file transfer protocol (TFTP), creating the Sorcerer's Apprentice Syndrome.

12.15.2.5 Concentrator object

12.15.2.5.1 General

A concentrator object represents an assembly of data, collected from multiple objects in the same UAP, that is to be published by a single publish request service. This object optimizes publication messages sent from a device. Multiple concentrator object instances may be used to represent multiple assemblies of data if required. A list of attributes is provided to indicate the data values that are published.

NOTE The published content represented by this object is established by configuration. This standard does not specify the device configuration tool.

A subscriber to data produced by a concentrator object shall only be a dispersion object. The data types associated with the list of attributes of the dispersion object should be configured to match those produced by the concentrator object.

When a concentrator object is configured by a host application, such as a gateway, the device is responsible for establishing contracts as needed to support the corresponding publications. The design is intended to support two use cases. In one case, the device joins the network and then the host configures the concentrator object. In the other case, the concentrator object is pre-configured and the device autonomously starts publication after it joins the network.

A UAP may have zero or more concentrator object instances.

12.15.2.5.2 Object attributes

A concentrator object has the attributes defined in Table 256.

The first time a UAP receives a read/write/execute request from an endpoint for which it has no contract, it shall request a contract so that it can send a service response to the requesting endpoint. The UAP shall, as necessary, delay the first service response to allow for time to establish/modify the contract.

Table 256 – Concentrator object attributes

Standard object type name: Concentrator object				
Standard object type identifier: 4				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: > 0	N/A
Reserved for future use	0	—	—	—
Concentrator ContentRevision	1	Tracks a change in what is published; ensures Concentrator (publisher) and Dispersion (subscriber) objects are in harmony	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	Revision shall be incremented when the complement of data to publish changes, i.e. CommunicationEndpoint or Array of ObjectAttributeIndexAndSize are changed. Attribute included in Table 347 header
CommunicationEndpoint	2	Serves to identify the object that receives the publication from this object	Type: Communication association endpoint structure Classification: Static Accessibility: Read/write Default value: The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	Write to this attribute last when configuring this object; see Table 265
Communication contract data for scheduled communication	3	Data corresponding to the communication contract	Type: Communication contract data Classification: Static Accessibility: Read only	Updated when the corresponding contract is established or terminated; see Table 266
MaximumItems Publishable	4	Maximum number of items that can be published	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: Local matter	If this attribute has a value of 0, it indicates it is not configured for publishing
NumberItemsPublishing	5	Actual number of items being published	Type: Unsigned8 Classification: Static Accessibility: Read only Default value: 0	Updated as ObjectAttributeIndexAndSize attributes are configured : incremented when another value to publish is added, and decremented when a value to publish is removed
ObjectAttributes	6	Array of data to identify each piece of data published	Type: Array of Object AttributeIndexAndSize Classification: Static Accessibility: Read/write Default value: Element size is 0	Object ID, attribute ID, attribute index, and size for each value published. See Table 264
Reserved for future use by this standard	7..63	—	—	—

Revision, NumItemsSubscribing, and ObjAttrIdx attributes can be implemented in a device in such a way that they can be written atomically in a single network transaction via concatenation of APDUs.

12.15.2.5.3 Standard object methods

A concentrator object has the methods defined in Table 257.

Table 257 – Concentrator object methods

Standard object type name: Concentrator object		
Standard object type identifier: 4		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.15.2.6 Dispersion object

12.15.2.6.1 General

A dispersion object is the subscribing object corresponding to a concentrator object. This object is configured to indicate how to parse a concentrator object's published content. If multiple disassemblies are required, multiple dispersion user objects are to be used. A UAP may have zero or more dispersion object instances.

NOTE Concentrator and dispersion objects are special objects supporting a publication proxy within a UAP. These objects are distinct from proxy application processes, which are able to distribute information across multiple UAPs within the UAL.

12.15.2.6.2 Object attributes

A dispersion object has the attributes defined in Table 258.

Table 258 – Dispersion object attributes

Standard object type name: Dispersion object				
Standard object type identifier: 5				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: > 0	N/A
Reserved for future use	0	—	—	—
Concentrator ContentRevision	1	Tracks changes to content subscribed. Ensures Concentrator publishing object and Dispersion subscribing object are in harmony	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	Updated when the complement of data to publish changes. In the event of a mismatched ContentRevision (Table 347), the publication shall not be processed
CommunicationEndpoint	2	Endpoint of concentrator object that publishes data to this dispersion object	Type: Communication association endpoint structure Classification: Static Accessibility: Read/write Default value: The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid) Valid range: See structure definition	Write to this attribute last when configuring this object
MaximumItemsSubscribing	3	Maximum number of items that can be subscribed	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: Local matter Valid range: >0	Maximum number of items in corresponding publication
NumItemsSubscribing	4	Number of items being subscribed to	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	Actual number of items in corresponding publication. A value of zero indicates the object is not configured to subscribe

Standard object type name: Dispersion object				
Standard object type identifier: 5				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Array of ObjectAttributeIndexAndSize	5	Array of data to identify each piece of data published	Type: Array of ObjectAttributeIndexAndSize Classification: : Static Accessibility: Read/write Default value: Element size is 0	Object ID, Attribute ID, Attribute index, and size of data for the destination within the application for the published information. NOTE To skip over data, the destination object and attribute may locally represent a Null object and Null attribute.
Reserved for future use by this standard	6..63	—	—	—

Revision, NumItemsSubscribing, and ObjAttrIdx attributes can be implemented in a device in such a way that they can be written atomically in a single network transaction via concatenation of APDUs.

12.15.2.6.3 Standard object methods

A dispersion object has the methods defined in Table 259.

Table 259 – Dispersion object methods

Standard object type name: Dispersion object		
Standard object type identifier: 5		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.15.2.7 Tunnel object

12.15.2.7.1 General

The tunnel object (TUN) is used to support the energy efficient transport of encapsulated messages over the network for a single non-native protocol. The tunnel service and a variation of the publication service are defined for this encapsulation. Support structures are provided for deconstruction, mapping and reconstruction of non-native protocol packets in order to reduce transactions and packet size.

NOTE The usage of the tunnel object to create protocol translators is intended to be defined by the organization that has defined the non-native protocol used in the tunnel.

12.15.2.7.2 Object attributes

A tunnel object has the attributes defined in Table 260.

Table 260 – Tunnel object attributes

Standard object type name: Tunnel object				
Standard object type identifier: 6				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: > 0	N/A
Reserved for future use	0	—	—	—
Protocol	1	Type of protocol supported by this object	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value: Local matter (protocol-specific) Valid range: See Annex M	Sets the specific protocol that is encapsulated in tunnel messages. Only matching protocol tunnels exchange meaningful data
Status (configuration status)	2	Communication configuration status of this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0 Named values: 0: not configured; 1: validly configured; 2: invalidly configured	The object status is not configured when the Protocol is set to None and no communication occurs. Once the object is configured and another protocol is set, the object attempts to apply the configuration and changes the status appropriately
Flow_Type	3	Communication service used by this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Named values: 0: 2-part tunnel; 1: 4-part tunnel; 2: publish; 3: subscribe	Configures the tunnel for a specific type of communication and role
Update_Policy	4	Periodic communication update policy for this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Named values: 0: periodic; 1: change of state	Sets the periodic publication policy for a linked publisher and subscriber. A periodic update publishes on every opportunity. Change of state publishes on fresh data or at least as often as Stale_Limit specifies

Standard object type name: Tunnel object				
Standard object type identifier: 6				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Period (data publication period)	5	Periodic communication update period for this object	Type: Integer16	Sets the periodic publication time for a linked publisher and subscriber. Isochronous publication is enabled by an implicit rule. Publication does not begin until a period is set. See 12.12.5
			Classification: Static	
			Accessibility: Read/write	
			Default value: 0	
			Named values: 0: not configured	
Phase (ideal publication phase)	6	Periodic communication phase within period for this object	Type: Unsigned8	Sets the requested publication time within the period for a linked publisher and subscriber. The actual phase may differ by contract requirements. Units should be indicated as a percentage (%)
			Classification: Static	
			Accessibility: Read/write	
			Valid range: 0..99	
Stale_Limit (stale data limit)	7	Periodic communication stale data limit for this object	Type: Unsigned8	Defines the maximum subscriber expected arrival time as a multiple of the period. Defines the minimum publication rate for change of state reporting as a multiple of the period
			Classification: Static	
			Accessibility: Read/write	
Max_Peer_Tunnels	8	Maximum number of correspondent tunnels with which this object can communicate	Type: Unsigned8	N/A
			Classification: Constant	
			Accessibility: Read only	
Num_Peer_Tunnels	9	Actual number of correspondent tunnels with which this object is communicating	Type: Unsigned8	Incremented / decremented as Tunnel endpoints array elements are added and deleted
			Classification: Static	
			Accessibility: Read/write	
			Default value: 0	
Array of Tunnel endpoint	10	Array of Protocol association endpoints	Type: Array of Tunnel endpoint	Links remote tunnel objects for communication with this tunnel object
			Classification: Static	
			Accessibility: Read/write	
			Valid range: Address information pointing to one or more tunnel objects	
Foreign_Source_Address	11	Foreign source address mapped to this objects communication	Type: IPv6Address	Holds static addressing information to be delivered to initiator or correspondent upon message receipt
			Classification: Static	
			Accessibility: Read/write	

Standard object type name: Tunnel object				
Standard object type identifier: 6				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Foreign_Destination_Address	12	Foreign destination address mapped to this objects communication	Type: IPv6Address Classification: Static Accessibility: Read/write	Holds static addressing information to be delivered to initiator or correspondent upon message receipt
Connection_Info[ ]	13	Foreign connection information mapped to this objects communication	Type: OctetString Classification: Static Accessibility: Read/write	Holds static information to be delivered to initiator or correspondent upon message receipt
Transaction_Info[ ]	14	Foreign transaction information mapped to this objects communication	Type: OctetString Classification: Dynamic Accessibility: Read/write	Holds transaction specific information to be delivered to initiator on completion of a transaction
Reserved for future use by this standard	15..63	—	—	

12.15.2.7.3 Standard object methods

A tunnel object has the methods defined in Table 261.

Table 261 – Tunnel object methods

Standard object type name: Tunnel object		
Standard object type identifier: 6		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.15.2.8 Interface object

12.15.2.8.1 General

The interface object provides a generic messaging end point for interfacing to a network. This object may be used as the source or destination object in native messaging interactions required for support of gateway protocol translation and various native messaging applications.

The interface object may be indicated in client/server communication services necessary for native read, write, and execute services. The interface object may also be referenced as the client object communicating with an upload/download object for bulk transfer.

Communications referencing the interface object as a client shall adhere to the client/server congestion control policies defined in this standard.

Where possible, implementers shall consider buffering client server retrieved values for local usage rather than creating additional communications over the wireless network to repeatedly retrieve these values from devices which need to preserve power. User application objects contained in the field devices provide guidance, via their specification of attribute data classification, regarding what object-related information should be buffered.

NOTE 1 Native object publication and subscription is accomplished by using the concentrator and dispersion objects.

NOTE 2 The actual structure of buffering/cache and local requirements for handling messages to the cache are considered local matters, outside the scope of this standard.

12.15.2.8.2 Object attributes

An interface object has the attributes defined in Table 262.

Table 262 – Interface object attributes

Standard object type name: Interface object				
Standard object type identifier: 7				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: > 0	N/A
Reserved for future use	0	—	—	—
Reserved for future use by this standard	1..63	—	—	—

12.15.2.8.3 Standard object methods

An interface object has the methods defined in Table 263.

Table 263 – Interface object methods

Standard object type name: Interface object		
Standard object type identifier: 7		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.16 Data types

12.16.1 Basic data types

The basic data types supported for attributes are:

- binary values;
- 8-, 16-, and 32-bit signed integers;
- 8-, 16-, 32-, 64- and 128-bit unsigned integers;
- ISO/IEC/IEEE 60559 (IEEE 754) 32-bit and 64-bit floating point values;

- strings representing visible text, a block of octets, or a sequence of bit values (BitString);
- time: TAINetworkTime, TAITimeDifference, TAITimeRounded.

12.16.2 Derived atomic data types

The derived atomic data types supported for attributes are:

- addresses:
  - IPv6Address (mapped to a 128-bit unsigned integer),
  - EUI64Address (mapped to a 64-bit unsigned integer),
  - DL16Address (mapped to a 16-bit unsigned integer);
- layer-specific identifiers (generally mapped to 8-bit or 16-bit unsigned integers): ;
- MIB indices (generally mapped to 8-bit or 16-bit unsigned integers): .

12.16.3 Industry-independent standard data structures

12.16.3.1 General

Standard data structures used shall be the data structures conveyed by the protocol defined by this standard. Industry-independent standard data structures are summarized in Annex L.

NOTE Vendor-specific data structure definitions are not supported.

12.16.3.2 Object, attribute, index, and size

The elements of ObjectAttributeIndexAndSize are shown in Table 264.

Table 264 – Data type: ObjectAttributeIndexAndSize

Standard data type name: ObjectAttributeIndexAndSize		
Standard data type code: 469		
Element name	**Element identifier**	**Element scalar type**
ObjectID	1	Type: Unsigned16 Classification: Static Accessibility: Varies by use
AttributeID	2	Type: Unsigned16 Classification: Static Accessibility: Varies by use
AttributeIndex	3	Type: Unsigned16 Classification: Static Accessibility: Varies by use
Size	4	Type: Unsigned16 Classification: Static Accessibility: Varies by use NOTE In practice, this maximum size depends on the capabilities of the device.

12.16.3.3 Communication association endpoint

The data structure shown in Table 265 is used for communication endpoints for both inputs and outputs.

Table 265 – Data type: Communication association endpoint

Standard data type name: Communication association endpoint		
Standard data type code: 468		
Element name	**Element identifier**	**Element scalar type**
Network address of remote endpoint	1	Type: IPv6Address This is a logical construct configured for the device by the system manager NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios. Classification : Static Accessibility: Read/write
T-port at remote endpoint	2	Type: Unsigned16 Classification : Static Accessibility: Read/write
Object ID at remote endpoint	3	Type: Unsigned16 Classification : Static Accessibility: Read/write
Stale data limit	4	Type: Unsigned8 Classification : Static Accessibility: Read/write NOTE 1 This attribute is primarily of interest to a subscriber. NOTE 2 This is a count of consecutive stale input values that a subscriber fails to receive before the subscriber considers the value previously received to be Bad (Table 299). Staleness is implied by an unchanging freshness sequence number (Table 347).
Data publication period	5	Type: Integer16 Classification : Static Accessibility: Read/write NOTE For units of time, see 12.12.5
Ideal publication phase	6	Type: Unsigned8 Classification : Static Accessibility: Read/write Valid range: 0..99 (as a percentage %) NOTE This attribute is primarily of interest to a publisher.
PublishAutoRetransmit	7	Type: Unsigned1 Classification: Static Accessibility: Read/write Named values: (Note 1) 0: Transmit only if application content changed since last publication; 1 : Transmit at every periodic opportunity (regardless of whether application content changed since last transmission or not)

Standard data type name: Communication association endpoint		
Standard data type code: 468		
Element name	**Element identifier**	**Element scalar type**
Configuration status	8	Unsigned8 Classification: Static Accessibility: Read access Named values: 0 : not configured (connection endpoint not valid); 1: configured (connection endpoint valid) NOTE The data owner sets this element to a value of 0 to indicate that the endpoint is not configured, and to 1 to indicate the endpoint is configured. An endpoint is considered not configured if the value of Object ID at remote endpoint is 0.
NOTE 1 The coding of this attribute is the inverse of the related attribute 12 of Table 27.		

12.16.3.4 Communication contract data

The data structure shown in Table 266 is used for the dynamic data important to the local application process that is associated with a particular communication contract.

NOTE 1 It is a local matter to ensure that applications are well-behaved in terms of the communication contracts they employ. The AL does not standardize the policing of compliance with requested contracts.

NOTE 2 As part of contract negotiation, sufficient information is provided to the contract requesting device in order to enable it to determine the maximum size APDU that the contract supports. For example, if contract negotiation determines the maximum network service data unit (NSDU) size, then the type of security in effect for the contract is locally determinable for the contract. If the type of security in use is known, the transport header size is locally acquired and subtracted from the maximum NPDU size, thus yielding the value for the maximum APDU size usable for communications employing that particular communication contract.

Table 266 – Data type: Communication contract data

Standard data type name: Communication contract data		
Standard data type code: 470		
Element name	**Element identifier**	**Element type**
ContractID	1	Type: Unsigned16 Classification: Static Accessibility: Read only Valid range: The set of valid values is defined by system management
Contract_Status	2	Type: Unsigned8: Classification: Static Accessibility: Read only Default value = 0 Named values: 0: endpoint_not_configured, 1: awaiting_contract_establishment, 2: contract_active_as_requested, 3: contract_active_negotiated_down, 4: awaiting_contract_termination, 5: contract_establishment_failed, 6: contract_inactive
Actual_Phase	3	Type:Unsigned8 Classification: Dynamic Accessibility: Read only Default value: 0 (indicating not assigned) Valid range: 0..99 (in units of percentage %)
Further information on the actual contract, such as the negotiated-down parameters, may be available from the DMAP and does not need to be maintained by the UAP.		

12.16.3.5 Alert communication endpoint

The data structure shown in Table 267 is used for communication endpoints for alert reports.

Table 267 – Data type: Alert communication endpoint

Standard data type name: Alert communication endpoint		
Standard data type code: 471		
Element name	**Element identifier**	**Element scalar type**
Network address of remote endpoint	1	Type: IPv6Address This is a logical construct configured for the device by the system manager NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios. Classification : Static Accessibility: Read/write
T-port at remote endpoint	2	Type: Unsigned16 Classification : Static Accessibility: Read/write
Object ID at remote endpoint	3	Type: Unsigned16 Classification : Static Accessibility: Read/write

12.16.3.6 Tunnel endpoint

The data structure shown in Table 268 is used in tunnel objects to identify remote tunnel endpoints for exchange of encapsulated payloads.

Table 268 – Data type: Tunnel endpoint

Standard data type name: Tunnel endpoint		
Standard data type code: 475		
Element name	**Element identifier**	**Element scalar type**
Network_Address (network address of remote endpoint)	1	Type: IPv6Address This is a logical construct configured for the device by the system manager NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios. Classification: Static Accessibility: Read/write
Transport_Port (T-port at remote endpoint)	2	Type: Unsigned16 Classification: Static Accessibility: Read/write
OID (object ID at remote endpoint)	3	Type: Unsigned16 Classification: Static Accessibility: Read/write

12.16.3.7 Alert report descriptor

Elements of the alert report descriptor are shown in Table 269.

Table 269 – Data type: Alert report descriptor

Standard data type name: Alert report descriptor		
Standard data type code: 499		
Element name	**Element identifier**	**Element type**
Alert report disabled	1	Type: Boolean8 Classification: Static Accessibility: Read/write Default value : Local matter
Alert report priority	2	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0 Valid range: 0..15, as specified in 12.17.5.2.2.22

12.16.3.8 Analog alarm descriptor

The analog alarm descriptor is used to define alarm reporting for an analog value with a single reference condition. Its elements are shown in Table 270.

Table 270 – Data type: Process control alarm report descriptor for analog with single reference condition

Standard data type name: Process control alarm report descriptor for analog with single reference condition		
Standard data type code: 498		
Element name	**Element identifier**	**Element scalar type**
Alert report disabled	1	Type: Boolean8 Classification: Static Accessibility: Read/write Default value : TRUE
Alert report priority	2	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0 Valid range: 0..15
Alarm limit	3	Type: Float32 Classification: Static Accessibility: Read/write

12.16.3.9 Binary alarm descriptor

The binary alarm descriptor is the same structure as the data type for the alert report descriptor, so no additional data type description is required.

12.16.3.10 ObjectIDandType

The elements of ObjectIDandType are shown in Table 271.

Table 271 – Data type: ObjectIDandType

Standard data type name: ObjectIDandType		
Standard data type code: 472		
Element name	**Element identifier**	**Element scalar type**
ObjectID	1	Unsigned16
ObjectType	2	Unsigned8
ObjectSubType	3	Unsigned8
VendorSubType	4	Unsigned8

12.16.3.11 Unscheduled correspondent

The elements of Unscheduled Correspondent are shown in Table 272.

Table 272 – Data type: Unscheduled correspondent

Standard data type name: Unscheduled Correspondent		
Standard data type code: 473		
Element name	**Element identifier**	**Element scalar type**
Address	1	IPv6Address
T-port	2	Unsigned16

12.17 Application services provided by application sublayer

12.17.1 General

All interfaces between the DLE and adjacent layer (or sublayer) entities or management entites are internal interfaces within the device, and thus are unobservable. Therefore they are not subject to standardization.

Application services are provided by the ASL (at the ASLDE-n SAP) for communication with native objects, which are either UAP objects or MP objects. These are the only services that should be used for behavior compliant with this standard. Not all devices will need to use all of the services defined herein, and not all objects will support all the services herein. However, if these services are employed for communication between or among native objects, they should be employed as defined in this standard.

Application processes using ASL services should be designed to tolerate receipt of duplicate ASL service indications and confirmations. For example, if a lower layer acknowledgment is lost when a response to a read request is sent, the lower layer may retry, and as a result the application client may receive a duplicate response to the read request.

It is left to the device to determine how best to handle congestion/back pressuring if locally indicated by the local lower protocol suite. This handling may, for example, limit transmission of messages from the device for a certain period of time, or for a certain set of communication priorities, or both. Congestion may occur, for example, in situations of network communication load, and handling by the device is intended to limit additional congestion.

NOTE 1 Capacity planning is a systems issue and is outside the scope of AL consideration.

Table 273 summarizes the services provided.

NOTE 2 Local services that do not result in network communication are not included in Table 273, as they are local matters and hence implementation-dependent.

NOTE 3 Local ASL service confirmation back to the AP is a local matter, and hence is not defined by this standard.

Table 273 – AL services

ASL-provided service	Applicable primitives	Description	How used
Object access services			
Read	Request Indication Response Confirmation	Read an attribute value from an object	Client/server
Write	Request Indication Response Confirmation	Write an value to an object	Client/server
Execute	Request Indication Response Confirmation	Execute a method on an object	Client/server
Publication services			
Publish	Request Indication	Publish a single or multiple values from one source object	Publish/subscribe NOTE Native content, as well as non-native content, is supported
Alert report-related services			
AlertReport	Request Indication	Report an alert	Source/sink (unicast only). The source of this service shall only be the ARMO. The sink of this service shall only be the alarm receiving object
AlertAcknowledge	Request Indication Response Confirmation	Acknowledge an individual alert reception	Client/server The source of this service may only be an alarm receiving object
Explicit support for tunneling			
Tunnel	Request Indication Response Confirmation	Tunnel payload without ASL parsing (for non-native protocol compatibility)	Tunnel payload without ASL parsing (for non-native protocol compatibility). This service shall be used only if the source and destination objects are both tunnel objects

NOTE 4 If a local service request is malformed, reporting the error to the requesting UAP is a local matter, and thus is not addressed in this standard. If desired, it is possible for an implementation to keep statistics regarding locally received services requests that are malformed.

12.17.2 Publish/subscribe application communication model

Publication is a communication process that is initiated by an object in the publishing UAL and received by an object in the subscribing UAL. Publication uses ASL services specific to the supporting publication. Publication occurs from a publisher object to a subscribing object. Any object may act as publisher or subscriber. To optimize communication bandwidth usage, special objects (a concentrator object for publishing and a dispersion object for subscribing) are defined to enable publication from/to a set of objects within a single UAP using a single publish service invocation.

The semantic reason for publishing is purely an application concern. This model supports communication for:

- schedule-triggered periodic buffered communications;
- application-triggered buffered communications; and
- change-of-state-triggered buffered communications.

All of the above published communications use the publish/subscribe communication flow paradigm. For scheduled periodic communications, subscriber applications may support timeout response methods to deal with a loss of individual publications and/or a loss of the publisher endpoint. For example, a subscribing application may use prior publication value content, but may degrade the corresponding quality of the value. Loss of individual messages may have a shorter timeout than the timeout used by the subscriber to determine the loss of the publisher.

Coordination of a publication with the network schedule is accomplished via appropriate endpoint configuration, which drives communication contract requests. A communication contract request is used to request that the system manager allocate scheduled bandwidth for publish communication.

NOTE 1 Determination of actual timeout policy when an expected publication is not received is an application process-specific matter that is not specified by this standard.

NOTE 2 Published messages with native content always contain a sequence counter and the current data value(s). If there is no change in value, the sequence counter indicates that the publishing application is still operating and has no new data to report (this is also known as a heartbeat). Some receiving devices retain this sequence counter to determine if the value has changed in order to limit reprocessing, while other receiving devices elect to ignore the sequence counter.

NOTE 3 For scheduled periodic communications, application processes often use common network time to synchronize their activities across the network. This synchronization is locally applicable by publishers and subscribers to synchronize their activities to the publication schedule.

NOTE 4 Continuous data and measurement in this standard employ a control system field proven publish/subscribe communication model and leverage use of scheduled bandwidth for more precise communication timing. If there is a more customized communication requirement, it is considered a custom situation outside the scope of this standard, or possibly a situation for future consideration.

12.17.3 Scheduled periodic buffered communication

12.17.3.1 General

The publish service is used for unidirectional buffered communication to at most one subscriber.

Publishing should be configured in accordance with the capabilities of the system. If a subscriber is not present, it is generally an interim or error situation and is not expected to be long-lived. Potential energy loss due to improperly configured or failed devices may be addressed by reconfiguration or device replacement. These are rare abnormal situations for which design optimization is not required; hence, the complexity required for publication only in the presence of an actual subscriber outweighs any communications savings.

No acknowledgments or retries are applied to publish/subscribe interactions. A publishing ASL is given an unconfirmed service request and constructs an appropriate APDU, which is then passed to the lower communication layers for communication transfer. If a publishing ASL receives a request for service before the prior request has been conveyed, the new request should overwrite the previous request and the previous request should not be transmitted. If a subscribing ASL receives a new request before the previous request was delivered to the destination object, the new request overwrites the previous request, and the previous request is lost.

The defined services support publication of an arbitrary attribute (for example, a process variable) from a simple device, or publication of a set of attributes from a more complex (for example, a multi-variable capable) wireless device. publish/subscribe communications take place over a configured communication relationship, as shown in Figure 125.

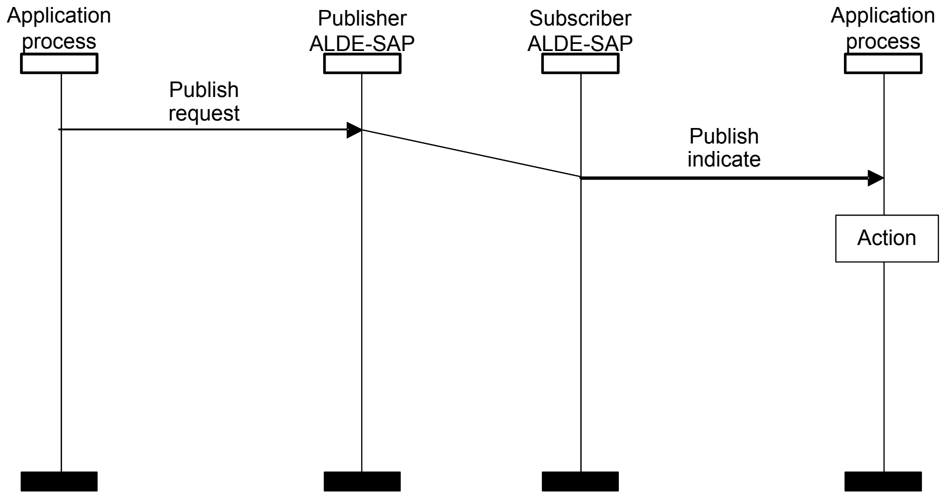


Figure 125 – Publish sequence of service primitives

The subscriber for a publication may be, for example, a gateway.

NOTE 1 The content of non-native publications is outside the scope of this standard.

Native publications include the attribute value itself, and status information for the value. In order to support duplicate detection and out-of-order delivery, a simple one-octet monotonic counter is included with each published value.

NOTE 2 Other schemes for uniquely identifying a published message, such as using a timestamp instead of a counter, were considered but eliminated because they incur more power to communicate. Time of TPDU construction, combined with a lower layer-provided freshness indication, often is locally available. Timestamps on data publications are useful in remote terminal unit (RTU)-style buffering gateways, but for such gateways, a reception timestamp is also usable for that purpose, particularly since all publications use a shared channel-hopping schedule, leaving a small variance between data generation time and data reception time.

Publish/subscribe message priority may be fixed; in that case, a local implementation may elect not to provide per-message priority. If an application requires publications with different priority levels, such as low priority publication for control monitoring and high priority publication for high-rate (1 Hz and 4 Hz) control loops variables, separate publish/subscribe relationships may be required.

The AL publisher and AL subscriber do not communicate explicitly to establish or break their relationship; however, establishment of secure communication relationships may force transport relationships to be established. Publishers and subscribers each establish their portion of the relationship independently (asynchronously). That is, either the publishing UAP or any of the subscribing UAPs may act first to establish its part of a publish/subscribe relationship. End-to-end messaging cannot commence before both the publisher and the subscriber(s) have established their respective sides of the communication relationship. Once the publisher creates one side of the communication relationship and a subscriber creates the other side of the communication relationship, publication messages can be sent and delivered to the corresponding application objects.

Locating publishers dynamically using either a tag discovery service or a centralized directory lookup service is outside the scope of this standard. Therefore, publish/subscribe is intended to be compatible with a static configuration mechanism. In future releases, a discovery mechanism may be employed. In either situation, the same information needs to be available to establish the publish/subscribe relationship.

Communication routes are formed transparently to the AL.

NOTE 3 The formation of transmission routes in support of publish/subscribe is important to ensure timely delivery, but is left as a responsibility for the system manager, which forms routes to use during communication contract establishment. See 6.3.11 for further details.

Publication is always an unconfirmed data transfer service request. Subscription always results in receiving an unconfirmed data transfer service indication.

NOTE 4 It is considered a management topic to ensure security configuration / changes support publish/subscribe without AL impact. It is understood that security considerations often constrain permitted relationships.

The timing of a scheduled publication is coordinated across the network, and as such depends on a coordinated view of time across the network. Bandwidth allocated for schedule-triggered publications needs to be reserved to ensure that subscribers can receive what their publishers send. The schedule should be configured to ensure best effort to meet delivery deadlines, but ultimately, the responsibility of the publisher to create new publications, and the subscriber to act on receipt of them, depends on the device’s internal scheduling of the application process.

Published communications rely on the publication service support provided by the communication protocol suite that is underlying the ASL. An important aspect of this lower communication protocol suite is the ability to provide specific communication timing in order to meet scheduling demands.

12.17.3.2 Publish

12.17.3.2.1 General

The publish service for this standard is a unicast service used to update data periodically from a single publication source in a single AP to (at most) a single subscriber destination.

The publish service may also be used in an aperiodic manner to support both application-triggered and change-of-state-triggered changes. Since buffer content is transmitted according to a schedule, native published communication includes a freshness indicator to enable the subscriber to determine whether or not a value has changed.

NOTE Freshness does not mean unchanged data, but rather that the value has been newly (freshly) acquired since it was last published.

Table 274 defines the service primitives.

Table 274 – Publish service

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	—
Priority	M	—
Discard eligible	M	—
End-to-end transmission time	—	M
Published data size	M	M
Subscriber T-port	M	—
Subscriber TDSAP	—	M
Subscribing object identifier	M	M(=)
Publisher IPv6Address	—	M
Publisher TDSAP	M	—
Publisher T-port	—	M(=)
Publishing object identifier	M	M(=)
DataStructureInformation	M	M(=)
NativeIndividualValue	S	S(=)
Freshness sequence number	M	M(=)
Individual analog value and status	S	S(=)
Individual digital value and status	S	S(=)
NativeValueList	S	S(=)
Publishing content version	M	M(=)
List of publish data	M	M(=)
Fresh value sequence number	M	M(=)
Analog value and status	S	S(=)
Digital value and status	S	S(=)
Non-native	S	S(=)
Non-native data	M	M(=)

12.17.3.2.2 Arguments

12.17.3.2.2.1 Service contract identifier

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

12.17.3.2.2.2 Priority

This parameter defines the message priority of service that is required of the communication. The permitted values for this service parameter may be an indication of either a high priority message or a low priority message. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

12.17.3.2.2.3 Discard eligible

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are TRUE (the message may be considered for discard), or FALSE (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

12.17.3.2.2.4 End-to-end transmission time

This is the transmission time from the TLE at the requesting device to the TLE at the receiving device. The interval is marked by two instants, the first instant being delivery to the TLE in the requesting device, and the second instant being receipt by the TLE in the destination device.

12.17.3.2.2.5 Published data size

This parameter provides the subscriber with the number of octets of the data to publish.

12.17.3.2.2.6 Subscriber T-port

This parameter identifies the subscriber UAP's associated T-port.

12.17.3.2.2.7 Subscriber TDSAP

This parameter identifies the subscriber TDSAP associated with the subscriber T-port.

12.17.3.2.2.8 Subscribing object identifier

This parameter specifies the object identifier destination in the application that is subscribed to this publication.

12.17.3.2.2.9 Publisher IPv6Address

This identifies the IPv6Address of the publisher.

12.17.3.2.2.10 Publisher TDSAP

This parameter uniquely identifies the publisher UAP's associated with the TDSAP. The TDSAP maps 1-to-1 to a UAP. The value shall be a member of the set of valid TDSAPs, as specified by the TL.

12.17.3.2.2.11 Publisher T-port

This parameter identifies the publisher UAP's associated T-port.

NOTE An implementation is able to infer this parameter from the publisher TDSAP. Thus it is included here for completeness, as required by this standard for the logical mapping to the transport data service request definition.

12.17.3.2.2.12 Publisher object identifier

This parameter identifies the publisher object that is the source of the published data.

NOTE If there is more than one entry in the list of published data, the publishing object source is an instance of the concentrator object type.

12.17.3.2.2.13 Data structure information

This parameter indicates the construct of the information to be conveyed via publication. It may indicate one of the following constructs:

- native individual value;
- native sequence of values; or
- non-native data (that is, information being tunneled via a publication service).

The details of these alternatives are as follows:

a) The data structure of each single native value is as follows:

1) Freshness value sequence number

This parameter is present if the data structure information indicates the structure of the data is a native individual value. This parameter indicates the freshness of the data.

2) Individual analog value and status

This parameter is present if the individual native value is an analog. This contains standard value status data structure that indicates information such as quality of the corresponding analog value and the analog value itself.

3) Individual digital value and status

This parameter is present if the individual native value is digital. This contains standard value status data structure that indicates information such as quality of the corresponding digital value and the digital value itself.

b) The data structure of a list of native values is as follows:

1) Publishing content version

This parameter is present if the publication is for native list data, such as sent from a concentrator object. This information ensures harmonious interpretation of the published information by the subscriber.

2) List of publish data

This parameter represents the list of data conveyed via the publish service.

3) Status and analog value

This contains standard value status data structure that indicates information such as quality of the corresponding analog value and the analog value itself.

4) Status and digital value

This contains standard value status data structure that indicates information such as quality of the corresponding digital value and the digital value itself.

c) The data contained in the publish service that is non-native is as follows:

This parameter contains the non-native data to publish. Non-native data is conveyed as a string of octets.

12.17.4 Client/server interactions

12.17.4.1 General

Client/server interactions are used for one-to-one aperiodic communications. These relationships employ on-demand queued bidirectional communication. client/server services defined by this standard are either two-part service (having two service primitives, .req and .ind), as in Figure 126, or four-part service (having four service primitives, .req, .ind, .rsp, and .cnf), as in Figure 127, Figure 128 and Figure 129.

When the ASL receives a client/server service request, it constructs a corresponding application protocol data unit (APDU) and requests queued transfer from the lower communication protocol suite. The server ASL is given a confirmed service response indication, which it delivers to the destination UAP and object.

For services with four-part primitives defined, the server UAP constructs a corresponding response. When the ASL receives the client/server service response, it constructs a response APDU and submits it to its lower communication protocol suite, which provides queue-oriented communication services to deliver the response.

In a client/server interaction, either endpoint of the communication can act as client or server or both. A client request is sent to a single destination (server). This request indicates the destination to which the response should be sent. A single response is then issued from the server.

Interactions as shown in Figure 127, Figure 128, and Figure 129 require a communication contract identifier and the communication protocol suite to be appropriately configured to support the client/server messaging requirements of the application. The bandwidth represented by the contract identifier is considered unscheduled shared bandwidth which need not be reserved solely for use by this contract.

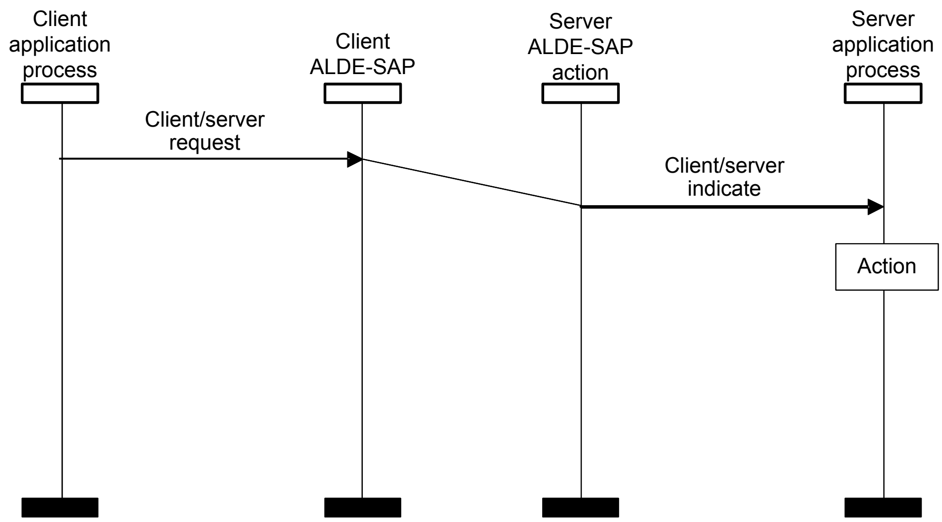


Figure 126 – client/server model two-part interactions

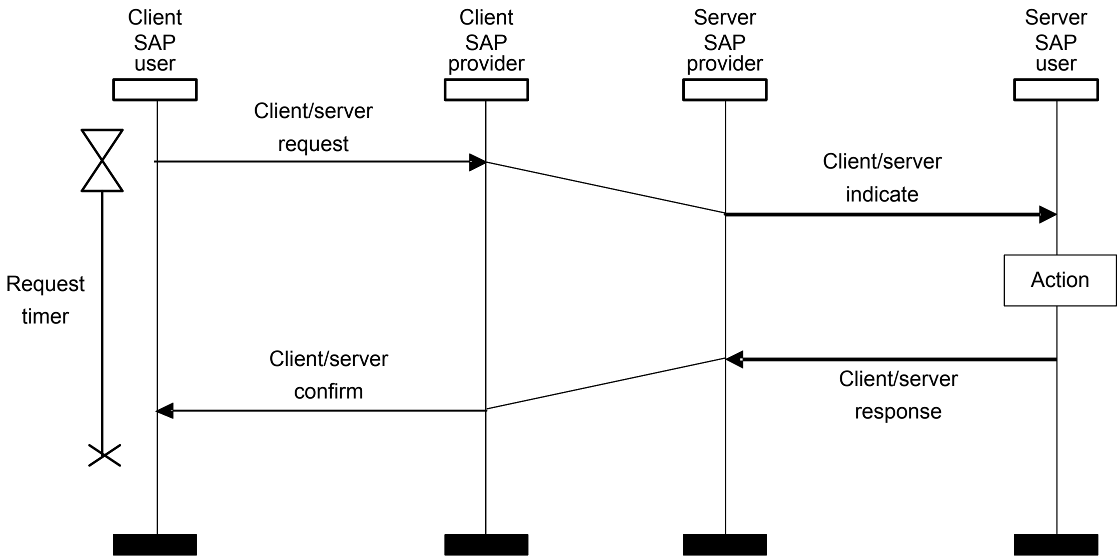


Figure 127 – client/server model four-part interactions: Successful delivery

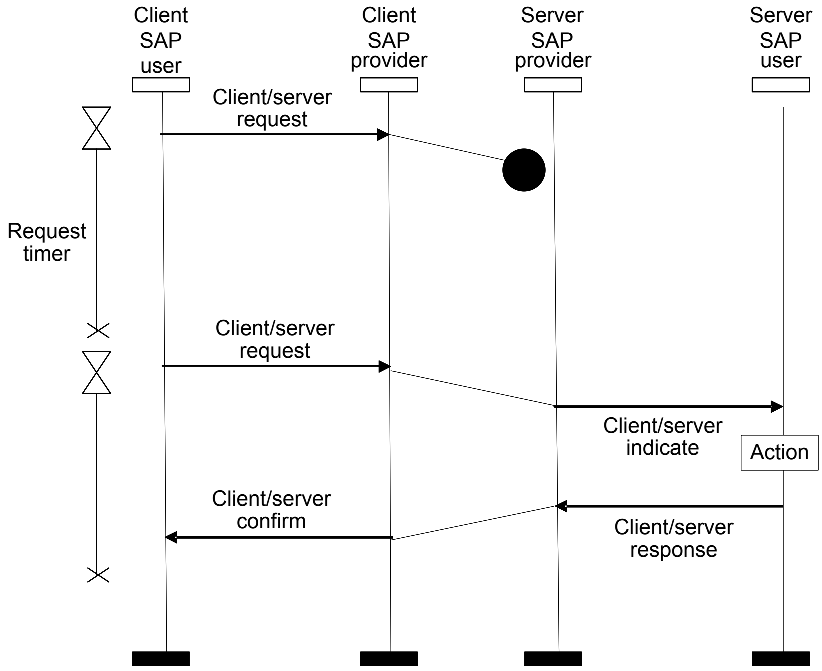


Figure 128 – client/server model four-part interactions: Request delivery failure

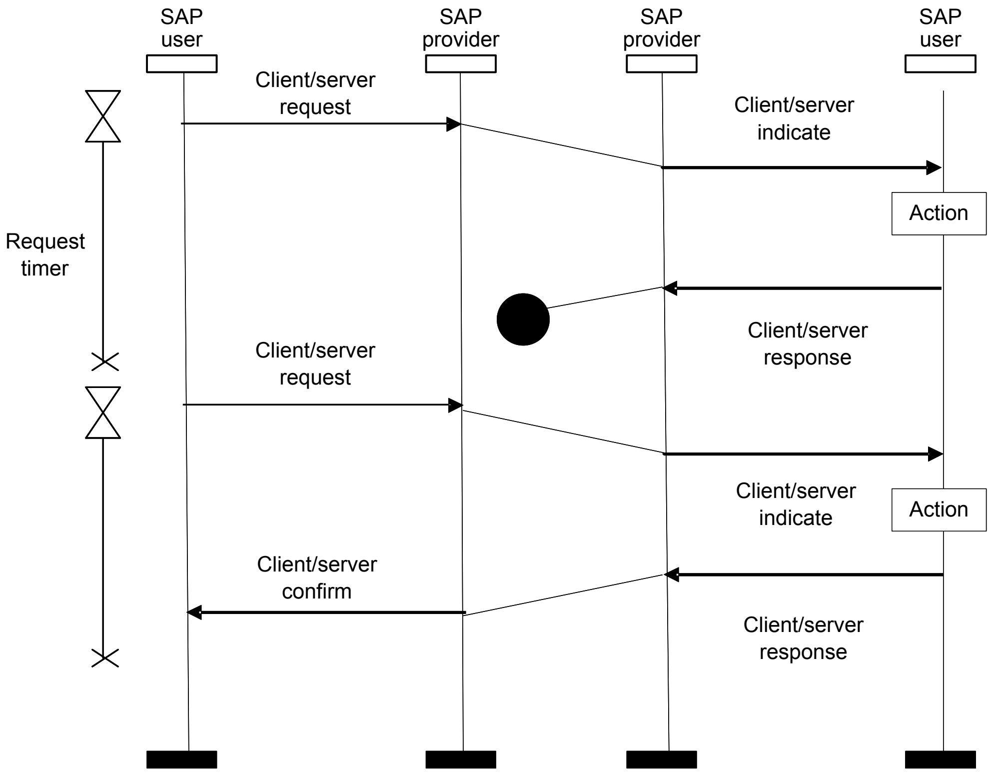


Figure 129 – client/server model four-part interactions: Response delivery failure

When a client/server association is secured, a security session is involved. To optimize elimination of connections that the UAL process knows are no longer required, a local interface to terminate the contract may be employed. Contract termination may be used to release a security session (if one exists).

To initiate communication, the client requests to send a message to a server. The client specifies the local contract identifier which indicates the server’s IPv6Address. The communication also identifies the particular source application and object making the request, the destination application and object intended to receive the request, as well as the actual service instance specific request information.

The server sends a response to the client specifying its local contract identifier, which indicates the client’s IPv6Address. The communication also specifies sufficient information to deliver the response to the appropriate application object and to collate the response with the original request.

A communication contract shall be established between client and server to carry the client’s request, and correspondingly from the server to the client to carry the server response.

Acting in the role of a client, a client may send requests to a server. Acting in the role of a server, a server may send responses to a client. The client is responsible for server timeout response and transactional integrity.

A simple server might support as few as one outstanding transaction with a particular client. If an extended delay occurs in receiving a response, the client may, for example, timeout and resend the request. If this occurs, duplicate responses may be received. If a server has resources to support multiple outstanding transactions with a client, requests and responses may arrive out of order. To support this situation, a request identifier is used to enable request/response collation.

If there is a need for multiple client or multiple server messages as part of a communication sequence, the implementation may consider employing ASL concatenation. Beyond concatenation, streaming of messages is an application process-specific responsibility outside the scope of this standard.

Communications characteristics for client/server interactions such as response timeout are local matters, beyond the scope of this standard. For example, they may be fixed by device construction, or determined by an application program within the device, or configured for the device on a per-application process basis or even a per contract basis. client/server interactions are usually used for configuration (such as process control related configuration or management object configuration) and ad-hoc exchange of information.

Client/server communications should not interfere with scheduled communications as it is essential that transmission bandwidth be allocated to support client/server messaging communication contracts. The intent is to ensure the ability to reconfigure a device.

Occasionally, only a single client/server exchange is required. This may entail substantial overhead in route and security establishment. At other times, multiple client/server exchanges occur between the same endpoints.

NOTE Any alteration of communication routes (for example, to compensate for interference) occurs transparently to the AL.

The client specifies the desired message priority for service requests; the server specifies the desired message priority for service responses.

Higher priority messages should ideally move to the front of prioritized communication protocol suite message queues supporting client/server communications. If possible, client/server message bandwidth allocation on the network should grant access first to higher priority message requests. Security is presumed to be on a contract basis, so per-message security is not provided.

Further considerations for transmission back-off, such as based on network congestion are an overall device responsibility, and are not a specific responsibility of the AL.

12.17.4.2 Client/server services

12.17.4.2.1 General

The following services are provided as client/server communications:

- read;
- write;
- execute; and
- tunnel.

NOTE Tunnel as a two-part primitive is also useful for source/sink communication.

12.17.4.2.2 Service feedback codes

Four-part client/server services provide a service feedback code to indicate the result of the service from the viewpoint of the server. A range of codes is reserved for vendor-specific additions.

12.17.4.3 Read

12.17.4.3.1 General

The read service is used to read an attribute of an object from a UAL process.

Table 275 defines the read service primitives.

Table 275 – Read service

Parameter name	Request	Indication	Response	Confirm
Argument	M	M	—	—
Service contract identifier	M	—	—	—
Priority	M	—	—	—
Discard eligible	M	—	—	—
End-to-end transmission time	—	M	—	—
Forward congestion notification	—	M	—	—
Server T-port	M		—	—
Server TDSAP	—	M	—	—
Server object identifier	M	M(=)	—	—
Client IPv6Address	—	M	—	—
Client TDSAP	M	—	—	—
Client T-port	—	M	—	—
Client object identifier	M	M(=)	—	—
Application request ID	M	M(=)	—	—
Data to be read	M	M(=)	—	—
Attribute identifier	M	M(=)	—	—
Attribute index(es)	C	C(=)	—	—

Parameter name	Request	Indication	Response	Confirm
Result	—	—	M	M
Service contract identifier	—	—	M	—
Priority	—	—	M	—
Discard eligible	—	—	M	—
End-to-end transmission time	—	—	—	M
Forward congestion notification	—	—	—	M
Forward congestion notification echo	—	—	M	M(=)
Server IPv6Address	—	—	—	M
Server TDSAP	—	—	M	—
Server T-port	—	—	—	M
Server object identifier	—	—	M	M(=)
Client T-port	—	—	M	—
Client TDSAP	—	—	—	M
Client object identifier	—	—	M	M(=)
Application request ID	—	—	M	M(=)
Value read	—	—	M	M(=)
Service feedback code	—	—	M	M(=)
Value size	—	—	C	C(=)
Data value	—	—	C	C(=)

12.17.4.3.2 Arguments

12.17.4.3.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.3.2.2 Priority

See 12.17.3.2.2.2.

12.17.4.3.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.3.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.3.2.5 Forward congestion notification

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

12.17.4.3.2.6 Server T-port

This parameter identifies the server UAP’s associated T-port.

12.17.4.3.2.7 Server TDSAP

This parameter identifies the server TDSAP associated with the server T-port.

12.17.4.3.2.8 Server object identifier

This parameter identifies a server object from which data is desired to be read.

12.17.4.3.2.9 Client/source address

This parameter identifies the IPv6Address for the client of this request.

12.17.4.3.2.10 Client/source TDSAP

This parameter identifies the client or source UAP's associated TDSAP. The UAP contains the object originating the request.

12.17.4.3.2.11 Client T-port

This parameter identifies the client UAP's associated T-port.

12.17.4.3.2.12 Client object identifier

This parameter identifies the client object that is initiating the service request.

12.17.4.3.2.13 Application request identifier

An identifier provided by the UAP to uniquely represent this request.

12.17.4.3.2.14 Data to be read

This parameter identifies the data values that the client desires to read.

12.17.4.3.2.15 Attribute identifier

This parameter identifies the attribute of the server object, the value of which is desired to be read.

12.17.4.3.2.16 Attribute index/indices

This parameter identifies the index/indices for the information of interest from the attribute. There may be:

- no index, such as for a scalar value;
- one index, for example, to access
  - an element of a singly-dimensioned array, or
  - a member of a standard data structure; or
- two indices, for example, to access
  - an element of a doubly-dimensioned array, or
  - a member of a data structure that is contained in a singly-dimensioned array of identical standard data structures, or
  - a singly-dimensioned of a doubly-dimensioned array, or
  - a singly-dimensioned slice of a singly-dimensioned array of identical standard data structures, extracting as a singly-dimensioned array a cross-sectional slice of a single member through those identical data structures.

12.17.4.3.3 Results

12.17.4.3.3.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.3.3.2 Priority

See 12.17.3.2.2.2.

12.17.4.3.3.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.3.3.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.3.3.5 Forward congestion notification

See 12.17.4.3.2.5.

12.17.4.3.3.6 Forward congestion notification echo

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

12.17.4.3.3.7 Server IPv6Address

This parameter identifies the IPv6Address for the server for this request.

12.17.4.3.3.8 Server TDSAP

See 12.17.4.3.2.7.

12.17.4.3.3.9 Server T-port

See 12.17.4.3.2.6.

12.17.4.3.3.10 Server object identifier

See 12.17.4.3.2.8.

12.17.4.3.3.11 Client T-port

The UAP contains the object originating the request. See 12.17.4.3.2.11.

12.17.4.3.3.12 Client TDSAP

The UAP contains the object originating the request. See 12.17.4.3.2.10.

12.17.4.3.3.13 Client object identifier

See 12.17.4.3.2.12.

12.17.4.3.3.14 Application request identifier

This parameter is an identifier provided by the client to uniquely represent this request.

12.17.4.3.3.15 Value read

The value read indicates the result of the requested operation, and if the read was successful, the size and value of the object attribute to be read.

12.17.4.3.3.16 Service feedback code

The service feedback code indicates if the requested operation was successful or not. If not successful, it provides information indicating why it was not successful.

12.17.4.3.3.17 Value size

Value size indicates the number of octets contained in the data value. It is present if and only if the corresponding service feedback code indicates success.

12.17.4.3.3.18 Data value

Data value is the data value that was read from the identified server object, attribute, and attribute index. It is present if and only if the corresponding service feedback code indicates success, and the value size is non-zero.

12.17.4.4 Write

12.17.4.4.1 General

The write service is used to write a value or set of values to one or more attributes of one or more objects in an application process.

A write to a structure containing both writeable and read-only elements is permitted. In this situation, the read-only elements shall be unaffected.

Table 276 defines the write service primitives.

Table 276 – Write service

Parameter name	Request	Indication	Response	Confirm
Argument	M	M	—	—
Service contract identifier	M	—	—	—
Priority	M	—	—	—
Discard eligible	M	—	—	—
End-to-end transmission time	—	M	—	—
Forward congestion notification	—	M	—	—
Server T-port	M	—	—	—
Server TDSAP	—	M	—	—
Server object identifier	M	M(=)	—	—
Client IPv6Address	—	M	—	—
Client TDSAP	M	—	—	—
Client T-port	—	M	—	—
Client object identifier	M	M(=)	—	—
Application request ID	M	M(=)	—	—
Data to write	M	M(=)	—	—
Attribute identifier	M	M(=)	—	—
Attribute index(es)	M	M(=)	—	—
Value size	M	M(=)	—	—
Data value	M	M(=)	—	—
Result	—	—	M	M
Service contract identifier	—	—	M	—
Priority	—	—	M	—
Discard eligible	—	—	M	—
End-to-end transmission time	—	—	—	M
Forward congestion notification	—	—	—	M
Forward congestion notification echo	—	—	M	M(=)
Server IPv6Address	—	—	—	M
Server TDSAP	—	—	M	—
Server T-port	—	—	—	M(=)
Server object identifier	—	—	M	M(=)
Client T-port	—	—	M	—
Client TDSAP	—	—	—	M
Client object identifier	—	—	M	M(=)
Application request ID	—	—	M	M(=)
Service feedback code	—	—	M	M(=)

12.17.4.4.2 Arguments

12.17.4.4.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.4.2.2 Priority

See 12.17.3.2.2.2.

12.17.4.4.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.4.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.4.2.5 Forward congestion notification

See 12.17.4.3.3.6.

12.17.4.4.2.6 Server T-port

See 12.17.4.3.2.6

12.17.4.4.2.7 Server TDSAP

See 12.17.4.3.2.7

12.17.4.4.2.8 Server object identifier

See 12.17.4.3.2.8

12.17.4.4.2.9 Client/source address

This parameter identifies the client or source UAP's associated IPv6Address. The UAP contains the object originating the request.

12.17.4.4.2.10 Client/source TDSAP

This parameter identifies the client or source UAP's associated TDSAP. The UAP contains the object originating the request.

12.17.4.4.2.11 Client T-port

See 12.17.4.3.2.11.

12.17.4.4.2.12 Client object identifier

See 12.17.4.3.2.12.

12.17.4.4.2.13 Application request identifier

An identifier provided by the UAP to uniquely represent this request.

12.17.4.4.2.14 Data to write

This parameter identifies the target attribute and data value that the client desires to write.

12.17.4.4.2.15 Attribute identifier

This parameter identifies the attribute of the server object, the value of which is desired to be read.

12.17.4.4.2.16 Attribute index/indices

This parameter identifies the index/indices for the information of interest from the attribute. See 12.17.4.3.2.16.

12.17.4.4.2.17 Value size

Value size indicates the number of octets contained in data value.

12.17.4.4.2.18 Data value

Data value is the data value that is desired to be written to the identified server object, attribute, and attribute index.

12.17.4.4.3 Results

12.17.4.4.3.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.4.3.2 Priority

See 12.17.3.2.2.2.

12.17.4.4.3.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.4.3.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.4.3.5 Forward congestion notification

See 12.17.4.3.2.5.

12.17.4.4.3.6 Forward congestion notification echo

See 12.17.4.3.3.6.

12.17.4.4.3.7 Server IPv6Address

See 12.17.4.3.3.7.

12.17.4.4.3.8 Server TDSAP

See 12.17.4.3.2.7.

12.17.4.4.3.9 Server T-port

See 12.17.4.3.2.6.

12.17.4.4.3.10 Server object identifier

This parameter identifies a server object to which data is desired to be written.

12.17.4.4.3.11 Client/source T-port

This parameter identifies the client UAP's associated T-port.

12.17.4.4.3.12 Client TDSAP

This parameter identifies the client TDSAP associated with the T-port. The UAP contains the object originating the request.

12.17.4.4.3.13 Client object identifier

See 12.17.4.3.2.12.

12.17.4.4.3.14 Application request identifier

See 12.17.4.3.3.14.

12.17.4.4.3.15 Service feedback code

See 12.17.4.3.3.16.

12.17.4.5 Execute

12.17.4.5.1 General

The execute service is used to execute a network visible method on an object.

NOTE Use of the execute service to establish a callback method is one way to provide a server with adequate time for a delayed response, providing information back to the client via a callback, rather than having to provide timely execution results in the response.

Table 277 defines the execute service primitives.

14985

Table 277 – Execute service

Parameter name	Request	Indication	Response	Confirm
Argument	M	M	—	—
Service contract identifier	M	—	—	—
Priority	M	—	—	—
Discard eligible	M	—	—	—
End-to-end transmission time	—	M(=)	—	—
Forward congestion notification	—	M	—	—
Server T-port	M	—	—	—
Server TDSAP	—	M	—	—
Server object identifier	M	M(=)	—	—
Client IPv6Address	—	M	—	—
Client TDSAP	M	—	—	—
Client T-port	—	M	—	—
Client object identifier	M	M(=)	—	—
Application request ID	M	M(=)	—	—
Method to execute	M	M(=)	—	—
Method identifier	M	M(=)	—	—
Size of input parameters	M	M(=)	—	—
Input parameters	C	C(=)	—	—
Result	—	—	M	M
Service contract identifier	—	—	M	—
Priority	—	—	M	—
Discard eligible	—	—	M	—
End-to-end transmission time	—	—	—	M
Forward congestion notification	—	—	—	M
Forward congestion notification echo	—	—	M	M(=)
Server IPv6Address	—	—	—	M
Server TDSAP	—	—	M	—
Server T-port	—	—	—	M
Server object identifier	—	—	M	M(=)
Client T-port	—	—	M	—
Client TDSAP	—	—	—	M
Client object identifier	—	—	M	M(=)
Application request ID	—	—	M	M(=)
Execution result	—	—	M	M(=)
Service feedback code	—	—	M	M(=)
Size of output parameters	—	—	M	M(=)
Output parameters	—	—	C	C(=)

14986

12.17.4.5.2 Argument

12.17.4.5.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.5.2.2 Priority

See 12.17.3.2.2.2.

12.17.4.5.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.5.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.5.2.5 Forward congestion notification

See 12.17.4.3.2.5.

12.17.4.5.2.6 Server T-port

See 12.17.4.3.2.6.

12.17.4.5.2.7 Server TDSAP

See 12.17.4.3.2.7.

12.17.4.5.2.8 Server object identifier

See 12.17.4.3.2.8.

12.17.4.5.2.9 Client/source address

See 12.17.4.3.2.9

12.17.4.5.2.10 Client/source TDSAP

12.17.4.3.2.10.

12.17.4.5.2.11 Client T-port

See 12.17.4.3.2.11.

12.17.4.5.2.12 Client object identifier

See 12.17.4.3.2.12.

12.17.4.5.2.13 Application request identifier

See 12.17.4.3.2.13.

12.17.4.5.2.14 Method identifier

This parameter identifies the method of the server object that is desired to be executed.

12.17.4.5.2.15 Size of input parameters

Size of input parameters indicates the number of octets contained in input parameters.

NOTE Execute requests and responses include the size in octets of the contained parameter stream to enable parsing (this is especially useful in APDU concatenation scenarios).

12.17.4.5.2.16 Input parameters

The input parameters' string is an octet string that contains the input parameters for the method that is being requested to be executed. This is present if and only if size of input parameters is present and has a value greater than zero.

12.17.4.5.3 Result

12.17.4.5.3.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.4.5.3.2 Priority

See 12.17.3.2.2.2.

12.17.4.5.3.3 Discard eligible

See 12.17.3.2.2.3.

12.17.4.5.3.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.4.5.3.5 Forward congestion notification

See 12.17.4.3.2.5.

12.17.4.5.3.6 Forward congestion notification echo

See 12.17.4.3.3.6.

12.17.4.5.3.7 Server IPv6Address

See 12.17.4.3.3.7.

12.17.4.5.3.8 Server TDSAP

See 12.17.4.3.2.7.

12.17.4.5.3.9 Server T-port

See 12.17.4.3.2.6.

12.17.4.5.3.10 Server object identifier

See 12.17.4.4.3.10.

12.17.4.5.3.11 Client/source T-port

See 12.17.4.4.3.11.

12.17.4.5.3.12 Client TDSAP

See 12.17.4.4.3.12.

12.17.4.5.3.13 Client object identifier

See 12.17.4.3.2.12.

12.17.4.5.3.14 Application request identifier

See 12.17.4.3.3.14.

12.17.4.5.3.15 Execution result

This contains the result of the method execution service request.

12.17.4.5.3.16 Service feedback code

The service feedback code indicates if the corresponding method execution was successful or not. If not successful, it provides information indicating why it was not successful.

12.17.4.5.3.17 Size of output parameters

Size of output parameters indicates the number of octets contained in output parameters.

12.17.4.5.3.18 Output parameters

The output parameters' string is an octet string that contains the output parameters for the method that was executed. This is present if and only if size of output parameters is present and has a value greater than zero.

12.17.5 Unscheduled acyclic queued unidirectional messages (source/sink)

12.17.5.1 General

Unscheduled acyclic queued unidirectional messaging is also sometimes referred to as source/sink messaging. This interaction type is used for alerts. Messages sent using this protocol are queued by the lower communication layers for transmission. Message receipt is unconfirmed. There is no application process flow or rate control or lost message detection for this mode of interaction. Like client/server communications, these communications require use of a communication contract, and specify message priority on a per-message basis.

Bandwidth for source/sink communications is not considered dedicated, but rather is considered to come from non-dedicated (i.e., shared) bandwidth.

Unscheduled acyclic unidirectional interactions in this standard support on-demand one-to-one queued message distribution. Alert reports for network communication are always issued from one initiator, the ARMO, and are always sent to one type of message recipient, an alert-receiving object (ARO).

Acknowledgment of reception of an individual alert may only be issued from one alert report recipient (ARO), and is sent to the (ARMO) object that reported the alert.

The following services are provided to support unscheduled acyclic queued unidirectional message communications:

- AlertReport;
- AlertAcknowledge; and
- Tunnel.

The Tunnel service is included as a source/sink service so that it will be able to take advantage of multi-cast capabilities in the future. Such a potential development is a subject for future standardization.

12.17.5.2 AlertReport service

12.17.5.2.1 General

AlertReport is used to report an alert using queued unidirectional communication services. The content of the alert report depends on the type of alert being reported and the category of the alert. AlertReports may be retried until an AlertAcknowledge for the AlertReport has been received.

Figure 130, Figure 131, and Figure 132 indicate alert reporting message sequencing.

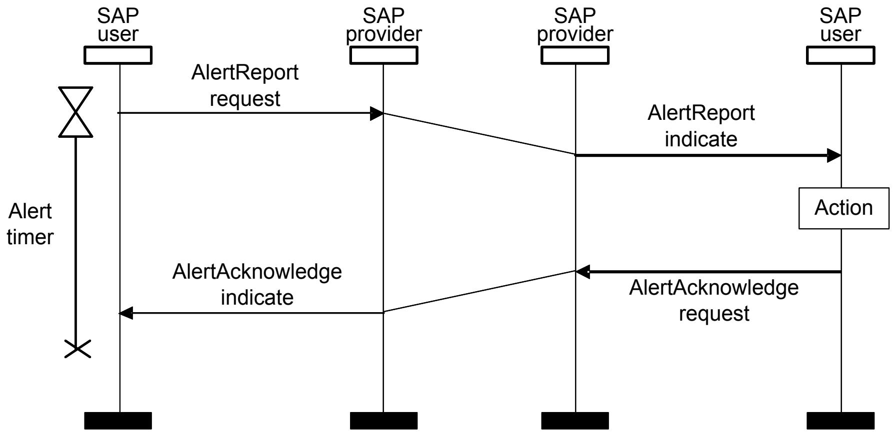


Figure 130 – AlertReport and AlertAcknowledge, delivery success

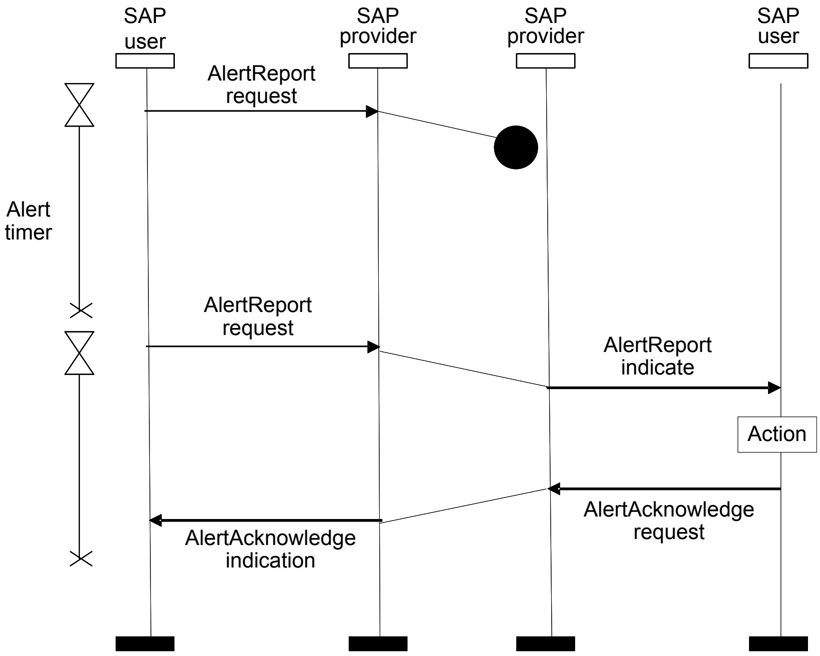


Figure 131 – AlertReport, delivery failure

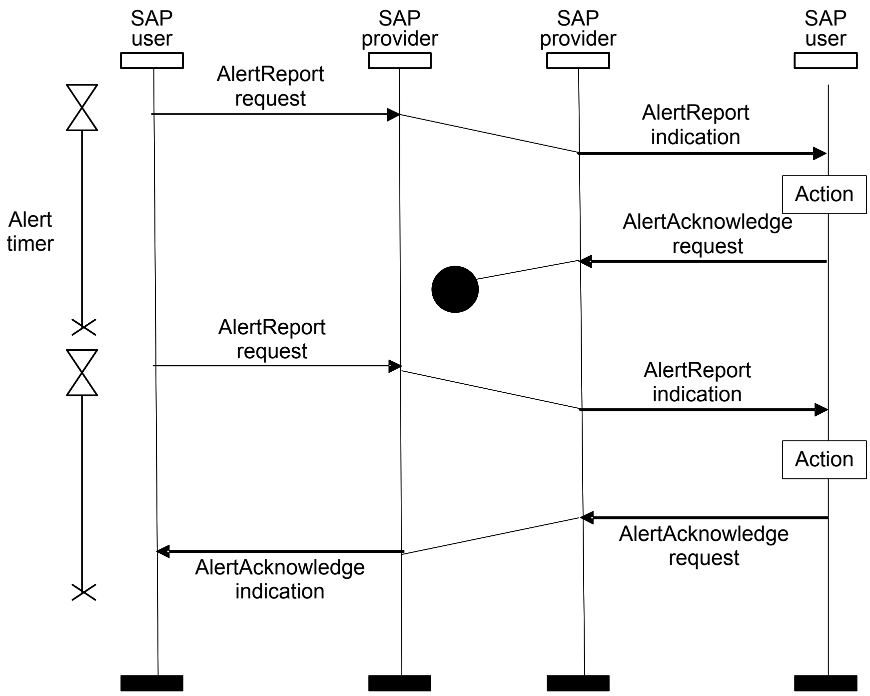


Figure 132 – AlertReport, acknowledgment failure

NOTE 1 AlertReport timeout/retry policy is defined on the ARMO. See 6.2.7.2 for further details.

Alert reporting employs two separate two-part application communication services. To report an alert, the AP uses the AlertReport service. In this version of the standard, the recipient of an AlertReport shall acknowledge the AlertReport by using the AlertAcknowledge service.

NOTE 2 The use of two separate services, AlertReport and AlertAcknowledgment, enable a single alert to be sent to multiple destinations in a future revision to this standard.

Monitoring and checking for acknowledgment, as well as re-reporting an alert condition for which acknowledge has not been received is the responsibility of the alert reporting device. Re-reporting an alert that is no longer prevalent, and for which an AlertAcknowledge indication has not been received, is a local matter. For example, if a diagnostic situation occurs and an alert is reported, and then the reporting device reboots such that the diagnostic situation is no longer prevailing, the device might not re-report the diagnostic alert that was in effect prior to reboot even though no AlertAcknowledge was received.

AlertReport employs the same communication model as a two-part client/server primitive. Table 278 defines the service primitives for the AlertReport service.

Table 278 – AlertReport service

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	—
Priority	M	—
Discard eligible	M	—
End-to-end transmission time	—	M
ARMO TDSAP	M	—
ARMO T-port	—	M
ARMO	M	M(=)
Sink T-port	M	—
Sink TDSAP	—	M
Sink object identifier	M	M(=)
Individual alert report	M	M(=)
Individual alert identifier	M	M(=)
Alert detector T-port	M	M(=)
Alert detector object	M	M(=)
Detection time	M	M(=)
Alert class	M	M(=)
Alarm direction	C	C(=)
Alert category	M	M(=)
Alert priority	M	M(=)
Alert type	M	M(=)
Associated-data size	M	M(=)
Associated data	O	O(=)

12.17.5.2.2 Arguments

12.17.5.2.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.5.2.2.2 Priority

See 12.17.3.2.2.2.

12.17.5.2.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.5.2.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.5.2.2.5 Alert reporting management object TDSAP

This parameter indicates the TDSAP of the application which is issuing this alert report.

12.17.5.2.2.6 Alert reporting management object T-port

This parameter indicates the T-port of the application which is issuing this alert report.

12.17.5.2.2.7 Alert reporting management object

This parameter represents the object identifier of the ARMO that is reporting the alert.

12.17.5.2.2.8 Sink T-port

This parameter identifies the sink UAP's associated T-port.

12.17.5.2.2.9 Sink TL data service access point

This parameter indicates the TDSAP corresponding to sink T-port.

12.17.5.2.2.10 Sink object identifier

This parameter specifies the destination sink object in the application to which this service request is to be sent.

12.17.5.2.2.11 Alert source IPv6Address

This parameter identifies the IPv6Address of the source of this request.

12.17.5.2.2.12 Alert source TDSAP

This parameter identifies the source UAP's associated TDSAP.

12.17.5.2.2.13 Source T-port

This parameter identifies the transmitting application source UAP associated with the T-port.

12.17.5.2.2.14 Individual alert report

This parameter contains an individual alert being reported by this service invocation.

12.17.5.2.2.15 Individual alert identifier

This parameter uniquely identifies the individual alert report. Separate identifier value sequences for the alert reporting categories shall be maintained. The value of this parameter shall be monotonically increasing, and shall wrap around when the maximum value is reached. It is included when an individual alert report is acknowledged. It also is used by an alert receiver to determine if an alert report or a set of alert reports of a particular category have been missed. If a missed report condition is detected, an alarm recovery operation should be performed. Refer to the Alarm_Regen attributes of the ARMO in 6.2.7.2 for further details on triggering the regeneration.

12.17.5.2.2.16 Alert source transport port

This parameter identifies the UAP containing the object that detected the alert via its associated T-port.

12.17.5.2.2.17 Alert source object

The alert source object indicates the object instance that detected the alarm condition.

NOTE The alert reporting management object reports alert conditions detected by one or more alert detecting objects.

12.17.5.2.2.18 Detection time

This parameter specifies the time at which the alert condition was detected. This value indicates the network time at which the alert was detected. How time information is made available to an application reporting an alert is a device local matter, not specified by this standard.

NOTE Translating network time to social time (wall clock time), when desired, is performed in the gateway. See 5.6 for further details.

12.17.5.2.2.19 Alert class

This parameter indicates if this is an event (stateless) or alarm (state-oriented) type of alert.

12.17.5.2.2.20 Alarm direction

For alerts that are state-oriented (alarms), this indicates if the report is for an alarm condition, or a return to normal from an alarm condition.

12.17.5.2.2.21 Alert category

Alert category indicates if the alert is a device diagnostic alert, a communication diagnostic alert, a security alert, or a process alert.

12.17.5.2.2.22 Alert priority

Alert priority is a value that suggests the importance of the alert. A larger value implies a more important alert. Host systems map device priorities into host alert priorities that usually include urgent, high, medium, low, and journal. The recommended mapping of alert priority values into these categories is as follows:

- 0..2: journal
- 3..5: low
- 6..8: medium
- 9..11: high
- 12..15: urgent

Since the interpretation of alert priorities occurs primarily in the originating and intended receiving devices, other assignments that reflect a differing categorization are permitted.

12.17.5.2.2.23 Alert type

Alert type provides additional information regarding the alert, specific to the alert category.

12.17.5.2.2.24 Associated-data size

Associated-data size specifies the size of any alert-specific data conveyed with the alert.

12.17.5.2.2.25 Associated data

Associated data provides a means of conveying alert-specific data.

12.17.5.3 AlertAcknowledge service

12.17.5.3.1 General

AlertAcknowledge is a two-part service that is used to acknowledge an individual alert to an alert reporting management object. For unicast alert reports, receipt of an AlertAcknowledge shall result in the ceasing of AlertReport retry requests for the corresponding individual alert.

An AlertAcknowledge shall be sent for every AlertReport received.

NOTE If a duplicate AlertReport has been received, either the application that sent the AlertReport did not receive the AlertAcknowledge within its timeout/retry time, or the AlertAcknowledgment message was not received. Since the application sending the AlertAcknowledge does not know which situation occurred, a duplicate acknowledgment is sent.

The AlertAcknowledge service is described in Table 279.

Table 279 – AlertAcknowledge service

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	—
Priority	M	—
Discard eligible	M	—
End-to-end transmission time	—	M
Source IPv6Address	—	M
Source TDSAP	M	—
Source T-port	—	M
Source object identifier	M	M(=)
Destination transport port	M	—
Destination TDSAP	—	M
Destination object identifier	M	M(=)
Individual alert identifier	M	M(=)

12.17.5.3.2 Arguments

12.17.5.3.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.5.3.2.2 Priority

See 12.17.3.2.2.2.

12.17.5.3.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.5.3.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.5.3.2.5 Source IPv6Address

This parameter identifies the IPv6Address for the source of this request.

12.17.5.3.2.6 Source TDSAP

This parameter identifies the service primitive source AP’s associated TDSAP. The TDSAP maps 1-to-1 to a UAP.

12.17.5.3.2.7 Source T-port

This parameter identifies the transmitting application source UAP associated with the T-port.

12.17.5.3.2.8 Source object identifier

This parameter identifies the object that is initiating the alert acknowledgment.

12.17.5.3.2.9 Destination T-port

This parameter identifies the application process to receive the alert acknowledgment as a single application is associated with a T-port.

12.17.5.3.2.10 Destination TDSAP

This parameter identifies the TDLE SAP corresponding to the destination transport port.

12.17.5.3.2.11 Destination object identifier

This parameter identifies the object in the application process of the device to receive the acknowledgment.

12.17.5.3.2.12 Individual alert identifier

This parameter identifies the individual alert that is being acknowledged.

12.17.6 Client/server and source/sink commonalities

12.17.6.1 Individual or concatenated messaging for client/server and/or source/sink

Client/server and source/sink messages may be sent as an individual transport service data unit (TSDU), or may be concatenated together within a single TSDU. Concatenation supports both four-part primitive messages (requests and responses) and two-part primitive messages (requests only). Concatenation allows client/server messages to be combined in the TSDU with source/sink messages. How APDU concatenation is determined and accomplished is a device local matter. It is recommended that concatenations refrain from including more than two services, as this may result in more bursty communication.

NOTE 1 The discussion of stretch acknowledgment violation in IETF RFC 2525 provides background on message acknowledgment concatenation and the ramifications of having more than two such acknowledgments in a PDU.

NOTE 2 Publish/subscribe already supports including multiple values from a UAP in a single message. See 12.15.2.5 and 12.15.2.6 for further details.

Concatenation may be used to reduce transmission overhead and/or deliver a set of messages to the corresponding ASL as a unit.

All messages within the concatenation shall have a common message priority, and shall indicate communication via a common communication contract.

The number of ASL services that may be concatenated by an ASL building the concatenated PDU is limited by the maximum APDU size corresponding to the communication contract to be used for the messages.

The ASL receiving a concatenated APDU is required to parse and handle each APDU individually from start to finish until the end of the TSDU has been met. If a protocol error is detected during parsing of a concatenated APDU, a single malformed APDU error is indicated and the remaining portion of the APDU shall be discarded.

An ASL concatenation of services may contain:

- homogeneous ASL service primitives (e.g., all service requests); or
- heterogeneous ASL service primitives (e.g., for client/server flows, requests and responses, which may be mixed).
- homogeneous application communication flow primitives (e.g., all client/server); or
- heterogeneous application communication flow type primitives (e.g., client/server and source/sink).

The AL itself makes no requirements on how responses to services included in a concatenation are returned; determination of such responses is at the discretion of the receiving application. For example, if a concatenated client/server request contains service requests (A, B, C) and another concatenated client/server service request contains service requests (D, E), the client/server responses for these may:

– not be required (e.g., A, B, and D may be four-part services that require a response, but C and E may be two-part services that do not require a response);

– not be concatenated at all, and returned in any order (e.g., response A, response B, response E, response D, response C, all in separate APDUs);

– be partially concatenated, in any order (e.g., response returned may be B,C in one APDU, A in another APDU);

– employ a single APDU to respond to a concatenated request, but responses may be concatenated in any order (e.g., response returned may be concatenated as BCA);

– be fully concatenated in the same order (e.g., response returned concatenated as ABC);

– be entirely differently concatenated than the requests received (e.g., response may be returned BD, ACE).

How and when an ASL initiating a communication determines when to create a concatenation as well as when to deliver the concatenation to the lower communication protocol suite for conveyance is a local matter. However, this standard shall specify the overall structure of a TSDU containing concatenated APDUs.

This standard does not prescribe the order in which services included in concatenated messaging are handled by the destination. Thus the order of responses is not required to be the same as the order of the requests included in the concatenation.

NOTE 3 An implementation that defines local services to bracket concatenated constructions is able to provide further control over concatenation content.

Figure 133 illustrates a concatenated response for multiple outstanding write requests with no message loss. Timeouts are described in 12.12.4.2.2.1.

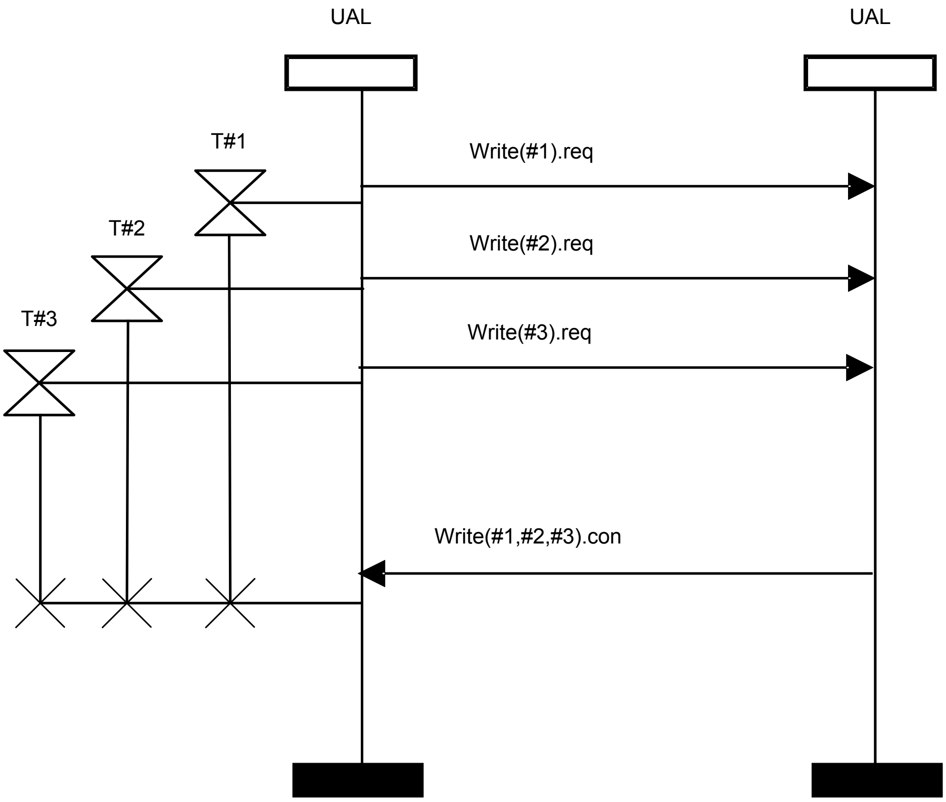


Figure 133 – Concatenated response for multiple outstanding write requests (no message loss)

12.17.6.2 Application sublayer common services for client/server and source/sink messaging – Tunnel

12.17.6.2.1 General

The tunnel service may use either a two-part (source/sink) or a four-part (client/server) communication service primitive model.

Information to enable matching service primitives, effecting appropriate application level handling of error situations such as duplicate message detection and detection of out-of-order delivery, etc. shall be the responsibility of the application process to include within the non-native payload of the tunnel messages.

The tunnel service can be used to encapsulate both client/server and source/sink communications as defined by this standard. This service identifies a message as destined for a non-native protocol tunnel. The service identifier is required so that the ASL can parse a message and determine whether to pass it on to a legacy protocol tunnel or handle it as a native message. The tunnel service provides a single level of message encapsulation for a protocol tunnel. The non-native APDU is passed through to the destination object specified in the tunnel service request.

A tunneling application may establish the retry policy for two-part (source/sink) tunnel requests that it sends.

Table 280 defines the tunnel service primitives.

Table 280 – Tunnel service

Parameter name	Request	Indication	Response	Confirm
Argument	M	M	—	—
Service contract identifier	M	—	—	—
Priority	M	—	—	—
Discard eligible	M	—	—	—
End-to-end transmission time	—	M	—	—
Forward congestion notification	—	M	—	—
Application destination T-port	M	—	—	—
Application destination TDLE SAP	—	M	—	—
Application destination object identifier	M	M(=)	—	—
Application source IPv6Address	—	M	—	—
Application source TLDE SAP	M	—	—	—
Application source T-port	—	M	—	—
Application source object identifier	M	M(=)	—	—
Payload size (in octets)	M	M(=)	—	—
Tunnel payload data	M	M(=)	—	—
Result	—	—	U	U
Service contract identifier	—	—	M	—
Priority	—	—	M	—
Discard eligible	—	—	M	—
End-to-end transmission time	—	—	—	M
Forward congestion notification	—	—	—	M
Forward congestion notification echo	—	—	M	M(=)
Application destination T-port	—	—	M	—
Application destination TDLE SAP	—	—	—	M
Application destination object identifier	—	—	M	M(=)
Application source IPv6Address	—	—	—	M
Application source TLDE SAP	—	—	M	—
Application source T-port	—	—	—	M(=)
Application source object identifier	—	—	M	M(=)
Payload size (in octets)	—	—	M	M(=)
Tunnel payload data	—	—	M	M(=)

12.17.6.2.2 Arguments

12.17.6.2.2.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.6.2.2.2 Priority

See 12.17.3.2.2.2.

12.17.6.2.2.3 Discard eligible

See 12.17.3.2.2.3.

12.17.6.2.2.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.6.2.2.5 Forward congestion notification

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

12.17.6.2.2.6 Application destination T-port

This parameter identifies the UAP at the destination for this service request.

12.17.6.2.2.7 Application destination TDSAP

This parameter represents the TDSAP corresponding to the AL transport destination port.

12.17.6.2.2.8 Application destination object identifier

This parameter identifies the object in the receiving application.

12.17.6.2.2.9 Application source IPv6Address

This parameter identifies the IPv6Address for the client of this request.

12.17.6.2.2.10 Application source TDSAP

This parameter identifies the application source UAP's associated TDSAP. The TDSAP maps 1-to-1 to a UAP. The UAP contains the object originating the request in the client.

12.17.6.2.2.11 Application source T-port

This parameter identifies the UAP that is the source for this service request.

12.17.6.2.2.12 Application source object identifier

This parameter indicates the application source object that is originating the tunnel service request.

12.17.6.2.2.13 Payload size

This parameter indicates the number of octets of the tunnel payload parameter.

12.17.6.2.2.14 Tunnel payload data

This parameter represents the data (e.g., legacy protocol APDU) that is to be conveyed to the server object.

12.17.6.2.3 Results

12.17.6.2.3.1 Service contract identifier

See 12.17.3.2.2.1.

12.17.6.2.3.2 Priority

See 12.17.3.2.2.2.

12.17.6.2.3.3 Discard eligible

See 12.17.3.2.2.3.

12.17.6.2.3.4 End-to-end transmission time

See 12.17.3.2.2.4.

12.17.6.2.3.5 Forward congestion notification

This parameter indicates if the service response encountered network congestion on its path from the server to the client.

12.17.6.2.3.6 Forward congestion notification echo

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

12.17.6.2.3.7 Application destination T-port

This parameter identifies the UAP at the destination for this service request.

12.17.6.2.3.8 Application destination TDSAP

This parameter represents the TDSAP corresponding to AL transport destination port.

12.17.6.2.3.9 Application destination object identifier

This parameter indicates the application destination object that is originating the tunnel service request.

12.17.6.2.3.10 Application source IPv6Address

This parameter identifies the IPv6Address for the client of this request.

12.17.6.2.3.11 Application source TDSAP

This parameter identifies the application source UAP's associated TDSAP. The TDSAP maps 1-to-1 to a UAP. The UAP contains the object originating the request in the client.

12.17.6.2.3.12 Application source T-port

This parameter indicates the application source object that is originating the tunnel service request.

12.17.6.2.3.13 Application source object identifier

This parameter indicates the application source tunnel object that is originating the tunnel service request.

12.17.6.2.3.14 Payload size

This parameter indicates the number of octets of the tunnel payload parameter.

12.17.6.2.3.15 Tunnel payload data

This parameter represents the data (e.g., legacy protocol APDU) that is to be conveyed to the server object.

12.18 AL flow use of lower layer services

12.18.1 General

All types of messaging (e.g., publication, client/server, source/sink, bulk transfer) and all qualities of service may flow through the common TDSAP for the application process. Table 281 indicates the mapping of the AL flows to the TL services provided.

Table 281 – Application flow characteristics

Application flow type	Buffered or queued	Periodic (scheduled)	Order sensitive	Reliable	Unacknowledged	Message importance		
						High	Medium	Low
Periodic publish/ subscribe	Buffered	Yes	Yes	No	Yes	a)	a)	a)
Client/ server	Queued	No	No	No	Yes	a)	a)	a)
Source/sink	Queued	No	No	No	Yes	a)	(Note 1)	a)

a) The message importance alternative that is selected is the one that applies.

Note 1 No use case identified.

12.18.2 AL use of TDSAPs

The ASL communicates with the lower layers of the communication protocol suite via TDSAPs. The information communicated to the TL for TDSAP mapping is a subset of the information communicated to the ASL from the UAP at the ASAP. There is one well-known TDSAP in this standard, TDSAP number 0, that is used for communications with the objects represented by the DMAP application.

TDSAPs that are not well known may be associated with one and only one particular application process. That is, there is a one-to-one mapping relationship between a TDSAP and a UAL process. Because TDSAPs are local, remote entities indicate a T-port that represents a corresponding application. T-ports map 1-to-1 to TDSAPs. UAL process / TDSAP / transport data port relationships shall survive application restart.

NOTE Fifteen TDSAPs are available for compressed transmission. See 11.6 for further details.

12.18.3 Mapping AL service primitives to TL service primitives

Table 282 indicates the mapping of application service primitives to transport services.

Table 282 – AL service primitive to TL service primitive mapping

Application service primitive	Transport service	Data conveyed between application sublayer and transport service
Publish.request Read.request, Read.response Write.request, Write.response Execute.request, Execute.response Tunnel.request, Tunnel.response AlertReport.request AlertAcknowledge.request	T-DATA.request	Contract_ID APDU_size APDU Message priority Discard eligibility Source TDSAP Destination T-port NOTE 1 The contract ID indicates destination address, and contract priority. NOTE 2 The source T-port can be determined from the source TDSAP, but is explicitly passed to match the interface provided by the TL.
Publish.indicate Read.indicate, Read.confirmation Write.indicate, Write.confirmation Execute.indicate, Execute.confirmation Tunnel.indicate, Tunnel.confirmation AlertReport.indicate AlertAcknowledge.indicate	T-DATA.indicate	source IPv6Address Source T-port APDU_size (equivalent to TSDU size) APDU (equivalent to TSDU) Explicit congestion notification (ECN) Destination TDSAP Destination T-port Transport time (one-way end-to-end delivery time, in seconds)

12.19 AL management

12.19.1 General

AL management supports the local DMAP application sublayer management object. Access to the attributes and methods of this object are defined by the ASL. For this standard, the ASL provides access to read a configured value from the ASL-MIB, to write a configured value to the ASL-MIB, and to support reset of the ASL.

12.19.2 Application sublayer handling of malformed application protocol data units

The ASL supports informing the local DMAP of a potential device/communication problem if an AL management-configured threshold is reached within a configured time period, for malformed APDUs received from a particular source device. Some examples of malformed APDUs are:

- APDU size incorrect (too long or too short);
- invalid service identifier; or
- service misuse (e.g., response primitive was indicated in the PDU for a two-part client/server or source/sink service).

The intent of this information is to enable the DMAP to provide information to higher level management. This may be important, for example, to enable detection of a malformed APDU attack occuring.

The ASL may be configured to advise the DMAP whenever a malformed APDU is received.

The ASL may be configured with non-zero values for a threshold and a time interval. When so configured, the ASL will maintain individual counters and timers internally for each network source address from which a malformed APDU has been received. Counting commences with the receipt of the first malformed APDU from a device and continues until either the malformed APDU threshold value is reached or the ASL_TimePeriodForMalformedAPDUs expires.

If the malformed APDU threshold value is reached prior to or at the expiration of the configured time interval, the DMAP is advised and the count and time interval for the device are reset.

If the malformed APDU threshold is not reached within the configured time interval, the counters and timers are internally reset.

NOTE How the DMAP is advised of a malformed APDU, or that a threshold has been reached within a specific time interval, is a local matter, and hence is not specified by this standard.

Figure 134 illustrates the handling of malformed APDUs.

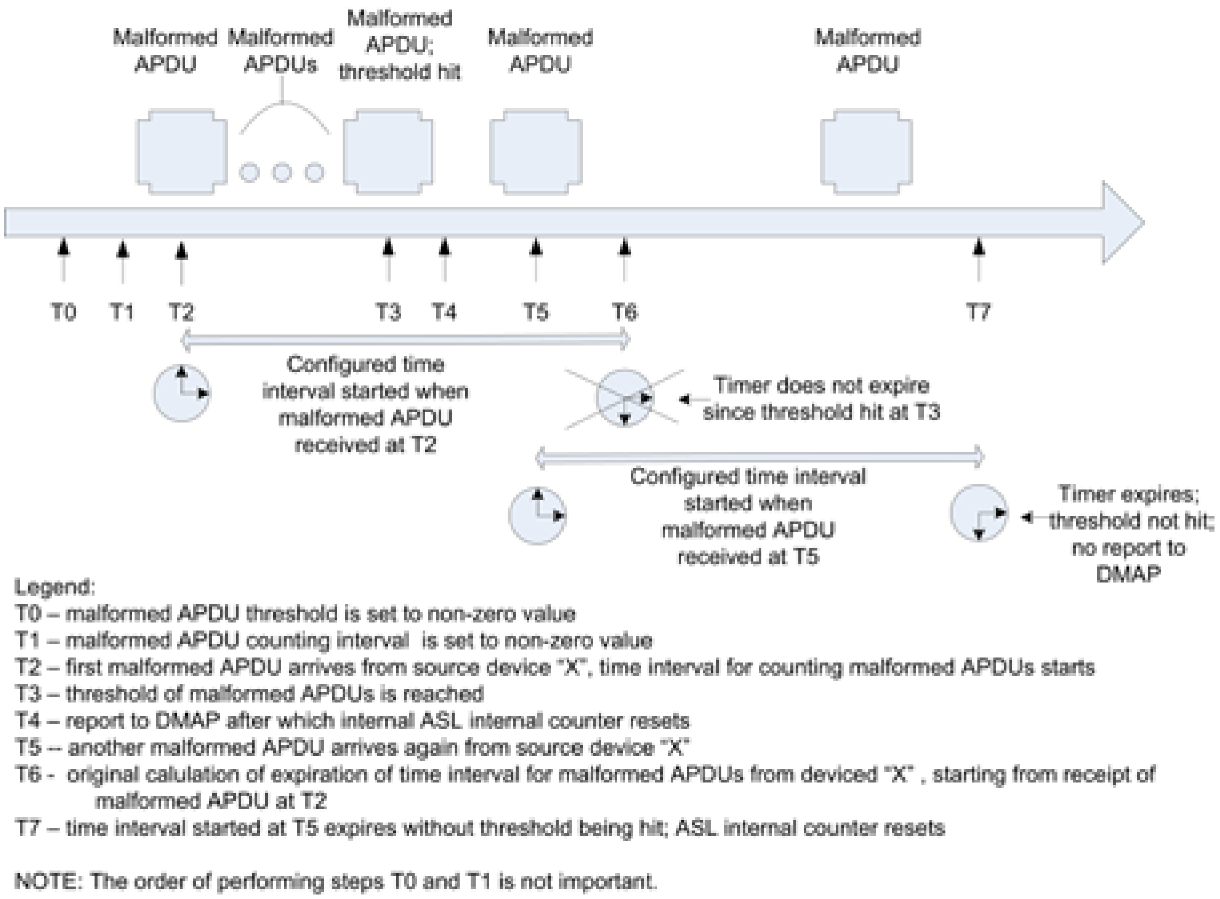


Figure 134 – Management and handling of malformed APDUs received from device X

12.19.3 Application sublayer management object attributes

Table 283 describes the attributes supported by the application sublayer management object (ASLMO).

Table 283 – ASLMO attributes

Standard object type name: Application sublayer management object (ASLMO)				
Standard object type identifier: 121				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Default value: 7 Valid range: 7	N/A
Reserved for future use	0	—	—	—
MalformedAPDUsAdvise	1	Indicates if ASL should indicate to local DMAP whenever a malformed APDU is encountered	Type: Boolean Classification: Static Accessibility: Read/write Initial default value : FALSE	If this parameter has a value of TRUE, the ASL shall pass each malformed APDU it receives on to the local DMAP
TimeIntervalForCounting MalformedAPDUs	2	This attribute specifies the time interval for the ASL to count malformed APDUs received from a particular device. Counting occurs from detection of the first malformed APDU from a device. This interval is applied commonly to APDUs from all source IPv6Addresses. The units of this attribute is seconds	Type: TAITimeDifference Classification: Static Accessibility: Read/write Initial default value : 0 Valid range: 0 ≤ value ≤ to 86 400 s Number of days is not included (it is always equal to zero)	If the time interval expires without the threshold being met, the corresponding internal malformed ASL counter and timer information shall be reset to zero (0)
MalformedAPDUsThreshold	3	Common threshold value to apply to malformed APDUs received from each device	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value : 0	If this threshold is met in the specified time interval, a communication alert shall be reported indicating the device that has been sending malformed APDUs. If a threshold value is set while counting is in progress, and the value set is lower than the prior threshold such that the new threshold has been exceeded, a malformed APDU alert shall be reported

Standard object type name: Application sublayer management object (ASLMO)				
Standard object type identifier: 121				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
MalformedAPDUAlertDescriptor	4	Describes how the malformed alert is reported on the network	Type : Alert report descriptor Classification: Static Accessibility: Read/write Default value: [TRUE, 7]	
MaxDevicesForWhichMalformed APDUsCanBeCounted	5	Describes the malformed APDU counting capability of the ASL in terms of the maximum number of devices for which counts can be simultaneously maintained	Type: Unsigned16 Classification: Constant Accessibility: Read only	A minimum value required may be established for example based on the role of the device
Reserved for future use by this standard	6..63	—	—	—

Attributes classified as either constant or static shall be preserved across restarts and power-fails.

12.19.4 Application sublayer management object methods

12.19.4.1 Standard object methods

An ASLMO has the methods defined in Table 284.

Table 284 – Application sublayer management object methods

Standard object type name: Application sublayer management object (ASLMO)		
Standard object type identifier: 121		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reset	1	Reset application sublayer
Reserved for future use by this standard	2..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.19.4.2 Reset method

Table 285 specifies the reset method primitives.

Table 285 – Reset method

<table>
<tr><th colspan="5">Standard object type name: Application sublayer management object (ASLMO)</th></tr>
<tr><th colspan="5">Standard object type identifier: 121</th></tr>
<tr><th>Method name</th><th>Method ID</th><th colspan="3">Method description</th></tr>
<tr><td rowspan="7">Reset</td><td>1</td><td colspan="3">Reset application sublayer</td></tr>
<tr><th colspan="4">Input arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type
(data type and size)</th><th>Argument description</th></tr>
<tr><td>1</td><td>ResetType</td><td>Type: Unsigned8
Named values:
0: not used;
1: reset to factory default settings;
2: reset to provisioned settings;
3: warm reset (reset to provisioned settings and any communication contract related information);
4: reset all dynamic data (e.g., related to statistics);
5..255: reserved</td><td>Type of reset desired</td></tr>
<tr><th colspan="4">Output arguments</th></tr>
<tr><th>Argument number</th><th>Argument name</th><th>Argument type
(data type and size)</th><th>Argument description</th></tr>
<tr><td colspan="4">None</td></tr>
</table>

12.19.4.3 Input arguments

The ResetType parameter indicates the type of reset desired. The sublayer may be reset to:

- factory default settings;
- only maintain provisioned settings (if any);
- only maintain the set of both provisioned settings and communication contract settings (if any); or
- only all dynamic statistics.

NOTE An example of default factory settings is:

– MalformedAPDUsAdvise configuration parameter, indicating disabled;

– TimeIntervalForCountingMalformedAPDUs configuration parameter, indicating 100 APDUs; and

– MalformedAPDUsThreshold configuration parameter, indicating a zero time interval.

12.19.4.4 Output arguments

There are no output arguments for this method.

12.19.4.5 Response codes

The following feedback codes are valid for this method:

- success;
- invalidArgument; and
- those that are vendor-defined.

12.19.5 Application sublayer management object alerts

Table 286 defines the alerts for the ASLMO.

Table 286 – ASLMO alerts

Standard object type name(s): Application sublayer management object (ASLMO)					
Standard object type identifier: 121					
Description of the alert: Malformed APDU alert					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority**	**Associated data**	**Description of associated data included with alert**
Event	Communication diagnostic	malformedAPDU CommunicationAlert	7 (i.e., a mid-range medium-priority alert)	No special standard type is defined, because the protocol content does not correspond to an attribute of the ASLMO. Rather, it is constructed within the protocol as an implicit sequence, identifiable by the combination of alert class, alert category and alert type	Three elements are included in the following sequence: a) Source address of malformed APDUS (IPv6Address) b) Threshold value exceeded (Unsigned16) c) Time interval in which threshold was exceeded (TAITimeDifference)

12.19.6 DMAP services invoked by application sublayer

If the configured ASL malformed APDUs interval and ASL malformed APDUs threshold are both non-zero, the ASL shall commence keeping malformed APDU statistics. If the threshold is reached prior to or upon expiration of the configured time interval, the ASL shall report to the DMAP that a communication diagnostic alert should be generated. The data provided by ASL to the DMAP shall include:

a) an indication that malformedAPDUsThresholdReached situation has been reached; and

NOTE This is indicated if and when the ASL malformed APDUs threshold is reached for malformed APDUs received from a single source IPv6Address within the configured ASL malformed APDUs interval.

b) diagnostic information regarding the malformed APDUs detected, such as:
   – number of APDUs received that did not have correct size,
   – number of APDUs received with an invalid service identifier, and
   – number of APDUs received with service identifier misused; and
c) the source IPv6Address of the malformed APDUs; and
d) the threshold value that was reached; and
e) the time duration over which the malformed APDUs were received.

   This time interval is calculated as the time from detection of the first malformed APDU from the indicated device by the ASL and the detection of the malformed APDU that resulted in the ASL threshold limit being reached for the indicated device. This time duration shall be less than or equal to the configured time interval established in the ASL management parameters.

12.19.7 Process industries standard objects

12.19.7.1 General

The standard objects defined in this standard are included to address the basic needs of the process industry. The unified field theory (for process control) that underlies this standard defines standard field objects by leveraging existing field device object definitions from field-proven object-oriented process control system standards. The set of objects has been selected based on commonality of use and are defined to facilitate interworkability and limited interoperability (within its domain of application) among devices.

NOTE 1 Terminology used here, including SOE, PV, OP, OOS, MAN and AUTO, is that common to the process industries.

NOTE 2 This standard presumes that any access restrictions to object attributes that are necessary to satisfy system usage requirements, are enforced by human interface devices and/or gateway devices.

To support timestamps used in process control industry alarm reports, the time value construct used to represent time shall support a coding accuracy of 1 ms. This accuracy is necessary when supporting the high speed resolution typical of the process industry SOE (sequence of events) applications.

12.19.7.2 Process industries user application objects

A basic list of user application objects is anticipated for the process control industries profile. The unified field objects (UFOs) defined in this standard are:

- analog input object,
- analog output object,
- binary input object, and
- binary output object.

Application object control mode supports the following modes:

- Target mode is the mode to which the device was commanded to transition. This may be different from the actual mode if the device cannot accept the target mode due to error, etc.
- Actual mode is the current mode of the object.
- Normal mode is the operating mode of the block that is desired by the responsible control engineer. Normal mode is one of the modes other than OOS, that is designated as “normal operation” for the block.
- Permitted modes represent the set of modes that are valid for this object. This is a filter that can be applied to limit the target mode of the block. For example, manual mode may be disabled this way. OOS is always included in the set of permitted modes.

The following modes are supported:

– OOS (out-of-service): The device is not actively measuring the PV (process variable) value or not accepting the OP (output point) value. Another common name for this mode is “inactive”. The value and associated status indicating the OOS condition are still communicated by the device. This is not intended to disable communication.

– MAN (manual): The PV can be manually entered by the operator and used for open loop control with a human in the loop or for overriding faulty measurement. MAN is also useful for testing.

– AUTO (automatic): Device is actively measuring its PV value or accepting its OP value.

A structured attribute is required to be added for each type of alert report supported by the object. This attribute supports enabling/disabling of an alert report and establishes the alert priorities.

For alarms, a structured attribute may additionally be required to establish alarm limits, to indicate if an alarm is present.

12.19.7.3 Analog input user object

12.19.7.3.1 General

A standard analog input user object representing an encapsulation of an analog input is defined. If multiple analog inputs are represented by a device, multiple analog input user objects should be instantiated. Object type-specific attributes of this object include:

- process value: a floating point value represented in engineering units and status;
- mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- corresponding concentrator object: specifies the concentrator associated to publish the PV; and
- scale: represents the range and units of the process value via a structured attribute that indicates the 0% and 100% limites of the nominal value range, a coded representation of engineering units, and the number of significant digits that should be used for display.

Standard alerts for this object will also be defined.

12.19.7.3.2 Object attributes

An analog input object has the attributes defined in Table 287.

Table 287 – Analog input object attributes

Standard object type name: Analog input object				
Standard object type identifier: 99				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: >0	N/A
Reserved for future use	0	—	—	—
PV	1	Measurement variable in engineering units of the sensor	Type: Process control value and status for analog value Classification: Dynamic Accessibility: Read only Default value: NaN Status: Unknown; Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for analog value structure	Analog process value and status of that value. Accessibility is read/write only when MODE.Target=MAN. See 12.19.7.2. When a write to PV is done, the device may implement this as a write to a non-network visible internal variable, and use the non-visible value when constructing the value it represents for the PV. As appropriate, the device may report a different status for the PV than that which was provided in the write request
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Default value: Actual mode value indicates OOS Valid range: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use	3	—	—	—

Standard object type name: Analog input object				
Standard object type identifier: 99				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Scale	4	Range and units of the measurement variable	Type: Process control scaling data	Scaling information for the analog process value
			Classification: Static	
			Accessibility: Read/write	
			Default value: Engineering units values for 0% and for 100% BOTH indicate 0	
			Valid range: See definition of scale structure type	
Reserved for future use by this standard	5..25	—	—	N octets of presently undefined content

12.19.7.3.3 Standard object methods

An analog input object has the methods defined in Table 288.

Table 288 – Analog input object methods

Standard object type name: Analog input object		
Standard object type identifier: 99		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.19.7.3.4 Alerts

An analog input may report the alerts shown in Table 289. If an alert is supported, a corresponding alert descriptor attribute shall be added to the analog input object to describe the characteristics of the alert.

Table 289 – Analog input alerts

Standard object type name(s):Analog input						
Standard object type identifier: 99						
Description of the alert: Analog input alerts						
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type(s) (Enumerated: based on alert category)**		**Alert priority**	**Associated data: type and size**	**Description of associated data included with alert**
Alarm	Process	1	High	Any	Type: Float32	Process variable
Alarm	Process	2	HighHigh	Any	Type: Float32	Process variable
Alarm	Process	3	Low	Any	Type: Float32	Process variable
Alarm	Process	4	LowLow	Any	Type: Float32	Process variable
Alarm	Process	5	DeviationLow	Any	Type: Float32	Process variable
Alarm	Process	6	DeviationHigh	Any	Type: Float32	Process variable
Alarm	Process	0	OutOfService	Any	Type: Float32	Process variable

12.19.7.4 Analog output user object

12.19.7.4.1 General

A standard analog output user object represents an encapsulation of an analog output. If multiple analog outputs are represented by a device, multiple analog output user objects should be instantiated. Object type-specific attributes of this object include:

- commanded output value: a floating point value represented in engineering units and status;
- mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- Readback: value and status of the actual position;
- provider of OP value: indicates the source of the OP value;
- corresponding concentrator object: specifies the concentrator associated to publish the Readback value; and
- scale: represents the range and units of the process value via a structured attribute that indicates the 0% and 100% limites of the nominal value range, a coded representation of engineering units, and the number of significant digits that should be used for display.

12.19.7.4.2 Object attributes

An analog output object has the attributes defined in Table 290.

Table 290 – Analog output attributes

Standard object type name: Analog output object				
Standard object type identifier: 98				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: >0	N/A
Reserved for future use	0	—	—	—
OP	1	Output value for the actuator	Type: Process control value and status for analog value structure Classification: Dynamic Accessibility: Read/write Default value: NaN Status: Unknown Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for analog value structure	This is the standard attribute that serves as the destination attribute for a publication from another object
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Default value: Actual mode value indicates OOS Valid range: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use	3	—	—	—
Readback	4	Readback value – the actual position of the OP	Type: Process control value and status for analog value Classification: Dynamic Accessibility: Read only Default value: NaN; Status: Unknown Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for analog value structure	Analog process value and status of that value. This is the standard attribute that is published from this object. If this object is extended, such that another attribute needs to be published, (a) concentrator object(s) represent(s) the resulting publication(s)
Reserved for future use	5	—	—	—

Standard object type name: Analog output object				
Standard object type identifier: 98				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Scale	6	Range and units of the output variable	Type: Process control scaling data structure	Scaling information
			Classification: Static	
			Accessibility: Read/write	
			Default value: Engineering units values for 0% and for 100% BOTH indicate 0	
			Valid range: See definition of Scale structure type	
Reserved for future use by this standard	7..25	—	—	N octets of presently undefined content

12.19.7.4.3 Standard object methods

An analog output object has the methods defined in Table 291.

Table 291 – Analog output object methods

Standard object type name: Analog output object		
Standard object type identifier: 98		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard.
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.19.7.4.4 Alerts

An analog output may report the alerts shown in Table 292. If an alert is supported, a corresponding alert descriptor attribute shall be added to the analog output object to describe the characteristics of the alert.

Table 292 – Analog output alerts

Standard object type name(s): Analog output						
Standard object type identifier: 98						
Description of the alert: Analog output alerts						
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type(s) (Enumerated: based on alert category)**		**Alert priority**	**Associated data: type and size**	**Description of associated data included with alert**
Alarm	Process	1	High	Any	Type: Float32	Process variable
Alarm	Process	2	HighHigh	Any	Type: Float32	Process variable
Alarm	Process	3	Low	Any	Type: Float32	Process variable
Alarm	Process	4	LowLow	Any	Type: Float32	Process variable
Alarm	Process	5	DeviationLow	Any	Type: Float32	Process variable
Alarm	Process	6	DeviationHigh	Any	Type: Float32	Process variable
Alarm	Process	0	OutOfService	Any	Type: Float32	Process variable

12.19.7.5 Binary input user object

12.19.7.5.1 General

A standard binary input user object represents an encapsulation of a single binary input. If multiple binary inputs are represented by a device, multiple binary input user objects should exist to represent them. Object type specific attributes of this object include:

- process value: binary value and status;
- mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode; and
- corresponding concentrator object: specifies the concentrator associated to publish the process value.

12.19.7.5.2 Object attributes

A binary input object has the attributes defined in Table 293.

Table 293 – Binary input object attributes

Standard object type name: Binary input object				
Standard object type identifier: 97				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: >0	N/A
Reserved for future use	0	—	—	—
PV_B	1	Discrete measurement variable	Type: Process control value and status for discrete value Classification: Dynamic Accessibility: Read only Default value: 0 Status: Unknown Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for discrete value structure	Binary process value and status of that value. This is the standard attribute that is published from this object. If this object is extended, such that another attribute needs to be published, (a) concentrator object(s) represent(s) the resulting publication(s). Accessibility is read/write only when in MAN mode. See 12.19.7.2. When a write to PV_B is done, the device may implement this as a write to a non-network visible internal variable, and use the non-visible value when constructing the value it represents for the PV_B. As appropriate, the device may report a different status for the PV_B than that which was provided in the write request.
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Default value: Actual mode value indicates OOS Valid range: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use by this standard	3..25	—	—	N octets of presently undefined content

12.19.7.5.3 Standard object methods

A binary input object has the methods defined in Table 294.

Table 294 – Binary input object methods

Standard object type name: Binary input object		
Standard object type identifier: 97		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific	128..255	These method identifiers are available for implementation-specific use

12.19.7.5.4 Alerts

An analog output may report the alerts shown in Table 295. If an alert is supported, a corresponding alert descriptor attribute shall be added to the binary input object to describe the characteristics of the alert.

Table 295 – Binary input alerts

Standard object type name(s): Binary input object						
Standard object type identifier: 97						
Description of the alert: Binary input alerts						
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type(s) (Enumerated: based on alert category)**		**Alert priority**	**Associated data: type and size**	**Description of associated data included with alert**
Alarm	Process	1	DiscreteAlarm	Any	Type: Boolean	Process value
Alarm	Process	0	OutOfService	Any	Type: Boolean	Process value

12.19.7.6 Binary output user object

12.19.7.6.1 General

A standard binary output user object represents an encapsulation of a single binary output. If multiple binary outputs are represented by a device, multiple binary output user objects should exist to represent them. Object type specific attributes of this object include:

- commanded output value: binary value and status;
- mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- provider of OP_B Value: indicates the source of the OP_B value;
- Readback value: binary value and status; and
- corresponding concentrator object: specifies the concentrator associated to publish the Readback_B value.

12.19.7.6.2 Object attributes

A binary output object has the attributes defined in Table 296.

Table 296 – Binary output attributes

Standard object type name: Binary output object				
Standard object type identifier: 96				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Valid range: >0	N/A
Reserved for future use	0	—	—	—
OP_B	1	Discrete measurement variable	Type: Process control value and status for discrete value structure Classification: Dynamic Accessibility: Read/write Default value: 0 Status: Unknown Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for discrete value structure	This is the standard attribute that is the destination for a publication from another object
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Default value: Actual mode value indicates OOS Valid range: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use	3	—	—	—
Readback_B	4	Readback value of actual position of the actuator	Type: Process control value and status for discrete value Classification: Dynamic Accessibility: Read only Default value: 0; Status: Unknown Substatus: Unknown Limit condition: Not limited Valid range: See definition of process control value and status for analog value structure	Analog process value and status of that value. This is the standard attribute that is published from this object. If this object is extended, such that another attribute needs to be published, a concentrator object(s) represents the resulting publication(s)
Reserved for future use by this standard	5..25	—	—	N octets of presently undefined content

12.19.7.6.3 Standard object methods

A binary output object has the methods defined in Table 297.

Table 297 – Binary output object methods

Standard object type name: Binary output object		
Standard object type identifier: 96		
Method name	**Method ID**	**Method description**
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0..127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128..255	These method identifiers are available for implementation-specific use

12.19.7.6.4 Alerts

A binary output may report the alerts shown in Table 298. If an alert is supported, a corresponding alert descriptor attribute shall be added to the binary output object to describe the characteristics of the alert.

Table 298 – Binary output alerts

Standard object type name(s): Binary output object						
Standard object type identifier: 96						
Description of the alert: Binary output alerts						
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type(s) (Enumerated: based on alert category)**		**Alert priority**	**Associated data: type and size**	**Description of associated data included with alert**
Alarm	Process	1	DiscreteAlarm	Any	Type: Boolean	Process value
Alarm	Process	0	OutOfService	Any	Type: Boolean	Process value

12.19.8 Factory automation industries profile

12.19.8.1 General

Additional standard objects to support the needs of factory automation may be added in future releases of this standard. More detailed thought specific to factory automation is deferred to a later release of this standard’s standardization activity. Examples of objects that may be defined to meet the needs of factory automation may include:

a) binary input user object (e.g., a contact);
b) binary output user object (e.g., a coil);
c) analog input user object;
d) analog output user object;
e) register input user object (e.g., to map 16 bits of two-state information often found in PLC input registers);
f) register output user object (e.g., to map 16 bits of two-state information often found in PLC output registers);

g) multi-state input user object (e.g., to map 8 bits of state information for a valve with enumerated states such as opening, open, closing, closed, or to map a unidirectional motor with states such as off, starting, running, stopping); and

h) multi-state output user object (e.g., to map 8 bits of state output information for an output device with enumerated states).

Standard alerts for these objects may also be defined.

12.19.8.2 Manufacturer specific objects

Vendors may also define vendor- or device-specific custom objects. An example of a vendor-specific object for process systems is an equipment-mounted display object, in which a string can be stored for display to a human near the device.

12.20 Process control industry standard data structures

12.20.1 General

The process control industry standard data structures used shall be the data structures conveyed by the protocol defined by this standard. Industry-independent standard data structures and process control industry data structures are both summarized in Annex L.

NOTE Vendor-specific data structure definitions are not supported.

12.20.2 Status for analog information

The status for analog information is a packed fixed format octet containing the items shown in Table 299.

Table 299 – Status octet

<table>
<tr><th>Bit 7
(MSB)</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0
(LSB)</th></tr>
<tr><th colspan="2">Quality</th><th><reserved></th><th colspan="3">Quality dependent substatus</th><th colspan="2">Limit condition</th></tr>
<tr><td colspan="2">0 = bad
(value should not be used)</td><td rowspan="4">This bit shall always be set to zero</td><td colspan="3">Named values when quality=bad:
0: non-specific;
1: configuration error;
2: not connected;
3: device failure;
4: sensor failure;
5: no communication with a lastUsableValue;
6: no communication without a lastUsableValue;
7: out of service (value is not being computed)</td><td colspan="2" rowspan="4">Named values:
0 = not limited;
1 = low limit
2 = high limit;
3 = constant (both high and low limited)</td></tr>
<tr><td colspan="2">1 = uncertain
(value of less than normal quality)</td><td colspan="3">Named values when quality=uncertain:
0: non-specific;
1: last usable value;
2: substituted or manual entry;
3: initial value;
4: sensor conversion inaccurate;
5: range limits exceeded;
6: sub normal;
7: reserved</td></tr>
<tr><td colspan="2">2 = good
(quality of value is good, but an alarm condition may exist)</td><td colspan="3">Named values when quality=good:
0: no special conditions exist;
1..7: reserved</td></tr>
<tr><td colspan="2">3 = reserved</td><td colspan="3">All values are reserved.
Within this standard, this shall always be set to zero</td></tr>
</table>

NOTE The definitions for the status octet are a subset of those defined by the HART Communication Foundation, the Fieldbus Foundation, and the OPC Foundation.

12.20.3 Value and status for analog information

As status does not indicate substitution, network-initiated writes using the analog data type structures are not permitted by this standard.

The structure of analog information is shown in Table 300.

Table 300 – Data type: Process value and status for analog value

Standard data type name: Process control value and status for analog value		
Standard data type code: 65		
Element name	**Element identifier**	**Element scalar type**
Status	1	Type: BitString8 Classification: Dynamic Valid value set: See Table 299
Value	2	Type: Float32 Classification: Dynamic

12.20.4 Value and status for binary information

As status does not indicate substitution, network-initiated writes to digital data type structures are not permitted.

The structure of digital information is shown in Table 301.

Table 301 – Data type: Process value and status for binary value

Standard data type name: Process control value and status for binary value		
Standard data type code: 66		
Element name	**Element identifier**	**Element scalar type**
Status	1	Type: BitString8 Accessibility: May vary by use Valid value set: See Table 299
Value	2	Type: Boolean

12.20.5 Process control mode

Elements in process control mode are shown in Table 302.

Table 302 – Data type: Process control mode

Standard data type name: Process control mode		
Standard data type code: 69		
Element name	**Element identifier**	**Element scalar type**
Target	1	Type: BitString8 Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303
Actual	2	Type: BitString8 Classification: Dynamic Accessibility: Read only Default value: OOS Valid value set : See Table 303
Permitted	3	Type: BitString8 Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303
Normal	4	Type: BitString8 Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303

The value of each element of the mode structure is represented by a bitstring containing the bits in Table 303, where the <reserved> bits shall be set to zero (0).

Table 303 – Data type: Process control mode bitstring

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<reserved>	<reserved>	<reserved>	AUTO	MAN	<reserved>	<reserved>	OOS

That is:

- OOS-only is equal to 0x01, with the equivalent decimal value of 1.
- MAN-only is equal to 0x08, with the equivalent decimal value of 8.
- AUTO-only is equal to 0x10, with the equivalent decimal value of 16.

12.20.6 Scaling

Elements in process control scaling are shown in Table 304.

Table 304 – Data type: Process control scaling

Standard data type name: Process control scaling		
Standard data type code: 68		
Element name	**Element identifier**	**Element scalar type**
Engineering units at 100% (the upper range of the associated object parameter)	1	Type: Float32 Classification: Static Accessibility: Read/write Default value : 0
Engineering units at 0% (the lower range of the associated object parameter)	2	Type: Float32 Classification: Static Accessibility: Read/write Default value : 0
Units index for both SI units and traditional (non-SI) units [a)] – range 1000..1999 – other codes reserved	3	Type: Unsigned16 Classification: Static Accessibility: Read/write
Location of the decimal point / decimal sign (represents the number of digits to the right of the decimal point / decimal sign, i.e., the significance of the fractional part of the associated value)	4	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0
a) As specified in *Standard Units Codes Table*, published in ISA *Handbook of Measurement Equations and Tables*, 2nd Edition, ISBN 978 1 55617 946 4. This index is also published by the Fieldbus Foundation as TN-016:2007, Table 3.1		

12.21 Additional tables

12.21.1 Process control profile standard objects

Table 305 lists process control profile standard objects.

Table 305 – Process control standard objects

Object type	Defined by	Standard object type identifier
Analog input	12.19.7.3	99
Analog output	12.19.7.4	98
Binary input	12.19.7.5	97
Binary output	12.19.7.6	96

12.21.2 Services

Table 306 provides a list of services.

Table 306 – Services

Service	Use
Read	Read the value of one or more attributes from one or more objects of a UAP
Write	Write the value to one or more attributes of one or more objects of a UAP
Execute	Execute a set of functions on object instances of a UAP
Publish	Periodically publish data to subscribers
AlertReport	Report one or more alert conditions
AlertAcknowledge	Acknowledge an AlertReport
Tunnel	Pass payload through

12.22 Coding

12.22.1 General

The conditions for encoding wireless APDUs in this standard include the following considerations:

- Some messages occur often, such as periodic publications.
- Messages should be short, to preserve battery power.
- There should be minimal selection of ASL service types.

The maximum size of an APDU (which is a TSDU) is determined by subtracting (the size of the information TL adds to the TSDU to form the TPDU) from (the Assigned_Max_NSDU_Size), where Assigned_Max_NSDU_Size is an output parameter of the method used to establish a communication contract. That is:

maxAPDUSize = Assigned_Max_NSDU_Size - sizeOF(TLInfoAddedtoAPDU)

See 6.3.11.2.5 for further details of communication contract establishment.

ASL coding shall ensure that the maximum APDU size is not exceeded.

NOTE IETF RFC 2348 provides recommendations on the maximum size of NPDUs.

12.22.2 Coding rules for application protocol data units

12.22.2.1 General

The coding rules defined in 12.22.2 represent bit 0 as the least significant bit (LSB) in the value represented.

All APDUs contain an AL header, and a service type-specific APDU content. Table 307 indicates the general coding construct of an APDU.

Table 307 – Application messaging format

May be 1, 2, 3, or 5 octets (see 12.22.2.3)	N octets
ASDU header (ensures routing to correct objects; provides service type identification)	Service-specific content

12.22.2.2 Concatenation

APDUs can be concatenated together, and the concatenation of individual APDUs may be given to the TL as a single TSDU. The size of this TSDU shall not exceed the maximum APDU size for communications relative to the corresponding communication contract between the sending and receiving applications.

Table 308 describes the format of concatenated APDUs in a single TSDU.

Table 308 – Concatenated APDUs in a single TSDU

APDU_1	...	APDU_n

Concatenation can be used to ensure that when one of the concatenated APDUs is received, all have been received, thus providing a basis for multi-component actions that are atomic with respect to inter-device communications.

NOTE How the ASL determines when and what to concatenate is a local matter.

12.22.2.3 AL header

The AL header supports four object identifier addressing modes which determine header construction beyond the first octet. The object identifier addressing modes are:

- four-bit object identifier addressing mode;
- eight-bit (1 octet) object identifier addressing mode;
- sixteen-bit (2 octets) object identifier addressing mode; and
- inferred addressing mode, which may be used only in the second and subsequent APDUs of a TSDU that contains multiple concatenated APDUs.

Identification of the UAP containing the object is a function of the TL.

The object identifier addressing mode in use for APDU interpretation is indicated in bits 5 and 6 of the first octet of the APDU header. Table 309 represents the coding of this first APDU header octet.

Table 309 – Object addressing

Octets	Bits							
	7	6	5	4	3	2	1	0
1	Service primitive type (.req = 0 .resp = 1)	Object identifier addressing mode Named values: 00: 4-bit mode 01: 8-bit mode 10: 16-bit mode 11: inferred mode		ASL service type				

12.22.2.4 Object identifier coding

12.22.2.4.1 General

In all header constructions, the source object identifier represents the object identifier in the application that is initiating the ASL service primitive (that is, the initiator of a .request or a .response primitive), and the destination object identifier represents the object identifier in the application that is receiving the ASL service primitive (that is, the recipient of an .indication or a .confirmation primitive).

12.22.2.4.2 Four-bit object identifier addressing mode

A four-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in a 4-bit unsigned integer. This addressing mode provides for optimal header compaction for application processes with a small number of objects. This mode is described in Table 310.

Table 310 – Four-bit addressing mode APDU header construction

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Service primitive type	0	0	ASL service type				
1	Source object identifier				Destination object identifier			

12.22.2.4.3 Eight-bit object identifier addressing mode

An eight-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in an 8-bit unsigned integer, and further that at least one of the object identifiers cannot be expressed in a 4-bit unsigned integer. This mode is described in Table 311.

Table 311 – Eight-bit addressing mode APDU header construction

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Service primitive type	0	1	ASL service type				
1	Source object identifier							
1	Destination object identifier							

12.22.2.4.4 Sixteen-bit object identifier addressing mode

A sixteen-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in a 16-bit unsigned integer, and further that at least one of the object identifiers cannot be expressed in an 8-bit unsigned integer. This mode is described in Table 312.

Table 312 – Sixteen-bit addressing mode APDU header construction

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Service primitive type	1	0	ASL service type				
2	Source object identifier							
2	Destination object identifier							

12.22.2.4.5 Inferred object identifier addressing mode for optimized concatenations

An inferred object identifier addressing mode is an optimization technique used only within a concatenated APDU. The intent of this technique is to save octets transmitted by eliminating redundant source and object identifiers, which can be determined from the most recently parsed APDU contained within the same APDU concatenation.

Inferred object addressing shall not be indicated in the first APDU of a concatenation.

NOTE Any APDU indicating an inferred object addressing mode in the first APDU met in ASL parsing is considered a malformed APDU.

An example is included in Table 313.

Table 313 – Inferred addressing use case example

APDU_1	APDU_2	APDU_3	APDU_4	APDU_5
00 object identifier addressing mode	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_1)	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_2, which is used the source and destination OIDs from APDU_1)	01 object identifier addressing mode	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_4)
APDU_1 includes: – 00 addressing mode; – service type; – 4-bit source object identifier; – 4-bit destination object identifier; service-specific payload	APDU_2 includes: – 11 addressing mode; – service type; – service-specific payload	APDU_3 includes: – 11 addressing mode; – service type; – service-specific payload	APDU_4 includes: – 01 addressing mode – service type – 8-bit source object identifier; – 8-bit destination object identifier; service-specific payload	APDU_5 includes: – 11 addressing mode; – service type; – service-specific payload

Table 314 describes the construction of the inferred addressing mode APDU header.

Table 314 – Inferred addressing mode APDU header construction

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Service primitive type	1	1	ASL service type				

12.22.2.5 Object attribute coding

12.22.2.5.1 General

Object attribute coding is determined by an attribute identifier format. The format may indicate:

- Six-bit, not indexed: The attribute fits into an Unsigned6 integer, and is not indexed.
- Six-bit, singly indexed: The attribute fits into an Unsigned6 integer, and requires one index. The attribute index is extensible, as indicated by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in size. If the first bit of the index is 1, the index is 15 bits in size.
- Six-bit, doubly indexed: The attribute fits into an Unsigned6 integer, and requires two indices. The attribute indices are individually extensible; that is, the first index may be 7

bits or 15 bits in size, and the second index also may be either 7 bits or 15 bits in size. The size of the index is determined by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in size. If the first bit of the index is 1, the index is 15 bits in size.

- Twelve-bit, not indexed: The attribute fits does not fit into an Unsigned12 integer. The attribute is not indexed.
- Twelve-bit, singly indexed: The attribute fits into an Unsigned12 integer, and requires one index. The attribute index is extensible, as indicated by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in size. If the first bit of the index is 1, the index is 15 bits in size.
- Twelve-bit, doubly indexed: The attribute fits into an Unsigned12 integer, and requires two indices. The attribute indices are individually extensible; that is, the first index may be 7 bits or 15 bits in size, and the second index also may be either 7 bits or 15 bits in size. The size of the index is determined by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in size. If the first bit of the index is 1, the index is 15 bits in size.

NOTE Refer to 12.23.1.3 for the definitions of Unsigned6 and Unsigned12.

12.22.2.5.2 Six-bit attribute identifier, not indexed

Table 315 indicates the coding for a six-bit attribute identifier that is not an indexed or structured attribute.

Table 315 – Six-bit attribute identifier, not indexed

Number of octets	**Number of octets**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 00)		Attribute identifier					

12.22.2.5.3 Six-bit attribute identifier, singly indexed forms

Table 316 and Table 317 indicate the coding for a six-bit attribute identifier that may be accessed using a single index.

Table 316 – Six-bit attribute identifier, singly indexed, with 7-bit index

Number of octets	**Bits**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 01)		Attribute identifier					
1	0	Index						

Table 317 – Six-bit attribute identifier, singly indexed, with 15-bit index

Number of octets	**bits**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 01)		Attribute identifier					
2	1	Index (high-order 7 bits)						
	Index (low-order 8 bits)							

12.22.2.5.4 Six-bit attribute identifier, doubly indexed forms

Table 318, Table 319, Table 320, and Table 321 indicate the coding for a six-bit attribute identifier that may be accessed using two indices.

Table 318 – Six-bit attribute identifier, doubly indexed, with two 7-bit indices

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 10)		Attribute identifier					
1	0	Index 1						
1	0	Index 2						

Table 319 – Six-bit attribute identifier, doubly indexed, with two 15-bit indices

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 10)		Attribute identifier					
2	1	Index 1 (high-order 7 bits)						
	Index 1 (low-order 8 bits)							
2	1	Index 2 (high-order 7 bits)						
	Index 2 (low-order 8 bits)							

Table 320 – Six-bit attribute identifier, doubly indexed, with first index seven bits long and second index fifteen bits long

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 10)		Attribute identifier					
1	0	Index 1						
2	1	Index 2 (high-order 7 bits)						
	Index 2 (low-order 8 bits)							

Table 321 – Six-bit attribute bit attribute identifier, doubly indexed, with first index fifteen bits long and second index seven bits long

Number of octets	bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Attribute short form (value = binary 10)		Attribute identifier					
2	1	Index 1 (high-order 7 bits)						
	Index 1 (low-order 8 bits)							
1	0	Index 2						

12.22.2.5.5 Twelve-bit attribute identifier, not indexed

Table 322 indicates the coding for a twelve-bit attribute identifier that is not indexed.

Table 322 – Twelve-bit attribute identifier, not indexed

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 00		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							

12.22.2.5.6 Twelve-bit attribute identifier, singly indexed coding forms

Table 323 and Table 324 indicate the coding for a twelve-bit attribute identifier that is accessed using a single index.

Table 323 – Twelve-bit attribute identifier, singly indexed with 7-bit index

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 01		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							
1	0	Index						

Table 324 – Twelve-bit attribute identifier, singly indexed with 15-bit index

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 01		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							
2	1	Index 1 (high-order 7 bits)						
	Index 1 (low-order 8 bits)							

12.22.2.5.7 Twelve-bit attribute identifier, doubly indexed coding forms

Table 325, Table 326, Table 327, and Table 328 indicate the coding for a twelve-bit attribute identifier that is accessed using two indices.

Table 325 – Twelve-bit attribute identifier, doubly indexed with two 7-bit indices

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute long form (value = binary 11)		Attribute long form, index form = binary 10		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							
1	0	Index 1						
1	0	Index 2						

Table 326 – Twelve-bit attribute identifier, doubly indexed with two 15-bit indices

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 10		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							
2	1	Index 1 (high-order 7 bits)						
	Index 1 (low-order 8 bits)							
2	1	Index 2 (high-order 7 bits)						
	Index 2 (low-order 8 bits)							

Table 327 – Twelve-bit attribute identifier, doubly indexed with first index seven bits long and second index fifteen bits long

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 10		Attribute identifier (high-order 4 bits)			
	Attribute identifier (low-order 8 bits)							
1	0	Index 1						
2	1	Index 2 (high-order 7 bits)						
	Index 2 (low-order 8 bits)							

Table 328 – Twelve-bit attribute identifier, doubly indexed with the first index fifteen bits long and the second index seven bits long

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	Attribute short form (value = binary 11)		Attribute long form, index form = binary 10		Attribute identifier (high-order 4 bits)			
	1	Index 1 (high-order 7 bits)						
2	Index 2(low-order 8 bits)							
	0	Index 2						

12.22.2.5.8 Reserved for future use

Table 329 identifies an attribute identifier form that is reserved for future use.

Table 329 – Twelve-bit attribute identifier, reserved form

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Attribute short form (value = binary 11)		Attribute long form, index form (reserved): binary 11		Reserved for future use			

12.22.2.6 Read

Table 330 provides coding rules for the service specific portion of a read service request APDU.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

Table 330 – Coding rules for read service request

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Request ID							
…	Attribute identifier (see coding rules for attribute identifier)							

Table 331 provides coding rules for a read service response with 7-bit size field.

Table 331 – Coding rules for read service response with seven bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Request ID							
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	ServiceFeedbackCode							
1	0	S=Size – conditionally included only when ServiceFeedbackCode indicates success						
S	Value – conditionally present only when ServiceFeedbackCode only if indicates success							

NOTE Refer to 12.23.3 for the definitions of ServiceFeedbackCode for AL services.

Table 332 provides coding rules for a read service response with 15-bit size field.

Table 332 – Coding rules for read service response with fifteen-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Request ID							
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	ServiceFeedbackCode							
2	1	S[14..8]=Size – high-order 7 bits, conditionally present only when ServiceFeedbackCode indicates success						
	S[7..0]=Size – low-order 8 bits, conditionally present only when ServiceFeedbackCode indicates success							
S	Value – conditionally present only when ServiceFeedbackCode indicates success							

12.22.2.7 Write

Table 333 and Table 334 provide coding rules for a write service request.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

Table 333 – Coding rules for write service request with 7- bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Request ID							
…	Attribute identifier (see attribute encoding rules)							
…	0	S=Size						
S	Value							

Table 334 – Coding rules for write service request with 15-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Request ID							
…	Attribute identifier (see attribute encoding rules)							
2	1	S[14..8]==Size (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Value							

Table 335 provides coding rules for a write service response.

Table 335 – Coding rules for write service response

Number of octets	bits							
	7	6	5	4	3	2	1	0
1	Request ID							
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	ServiceFeedbackCode							

12.22.2.8 Execute

Table 336 and Table 337 provide coding rules for an execute service request.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

Table 336 – Coding rules for execute service request with 7-bit size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Request identifier							
1	Method identifier							
1	0	S=Size in octets of request parameters						
S	Request parameters							

Table 337 – Coding rules for execute service request with 15-bit size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Request identifier							
1	Method identifier							
2	1	S[14..8]=Size in octets of response parameters (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Response parameters							

Table 338 and Table 339 provide coding rules for an execute service response.

Table 338 – Coding rules for execute service response with 7-bit size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Request identifier							
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	ServiceFeedbackCode							
1	0	S=Size in octets of response parameters						
S	Response parameters							

Table 339 – Coding rules for execute service response with 15-bit size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Request identifier							
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	ServiceFeedbackCode							
2	1	S[14..8]=Size in octets of response parameters (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Response parameters							

12.22.2.9 Tunnel

Table 340 and Table 341 provide coding rules for a tunnel service request.

Table 340 – Coding rules for tunnel service request with 7-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	0	S=7-bit size						
S	Payload							

Table 341 – Coding rules for tunnel service request with 15-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
2	1	S[14..8]=Size in octets of response parameters (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Payload							

Table 342 and Table 343 provide coding rules for a tunnel service response.

Table 342 – Coding rules for tunnel service response with 7-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
1	0	S=Size						
S	Payload							

Table 343 – Coding rules for tunnel service response with 15-bit size field

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
2	1	S[14..8]=Size in octets of response parameters (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Payload							

12.22.2.10 AlertReport

Table 344 and Table 345 provide coding rules for an AlertReport service request.

Table 344 – Coding rules for AlertReport service with 7-bit associated-data size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Alert report ID							
2	Detecting object application process identifier (T-port)							
2	Detecting object identifier							
6	TAINetworkTime							
1	Class	Direction	Category		Alert Priority			
1	Type							
1	0	S=Size of associated data						
S	Associated data							

Table 345 – Coding rules for AlertReport service with 15-bit associated-data size field

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Alert report ID							
2	Detecting object application process identifier (T-port)							
2	Detecting object identifier							
6	TAINetworkTime							
1	Class	Direction	Category		Alert Priority			
1	Type							
2	1	S[14..8]=Size in octets of response parameters (high-order 7 bits)						
	S[7..0]=Size (low-order 8 bits)							
S	Associated data							

12.22.2.11 AlertAcknowledge

Table 346 provides coding rules for an AlertAcknowledge service request.

Table 346 – Coding rules for AlertAcknowledge service

Number of octets	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	Alert report ID							

12.22.2.12 Publish

Table 347 provides coding rules for a native publish service.

When used in conjunction with a concentrator object, “Data” in the payload comprises the entire data communicated, which is a configured sequence of process control variables. The process control variables include both status information and process values. The structure of the data is indicated by the publishing content version. The freshness sequence number is within the scope of a particular concentrator object.

Table 347 – Coding rules for publish service for a native sequence of values

Number of octets	Bits							
	7	6	5	4	3	2	1	0
1	Publishing content version							
1	Freshness sequence number							
S	Data							

Table 348 provides coding rules for a publish service used to convey either an internally encoded octet string, or non-native data. Use of this service for non-native data enables support for tunneling.

Table 348 – Coding rules for publish service – non-native (for tunnel support)

Number of octets	Bits							
	7	6	5	4	3	2	1	0
S	Data							

The coding rules for uninterpreted varying-size data, given in Table 351, apply to a published healthReport (see 12.23.1.6).

12.22.2.13 Concatenation

Table 349 provides coding rules for a constructing a single TSDU which contains multiple logical APDUs.

Table 349 – Coding rules for concatenate service

Number of octets	Bits							
	7	6	5	4	3	2	1	0
S	SEQUENCE OF APDUs							

12.22.3 Coding of application data

12.22.3.1 General

Coding of single application data elements is always primitive. In the tables of 12.22.3, octet 1 represents the most significant octet, bit 7 represents the most significant bit within an octet, and bit 0 represents the least significant bit within an octet.

The semantics of user data is known by:

- prior agreement (e.g., tunnel payload content);
- position in the APDU with fixed field size for content; or
- existing fields in the APDU.

In these situations, no additional decoding information is added to the APDU.

Coding rules for application data are provided in Table 350 and Table 351. If the size is fixed, such as for data type OctetStringN for a given fixed value of N, then size information is implicit in the declaration, so is not explicitly conveyed in the APDU, as shown in Table 350.

Table 350 – General coding rule for size-invariant application data

Data (fixed size)

In contrast, if the size may vary, such as for data type OctetString (and not OctetStringN for any N), then the size of the actual field is explicitly conveyed in the APDU. Often that is done by prefixing the data with the size, as shown in Table 351. In other cases, the size field either is found directly in, or is computable from, some earlier-parsed field in the APDU.

Table 351 – General coding rule for size-varying application data of 0..255 octets

Unsigned8 N, size of data (in octets)	Data (size N octets)

12.22.3.2 through 12.22.3.8 define the data coding for standard data types.

12.22.3.2 Boolean values

NOTE The type name honors the logician George Boole, hence its capitalization.

12.22.3.2.1 Coding of Boolean values

Booleans are coded as zero / non-zero values in either a 1-bit, for packed data structures, or an 8-bit field, for relatively unpacked data structures.

12.22.3.2.2 Boolean8

The coding of a Boolean8, which is used in relatively unpacked data structures, is:

- Data type: Boolean
- Size: 1 octet

An all-zero value of the underlying Unsigned8 representation codes the value FALSE; any non-zero value codes the value TRUE.

12.22.3.2.3 Boolean1

The coding of a Boolean1, which is used in packed data structures is:

- Data type: Boolean
- Size: 1 bit

A zero value of the underlying Unsigned1 representation codes the value FALSE, whereas the non-zero value one (1) codes the value TRUE.

12.22.3.3 Integer values

12.22.3.3.1 Coding of signed integer values

12.22.3.3.1.1 General

Signed integers are coded as 2's-complement numbers. In 2's-complement arithmetic, negative numbers are represented by the 2's-complement of the absolute value. In this system, zero has a single representation.

In the 2's-complement representation, positive numbers are represented as simple binary, and negative 2's-complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

The most significant bit (i.e., bit 7 for an Integer8 value, bit 15 for an Integer16) indicates the sign of the integer, and is therefore called the sign bit. If the sign bit is zero, then the number represented is greater than or equal to zero (i.e., zero, or a positive number). If the sign bit is one, then the number represented is less than zero (i.e., a negative number).

NOTE To calculate the 2's-complement of an integer, invert the binary equivalent of the number by changing all of the ones to zeroes and all of the zeroes to ones (also called 1's-complement), and then add one.

Example: Form the 2's-complement of the value 17.

0x 0001 000 1 (binary 17)

To form the 2's-complement:

First: NOT (0x 0001 000 1) = 0x 1110 111 0, where the NOT operation results in inverting the bits

Then add 1: (0x 1110 111 0) + (0x 0000 0001) = 0x 1110 1111 (2's-complement = -17).

12.22.3.3.1.2 Integer8

The coding of an Integer8 is:

- Data type: Integer8
- Range: $-2^{7} \le k \le 2^{7}-1$ (i.e., $-128 \le k \le 127$)
- Size: 1 octet

12.22.3.3.1.3 Integer16

The coding of an Integer16 is:

- Data type: Integer16
- Range: $-2^{15} \le k \le 2^{15}-1$ (i.e., $-32\,768 \le k \le 32\,767$)
- Size: 2 octets

12.22.3.3.1.4 Integer32

The coding of an Integer32 is:

- Data type: Integer32
- Range: $-2^{31} \le k \le 2^{31}-1$ (i.e., $-2\,147\,483\,648 < k < 2\,147\,483\,647$)
- Size: 4 octets

12.22.3.3.1.5 IntegerN

The coding of an IntegerN, which is used in packed data structures is:

- Data type: IntegerN
- Range: $-2^{-(N-1)} \le k \le 2^{(N-1)}-1$
- Size: N bits

12.22.3.3.2 Coding of unsigned integer values

12.22.3.3.2.1 Unsigned8

The coding of an Unsigned8 is:

- Data type: Unsigned8
- Range: $0 \le k \le 2^{8}-1$ (i.e., $0 \le k \le 255$)
- Size: 1 octet

Table 352 provides coding rules for Unsigned8 data.

Table 352 – Coding rules for Unsigned8

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

12.22.3.3.2.2 Unsigned16

The coding of an Unsigned16 is:

- Data type: Unsigned16
- Range: $0 \le k \le 2^{16}-1$ (i.e., $0 \le k \le 65\,535$)
- Size: 2 octets

Table 353 provides coding rules for Unsigned16 data.

Table 353 – Coding rules for Unsigned16

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

12.22.3.3.2.3 Unsigned32

The coding of an Unsigned32 is:

- Data type: Unsigned32
- Range: $0 \le k \le 2^{32}-1$ (i.e., $0 \le k \le 4\,294\,967\,295$)
- Size: 4 octets

Table 354 provides coding rules for Unsigned32 data.

Table 354 – Coding rules for Unsigned32

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

12.22.3.3.2.4 Unsigned64

The coding of an Unsigned64 is:

- Data type: Unsigned64
- Size: 8 octets
- Range: $0 \le k \le 2^{64}-1$ (i.e., $0 \le k \le 18\,446\,744\,073\,709\,551\,615$)

Table 355 provides coding rules for Unsigned64 data.

Table 355 – Coding rules for Unsigned64

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

12.22.3.3.2.5 Unsigned128

The coding of an Unsigned128 is:

- Data type: Unsigned128
- Size: 16 octets
- Range: $0 \leq k \leq 2^{128}$ -1 (i.e., $0 \leq k \leq$ 340 282 366 920 938 463 463 374 607 431 768 211 455)

Table 356 provides coding rules for Unsigned128 data.

Table 356 – Coding rules for Unsigned128

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2 127	2 126	2 125	2 124	2 123	2 122	2 121	2 120
2	2 119	2 118	2 117	2 116	2 115	2 114	2 113	2 112
3	2 111	2 110	2 109	2 108	2 107	2 106	2 105	2 104
4	2 103	2 102	2 101	2 100	2^{99}	2^{98}	2^{97}	2^{96}
5	2^{95}	2^{94}	2^{93}	2^{92}	2^{91}	2^{90}	2^{89}	2^{88}
6	2^{87}	2^{86}	2^{85}	2^{84}	2^{83}	2^{82}	2^{81}	2^{80}
7	279	2^{78}	2^{77}	2^{76}	2^{75}	2^{74}	2^{73}	2^{72}
8	2^{71}	2^{70}	2^{69}	2^{68}	2^{67}	2^{66}	2^{65}	2^{64}
9	2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}
10	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
11	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}
12	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}
13	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
14	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
15	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
16	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

12.22.3.3.2.6 UnsignedN

The coding of an UnsignedN, which is used in packed data structures is:

- Data type: UnsignedN
- Range: $0 \leq k \leq 2^{N)}-1$
- Size: N bits

12.22.3.4 Floating point values

12.22.3.4.1 Coding of floating-point values

This standard uses the encoding defined by ISO/IEC/IEEE 60559 (based on IEEE 754) for normalized floating-point values and NaNs. Each value is represented by three contiguous fields:

- S, the sign of the floating-point value, where 0 and 1 represent positive and negative, respectively, conveyed in a 1-bit field;
- E, the exponent of the value, in base 2, plus a bias B, conveyed in a field occupying N_E = about 1/4 of the total number of bits of the representation of the floating-point value, where the value of B is $2^{(N_E-1)}-1$;
- F, the fractional part of the value's mantissa, also in base 2, conveyed in the remaining N_F bits of the value's representation.

When E is not all zero bits or all one bits, the resulting numeric value is $(-1)^S \times 2^{(E-B)} \times (1.F)$. When E and F are both all zero bits the value represented is a signed zero.

When E is all one bits and F is all zero bits the value represented is a signed infinity. See ISO/IEC/IEEE 60559 for further information regarding real number representation, range and precision, including encoding of signed zero, signed infinity (overflow), de-normalized numbers (underflow), and NaNs.

12.22.3.4.2 Single-precision float

Single-precision floating-point values are represented contiguously as shown in Table 357, where N_E = 8, B = 127 and N_F = 23. This permits a single-precision floating point value to be calculated by the following equation, which applies when E is not all zero bits or all one bits:

$$(-1)^S \times 2^{(E-127)} \times (1,F)$$

Table 357 – Coding rules for single-pecision float

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Sign (S)	Exponent (E)						
1	+/-	2^7	2^6	2^5	2^4	2^3	2^2	2^1
	(E)	Fraction (F)						
2	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
3	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
4	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

12.22.3.5 Double-precision float

Double-precision floating-point values are represented contiguously as shown in Table 358, where N_E = 11, B = 1023 and N_F = 52. This permits a double-precision floating point value to be calculated by the following equation, which applies when E is not all zero bits or all one bits:

$$(-1)^S \times 2^{(E-1023)} \times (1,F)$$

Table 358 – Coding rules for double-precision float

Octet	Bits							
	7	6	5	4	3	2	1	0
1	Sign (S)	Exponent (E)						
	+/-	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}
2	Exponent (E) (continued)				Fraction (F)			
	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}
3	Fraction (F) (continued)							
	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}
4	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}
5	2^{-21}	2^{-22}	2^{-23}	2^{-24}	2^{-25}	2^{-26}	2^{-27}	2^{-28}
6	2^{-29}	2^{-30}	2^{-31}	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}
7	2^{-37}	2^{-38}	2^{-39}	2^{-40}	2^{-41}	2^{-42}	2^{-43}	2^{-44}
8	2^{-45}	2^{-46}	2^{-47}	2^{-48}	2^{-49}	2^{-50}	2^{-51}	2^{-52}

12.22.3.6 VisibleString

The coding of a visible string is:

- Type: VisibleString
- Range: See ISO/IEC 646 and ISO/IEC 2375: Defining registration number 2 + SPACE
- Coding: See ISO/IEC 646

NOTE See ISO/IEC 2375 for further details.

Table 359 provides coding rules for VisibleString data. If the size of the string is not determinable from other factors, then the size in octets is coded in one octet that immedately precedes the OctetString, as specified in Table 351.

Table 359 – Coding rules for VisibleString

Octet	Bits							
	7	6	5	4	3	2	1	0
1	First character in string							
2	Second character in string							
…	….							
N	Last character in string							

12.22.3.7 OctetString

The coding of an octet string is:

- Type: OctetString
- Coding: Binary

Table 360 provides coding rules for OctetString data. If the size of the string is not determinable from other factors, then the size in octets is coded in one octet that immedately precedes the OctetString, as specified in Table 351.

Table 360 – Coding rules for OctetString

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	First octet in string							
…	…							
N	Last octet in string							

12.22.3.8 BitString

The coding of a bit string that is not part of a superior packed structure is:

- Type: BitString
- Coding: Binary
- Size: Only multiples of 8 bits (i.e., multiples of octets) are supported for BitStrings that are not part of superior packed structures

Table 361 provides the general coding rule for BitString data. If the size of the string is not determinable from other factors, then the size in octets is coded in one octet that immedately precedes the BitString, as specified in Table 351.

Table 361 – Coding rules for BitString

Octet	Bits							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	(8xN-1) th	(8xN-2) th	(8xN-3) th	(8xN-4) th	(8xN-5) th	(8xN-6) th	(8xN-7) th	(8xN-8) th
	(bit position in string)							
2	(8xN-9) th	(8xN-10) th	(8xN-11) th	(8xN-12) th	(8xN-13) th	(8xN-14) th	(8xN-15) th	(8xN-16) th
…								
N	etc.							

12.22.3.9 SymmetricKey

A SymmetricKey is opaque. In this edition of this standard it is 128 bits. As such it is mapped, without interpretation, to an Octet16, which is sixteen octets in size.

12.22.4 Time-related data types

12.22.4.1 General

Time is continuous, potentially represented to nearly infinite precision in a nearly infinite range. Thus any reasonable representation of time has a specified finite resolution (e.g., 1 h, 1 s, 1 ns, 10^{-20} s, etc) and a specified range, such as [0..1 d) or (0..10 000 yr], modulo which any value of time must be represented.

Within this standard, TAINetworkTime is represented with a resolution of 2^{-16} s and a range of [0..2^{32}) s, modulo 2^{32} s. TAITimeRounded has the same range but rounds to the nearest 1 s and has a resolution of 2^{0} s (i.e., 1 s).

TAITimeDifference is intended for use to represent the difference between two diffferent values of TAINetworkTime. That difference is also represented modulo 2^{32} s, so that very large numeric values likely represent negative differences. The determination of what part of the 2^{32} s range of a TAITimeDifference value is interpeted as a positive difference, versus the part that is interpreted as a negative difference, is determined by the use of that difference.

EXAMPLE When differencing two TAINetworkTime values during processing of a TPDU nonce, the specified logic specifically addresses differences in a small signed range and then classifies all other differences as “out of range” without attempting to assign them to either the relatively distant past or the relatively distant future relative to the referenced TAI time instant.

12.22.4.2 TAINetworkTime

TAINetworkTime represents the network time in TAI time as a six-octet fixed-point binary value with a resolution of 2^{-16} s modulo 2^{32} s. Thus the high-order four octets represent the current TAI time in units of 1 s while the low-order two octets represent the fractional TAI time in units of 2^{-16} s.

- Data type: TAINetworkTime

  NOTE 1 This representation also applies to TAITimeDifference, which is a modulo difference.

- Valid range, expressed as an unsigned binary fixed-point value
  – whose integral component has the range 0..2^{32}-1 s (modulo 2^{32} s);
  – and whose fractional component has a resolution of 2^{-16} s.

  NOTE 2 Because all possible values occur repeatedly (cyclically) in a modulo representation such as TAINetworkTime, it is not possible to code special-meaning values within this range, as can be done with the endpoints of a linear range.

Table 362 shows the representation for TAINetworkTime, and for TAITimeDifference when interpreted as a modulo difference.

Table 362 – Coding rules for TAINetworkTime, and for TAITimeDifference when interpreted as a modulo difference

Octet	Bits								Interpretation
	7	6	5	4	3	2	1	0	
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Integral part of TAI time with granularity of 1 s
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	Fractional part of TAI time with granularity of 2^{-16} s
6	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	

12.22.4.3 TAITimeDifference

The coding of a TAITimeDifference is identical to that of TAINetworkTime. However, since it is a modulo value, it has potential interpretations as signed values. Those interpretations are:

- Data type: TAITimeDifference
- Valid range, expressed as a two’s-complement binary fixed-point value
  – whose integral component has the range -2^{32}..2^{32}-1 s;
  – whose fractional component has a resolution of 2^{-16} s; and
  – which is considered to “wrap” from positive to negative values at some Unsigned32 value for the integral component that is dependent on the specific usage scenario.

12.22.4.4 TAITimeRounded

TAITimeRounded represents the TAI time in integral seconds modulo the period of the representation, rounded to the nearest second. Its coding is:

- Data type: TAITimeRounded

- Valid Range: 0..2^{32}-1 s (modulo 2^{32} s)

Table 363 shows the representation for TAITimeRounded.

Table 363 – Coding rules for TAITimeRounded

Octet	Bits								
	7	6	5	4	3	2	1	0	Interpretation
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	TAI time with granularity of 1 s
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

12.22.4.5 Standard data structures

Standard data structures are coded by concatenating the coded values for the structure elements in order from the lowest numbered element to the highest numbered element, beginning at octet 1 of the coded result.

12.22.4.6 Null

The data type null has a size of zero (0) octets. The value null is often used for semantic consistency, where it represents the potential for content when no content has been identified.

12.22.4.7 Packed

The data type packed indicates that one or more elements of the standard data types have been concatenated together without gap to maintain octet alignment. Additionally, packed elements of BitString and BooleanArray type need not occupy an integral number of octets. The structure and composition of packed data is implicitly known by the correspondents.

NOTE BooleanArrays are usually represented as packed BitStrings.

12.22.4.8 Structured data

12.22.4.8.1 SEQUENCE

SEQUENCE is used to indicate structured data of the same or different standard data type(s). This is akin to a record construct.

This standard does not support sequences that contain optionally-present members. If such a need is identified, a separate sequence (structure) shall be defined for each such required sequence of members. Correspondents are required to have prior knowledge of the structure of the sequence; thus no mechanism is provided to convey its structure explicitly.

12.22.4.8.2 SEQUENCE OF

For data, SEQUENCE OF is used to indicate an array construct. Array content may either be conveyed in its entirety, or a specified individual element of an array may be conveyed.

For conveyance of an individual element, the data type of the element is implicitly known by the correspondents. Since some data types are variable in size, the size of the element is conveyed with the element data.

When arrays are conveyed in their entirety, they are encoded in row-major-order. The size of the array in octets shall also be included. The data type of the elements is also known implicitly by the corresponding endpoints, and is not explicitly indicated in the APDU. The dimension(s) of the array is(are) also implicitly known by the corresponding endpoints, and hence is(are) not explicitly included in the APDU.

NOTE Following standard matrix notation, rows are identified by the first index of a two-dimensional array and columns by the second index. For example, for the “C” programming language, a two-dimensional array consisting of two rows and three columns, which visually would be

$$\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\end{matrix},$$

might be defined as

```
int A[2][3] = { {1, 2, 3}, {4, 5, 6} }
```

The encoding of this standard would convey the elements of this array in the following order: 1, 2, 3, 4, 5, 6.

12.22.4.8.3 CHOICE

CHOICE represents a selection chosen from a predefined enumeration of acceptable possibilities. Content of the data varies based on the choice selected.

12.22.4.8.4 OPTIONAL

OPTIONAL specifies that the designated component need not be included in the containing structure.

12.22.4.8.5 IMPLICIT

IMPLICIT specifies that those coding aspects that identify type, size, choice selection, and presence or absence as an optional element are to be suppressed when that information is otherwise determinable from context, such as from other elements of the data structure.

NOTE When a data type declaration ends with an explicit integer specifying the size of the atomic type (e.g., Unsigned12) or number of elements of an array type (e.g., OctetString2), that integer is implicit in the declaration and is not carried in the PDU as a size explicitly-conveyed within the item itself. Thus an OctetStringN does not contain a field specifying N, but an OctetString does contain such a field (because the size is not implicit in the declaration).

12.23 Syntax

12.23.1 Application protocol data unit

12.23.1.1 Start of containing module

NOTE 1 The object identifier root for the following definitions was changed to an IEC-based root to support correction and evolution of the TSDU structure relative to that of the original ISA100.11a:2011 TSDU structure.

NOTE 2 The ASN.1 extensibility declaration “...” is used in each production that may be extended in future editions of this standard, or in industry-specific or vendor-specific ways for this edition.

```
IEC62734 (1 0 62734) edition (1) TSDU (1) ) DEFINITIONS
    IMPLICIT TAGS
    EXPORTS IEC62734_TSDU;
    ::= BEGIN
```

NOTE 3 For this edition of IEC 62734, the bit-level structure of IEC62734_TSDU is identical to that of ISA100_TSDU in ISA100.11a:2011, 11.23, so either prefix designates a data structure with identical concrete representation and similar associated semantics. The equivalent prefix declaration for ISA100.11a:2011 was

```
--  ( ISA ( ) ISA100.11a:2011 (71) ) DEFINITIONS
--      IMPLICIT TAGS
--      EXPORTS ISA100_TSDU;
--      ::= BEGIN
```

12.23.1.2 Top level definitions

```
IEC62734_TSDU :: = IMPLICIT CHOICE (
        individualAPDU                    ASLIndividualAPDU,
        concatenatedAPDU                  ASLConcatenatedAPDU
)
ASLIndividualAPDU ::= IMPLICIT CHOICE(
    confirmedRequestAPDU                  ASLConfirmedServiceRequest,
    confirmedResponseAPDU                 ASLConfirmedServicerResponse,
    unconfirmedRequestAPDU                ASLUnconfirmedServiceRequest,
    publicationAPDU                       ASLPublicationRequest
)
ASLConcatenatedAPDU ::= IMPLICIT SEQUENCE (
    IMPLICIT CHOICE (
        -- implicit based on the content of the APDU header, which is common across the choices
        confirmedRequest                  ASLConfirmedServiceRequest,
        confirmedResponse                 ASLConfirmedServiceResponse,
        unconfirmedRequest                ASLUnconfirmedServiceRequest
    )
)
```

NOTE This concatenation works because the size of each aperiodic APDU is either determined by explicit information or is implicit by service primitive definition.

12.23.1.3 Common substitutions

```
Float32 ::= REAL (WITH COMPONENTS(, base(2)) SIZE 32)        -- single-precision binary float
Float64 ::= REAL (WITH COMPONENTS(, base(2)) SIZE 64)        -- double-precision binary float

Integer8 ::= INTEGER (-128..127)                             -- 8-bit integer
Integer16 ::= INTEGER(-32 768..32 767)                       -- 16-bit integer
Integer32 ::= INTEGER(-4 294 967 296..4 294 967 295)         -- 32-bit integer

Unsigned8 ::= INTEGER(0..255)                                -- 8-bit unsigned
Unsigned16 ::= INTEGER(0..65 535)                            -- 16-bit unsigned
Unsigned32 ::= INTEGER(0..4 294 967 295)                     -- 32-bit unsigned
Unsigned64 ::= INTEGER(0..18 446 744 073 709 551 615)        -- 64-bit unsigned
Unsigned128 ::= INTEGER(0..340 282 366 920 938 463 374 607 431 768 212 455)  -- 128-bit unsigned

Octet1 ::= Unsigned8
DL16Address ::= Unsigned16
EUI64Address ::= Unsigned64
IPv6Address ::= Unsigned128
SymmetricKey ::= PACKED ARRAY [128] OF BIT  -- opaque, uninterpretable bit string

TAINetworkTime::= SEQUENCE (            -- referenced to TAI start time instant
    Seconds                             Unsigned32,
    FractionalSecond                    Unsigned16
)

TAITimeRounded ::= SEQUENCE (           -- referenced to TAI start time instant
    Seconds                             Unsigned32,
)

TAITimeDifference ::= SEQUENCE (        -- not referenced to TAI start time instant
    Seconds                             Unsigned32,
    FractionalSecond                    Unsigned16
)                                       -- See NOTE 1
```

NOTE 1 Since the representation of TAI time in this standard is modulo 2^{32} s, a value of type TAITimeDifference can be interpreted as either a positive or negative difference, with the two differing by 2^{32} s. Different uses of this type impose differing local limits on the expected range of numeric difference, which in turn determine how the modulo difference is interpreted. (E.g., -2^{31} s .. $+(2^{31}-1)$ s, or -2^{k} s .. $+(2^{32}-2^{k}-1)$ s for $0 \le k < 32$, etc.)

NOTE 2 The following are only used within packed data structures.

```
Unsigned1 ::= INTEGER(0..1)                                  -- 1-bit unsigned
Unsigned2 ::= INTEGER(0..3)                                  -- 2-bit unsigned
Unsigned3 ::= INTEGER(0..7)                                  -- 3-bit unsigned
Unsigned4 ::= INTEGER(0..15)                                 -- 4-bit unsigned
Unsigned5 ::= INTEGER(0..31)                                 -- 5-bit unsigned
Unsigned6::= INTEGER(0..63)                                  -- 6-bit unsigned
Unsigned7 ::= INTEGER(0..127)                                -- 7-bit unsigned
```

```
Unsigned9 ::= INTEGER(0..511)                                  -- 9-bit unsigned
Unsigned10 := INTEGER(0..1 023)                                -- 10-bit unsigned
Unsigned11 ::= INTEGER(0..2 047)                               -- 11-bit unsigned
Unsigned12 ::= INTEGER(0..4 095)                               -- 12-bit unsigned
Unsigned13 ::= INTEGER(0..8 191)                               -- 13-bit unsigned
Unsigned14 ::= INTEGER(0..16 383)                              -- 14-bit unsigned
Unsigned15 ::= INTEGER(0..32 767)                              -- 15-bit unsigned

Unsigned63 ::= INTEGER(0..9 223 372 036 854 775 807)           -- 63-bit unsigned
```

12.23.1.4 Application sublayer header

```
RequestResponse ::= Unsigned1 (
   request        (0),
   response       (1)
)

ObjectAddressingMode ::= Unsigned2 (
   compact        (0)                -- indicates 4-bit object identifiers
   midSize        (1)                -- indicates 8-bit object identifiers
   fullSize       (2)                -- indicates 16-bit object identifiers
   inferred       (3)                -- shall only be used as specified in 12.22.2.4.5.
)

ASLService ::= Unsigned5 (
   Publish                           0,
   AlertReport                       1,
   AlertAcknowledge                  2,
   Read                              3,
   Write                             4,
   Execute                           5,
   Tunnel                            6,
                                        -- values 7..31 are reserved for future use by this standard
)

ASLConfirmedServiceRequest ::= CHOICE(
   -- the first octet of the ConfirmedServiceRequest is constructed with
         -- bit 7 containing RequestResponse
         -- bits 6 and 5 containing ObjectAddressingMode
         -- bits 4..0 containing ASLService
   readCompact                       [3]     IMPLICIT ReadRequestPDU,       -- bit pattern: 0000 0011
   readMidSize                       [35]    IMPLICIT ReadRequestPDU,       -- bit pattern: 0010 0011
   readFull                          [67]    IMPLICIT ReadRequestPDU,       -- bit pattern: 0100 0011
   readInferred                      [99]    IMPLICIT ReadRequestPDU,       -- bit pattern: 0110 0011
   writeCompact                      [4]     IMPLICIT WriteRequestPDU,      -- bit pattern: 0000 0100
   writeMidSize                      [36]    IMPLICIT WriteRequestPDU,      -- bit pattern: 0010 0100
   writeFull                         [68]    IMPLICIT WriteRequestPDU,      -- bit pattern: 0100 0100
   writeInferred                     [100]   IMPLICIT WriteRequestPDU,      -- bit pattern: 0110 0100
   executeCompact                    [5]     IMPLICIT ExecuteRequestPDU,    -- bit pattern: 0000 0101
   executeMidSize                    [37]    IMPLICIT ExecuteRequestPDU,    -- bit pattern: 0010 0101
   executeFull                       [69]    IMPLICIT ExecuteRequestPDU,    -- bit pattern: 0100 0101
   executeInferred                   [101]   IMPLICIT ExecuteRequestPDU,    -- bit pattern: 0110 0101
   tunnelCompact                     [6]     IMPLICIT TunnelRequestPDU,     -- bit pattern: 0000 0110
   tunnelMidSize                     [38]    IMPLICIT TunnelRequestPDU,     -- bit pattern: 0010 0110
   funnelFull                        [70]    IMPLICIT TunnelRequestPDU,     -- bit pattern: 0100 0110
   tunnelInferred                    [102]   IMPLICIT TunnelRequestPDU      -- bit pattern: 0110 0110
)
```

```
ASLConfirmedServiceResponse ::= CHOICE(
    -- the first octet of the ConfirmedServiceResponse is constructed with
            -- bit 7 (MSBO containing RequestResponse) = 1 -- only response form is valid
            -- bits 6 and 5 containing ObjectAddressingMode
            -- bits 4..0 containing ASLService
    readCompact             [131]   IMPLICIT ReadResponsePDU,      --bit pattern: 1000 0101
    readMidSize             [163]   IMPLICIT ReadResponsePDU,      --bit pattern: 1010 0011
    readFull                [195]   IMPLICIT ReadResponsePDU,      --bit pattern: 1100 0011
    readInferred            [227]   IMPLICIT ReadResponsePDU,      --bit pattern: 1110 0011
    writeCompact            [132]   IMPLICIT WriteResponsePDU,     --bit pattern: 1000 0100
    writeMidSize            [164]   IMPLICIT WriteResponsePDU,     --bit pattern: 1010 0100
    writeFull               [196]   IMPLICIT WriteResponsePDU,     --bit pattern: 1100 0100
    writeInferred           [228]   IMPLICIT WriteResponsePDU,     --bit pattern: 1110 0100
    executeCompact          [133]   IMPLICIT ExecuteResponsePDU, --bit pattern: 1000 0101
    executeMidSize          [165]   IMPLICIT ExecuteResponsePDU, --bit pattern: 1010 0101
    executeFull             [197]   IMPLICIT ExecuteResponsePDU, --bit pattern: 1100 0101
    executeInferred         [229]   IMPLICIT ExecuteResponsePDU,  --bit pattern: 1110 0101
    tunnelCompact           [134]   IMPLICIT TunnelResponsePDU,   --bit pattern: 1000 0110
    tunnelMidSize           [166]   IMPLICIT TunnelResponsePDU,   --bit pattern: 1010 0110
    funnelFull              [198]   IMPLICIT TunnelResponsePDU,   --bit pattern: 1100 0110
    tunnelInferred          [230]   IMPLICIT TunnelResponsePDU   --bit pattern: 1110 0110
)

ASLUnconfirmedServiceRequest ::= CHOICE (
    -- the first octet of the UnconfirmedServiceRequest is constructed with
            -- bit 7 (MSBO containing RequestResponse) = 0 -- only request form is valid
            -- bits 6 and 5 containing ObjectAddressingMode
            -- bits 4..0 containing ASLService
    alertReportCompact          [1]     IMPLICIT AlertReportRequestPDU,    --bit pattern: 0000 0001
    alertReportMidSize          [33]    IMPLICIT AlertReportRequestPDU,    --bit pattern; 0010 0001
    alertReportFull             [65]    IMPLICIT AlertReportRequestPDU,    --bit pattern: 0100 0001
    alertReportInferred         [97]    IMPLICIT AlertReportRequestPDU,     --bit pattern: 0110 0001
    alertAcknowledgeCompact     [2]     IMPLICIT AlertAcknowledgeRequestPDU, --0x 0000 0010
    alertcknowledgeMidSize      [34]    IMPLICIT AlertAcknowledgeRequestPDU, --0x 0010 0010
    alertReportFull             [66]    IMPLICIT AlertAcknowledgeRequestPDU, --0x 0100 0010
    alertReportInferred         [98]    IMPLICIT AlertAcknowledgeRequestPDU, --0x 0110 0010
    tunnelCompact               [6]     IMPLICIT TunnelRequestPDU,      --bit pattern: 0000 0110
    tunnelMidSize               [38]    IMPLICIT TunnelRequestPDU,      --bit pattern: 0010 0110
    funnelFull                  [70]    IMPLICIT TunnelRequestPDU,      --bit pattern: 0100 0110
    tunnelInferred              [102]   IMPLICIT TunnelRequestPDU     --bit pattern: 0110 0110
)

ASLPublicationServiceRequest ::= CHOICE (
    -- the first octet of the PublicationServiceRequest is constructed with
            -- bit 7 (MSBO containing RequestResponse) = 0 -- only request form is valid for publication)
            -- bits 6 and 5 containing ObjectAddressingMode
    publishCompact              [0]     IMPLICIT PublishRequestPDU,     bit pattern: 0000 0000
    publishMidSize              [32]    IMPLICIT PublishRequestPDU,     bit pattern: 0010 0000
    publishFull                 [64]    IMPLICIT PublishRequestPDU      bit pattern: 0100 0000
                    -- inferred addressing is not used as there is no concatenation of publications
                    --  (see concentrator / dispersion objects)
)
```

12.23.1.5 Individual APDUs

```
SourceAndDestinationOIDs:: = IMPLICIT SEQUENCE (OCTET ALIGNED)(
        IMPLICIT CHOICE ( -- as determined by objectAddressingMode in bits 5 and 6 of first octet of APDU
                -- source object represents the initiator of the service primitive (.req or .rsp)
                -- destination object represents the recipient of the service primitive (.ind or .cnf)
                compact IMPLICIT PACKED SEQUENCE (
                        compactSourceObject         Unsigned4,
                        compactDestinationObject    Unsigned4
                )
                midSize IMPLICIT SEQUENCE (
                        midSizeSourceOID            Unsigned8,
                        midSizeDestinationOID       Unsigned8
                )
                fullSize IMPLICIT SEQUENCE (
                        fullSizeSourceOID           Unsigned16,
                        fullSizeDestinationOID      Unsigned16
                )
                inferred NULL
        )
```

```
ReadRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   readRequest                           ReadRequest
)

ReadResponsePDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   readResponse                          ReadResponse
)

WriteRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   writeRequest                          WriteRequest
)

WriteResponsetPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   writeResponse                         WriteResponse
)

ExecuteRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   executeRequest                        ExecuteRequest
)

ExecuteResponsePDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   executeResponse                       ExecuteResponse
)

TunnelRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   tunnelRequest                         TunnelRequest
)

TunnelResponsePDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   tunnelResponse                        TunnelResponse
)

AlertReportRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   alertReportRequest                    AlertReportRequest
)

AlertAcknowledgeRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   alertAcknowledgeRequest               AlertAcknowledgeRequest
)

PublishRequestPDU ::= IMPLICIT SEQUENCE
   soidDoid                              SourceAndDestinationOIDs,
   publishRequest                        PublishRequest
)
```

12.23.1.6 Periodic APDUs

```
PublishRequest ::= IMPLICIT SEQUENCE (
   IMPLICIT CHOICE ( -- implicitly determined by the corresponding application processes
      NativeValue                        IMPLICIT PublishedValue,             -- single published value
      NativeSequence                     IMPLICIT PublishedValueSequence,    -- sequence of published values
      HealthReportSequence               IMPLICIT HealthReportSequence,       -- publication HRCO
      nonNativeSequence                  IMPLICIT NonNativeSequence           -- publication tunnel
       )
)

PublishedValue ::= IMPLICIT SEQUENCE (
   contentVersion                        Unsigned8,    -- version of configuration of content published
   freshValueSequenceNumber              Unsigned8,    -- freshness of this set of data
   value                                 ProcessValueAndStatus
 )
```

```
PublishedValueSequence ::= IMPLICIT SEQUENCE (
   contentVersion                    Unsigned8,   -- version of configuration of content published
   freshValueSequenceNumber          Unsigned8,   -- freshness of this set of data
   publishedValues                   SEQUENCE OF ProcessValueAndStatus
 )

HealthReportSequence ::= IMPLICIT SEQUENCE (
   contentVersion                    Unsigned8,   -- version of configuration of content published
   freshValueSequenceNumber          Unsigned8,   -- freshness of this set of data
   healthReportSize                  Unsigned8,
   healthReport                      OCTET STRING
 )

NonNativeSequence ::= IMPLICIT OCTET STRING

ProcessValueAndStatus ::= IMPLICIT CHOICE ( -- based on publisher and subscriber application configuration
   analog       AnalogProcessValueAndStatus,
   boolean      BooleanProcessValueAndStatus
                -- NOTE   This choice element can be extended by industry consortia and vendors
 )

AnalogProcessValueAndStatus ::= IMPLICIT SEQUENCE (
      valueStatus                    PV_Status,
      analogProcessValue             Float32
 )

BooleanProcessValueAndStatus ::= IMPLICIT SEQUENCE (
      valueStatus                    PV_Status,
      booleanProcessValue            Boolean8     -- single Boolean value represented by a full octet
 )

PV_Status ::= PACKED SEQUENCE (OCTET ALIGNED) ( -- 1 octet (bit field sizes are: 2 + 1 + 3 + 2)
   quality                           PV_Quality,            -- 2 bits
   reservedSpareBit                  Unsigned1,             -- 1 bit
   IMPLICIT CHOICE ( -- selected by quality; all are        -- 3 bits
      [0] BadValueSubstatus          BadValueSubstatus,
      [1] UncertainValueSubstatus    UncertainValueSubstatus,
      [2] GoodValueSubstatus         GoodValueSubstatus,
      [3] otherSubstatus             Unsigned3                            -- reserved for future use
    ),                                                                    -- 1 spare code point
   limitStatus                       LimitStatus            -- 2 bits control anti-windup information
 )

PV_Quality ::= Unsigned2 (                 -- 2 bits
   badValue,                        (0),                                  -- value is bad
   uncertainValue                   (1),                                  -- value is uncertain
   goodValue                        (2),                                  -- value is good
   otherValue                       (3)                                   -- reserved for future use
 )                                                                        -- 1 spare code point

BadValueSubstatus ::= Unsigned3 (          -- 3 bits
   badValue_NonSpecific,            (0),
   badValue_ConfigurationError,     (1),
   badValue_NotConnected            (2),
   badValue_DeviceFailure,          (3),
   badValue_SensorFailure,          (4),
   badValue_NoCommunicationWithLUV  (5),
   badValue_NoCommunicationNoLUV    (6),
   badValue_OutOfService            (7)
 )                                                                        -- no spare code points

UncertainValueSubstatus ::= Unsigned3 ( -- 3 bits
   uncertainValue_NonSpecific,               (0),
   uncertainValue_LastUsableValue            (1),
   uncertainValue_SubstitutedOrManualEntry   (2),
   uncertainValue_InitialValue               (3),
   uncertainValue_SensorConversionInaccurate, (4),
   uncertainValue_RangeLimitsExceeded        (5)
   uncertainValue_SubNormal,                 (6),
   uncertainValue_Spare                      (7)                          -- reserved for future use
 )                                                                        -- 1 spare code point
```

```
GoodValueSubstatus ::= Unsigned3 (          -- 3 bits
   goodValue_NoSpecialConditionsExist   (0),
   goodValue_SpecialCondition1          (1),                    -- reserved for future use
   goodValue_SpecialCondition2          (2),                    -- reserved for future use
   goodValue_SpecialCondition3          (3),                    -- reserved for future use
   goodValue_SpecialCondition4          (4),                    -- reserved for future use
   goodValue_SpecialCondition5          (5),                    -- reserved for future use
   goodValue_SpecialCondition6          (6),                    -- reserved for future use
   goodValue_SpecialCondition7          (7)                     -- reserved for future use
 )                                                              -- 7 spare code points

LimitStatus ::= Unsigned2 ( -- 2 bits
   notLimited                           (0),
   lowLimited                           (1),
   highLimited                          (2),
   constant                             (3)      -- both high limited and low limited
 )                                                              -- no spare code points

highLowLimited LimitStatus ::= LimitStatus   constant        -- alternative symbolic name
lowHighLimited LimitStatus ::= LimitStatus   constant        -- alternative symbolic name
```

12.23.1.7 Aperiodic APDUs

```
CompactObjectIdentifier ::= Unsigned4
MidSizeObjectIdentifier ::= Unsigned8
FullSizeObjectIdentifier ::= Uinsigned16

ExtensibleInteger ::= IMPLICIT SEQUENCE (OCTET ALIGNED) (
      format                            Boolean1, -- 1 bit, FALSE for short form, TRUE for long form
      IMPLICIT CHOICE ( -- choice is established by the format field
         shortForm                      Unsigned7,        -- 7 bits     -- value shall be < 0x80
         longForm                       Unsigned15,       -- 15 bits    -- value shall be ≥ 0x80 and
       )                                                         -- <0x800; value < 0x80 are invalid
 )
```

An ExtensibleInteger shall use a minimal-size encoding. Use of a longForm to encode a value that could be encoded as a shortForm is invalid and shall be rejected as a protocol error.

```
AttributeClass ::= Unsigned2 ( -- code points for attribute alternatives
      sixBitNoIndexing                  (0),   -- 6-bit attribute identifier, no index
      sixBitOneDimension                (1),   -- 6-bit attribute identifier, one index (8 or 16 bits)
      sixBitTwoDimensions               (2),   -- 6-bit attribute identifier, two indices (each 8 or 16 bits)
      twelveBitExtended                 (3)    -- 12-bit attribute identifier
)

TwelveBitIndexClass ::= Unsigned2 ( - code points for 12-bit AID indexing alternatives
      twelveBitNoIndexing               (0),
      twelveBitOneDimension             (1),
      twelveBitTwoDimensions            (2),
      twelveBitReserved                 (3)
)
```

```
ExtensibleAttributeIdentifier ::= IMPLICIT PACKED SEQUENCE (OCTET ALIGNED) (
        attributeFormat                     AttributeClass              --2 bits
        IMPLICIT CHOICE ( -- choice is established by element attributeFormat
                sixBitNoIndexing            Unsigned6,
                sixBitOneDimension IMPLICIT SEQUENCE (OCTET ALIGNED) (
                        sixBitOneIndexAID   Unsigned6,
                        sixBitOneIndex      ExtensibleInteger,
                ),
                sixBitTwoDimensions IMPLICIT SEQUENCE (OCTET ALIGNED) (
                        sizBitTwoIndexAID   Unsigned6,
                        sixBitTwoIndexNo1   ExtensibleInteger,
                        sixBitTwoIndexNo2   ExtensibleInteger
                ),
                twelveBitExtended IMPLICIT SEQUENCE (OCTET ALIGNED) (
                        twelveBitIndex      TwelveBitIndexClass,
                        twelveBitAID        Unsigned12
                        CHOICE ( -- choice is established by the twelveBitIndexClass
                                            twelveBitNoIndexing NULL,
                                            twelveBitOneDimension : ExtensibleInteger,
                                            twelveBitTwoDimensions IMPLICIT SEQUENCE (OCTET ALIGNED) (
                                                TwelveBitTwoIndexNo1        ExtensibleInteger,
                                                TwelveBitTwoIndexNo2        ExtensibleInteger
                                            )
                        )
                )
        )
)
```

NOTE The four bits in the first octet and eight bits of the second octet of the attributeID are concatenated to form a longer Unsigned12 value when the 12-bit attributeID alternative is indicated. The four bits in the first octet are the most significant, and the eight bits in the second octet are the least significant.

```
ScalarType :::= Unsigned12 (
    Null                        (0)
    Boolean8                    (1),    -- single Boolean value represented by a full octet
    Integer8                    (2),
    Integer16                   (3),
    Integer32                   (4),
    Unsigned8                   (5).
    Unsigned16                  (6).
    Unsigned32                  (7),
    Float32                     (8),
    VisibleString               (9),    -- GenericSizeAndValue format
    OctetString                 (10),   -- GenericSizeAndValue format


    BitString                   (14),

    Float64                     (30),
    TAITimeDifference           (31),
    TAINetworkTime              (32)
)                                   -- all other code points are reserved for this standard
```

Primitive encoding shall be used for ScalarData, ArrayData, and StructureData value elements. No type information is included in the encoding.

```
GenericSizeAndValue ::= IMPLICIT SEQUENCE OF (
        SizeInOctets            ExtensibleInteger,      -- necessary for parsing (e.g., concatenations)
        DataValue               IMPLICIT SEQUENCE OF Octet1
 )

ServiceFeedbackCodeGenericSizeAndValue ::= GenericSizeAndValue
```

12.23.2 Alert reports and acknowledgments

```
AlertClass ::= Unsigned1 (                              -- 1 bit
    event                       (0),
    alarm                       (1)
 )
```

```
AlertCategory ::= Unsigned2 ( -- 2 bits
   deviceDiagnostic                         (0),
   communicationsDiagnostic                 (1),
   security                                 (2),
   process                                  (3)
 )

AlarmDirection ::= Unsigned1 ( -- 1 bit
   returnToNormalOrNoAlarm                  (0),    -- for alerts, set this value to 0; for alarm returns set this to zero
   inAlarm                                  (1)     -- to report an alarm condition, set this value to 1.
)
```

This standard presently does not define standard alerts for the following industry-independent AL-defined objects:

- UAPMO;
- ARO;
- UDO;
- Concentrator;
- Dispersion;
- Tunnel;
- Interface.

```
ASLMO_Communication_Alerts ::= ENUMERATED (
   malformed_APDU                            (0),
                     -- values 1..50 are reserved for future use by this standard
                     -- values 51..100 are reserved for future use by standard profiles
                     -- vendor-specific codes range 101..255
)

AI_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
   outOfServiceAlarm                        (0),
   highAlarm                                (1),
   highHighAlarm                            (2),
   lowAlarm                                 (3),
   lowLowAlarm                              (4),
   deviationLowAlarm                        (5),
   deviationHighAlarm                       (6)
                     -- values 7..50 are reserved for future use by this standard
                     -- values 51..100 are reserved for future use by standard profiles
                     -- vendor-specific codes range 101..255
)

AO_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
   outOfServiceAlarm                        (0),
   highAlarm                                (1),
   highHighAlarm                            (2),
   lowAlarm                                 (3),
   lowLowAlarm                              (4),
   deviationLowAlarm                        (5),
   deviationHighAlarm                       (6)
                     -- values 7..50 are reserved for future use by this standard
                     -- values 51..100 are reserved for future use by standard profiles
                     -- vendor-specific codes range 101..255
 )

BI_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
   outOfServiceAlarm                        (0),
   discreteAlarm                            (1)
                     -- values 2..50 are reserved for future use by this standard
                     -- values 51..100 are reserved for future use by standard profiles
                     -- vendor-specific codes range 101..255
 )
```

```
BO_ProcessAlerts ::= ENUMERATED (
   outOfServiceAlarm                          (0),
   discreteAlarm                              (1)
                    -- values 2..50 are reserved for future use by this standard
                    -- values 51..100 are reserved for future use by standard profiles
                    -- vendor-specific codes range 101..255
 )

ARMO_Alerts ::= ENUMERATED (
   AlarmRecoveryStart                         (0),
   AlarmRecoveryEnd                           (1)
                    -- values 2..50 are reserved for future use by this standard
                    -- values 51..100 are reserved for future use by standard profiles
                    -- vendor-specific codes range 101..255
)

IndividualAlertID :: = Unsigned8                     -- unique ID associated with an individual alert.
                                                     --  Assigned by the application process in the UAL.

statusSignalNamur107  ::= Unsigned8 (
   failure                                    (0),     --
   checkFunction                              (1),     --
   offSpec                                    (2),     --
   maintenanceRequired                        (3),     --
)

IndividualAlert ::= IMPLICIT PACKED SEQUENCE (OCTET ALIGNED)(
      individualAlertID                       IndividualAlertID,
      DetectingObjectTransportLayerPort       Unsigned16,
      DetectingObject                         Unsigned16,
      DetectingObjectType                     Unsigned16,
      detectionTimeTAINetworkTime,            -- 48 bits
      alertClass                              AlertClass,           -- 1 bit
      alarmDirection                          AlarmDirection,       -- 0: event or alarm return; 1: alarm report
      alertCategory                           AlertCategory,        -- 2 bits: device, comm, security, process
      alertPriority                           AlertPriority,        -- 4 bits
      alertType                               Unsigned8,            -- object category and type dependent
      associatedDataSize                      ExtensibleInteger,
      associatedData                            -- present when associatedDataSize > 0
         CHOICE  ( -- choice is based on AlertCategory
            communicationsDiagnostic IMPLICIT SEQUENCE OF Octet1 OPTIONAL,
            security  IMPLICIT SEQUENCE OF Octet1 OPTIONAL,
            process IMPLICIT SEQUENCE OF Octet1 OPTIONAL,
            deviceDiagnostic IMPLICIT SEQUENCE
                  (   namur107Status             statusSignalNamur107,
                      detailedInformation        IMPLICIT SEQUENCE OF Octet1 OPTIONAL
                  )                                 -- may include additional information per NAMUR-107
            ) OPTIONAL
)

AlertReportRequest ::= ( -- note: client OID not present; ARMO is implied
      alert          IndividualAlert
 )

AlertAcknowledgeRequest ::= (
      alertID        IndividualAlertID                 -- server is always ARMO
 )

AlertPriority ::= Unsigned4
```

Alert priority is a value that indicates the importance of the alert. A larger value implies a more important alert. Host systems map device priorities into host alert priorities that usually include the categories:

- urgent,
- high,
- medium,
- low, and
- journal.

The recommended mapping of alert priority values into these categories is specified in 12.17.5.2.2.22.

```
MalformedAPDUClass ::= AlertClassevent;

MalformedAPDUAlertCategory ::= AlertCategorycommunicationsDiagnostic

MalformedAPDUAlertType ::= AlertTypemalformedAPDUCommunicationAlert

MalformedAPDUAlertPriority = 7              -- Mid-range of medium priority alerts

MalformedPDUAlertValue ::= IMPLICIT SEQUENCE ( -- alert value sent by ASL to DMAP
   sourceAddress                  IPv6Address,          -- 128 bits to ensure address uniqueness.
   thresholdExceeded              Unsigned16,
   TimeWindow                     TAITimeDifference
 )

MalformedPDUAlertValueSize ::= 24       -- sizeof(MalformedPDUAlertValue)
```

12.23.3 Service feedback code

NOTE Service feedback code is used to indicate status (e.g., success), warning (e.g., value limited), or error (e.g., incompatible mode).

```
ServiceFeedbackCode ::= Unsigned8 (       -- 1octet
   -- standard error codes, range 0..127
   success                          (0)     -- success
   failure                          (1)     -- generic failure
   other                            (2),    -- reason other than that listed in this enumeration
   invalidArgument                  (3),    -- invalid attribute to a service call
   invalidObjectID                  (4),    -- invalid object ID
   invalidService                   (5),    -- unsupported or illegal service
   invalidAttribute                 (6),    -- invalid attribute index
   invalidElementIndex              (7),    -- invalid array or structure element index (or indices)
   readOnlyAtrribute                (8),    -- read-only attribute
   valueOutOfRange                  (9),    -- value is out of permitted range
   inappropriateProcessMode         (10),   -- process is in an inappropriate mode for the request
   incompatibleMode                 (11),   -- value is not acceptable in current context

   invalidValue                     (12),   -- value (data) not acceptable for other reason
                                            --  (e.g., too large, too small, invalid engineering units code)
   internalError                    (13),   -- device internal problem
   invalidSize      (14),                   -- size is not valid (may be too big or too small)
   incompatibleAttribute            (15),   -- attribute not supported in this version
   invalidMethod                    (16),   -- invalid method identifier
   objectStateConflict              (17),   -- state of object in conflict with action requested
   inconsistentContent              (18),   -- the content of the service requested is inconsistent
   invalidParameter                 (19),   -- value conveyed is not legal for method invocation
   objectAccessDenied               (20),   -- object is not permitting access
   typeMismatch                     (21),   -- data not as expected (e.g., too many or too few octets)
   deviceHardwareCondition          (22),   -- device specific hardware condition prevented request from
                                            --  succeeding (e.g., memory defect problem)
   deviceSensorCondition            (23),   -- problem with sensor detected
  deviceSoftwareCondition           (24),   -- device specific software condition prevented request from
                                            --  succeeding (e.g., local lockout, local write protection,
                                            --   simulating in progress)
   fieldOperationCondition          (25),   -- field specific condition prevented request from succeeding
                                            --  (e.g., lockout, or environmental condition not in range)
   configurationMismatch            (26),   -- a configuration conflict was detected
   insufficientDeviceResources      (27),   -- e.g., queue full, buffers / memory unavailable
   valueLimited                     (28),   -- e.g., value limited by device
   dataWarning                      (29),   -- e.g., value has been modified due to a device specific reason
   invalidFunctionReference         (30),   -- function referenced for execution is invalid
   functionProcessError             (31),   -- function referenced could not be performed due to a device
                                            --  specific reason
```

```
   warning                                (32),   -- successful, but there is additional information that may be of
                                                  --  interest to the user which may, for example be conveyed via
                                                  --  accessing an attribute or by sending an alert.
  writeOnlyAttribute                      (33),   -- write-only attribute (e.g., a command attribute)
   operationAccepted                      (34),   -- method operation accepted
   invalidBlockSize                       (35),   -- upload or download block size not valid
   invalidDownloadSize                    (36),   -- total size for upload not valid
   unexpectedMethodSequence               (37),   -- required method sequencing has not been followed
   timingViolation                        (38),   -- object timing requirements have not been satisfied
   operationIncomplete                    (39),   -- method operation, or method operation sequence not
                                                  --  successful
   invalidData                            (40),   -- data received is not valid
                                                  --  (e.g., checksum error, data content not as expected, etc.)

   dataSequenceError                      (41),   -- data is ordered; data received is not in the order required
                                                  --  example: duplicate data was received
   operationAborted                       (42),   -- operation aborted by server
   invalidBlockNumber                     (43),   -- invalid block number
   blockDataError                         (44),   --error in block of data, example, wrong size, invalid content
   blockNotDownloaded                     (45),   -- the specified block of data has not been successfully
                                                  --  downloaded
   writeProtected                         (46),   -- data is write protected, so write operation is invalid
   invalidMode                            (47),   -- operation did not succeed due to invalid mode
      -- ...                              -- range 48..127 is reserved for future use of this standard
                                          -- vendor-specific device-specific feedback codes, range 128..255
   vendorDefinedError_128                 (128),  -- redefinable by each device vendor for each device type
      -- ...
   vendorDefinedError_254                 (254),  -- redefinable by each device vendor for each device type
   extensionCode                          (255)   -- indicates a two octet field size follows for an extended service
                                                           feedback code value
)                                         -- 123 values redefinable by each device vendor for each device type
```

12.23.4 Read, write, and execute

```
RequestID ::= Unsigned8
ReadRequest ::= IMPLICIT SEQUENCE (
   requestID                              RequestID,
   targetAttribute                        ExtensibleAttributeIdentifier
 )

ApduResponseControlData ::= PACKED IMPLICIT SEQUENCE (
   ,
   Spare                                  Unsigned7,           -- redefinable in future editions of this standard
   ForwardCongestionNotificationEcho      Boolean1  -- TRUE when congestion in forward (request) path detected
)

ReadResponse ::= IMPLICIT SEQUENCE (
   requestID                              RequestID,           -- matches corresponding ReadRequest
   apduControl                            ApduResponseControlData,
   readValue                              ServiceFeedbackCodeGenericSizeAndValue
 )

WriteRequest ::= IMPLICIT SEQUENCE (
   requestID                              RequestID,
   targetAttribute                        ExtensibleAttributeIdentifier
   value                                  GenericSizeAndValue
 )

WriteResponse ::= IMPLICIT SEQUENCE (
   requestID                              RequestID,           -- matches corresponding WriteRequest
   apduControl                            ApduResponseControlData,
   serviceFeedbackCode                    ServiceFeedbackCode
)

MethodInvocationRequest ::= IMPLICIT SEQUENCE (
   methodID                               Unsigned8,
   requestParametersSize                  ExtensibleInteger,
   requestParameters                      IMPLICIT SEQUENCE of Octet1 OPTIONAL
      -- primitive encoding; data type known by correspondents
      -- requestParameters only present if requestParametersSize >0
)
```

```
MethodInvocationResponse ::= IMPLICIT SEQUENCE (
   responseParametersSize                ExtensibleInteger,
   responseParameters                    IMPLICIT SEQUENCE of Octet1 OPTIONAL
      -- primitive encoding; data type known by correspondents
      -- responseParameters only present if responseParametersSize >0
)

ExecuteRequest :: = IMPLICIT SEQUENCE (
      requestID                          RequestID,
      methodinvocationRequest            MethodInvocationRequest    -- data type(s) specified by standard
 )

ExecuteResponse ::= IMPLICIT SEQUENCE (
      requestID                          RequestID,
      apduControl                        ApduResponseControlData,
      serviceFeedbackCode                ServiceFeedbackCode,
      methodInvocationResponse           MethodInvocationResponse    -- data type(s) specified by standard
 )
```

12.23.5 Tunnel

```
TunnelRequest ::= IMPLICIT SEQUENCE (
   length                                ExtensibleInteger,
   tunnelPayload                         SEQUENCE OF Octet1
 )

TunnelResponse ::= IMPLICIT SEQUENCE (
   apduControl                           ApduResponseControlData,
   length                                ExtensibleInteger,
   tunnelPayload                         SEQUENCE OF Octet1
 )
```

12.23.6 End of contained module

```
END
```

12.24 Detailed coding examples (INFORMATIVE)

12.24.1 Read

Scenario: Client object 11 wishes to read data from server object 12, attribute 3. The response indicates the read is successful and returns a value of size two octets.

Table 364 illustrates an example of a request to read multiple values.

Table 364 – Coding example: Read request for a non-indexed attribute

Encoding of octets in hexadecimal	Semantic
03	Read request
BC	Client (source)object ID = 11_{10} Server (destination) object ID = 12_{10}
XX	Request identifier
03	Attribute ID = 3 (attribute is scalar)

Table 365 illustrates an example of a response to a request to read multiple values.

Table 365 – Coding example: Read response for a non-indexed attribute

Encoding of octets in hexadecimal	Semantic
83	Read response
CB	Server (source) object iD = 12_{10} Client (destination) object ID = 11_{10}
XX	Request identifier (same value as for Request identifier that was included in the corresponding service request)
00	Success
02	Value is two octets long
YY YY	Value

12.24.2 Tunnel

Scenario: Object 16 in the client is sending a message to object 20 in the server. The content of the message is to be passed through to the server object.

Table 366 illustrates an example of a tunnel service request that has payload size of 9 octets.

Table 366 – Coding example: Tunnel service request

Encoding of octets in hexadecimal	Semantic
06	Tunnel request
09	Size
(9 octets of tunneled data)	Data being tunneled

13 Provisioning

13.1 General

A device conforming to this standard is considered provisioned when the device has the information required to communicate with a target network and initiate a join request to the system/security manager of the target network. In this document, a target network is defined as a network the device is being provisioned to join. The information required to initiate the join request includes both security (trust-related) information and network-related information. Clause 13 specifies the over-the-air provisioning procedure and message format where the Type A field medium is used and out-of-band message formats where the Type A field medium is not used to provision the trust-related and network-related information.

Over-the-air provisioning uses join processes to set up a connection between the provisioning device and the device being provisioned. The join process is described in 6.3.9.2 and follows two optional paths, one defined for a device joining with trust-related information based on a symmetric key, and another defined for a device joining with trust-related information based on an asymmetric key. Out-of-band provisioning may not use join processes and provision the information via another wired or wireless means.

The goal of the provisioning process is to provide enough information so that one of these paths can be taken by the device.

The provisioning process involves a device that implements the provisioning role by providing the network-related and trust-related information to the new device. During provisioning, the operator can use the provisioning device, acting as a proxy for the system manager, to decide if a new device should be connected to the network or not, with information from the security manager. In another example, a copy of the list of allowed devices can be obtained from the security manager, allowing the provisioning device to make a local decision. When the target network is a secure network both trust-related and network-related information needs to be provisioned; for unsecured networks the default key (K_global) is used as the trust-related information. Once a device is provisioned, it is ready to join the target network. Thereafter, usually without human intervention, the security manager of the target network may either accept or reject the join request to the target network from the device based on the provisioned information.

NOTE In this standard, various aspects related to installation of the trust-related and network-related information in a device, conveyance of this information to the security manager, and establishment of trust are described in different clauses. Installation of the trust-related and network-related information is described in Clause 13. Conveyance of the information to the security manager is described in Clause 9 and Clause 10. Establishment of trust is described in 7.4.4.3.2.

13.2 Terms and definitions for devices with various roles or states

The following terms are defined for devices with various roles or states:

- **Device being provisioned (DBP)**: A device that needs to be provisioned, or is in the process of being provisioned. The device may be missing all or part of the information required to join a network.

  NOTE 1 A device that contains old information relating to a network often is updated by provisioning it with new information.

- **Target network**: The network that the DBP is being provisioned to join.
- **Provisioning device (PD)**: A device that implements the role of provisioning another device to allow that device to join the target network. A PD need not be a device implementing only the provisioning role; rather, it could be:
  - the system/security manager of the target network;

    NOTE 2 The system/security manager role is distributable, e.g., to a designated set of devices in the target network.

  - a device, such as a handheld device containing a system/security manager, that uses the protocol suite specified by this standard to provision the DBP through a separate, temporary mini-network; or
  - a device that uses out-of-band (OOB) communication, such as infrared, near field communications (NFC), or plugs, to provision the DBP, where the OOB communication is outside the scope of this standard.
- **Default network**: The network whose network identifier is 1.
- **Provisioning network**: A network formed between the PD and the DBP. If the PD is a handheld, then the provisioning mini-network is the network formed between the handheld and the DBP. If the provisioning device is the security manager of the target network, then the provisioning network may be a separate logical network on the target network itself.
- **Join key (K_join)**: A symmetric join key used to join a secure target network. The value of key K_join is intended to be secret, and thus is intended to offer data confidentiality. The value of key K_join is updated during provisioning to a new value that is known only to the target network security manager and the device.[10]
- **Default join key value (K_global)**: A symmetric join key with a published value. The value of K_global is not intended to be a secret; its value is well known. It therefore does not offer data confidentiality, but does help improve data integrity. Its purpose is to establish connectivity between devices compliant to this standard that do not share a secret join key. Such connectivity is needed for:
  - over-the-air (OTA) provisioning of target network related information;

[10] Appropriate mechanisms are provided so that the protocol suite defined by this standard cannot be used to read the current value of the join key from a device. Note that the secrecy of join keys cannot be enforced by this standard.

  – OTA reading of device identity and configuration settings;
  – OTA authentication of device credentials; and
  – OTA updating of join key K_join (the latter two steps using asymmetric cryptography).

  The value of the default join key shall be K_global, as defined in 7.2.2.2.

- **Open join key (K_join = K_open)**: A published non-secret value for the join key (K_join). This special value for the join key is used to join a provisioning network so that certain OTA symmetric-key only provisioning methods can be facilitated. The actual value for this key is defined in 7.2.2.2.
- **Physical and logical networks**: A physical network is a set of physical devices that communicate with each other, possibly through multiple hops. A logical network is a network instance that runs on the physical network. One physical network may support multiple logical networks. Logical networks have different individual priority and security properties. For example, the target network and the provisioning network are two logical networks that exist on a physical network.
- **Idle state**: Device state that is not actively participating in the wireless network,
- **Provisioning state**: The device is in the provisioning phase.
- **Provisioned state**: The device received enough information to join target network, and got the DMO.Join_Command=1.
- **Factory defaults**: The default configuration of a field device as it comes out of a manufacturing facility. The default configuration has K_global and K_join equal to their default values, and OTA provisioning allowed. An operational device may be reset to factory default, either by the system manager when it is part of a secured network or by OOB means using a provisioning device. Factory defaults for the provisioning process are summarized in Table 367. Only the system manager shall have the authority to reset a device to factory defaults via the network.

This specification does not preclude devices that do not allow reset to factory defaults.

13.3 Provisioning procedures

All field devices compliant with this standard shall implement a standard object called the device provisioning object (DPO). Attributes of the DPO in the DBP shall specify the information required to initiate a join request to the target network. The device shall retain all attributes of the DPO through a power cycle or battery replacement. The device provisioning object is described in detail in 13.9.1.

This specification does not preclude that the system manager can have the DPO, for example, store the security manager's EUI64Address.

PDs shall implement a device provisioning service object (DPSO) that contains information intended for the DBPs that are serviced by the PD.

Provisioning involves setting the attributes of the DPO. The attributes in the DPO contain both network-related and trust-related information. These attributes can be set via three different means:

- they may be pre-installed during device manufacture; or
- they may be set using OOB means; or
- they may be set by a PD using a provisioning network, where the PD acts as a proxy for a security manager/system manager of a target network to provide the trust-related and network-related information for that target network.

All devices complying with this standard shall support the formation of the provisioning network using only the full protocol suite defined by this standard (PhL, DL, NL, and TL), i.e., not requiring any other mechanism. However, this standard does not disallow provisioning by OOB communication means.

When using the Type A field medium (5.2.6.4) in the provisioning network, standard PDUs shall be used to set the attributes of the DPO, which defines a set of default read-only attributes for the formation of either the provisioning network or another unsecured network. The default attributes include published default symmetric keys (K_join = K_global and K_join = K_open), a default D-subnet identifier, and a default set of channels. Since this set of default attributes is known and contained in the DPO of all devices conforming to this standard, those attributes provide a means for all devices to join a provisioning network.

The DPO includes an attribute, DPO.Allow_Provisioning, that specifies whether access to the attributes of the object via the default open instance is either allowed or blocked.

Some devices may implement an external mechanism (i.e., a switch) that will lock the provisioning state (either blank or provisioned) of the device, to minimize battery consumption and also to minimize the likelihood that a rogue PD will re-provision a device.

13.4 Pre-installed symmetric keys

The formation of a provisioning network is not a necessary step for provisioning; the trust-related information can be pre-installed in a device. For example, it is possible for a user to delegate (partly) the provisioning of devices to a device manufacturer or to a third party. A device manufacturer may pre-program secret symmetric join keys into devices, and may supply this same secret symmetric join key data to the user so that the data can be loaded to the system/security manager of the target network. Alternatively, the user may stipulate to the device manufacturer what symmetric key shall be loaded. In this case, the DPO of a device shall be pre-installed with the target network information and the target symmetric join key K_join. Depending on the application, two or more devices may share the same secret information. Devices with pre-installed trust information and target network information can proceed directly to the join process.

When a devices has pre-installed trust-related information but no target network-related information, it shall be possible to provision the device with necessary network information. This facilitates having the device receive advertisements from the target network on the intended channels, expediting the join process and present join requests only to the target network. If the network-related information is not provisioned, a device may use the default network settings to scan for advertisements from all networks in its vicinity (including those of competitors of the device's owning organization).

13.5 Provisioning using out-of-band mechanisms

Devices without pre-installed symmetric keys need to be authenticated and then provisioned with trust information. As noted earlier, this can be accomplished either through the provisioning network Type A field medium over-the-air or through OOB mechanisms.

OOB communication means include, but are not limited to, infrared, wired connectors, memory cards, keyboards on devices, NFC, and plugs. The mechanism of OOB communications is outside the scope of this standard. The attributes of the DPO that specify the joining to the target network should be set to the same values regardless of the means used (over-the-air or OOB).

13.6 Provisioning networks

13.6.1 General

In addition to OOB-provisioning and factory pre-provisioning, this standard defines the formation of a standard network for provisioning devices over-the-air (OTA) using the Type A field medium. The default symmetric join key (K_global) or open symmetric join key (K_join = K_open) may be used as the trust-related information for the formation of the OTA provisioning network. The default join key (K_global) is used for the formation of the provisioning network to obtain target network-related information and target network join key and for devices with asymmetric cryptographic capability. The join key (K_join = K_open) is used for the formation of a provisioning network where both trust-related and network-related information is provisioned over-the-air. This form of provisioning is insecure and by default system managers and provisioning devices shall not allow joining with this join key.

A PD that has asymmetric cryptographic capabilities distinguishes the method with the key used to generate the MIC in the Security_Sym_Join().request. In the PD, the MIC generated by the device joining the default network needs to be validated a maximum of twice – one for K_open and the other for K_global. If the security manager detects that K_global is used for the MIC, the DBP shall be provisioned using asymmetric cryptography. Otherwise, the DBP shall be provisioned using the K_open symmetric key.

The provisioning network can either be an isolated mini-network formed with a handheld device, or it can be a separate logical network on the target network itself. In the latter case, connectivity from the DBP to the advertising router is open but connectivity further on, from that advertising router to the system manager, is protected by the existing session and thus secured. If the logical provisioning network is on the target network, the application objects of the system/security managers on the target network and the logical provisioning network (e.g., DPSO) can communicate with each other within the same device.

Figure 135 illustrates the provisioning (mini-)network.

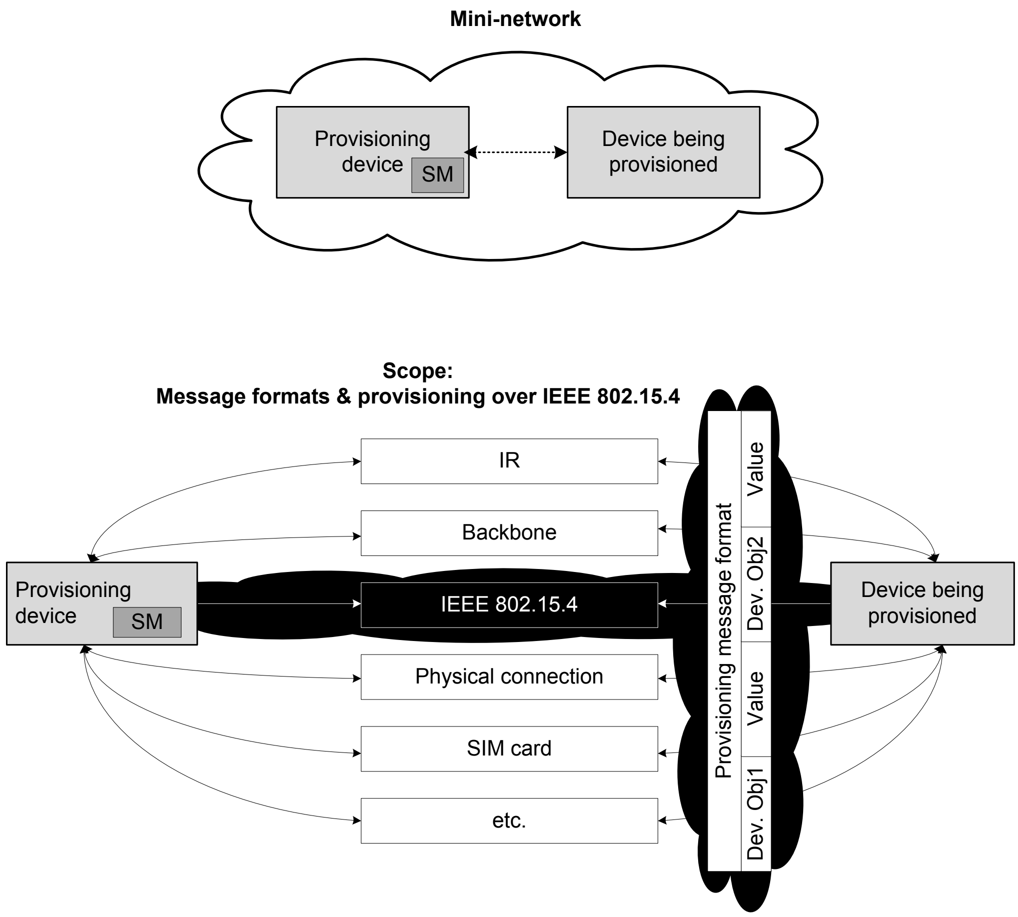


Figure 135 – The provisioning network

OTA provisioning uses a PD that can be either:

- a handheld configurator that forms an isolated mini-network with the DBP. This handheld has its own system/security manager and an advertising router functionality; or
- the system/security manager of the logical provisioning network on the target network.

NOTE When a PD is used for OTA provisioning, it forms a mini-network and functions temporarily as both the system manager and security manager for the DBP.

13.6.2 Provisioning over the air using asymmetric cryptography

DBPs that are capable of performing asymmetric cryptographic calculations shall use the default join key (K_global) to join a provisioning network. The DBP receives advertisements whose D-subnet ID = 1 from nearby advertising routers and initiates a join request using the default symmetric key. A successful join process results in the PD and the DBP having established a contract for further communication. The PD then uses standard AL primitives (such as read and write) to transfer the network-related information contained in its DPSO to the DPO of the DBP.

For provisioning the trust-related information the PD interrogates the DBP; i.e., it reads its credentials (e.g., DPO.PKI_Certificate or multiple DPO.PKI_Certificates; see Annex G), and sends those credentials to the security manager. The security manager of the provisioning network checks the credentials of the DBP and validates the authenticity of the DBP through a challenge-response mechanism. The security manager/system manager may ask for further confirmation from the user through a GUI to provision the DBP. Once accepted, the system manager provides the DBP with the secret join key, K_join, so that the DBP can join the target network either immediately or at a later time, using that join key. When this new key is transmitted over-the-air, it shall be encrypted by the asymmetric key of the DBP, which is part of its certificate, so that it cannot be recovered by an eavesdropper while in transit.

Security managers conforming to this standard are not required to have asymmetric cryptographic capabilities; hence, some security managers may not be able to accept or provision devices using asymmetric cryptographic capabilities. When the DBP joins the provisioning network using K_global, security managers and PDs not capable of asymmetric cryptographic calculations shall not transmit the trust-related information to the DBP.

In addition to a high level of security, asymmetric cryptographic modules and certificates provide a convenient and easy means for devices to establish communication with the security manager of the target network and to be provisioned without the use of additional tools. It is recommended that manufacturers of security manager devices that lack support for asymmetric cryptography provide adequate means (e.g., memory, processing power, or optional peripherals, etc.) to upgrade such security managers, upon user request, to support asymmetric cryptography.

13.6.3 Provisioning over the air using an open symmetric join key

This standard allows PDs to provision devices that do not have asymmetric cryptographic capabilities to be provisioned over-the-air. For this purpose, a well-known open symmetric join key (K_join = K_open) is used. By default, the security manager in the PD shall not permit OTA provisioning with the open symmetric key, K_open. The provisioning network is not secure in itself, since it uses a published open key and join key for the target network, and thus requires compensating measures, such as a secure physical connection or use of asymmetric cryptography, for security.

NOTE 1 In OTA provisioning with K_global, the security information (i.e., join key) is encrypted with an asymmetric key while transmitting.

NOTE 2 The use of an open symmetric join key for provisioning is not a secure procedure. An eavesdropping device may be able to obtain the join keys to the target network and pose a security risk when this provisioning procedure is used. This provisioning procedure can only be used in applications where the security risk is minimal and the user is either not concerned or has taken sufficient measures to avoid eavesdropping. Such exposure can be avoided by using asymmetric crypto-based provisioning or OOB provisioning.

Use of the symmetric join key K_open for provisioning is a configuration option. DBPs may be pre-configured not to use OTA provisioning with this key. By default, security managers and PDs shall reject join requests from all devices that send join requests using the K_open symmetric join key. Security managers and PDs need to be configured to accept join requests using the K_open join key. It is permissible for security managers and PDs not to permit such configuration.

A device that joins a provisioning network using the K_open join key may be provisioned by the PD with the join key for a target network. However, once provisioned with a new join key for the target network, the device shall not be allowed to use the K_open symmetric join key unless the device is reset to factory defaults. Thus the only permissible means for the device to reuse this key for provisioning is to reset the device to factory defaults.

The provisioning procedure using the K_open symmetric join key can be used either in the provisioning (mini-)network or through a separate logical network on the target network. The DBP receives an advertisement from a provisioning network and a standard join request is sent using a symmetric key (K_join = K_open). If the request is accepted, the DBP joins the provisioning network and a contract is established between the DBP and PD. Application-level read/write primitives and methods are available to the PD to provision the trust-related and network-related information; this includes, for example, provisioning the target join key using the DPO.Write_Join_Key method.

The provisioning (mini-)network can also be used for device configuration. Since a contract has already been established, the PD may also use the network (either OTA or OOB) to configure the DBP.

13.7 State transition diagrams

The options discussed thus far for provisioning are shown below in state transition diagrams.

Figure 136 depicts the state transitions relevant for provisioning through the lifecycle of a field device. The diagram depicts states at a manufacturing site and a user site.

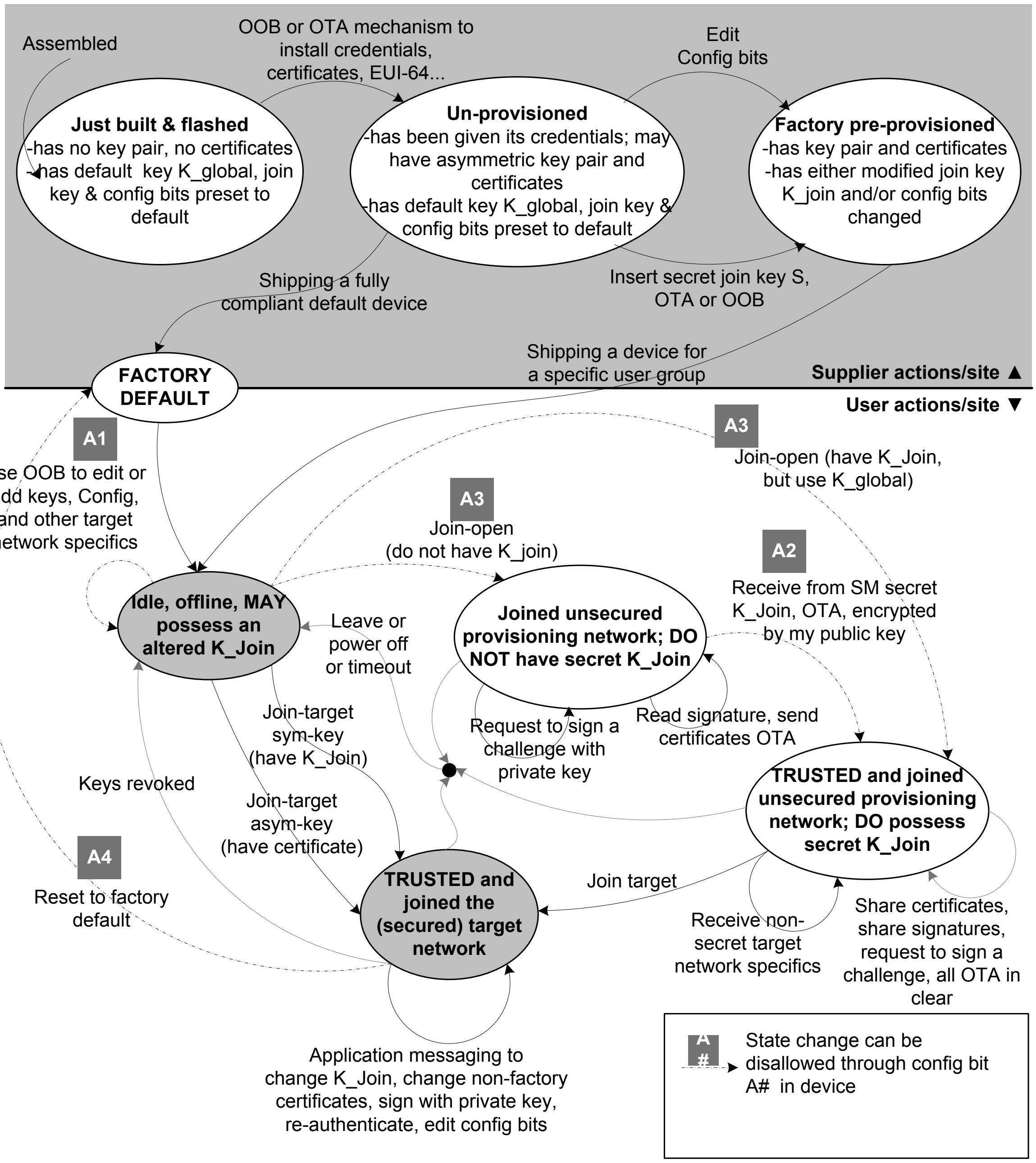


Figure 136 – State transition diagrams outlining provisioning steps during a device lifecycle

A field device that is newly manufactured transitions to the un-provisioned state when its identity (e.g., it EUI64Address) and credentials are provided to it. In this state, the device has the default settings as defined in Table 367.

Table 367 – Factory default settings

Attribute	Description	Default value
Default symmetric join key K_global	The symmetric join key used to join a default network	Specified in 7.2.2.2.
Open symmetric join key (K_join = K_open)	The symmetric join key used to join a provisioning network, then to receive new target join keys	Specified in 7.2.2.2
Allow OOB provisioning (A1)	This configuration bit allows the use of OOB mechanisms for provisioning the device. This bit is irrelevant if the device does not have any OOB means for provisioning	1 = allowed
Allow asymmetric-key-based provisioning (A2)	This configuration bit allows the use of asymmetric crypto for OTA provisioning of K_join. This bit is irrelevant if the device does not support asymmetric cryptography	1 = allowed
Allow default join (A3)	This configuration bit allows the device to join a default network. Some devices may choose not to allow a default join at all	1 = allowed
Allow reset to factory defaults (A4)	This configuration bit allows execution of OTA commands that reset the device to the factory default configuration	1 = allowed

The manufacturer may ship devices with these default settings.

Alternatively, the device may be pre-provisioned for a particular user at the manufacturing site. When a device is pre-provisioned, the default settings of the device are changed. The device may be given a new target symmetric join key specific to a target network at the user site. In addition, any of the configuration bits (A1, A2, A3 and A4) may be changed. For example, the default network join and reset to factory defaults may be disabled (A3, A4 = 0). Such a device shall not be able to be provisioned through the open symmetric join key (K_join = K_open).

The device arrives at the user site either pre-provisioned or with factory defaults and is in the idle state.

At a user site, a device may be provisioned, using OOB mechanisms (A1 enabled), with a symmetric join key and network-related information for joining the target network. Alternatively, the device may already have preinstalled secret join keys and/or network-related information established by the device manufacturer. If the network-related information is not provisioned into the device at manufacturing, the device may join a provisioning network using the default symmetric join key K_global, specified in 7.2.2.2 (if A3 is enabled), and may be provisioned with network-related information using over-the-air mechanisms.

Devices that fail to join the target network using their provisioned information may seek to join a provisioning network (if A3 is enabled) using K_global. After joining using the default join key (K_global), the PD may use the Write_Symmetric_Join_Key method to update the K_join only if it is sent encrypted with the asymmetric key of the DBP.

If the device in an idle state does not have a valid installed symmetric join key and is allowed to join a default network, and A4 is enabled, the device shall start scanning for advertisements in order to reach a security manager/system manager of a default network in its vicinity.

If an advertisement is found and the device has asymmetric cryptographic capabilities and PKI certificates, it shall forward its credentials to the security manager associated with the advertising router. The advertising routers shall forward join requests to their security managers using an established contract that the advertising router has with the security manager/system manager.

When the security manager receives new device credentials, it first checks whether devices with those credentials are expected and authorized for the target network. This may be accomplished via lookup in pre-populated white lists with the EUI64Address of the individual device. The device credentials are used by the system manager to decide on the CA (and its asymmetric key) to use in subsequent authentication steps.

If the device is authorized, then the authenticity of the credentials is checked by the system manager. The device credentials include the device certificate or multiple certificates. When using multiple certificates, the check on the device data may[11] consist of two asymmetric crypto steps, one using the CA’s public key that is already present inside the security manager (the PD) to read the first certificate (termed the issuer certificate) and hence the issuer’s public key, followed by the second certificate (termed the device certificate) and hence the device’s public key, using the issuer’s public key. Once the device’s public key is obtained, a challenge/response mechanism (see 7.4.6) is used by the PD to establish the authenticity of the DBP.

A copy of data exchanged in the preceding steps may be logged in public files in the PD for future audit purposes.

User input to accept the device may be solicited before the device is accepted. A dialog on a human-machine interface (HMI) connected to the system manager may seek confirmation that the trustworthy device should be allowed to join the target network. This can be a yes/no dialog that asks if a specific device, with a specific authenticated identity, that is a member of a family of expected and deemed welcome devices, should indeed now be prepared for a secure join to the secured target network. When this user-input step is implemented, and the user response is not received and no response is sent within the join response timeout period, the join request shall be considered to have failed.

If the device is authorized (present in the white list) and authentic, the PD generates a new key for the DBP, encrypts it using the DBP asymmetric key and transmits it to the DBP. A copy of that may be logged in public files in the PD for future audit purposes.

Failure in any of the steps above can be due to loss of connectivity, timeouts, or denial of join request from the DBP. Examples of the latter include a negative status on the white lists, a mismatch while authenticating, or a reject from a dialog on an HMI. When it is clear that a DBP should be rejected for any of those reasons, an alert is generated by the security manager. No join response shall be sent back to the device indicating a join failure to the device.

If the DBP does not have asymmetric cryptographic modules but has the open symmetric join key, it can join a provisioning network with the open symmetric join key (K_join = K_open). The right to accept or reject provisioning of DBPs that use the open symmetric join keys (K_join = K_open) rests with the PD. By default, the PD shall not provision devices that join with the open join key; however, the PD may be configured to provision such devices. If the PD is configured to allow open OTA provisioning, then the DBP will be provisioned with a new join key K_join for joining the target network. Once provisioned, the device shall not use the open key again unless it is reset to factory defaults (A4 is enabled).

Once provisioned, the device can proceed to join the target network with its provisioned information. As part of the join process, the device receives a master key, T-keys, and D-keys, in addition to establishing a contract with the system/security manager of the target network, and normal operation of the standard secured network follows.

As part of the normal operation of a network, the system manager of the network may provision the device with sufficient information to join another network when the device leaves the current network. This process enables a device to join and leave multiple networks. Provisioning for another network using a current target network is accomplished as follows.

---

[11] The two-certificate chain described here is only one of the many certificate topologies possible with multiple certificates. The DPO provides attributes to include multiple certificates.

a) The DPSO in the current system manager retrieves network information and security keys from the system/security manager of the other network.

NOTE The interface for such inter-manager communication is beyond this scope of this standard.

b) The DPSO in the current system manager installs information into the DPO of the device.

c) The DBP leaves the current network.

d) When the device leaves the current network, it joins the next network with network and security information installed in its DPO.

As described herein, there are multiple paths (and state transitions) available for an un-provisioned device to be provisioned and ultimately to join a secured network. These paths are illustrated via the state transition diagram in Figure 137. Figure 137 is related (and equivalent to) to Figure 136; however, Figure 137 is depicted from the perspective of a device internal state.

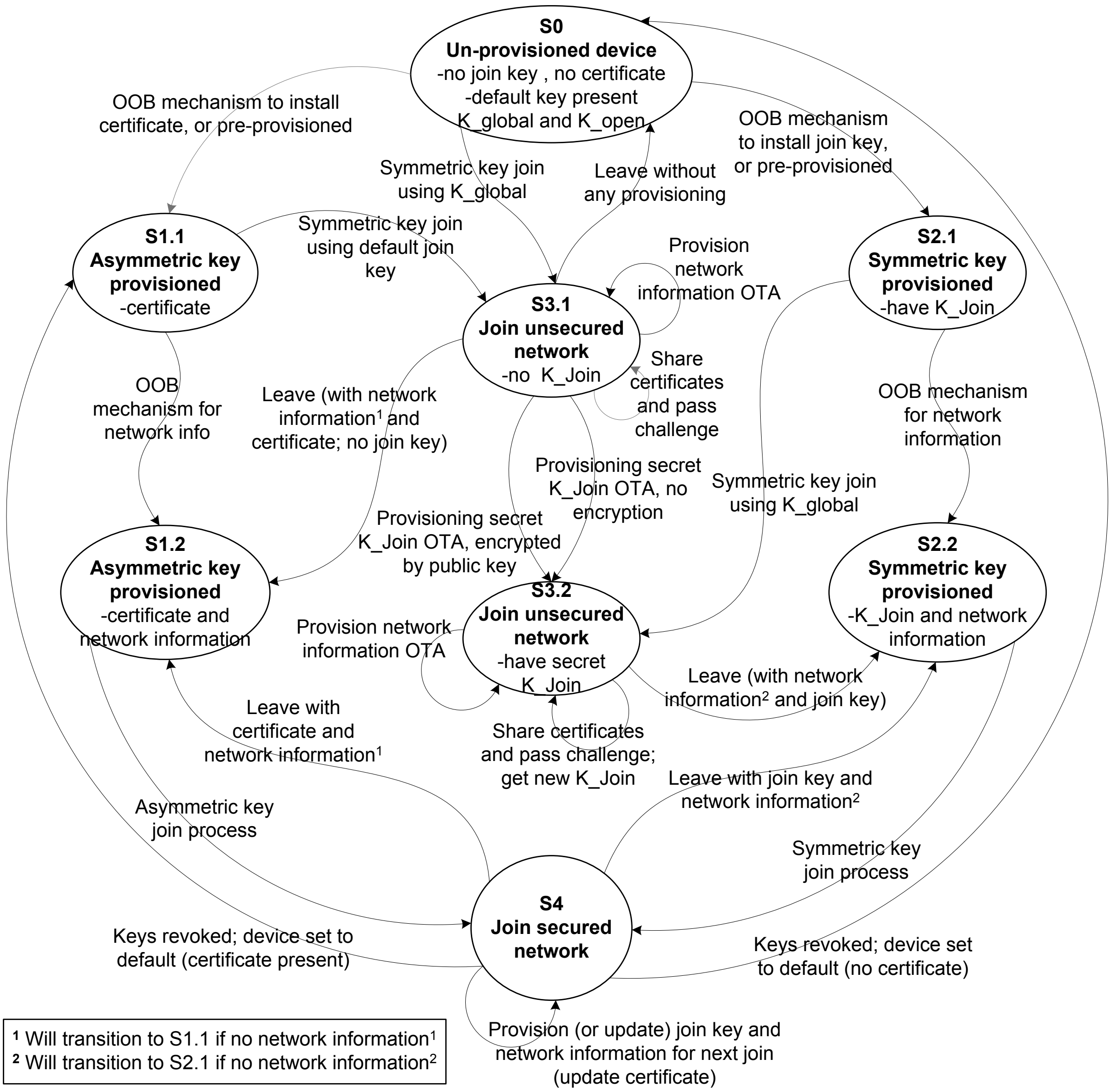


Figure 137 – State transition diagram showing various paths to joining a secured network

The transitions and paths addressed in Figure 137 include:

a) OOB provisioning of symmetric key and network information:
   1) State transitions : S0 → S2.1 → S2.2 → S4
   2) Synopsis: OOB mechanisms are used to provision a device with the target network join key (S0 –> S2.1) and network information (S2.1 → S2.2). Then, the device uses the symmetric join procedure (S2.2 → S4) to join the secured network.
b) Factory pre-provisioned (OOB or otherwise): Asymmetric keys and certificates and OOB provisioning of network information:
   1) State transitions : S0 → S1.1 → S1.2 → S4
   2) Synopsis: A device is factory pre-provisioned with asymmetric keys and certificates (S0 → S1.1). The device has the necessary information to initiate an asymmetric-key join procedure. However, it does not have enough network-related information. This information is provisioned using OOB mechanism (S1.1 → S1.2). Then, the device uses the asymmetric join procedure to join the secured network (S1.2 → S4).
c) OOB provisioning of symmetric-key information and OTA provisioning of network information:
   1) State transitions : S0 → S2.1 → S3.2 → S2.2 → S4
   2) Synopsis: A device is provisioned using OOB mechanism (or pre-provisioned) with the symmetric join key for the target network (S0 → S2.1). The device then joins a default provisioning network using the default join key, K_global (S2.1 → S3.2). The PD in the provisioning network provides the network information for the target network. The device leaves the provisioning network (S3.2 → S2.2) and joins the secured network (S2.2 → S4) using the symmetric join procedure.
d) Factory pre-provisioned (OOB or otherwise) asymmetric keys and certificates and OTA provisioning of symmetric keys:
   1) State transitions: S0 → S1.1 → S3.1 → S3.2 → S2.2 → S4
   2) Synopsis: A device is factory pre-provisioned with asymmetric keys and certificates (S0 → S1.1). The device has the necessary information to initiate an asymmetric-key join procedure. However, it cannot join a target network that does not support an asymmetric join process. The device then joins a default provisioning network that is different from the target network using the default join key, K_global (S1.1 → S3.1). As part of this provisioning network, the device exchanges its credentials, passes a challenge-response mechanism, and receives the target network join key, encrypted with the device's public key, from the PD (S3.1 → S3.2). The device is then provisioned with the network information OTA. Then, the device leaves the provisioning network (S3.2 → S2.2) and joins the secured network (S2.2 → S4) using the symmetric join procedure.
e) Open join key-based provisioning in the clear:
   1) State Transitions : S0 → S2.1 → S2.2 → S4(1) → S2.2 → S4(2)
   2) Synopsis: A device that has the default open symmetric join key. It uses the symmetric join key procedure for joining a provisioning network (S2.2. → S4(1)). As part of this provisioning network, the device is provisioned with the target network join key and network information. The device then leaves the provisioning network (S4(1) → S2.2). The device is now provisioned to join the target network; it joins the secured target network using the symmetric-key join process (S2.2 → S4(2)). In this transition, the first time the device has joined a provisioning network is indicated by state S4(1), and the second time it is joined to the target network is indicated by state S4(2). After the device has reached S4(2), the device cannot use the open symmetric join key unless it is reset to factory defaults.

13.8 Device management application protocol objects used during provisioning

This standard uses one DMAP object and one SMAP object during provisioning. The device provisioning object (DPO) holds the configuration settings. Figure 138 illustrates provisioning objects and the interactions between them.

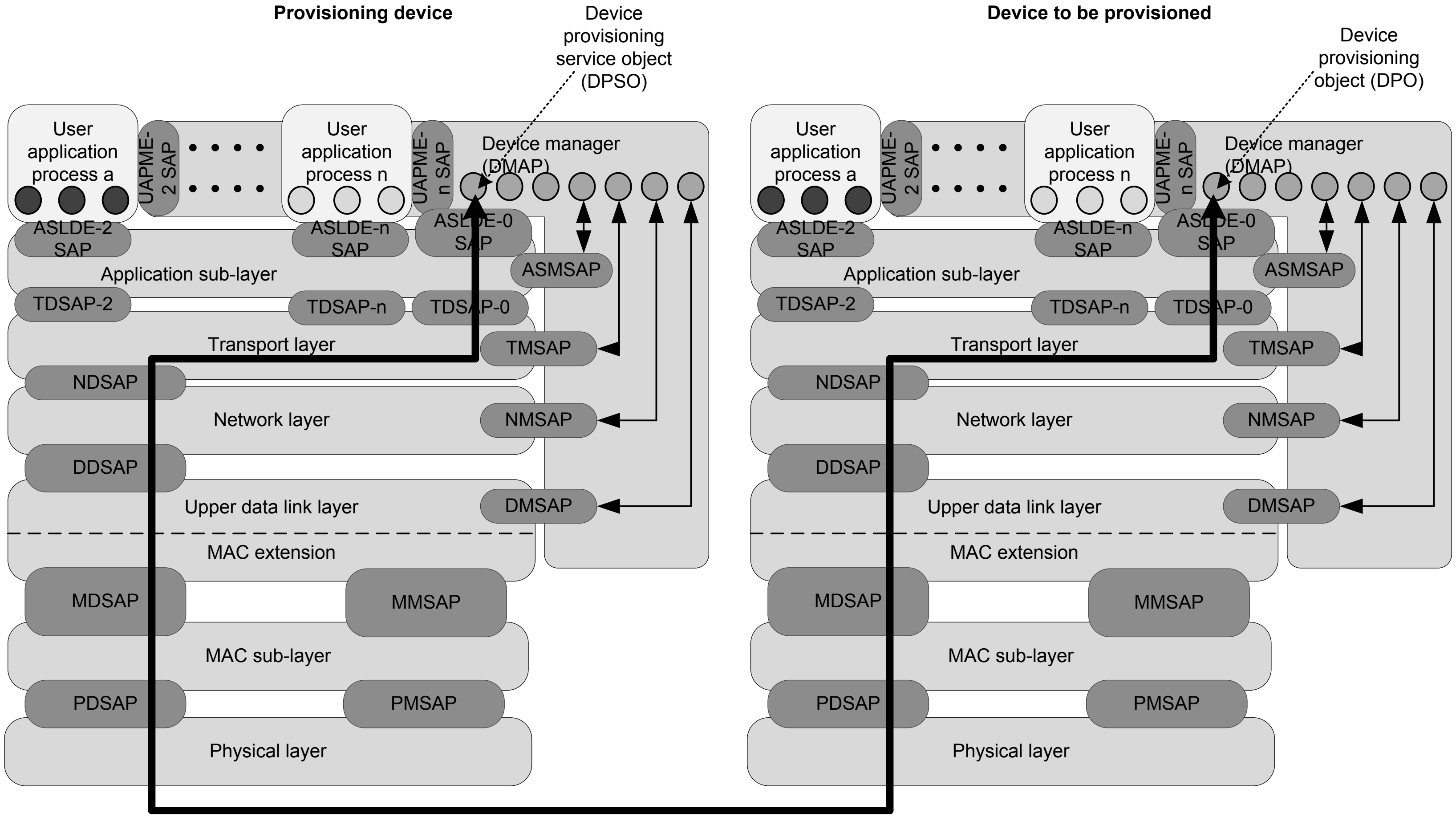


Figure 138 – Provisioning objects and interactions

Whether it is the system manager/security manager in a handheld device or the system manager of the target network, the PD shall implement a device provisioning service object (DPSO) with attributes and methods to provision the DBP. The DPSO may have a list of symmetric keys, used to provision devices that do not have pre-installed keys.

The white list, symmetric-key information, and target network information in the DPSO can be maintained with information specific to a device in the White_List_Array attribute in Table 372. Alternatively, a pool of valid symmetric keys can be maintained.

When the DBP joins the provisioning network using K_global, a contract is established between the PD and DBP. The DPSO in the PD can use the established contract to communicate with the DPO in the DBP. Read and write primitives are used for accessing and setting the attributes of the DPO. A subset of network and trust information can now be provisioned in the DPO using the DPSO. To write the new symmetric join key to the device the DPSO invokes the Write_Join_key method of the DPO. This method is allowed if the new key value was received under protection of asymmetric crypto. The attributes in the DPO include both network-related and trust-related information.

Users that want additional security while provisioning should use the asymmetric crypto-based authentication and secured key loading technique for the trust-related steps. Alternatively, out-of-band mechanisms may be used for provisioning join keys

Once the appropriate trust and network information has been provisioned in the DPO, the device is ready to join the target network. The provisioning network can also be used for device configuration. Since a contract has already been established, the PD may also use the network (or OOB means) to configure the appropriate UAP and DMAP objects of the DBP.

The device provisioning object (DPO) provides an attribute called the Target_DL_Config in the DL_Config_Info format. DL_Config_Info is described in Clause 9 (see Table 102) to configure various attributes of the DL. Once provisioned with this attribute, the DPO provides the DL with an OctetString encapsulating DL_Config_Info that includes at least one superframe, and at least one link, that can be used by the DL in searching for advertisements. Target network-specific (e.g., non-default) timeslot templates, channel-hopping patterns, superframes and links can also be provided to the DBP through the Target_DL_Config attribute. Such configuration helps reduce the amount of information (e.g., join superframes) that is otherwise required to be advertised by target network advertisement routers.

The DL of the device plays a major role during the provisioning and joining of the device. The state machine of the DL when it is going through the provisioning process is described in 9.1.14.2.

If the provisioning process is successful, the DPO provides the DL with the set of attributes, including D-subnet information, superframes, and links, that the DL can use to search for the target network and corresponding D-subnet(s). In the provisioned state the DL operates its state machine as configured in the superframes and links that were provided by the DPO. Superframe operation may be delayed or disabled by setting the IdleTimer field within the superframe.

Since the device retains the information that was used to provision the DL (all attributes of the DPO), this ensures a means to reset the DL back to its provisioned state by putting the DL into its default state and then adding the provisioned attributes.

The DPO shall be accessible to the system manager of the target network after joining with Key_Join. Once the device joins a target network, the system manager of the target network has the ability to change the attributes of the DPO. The system manager of the target network has the ability to instruct the device to join another target network by providing network and trust information of the other network. Depending on the value of configuration bit A4, the system manager of the target network has the ability to invoke a DPO.Reset_To_Defaults method to remove trust information from the device.

NOTE 2 In the provisioning phase, the DPO in the DBP is accessed by the system/security manager functionality in the PD.

13.9 Management objects

13.9.1 Device provisioning object

Table 368 describes the attributes of the DPO. The data type, default value, and a brief description are provided for each attribute. Each attribute also has accessibility of read only or read/write. The attributes of the DPO are accessible only to the system/security manager. The value of a read-only attribute can be set only at the device manufacturing time (i.e., at a time before the device is certified) or internally by the device; no entity external to the device can change this attribute. Read/write accessibility implies that entities external to the device can change the value of the attribute. The attributes of the DPSO are accessible (read/write) only to the system manager.

The attributes classified as "constant" have a value that is not changed during the device lifecycle, neither internally nor externally. The definition of the classification is found in 12.6.3.

Table 368 – Device provisioning object

Standard object type name: Device provisioning object (DPO)				
Standard object type identifier: 120				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Default_NWK_ID	1	A published network identification for the default network	Type: Unsigned16 Classification: Constant Accessibility: Read only Default value: 0x0001	This is the network identification for the default network. The default network may be used to form the provisioning mini-network
Default_SYM_Join_Key	2	A published join key for the default network	Type: SymmetricKey Classification: Constant Accessibility: Read only Default value: K_global, 7.2.2.2	This key is used by devices to join the default network. The default keys may be used to form the provisioning mini-network
Open_SYM_Join_Key	3	A published join key for the default network	Type: SymmetricKey Classification: Constant Accessibility: Read only Default value: K_open, 7.2.2.2	This key is used by devices to join the unsecured provisioning network

Table 368 *(continued)*

Standard object type name: Device provisioning object (DPO)				
Standard object type identifier: 120				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Default_Channel_List	4	The list of 2,4 GHz channels used by the default network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: Unsigned16 Classification: Constant Accessibility: Read only Default value: 0x7FFF	The list of channels used by the advertising routers of the default network. To join the default network the device may receive advertisements on any of these frequencies
Join_Method_Capability	5	The join capabilities of the device.	Type: Unsigned2 Classification: Constant Accessibility: Read/write Default value: 00 Named values: 00: default join only; 01: symmetric-key join only; 10: asymmetric-key join only; 11: any key join	This attribute defines the capability of a device to join. The device can either use symmetric keys or asymmetric-key infrastructure to join a target network. This attribute merely defines the capabilities of the device. The actual method used to join the target network shall be set by the PD
Allow_Provisioning	6	A Boolean value set to indicate if a device is allowed to be provisioned or not	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	This flag is used to lock the state of an already provisioned device. If this value is set the device will not accept any reads or writes to the target network attributes
Allow_Over_The_Air_ Provisioning	7	A Boolean value set to indicate if a device is allowed to be provisioned or not	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	This Boolean indicates whether over-the-air provisioning is enabled or disabled. If over-the-air provisioning is disabled the device needs to be provisioned using out of band methods. Backbone devices shall have this value set to FALSE. In all cases, provisioning is allowed only if the Allow_Provisioning attribute is enabled

Table 368 *(continued)*

Standard object type name: Device provisioning object (DPO)				
Standard object type identifier: 120				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Allow_OOB_Provisioning	8	A Boolean value set to indicate if a device is allowed to be provisioned using OOB means	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	The Boolean is used to block the devices from accepting provisioning information from OOB means
Allow_Reset_to_Factory_ Defaults	9	A Boolean value set to indicate if a device is allowed to be reset to factory defaults	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	This Boolean is used to block devices from being reset to factory defaults by a system manager
Allow_Default_Join	10	A Boolean value set to indicate if a device is allowed to join a network using the default keys	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	The Boolean is used to force the devices to join a particular target network and not join to any default network. Devices choosing not to join a Default network can set this attribute to FALSE
Target_NWK_ID	11	The network ID of the target network that this device is provisioned to join	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 0	This attribute indicates the target network that this device has to join [a)]
Target_NWK_BitMask	12	A bit mask for matching of the bits of the Target network ID	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 0xFFFF	The bit mask is useful for matching multiple target networks. If the value of a bit in the bit mask is 1 then the bit has to be exactly matched to the corresponding bit in the Target Network. The default value of all 1s indicates that all bits of network ID need to match
Target_Join_Method	13	Indicate whether the device should use symmetric-key join or asymmetric-key join mechanism to join the target network.	Type: Unsigned1 Classification: Static Accessibility: Read/write Default value: 1 Named values: 0: Symmetric key 1: Asymmetric key	7.4.4 defines two different methods for join depending on the use of either symmetric keys or asymmetric-key certificates. This attribute sets method to be used to join a target network

Table 368 *(continued)*

Standard object type name: Device provisioning object (DPO)				
Standard object type identifier: 120				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Target_Security_Manager_EUI	14	The EUI64Address of the security manager in the target network that the device is intended to join	Type: EUI64Address Classification: Static Accessibility: Read/write Default value: 0xFF…FF (all 0xFF)	Set to the EUI64Address of the security manager that the device is provisioned to join
Target_System_Manager_Address	15	The IPv6Address of the system/security manager in the target network that the device is intended to join	Type: IPv6Address Classification: Static Accessibility: Read/write	The IPv6Address is required for backbone devices to join the network. The backbone devices do not have an advertising router - hence a join request is sent to the IPv6Address of the system/security manager to begin the join process. I/O devices and routing devices need not be provisioned with this attribute
Target_Channel_List	16	The list of channels used by the target network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: BitArray16 Classification: Static Accessibility: Read/write	The target network may be using only a subset of channels for advertisements by the join routers. By using only a subset of frequencies battery powered devices can quickly join the target network by listening in that subset of frequencies only
Target_DL_Config	17	The DL configuration information for this device	Type: OctetString Classification: Static Accessibility: Read / Write	This attribute indicates the various configuration settings for the DL of the device. The structure of this attribute is defined in DL_Config_Info defined in Clause 9
PKI_Certificate_Type	18	(Asymmetric-crypto option) The type of certificate stores in PKI_Root_Certificate and PKI_Certificates	Type: Unsigned8 Classification: Static Accessibility: Read/Write Default value: 0 Named values: 0: implicit cert; 1: manual cert	This field indicates a type of Certificate in PKI_Root_Certificate and PKI_Certificates.

Table 368 *(continued)*

Standard object type name: Device provisioning object (DPO)				
Standard object type identifier: 120				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
PKI_Root_Certificate	19	(Asymmetric-crypto option) The root certificate of the certificate authority issuing the certificate to the device.	Type: OctetString Classification: Static Accessibility: Read/write	The root certificate of the certificate authority and its corresponding asymmetric key is used to verify certificates of the peer nodes. The root certificate may updated by the system manager
Number of PKI_Certificates	20	(Asymmetric-crypto option) The number of certificates stored in the PKI_certificat e attribute	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	This field indicates the number of certificates available in attribute PKI_Certificate
PKI_Certificate	21	(Asymmetric-crypto option) The certificate issued to this device for joining using the asymmetric-key infrastructure	Type: Array of OctetString Classification: Static Accessibility: Read/write	If Target_Join_ Method is set to Asymmetric-key, this attribute contains the certificate (which includes the asymmetric key, device ID, and other text) signed by a certificate authority which is required for joining the target network
Current_UTC_Adjustment	22	The current value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Static Accessibility: Read/write Default value: 35	See Table 25 attribute 1 and footnote
[a] If the Target_NWK_BitMask (attribute 12) is set to 0xFFFF, the device shall ignore advertisements from routers belonging to any other network except the indicated target network. Otherwise a combination of network ID and bit mask shall be used. (See description of attribute 12 on how the NetworkID and bitmask are combined). This helps with fast joins and also prevents devices from trying to join all networks in their vicinity. This value can be set to 0 to allow responses to any advertising router				

13.9.2 Device provisioning object methods and alerts

Several methods and alerts are available in the DPO. Table 369 describes the Reset_To_Default method.

Table 369 – Reset_To_Default method

Standard object type name(s): Device provisioning object (DPO)				
Standard object type identifier: 120				
Method name	**Method ID**	**Method description :**		
Reset_To_Default	1	This method is used to reset to default settings for the provisioning. This method shall be executed only when Allow_Provisioning is enabled.		
	Input arguments (None)			
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Status	Unsigned8	Named values: 0: success; other: failure

Table 370 describes the method to write a symmetric join key. The join key shall not be exposed to a remote device, and may be exposed to limited internal process; DPO.Write_Symmetric_Join_Key() method installs a join key to a memory area that is not used for attributes (e.g., secure storage).

Table 370 – Write symmetric join key method

Standard object type name(s): Device provisioning object				
Standard object type identifier: 120				
Method name	**Method ID**	**Method description :**		
Write_SYM_join_key	2	This method is used to write a symmetric join key to a device. This method is evoked by the DPSO to provision a DBP with the target join key. Depending on the provisioning method used this method call APDU and hence the join key may be encrypted by the T-key between the device and PD alone or the device’s asymmetric key in the APDU in addition to the APDU being encrypted by the T-key. This method shall be executed only when Allow_Provisioning is enabled.		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	New_Key_Value	SymmetricKey	New join key to be installed.
	2	Encrypted By	Unsigned8	Named values: 0: TL_Session_Key_Only, 1: Asymmetric_Key
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Status	Unsigned8	Named values: 0: success; >0: failure

13.10 Device provisioning service object

13.10.1 Device provisioning service object attributes

Table 371 describes the attributes of the DPSO.

The system manager can either choose particular provisioning information for each EUI64Address or a set of join keys for a set of EUI64Addresses with no one-to-one mapping. The Boolean1 attribute, DPSO.Enable_White_List_Array, is used to specify which method is used.

In the DPSO.Enable_White_List_Array set, DPSO.White_List_Array is used to install particular provisioning information for each EUI64Address of the DBP.

If DPSO.Enable_White_List_Array is not set, the PD shall check if there are at least as many entries in DPSO.SYM_Key_List as entries in DPSO.White_List. This standard does not specify how each entry in DPSO.SYM_Key_List and DPSO.White_List is mapped.

Table 371 – Device provisioning service object

Standard object type name: Device provisioning service object (DPSO)				
Standard object type identifier: 106				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
White_List	1	A list of devices permitted to be provisioned by this object	Type: Array of EUI64Address Classification: Static Accessibility: Read/write Default value: [ ] -- empty	This list contains EUI64Addresses of device being provisioned. This list can be used to restrict a provisioning device to the specific set of devices whose EUI64Addresses are in this list. If this list is empty, then the provisioning device can provision any device
Symmetric_Key_List	2	A list of valid join keys with which a device can be provisioned.	Type: Array of SymmetricKey Classification: Static Accessibility: Read/write Default value: {K_global] -- 7.2.2.2	This key is used by devices to join the target network, that have suitable entropy
Symmetric_Key_Expiry_Times	3	The expiration time for each key	Type: Array of TAITimeRounded Classification: Static Accessibility: Read/write Default value: [0xFFFF FFFF] (Note 1)	This attribute sets the expiry time for each of the symmetric keys. The key is only used for provisioning if it has not expired

Standard object type name: Device provisioning service object (DPSO)				
Standard object type identifier: 106				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Target_NWK_ID	4	The network ID of the target network that the devices provisioned by this object are supposed to join	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: 0	This attribute indicates the target network (subnet ID) that a provisioned device has to join

Table 371 *(continued)*

Standard object type name: Device provisioning service object (DPSO)				
Standard object type identifier: 106				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
Target_Join_Method	5	A Boolean value to indicate if the devices provisioned by this object should use symmetric-key join or asymmetric-key join mechanism to join the target network	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: 0	Clause 7 defines two different methods for join depending on the use of either symmetric or asymmetric keys. This attribute sets the method to be used to join a target network. Named values: 0: symmetric key; 1: asymmetric key
Target_Security_Manager_EUI	6	The EUI64Address of the Security Manager in the target network that the device provisioned by this object is intended to join	Type: EUI64Address Classification: Static Accessibility: Read/write	Set to the EUI64Address of the security manager that the device is provisioned to join
Target_System_Manager_Address	7	The IPv6Address of the system/ security manager in the target network that the device provisioned by this object is intended to join	Type: IPv6Address Classification: Static Accessibility: Read/write Valid range: all with highest bit reset	The IPv6Address is required for backbone devices to join the network
Target_Channel_List	8	The list of channels used by the default network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: BitArray16 Classification: Static Accessibility: Read/write	The target network may be using only a subset of channels for advertisements by the join routers
Target_DL_Config	9	The DL configuration information for	Type: OctetString Classification: Static	This attribute indicates the various configuration settings

Table 371 *(continued)*

Standard object type name: Device provisioning service object (DPSO)				
Standard object type identifier: 106				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
		the device to be provisioned by this object	Accessibility: Read / Write	for the DL of the device. The structure of this attribute is defined in the DL_Config_Info defined in Clause 9, Table 102
Allow_Provisioning	10	A Boolean value set to indicate if a device is allowed to be provisioned again or not	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: TRUE	This flag is used to lock the future state of a provisioned device
Allow_Default_Join	11	A Boolean value set to indicate if a device provisioned by this object is allowed to join a network using the default keys	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: Not_allowed (0)	The flag is used to force the provisioned devices to join a particular target network and not join to a default network. Once provisioned the device should join the target network
Enable_White_List_Array	12	A Boolean value set to indicate if the provisioning object is designed to set device specific provisioning information	Type: Boolean1 Classification: Static Accessibility: Read/write Default value: FALSE	If this flag is set the DPSO is capable of provisioning different devices (based on the EUI64Address) with different provisioning information. This can be used to provision a particular device, with a particular security manager and a particular Target_Join_Time, for example
White_List_Array	13	An array of the EUI addresses that the DPSO intends to provision along with the corresponding provisioning information for that device	Type: Array of DPSOWhiteListTbl Classification: Static Accessibility: Read/write	This attribute shall be used only if Device_specific_provisioning_flag is set. It contains the device specific provisioning information like device specific join keys, device specific target security manager etc.
White_List_Array_Meta	14	Metadata for White List Array (Attribute 13) or set of White_List (Attribute 1), SYM_Key_List (Attribute 2)	Type: Metadata_attribute Classification: Static Accessibility: Read only	Metadata containing a count of the number of entries and capacity (the total number of rows allowed) of White_List_Array table or set of White_List. (Note 2)
DPSO_Alerts_AlertDescriptor	15	Used to change the priority of DPSO alerts that belong to the security	Type : Alert report descriptor Classification: Static Accessibility: Read/write	See description of alerts in Table 374 and Table 375

Table 371 *(continued)*

Standard object type name: Device provisioning service object (DPSO)				
Standard object type identifier: 106				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of behavior of attribute**
		category; these events can also be turned on or turned off	Default value: [FALSE, 6]	
Current_UTC_Adjustment	16	The current value of the UTC accumulated leap second adjustment	Type: Integer16	See Table 25 attribute 1 and footnote
			Classification: Static	
			Accessibility: Read/write	
			Default value: 35	
NOTE 1 When a computed key expiry time results in the value 0xFFFF FFFF, it is increased (circularly) to the modulo value 0x0000 0000, so that the value 0xFFFF FFFF can be used to designate a key that never expires. NOTE 2 If Enable_White_List_Array is enabled, this attribute specifies a count of the number of entries and capacity for White_List_Array. If Enable_White_List_Array is disabled, this attribute specifies a count of the number of entries and capacity for the set of White_List and SYM_Key_List				

13.10.2 Device provisioning service object structured attributes

Table 372 describes the structured attributes of the DPSO. The White_List_Array is used when the PD has device-specific information, i.e., if the PD has symmetric join keys and other DPO attributes that are specific to a device. In this case, a structured array is required that stores the provisioning information indexed by the EUI64Address identifier of the DBP. This indexed array is described in Table 372.

After the PD receives the Device_SYM_Key from the security manager, the PD shall not expose the Device_SYM_Key attribute externally.

NOTE The interface between the PD and the security manager is beyond the scope of this standard.

Table 372 – DPSOWhiteListTbl data structure

Standard data type name: DPSOWhiteListTbl		
Standard data type code: 440		
Element name	**Element identifier**	**Element scalar type**
Device_EUI	1	Type: Array of EUI64Address
		Classification: Static
		Accessibility: Read only
		Default value: [ ] -- empty
Device_Tag	2	Type: Array of VisibleString
		Classification: Static
		Accessibility: Read/write
		Default value: [“”]
Symmetric_Key_List	3	Type: Array of SymmetricKey
		Classification: Static
		Accessibility: Read only
		Default value: {K_global] -- 7.2.2.2

Standard data type name: DPSOWhiteListTbl		
Standard data type code: 440		
Element name	**Element identifier**	**Element scalar type**
Symmetric_Key_Expiry_Times	4	Type: Array of TAIRounded
		Classification: Static
		Accessibility: Read only
		Default value: [0xFFFF FFFF] (Note 1)
Target_NWK_ID	5	Type: Unsigned16
		Classification: Static
		Accessibility: Read only
		Default value: 0
Target_Join_Method	6	Type: Unsigned8
		Classification: Static
		Accessibility: Read only
		Default value: 1
		Named values: 0: symmetric key; 1: asymmetric key
Target_Security_Manager_EUI	7	Type: EUI64Address
		Classification: Static
		Accessibility: Read only
Target_System_Manager_Address	8	Type: IPv6Address
		Classification: Static
		Accessibility: Read only
		Valid range: all with highest-bit reset
Target_Channel_List	9	Type: Array of Unsigned8
		Classification: Static
		Accessibility: Read only
Target_DL_Config	10	Type: OctetString (See DL_Config_Info for format)
		Classification: Static
		Accessibility: Read only
Allow_Provisioning	11	Type: Boolean1
		Classification: Static
		Accessibility: Read/write
		Default value: TRUE
Allow_Default_Join	12	Type: Boolean1
		Classification: Static
		Accessibility: Read/write
		Default value: TRUE
NOTE 1 When a computed key expiry time results in the value 0xFFFF FFFF, it is increased (circularly) to the modulo value 0x0000 0000, so that the value 0xFFFF FFFF can be used to designate a key that never expires.		

When not null, the Device_Tag specifies a Tag_Name assigned to the device by a user. This value shall be written to the Tag_Name attribute of the DMO (see 6.2.8).

13.10.3 Device provisioning service object methods

Several methods are available for manipulating the DPSO. Standard methods such as read and write can be used for scalar or structured MIBs (SMIBs) in their entirety. The methods described herein are used to manipulate tables. These methods allow access to a particular row of a SMIB based on a unique key field.

It is assumed that the tables have a unique key field, which may either be a single element or the concatenation of multiple elements. The key field is assumed to be the (concatenation of) the first (few) element(s) of the table.

Table 373 describes the methods for manipulation of structured MIBs. These methods are based on the Read_Row, Write_Row and Delete_Row templates defined in Annex J.

Table 373 – Array manipulation table

Standard object type name(s): Device provisioning service object (DPSO)		
Standard object type identifier: 106		
Method name	**Method ID**	**Method description**
Setrow_WhiteListTbl	1	Method to set (either write or edit) the value of a single row of the white list array. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)
Getrow_WhiteListTbl	2	Method to get the value of a single row of the white list array. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)
Deleterow_WhiteListTbl	3	Method to delete the value of a single row of the white list array. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)

13.10.4 Device provisioning service object alerts

Table 374 describes an alert to indicate a join attempt by a device that is not on the white list.

Table 374 – DPSO alert to indicate join by a device not on the WhiteList

Standard object type name(s): Device provisioning service object (DPSO)					
Standard object type identifier: 106					
Description of the alert: Alert to indicate provisioning request by a device not on white list					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	2 = Security	0 = Not_On_ Whitelist_Alert	6 = Medium	Type: EUI64Address	EUI64Address of a device not on the white list

Table 375 describes an alert to indicate that inadequate capability is available for a device to join the network.

Table 375 – DPSO alert to indicate inadequate device join capability

Standard object type name(s): Device provisioning service object (DPSO)					
Standard object type identifier: 106					
Description of the alert: Alert to indicate inadequate device join capability					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority (Enumerated: high, med, low, journal only)**	**Value data type**	**Description of value included with alert**
0 = Event	2 = Security	1 = Inadequate_Join_ Capability_Alert	6 = Medium	Type: struct { reason: Unsigned8; rejectedDevice: EUI64Address}	The reason field provides a diagnostic code. Named values: 1: Bad join key; 2: Expired join key; 3: Authentication failed. The rejectedDevice field specifies the EUI64Address of the device that attempted to join

13.10.5 Summary of attributes that can be provisioned

The following is a summary of the attributes that are provisioned by the PD so that a new device can join a target network. These attributes can be provisioned either using over-the-air (OTA) methods or OOB methods. The list of provisioned attributes includes the follow items. The full list is defined by Table 368, Table 371, and Table 372.

- Trust-related information :
  - symmetric join key (K_join);
  - EUI64Address of the security manager;
  - network join method.

- Network-related information:
  - network ID and bitmask;
  - IPv6Address of the system manager;
  - DL configuration (contains the superframes, link TsTemplate, channel information, etc., needed to join the target network).

In addition, the configuration bits (attributes 6..10 of the DPO), describing the behavior of the device, can be set by the provisioning device.

13.11 Provisioning functions [INFORMATIVE]

13.11.1 General

The provisioning interface and procedures described herein do not describe a human-machine interface (HMI). This standard does not specify a specific HMI, but does describe how such tools can be designed for provisioning devices conforming to this standard. In plant operations, a user may enter provisioning data and accept or reject devices wishing to be provisioned, using a handheld device or an interface at some central location.

Provisioning scenarios in Clause 13 are examples of provisioning using methods described by this standard.

13.11.2 Examples of provisioning methods

13.11.2.1 General

Examples are discussed herein of how the described management objects and procedures can be used to provision a device. These examples use the following provisioning methods:

- provisioning over-the-air using pre-installed join keys;
- provisioning using out of band mechanisms;
- provisioning over-the-air using asymmetric-key (e.g., PKI) certificates;
- provisioning over-the-air using dual role advertisement routers; and
- provisioning backbone devices.

13.11.2.2 Provisioning over-the-air using pre-installed join keys

The steps for provisioning a device with pre-installed trust information may include:

a) The device arrives at the deployment site with pre-installed keys. The keys are symmetric join keys.
b) A WhiteList of device addresses and their corresponding symmetric keys is installed in the security manager of the target network. The mechanism by which these keys are installed in the security manager is beyond the scope of this standard. The WhiteList and corresponding symmetric keys may be securely emailed, sent on CDs, hand delivered and keyboard entered, or delivered using any other appropriate tool.
c) The target network may be using a subset of frequencies allowed and may be operating in the vicinity (i.e., in the interference range) of multiple distinct networks conforming to this standard. In this case, network-related information may be provisioned in the device. This information allows the device to respond to advertisements from target networks only, and also to listen for advertisements at the correct frequencies and at the correct time, thereby decreasing interference and decreasing join times. The network-related information may be provisioned using a PD via an out-of-band communication mechanism or via over-the-air mechanisms. When not provisioned with specific target network information, the device may try all channels and attempt to join all networks in its vicinity.

d) The device listens for advertisements from an advertising router in the target network. Once an advertisement is heard, the device sends a join request to the system/security manager of the target network. (The join process is described in 7.4.)

e) The system/security manager checks its WhiteList and then checks to see if the join key of the device matches the join key for the device provided to the security manager. If the join keys match, the security manager provides a master key to the device that is a shared secret key between the security manager and the device. In addition, T-keys and D-keys are provided, and a contract is established with the new device to complete the join process.

13.11.2.3 Provisioning using out-of-band mechanisms

The steps for provisioning a device using an OOB provisioning device might include:

a) A fresh device arrives at the user site. The device has default settings and no-pre-installed keys.

b) A PD (e.g., a handheld) obtains a list of symmetric keys generated by the security manager/system manager of the target network. The symmetric keys are time-bounded. The keys may be an array of keys or device-specific key/EUI64Address pairs at the discretion of the security manager/system manager. This information is stored in the PD.

c) The handheld device, loaded with the symmetric keys, is brought near the new device or connected to the new device and an OOB connection is made between the handheld device and the DBP. The handheld device then uses the OOB communication interface to populate the attributes of the DPO in the new device. OOB communication may occur over infrared, physical connection, near field communication, or other means.

d) The device is now ready to join the target network and listens for advertisements from the target network. The device responds to the advertisements by sending a join request through the advertising router to the target system manager. The security manager checks its WhiteList, if applicable. The security manager also checks the validity of the join key and verifies that the key has not expired. If the key is valid, the security manager accepts the join request and provides the device with a master key that is a shared secret key between the security manager and the device. In addition, T-keys and D-keys are also provided and a contract is established with the new device to complete the join process.

13.11.2.4 Provisioning over-the-air using asymmetric key infrastructure certificates

The steps for provisioning a device that has pre-installed trust information might include:

a) The device arrives at the deployment site with an installed security module. The module contains a factory-signed certificate and a public/private key pair. A certificate authority (CA) has signed the issuer key of the factory.

b) A WhiteList of device addresses and a list of asymmetric keys of certificate authorities are installed in the security manager of the target network. The mechanism by which these keys are installed in the security manager is beyond the scope of this standard. The WhiteList may be securely emailed, sent on CDs, hand delivered and entered via keyboard, or delivered using any other appropriate tool.

c) The target network may be using a subset of the allowed channels, because it may be operating in the vicinity (i.e., within the interference range) of multiple networks. In such a case, network-related information may be provisioned to the device, enabling the device to respond to advertisements only from target networks and to listen for advertisements on the intended channels at appropriate times, thereby decreasing interference and increasing join rates. If the device is not provisioned with specific target network information, the device may try all channels and try to join all networks in its vicinity. The network-related information is provisioned using a provisioning device via an OOB communication mechanism or via over-the-air mechanisms.

d) The device now listens for the advertisements from the advertising router in the target network. Once an advertisement is heard, the device shares its certificates with the security manager. With the CA's public certificate, the security manager decodes the device certificate and checks that it is valid. The procedure also involves a challenge-response mechanism on part of the system/security manager to confirm the identity of the joining device. The security manager checks its WhiteList to confirm that the device is intended to join the network. The confirmation step may involve a pop-up on a GUI of the security manager for manual confirmation by a user. Once confirmed, the security manager may issue T-keys for the device if the device wishes to join the network immediately. Alternatively, the security manager may issue symmetric join keys for the device to join the network at a later time. In either case, the issued keys are sent back to the device, encrypted with the public key of the device.

13.11.2.5 Provisioning over-the-air using dual role advertisement routers

The steps for provisioning a device over-the-air using a dual role advertisement router might include:

a) The device arrives with factory default settings at a user site. The user site requires very low levels of security.
b) Some of the advertisement routers at the user site have a dual role and function as provisioning devices. Using the open symmetric join key (K_join = K_open), the dual role advertisement router (i.e., the logical PD side of the dual role advertisement router) forms a mini-network with the new device and provides the new device with the network settings and join key for the target network. These settings, including the keys, are sent in the clear over-the-air. The dual role device may also inform the security manager of the target network to update its white list by adding the device that has just been provisioned. The dual role provisioning device may be operational in a place where the user is fairly confident that transmission of join keys over-the-air poses little risk. (This step poses similar risk as that in binding garage door openers to remote controls.)
c) The DPO of the new device now has the trust information and network information to join the target network. It can use the advertisement routers (either the same dual role advertisement router that provisioned it, or some other advertisement router of the target network if the device was moved) and sends a join request to the system manager of the target network.
d) The system manager of the target network accepts the join request and provides a contract to the new device.

13.11.2.6 Provisioning backbone devices

The steps for provisioning backbone devices might include:

a) A fresh device arrives at the user site. The device has default settings and no pre-installed keys.
b) A PD (e.g., a handheld or a device connected via the backbone interface to the DBP) obtains a list of symmetric keys generated by the security manager/ system manager of the target network. The symmetric keys are time-bounded. The keys may be an array of keys or device-specific key/EUI64Address pairs at the discretion of the security manager/system manager. This information is stored in the DPSO of the PD.
c) The PD loaded with the symmetric keys is brought near the new device or connected to the new device and an OOB connection (which can, in this case, be the backbone) is made between the handheld device and the DBP. The PD then uses the OOB communication interface (most probably the backbone interface) to populate the attributes of the DPO in the new device.
d) The device is now ready to join the target network. However, unlike a field device, a backbone device may not have a DL interface. For example, the device may be a gateway residing on the backbone. Alternatively, the backbone device may be the first advertising router connected to the network. For example, the device may be a backbone router with advertisement router functionality on the IEEE 802.15.4 physical layer interface. However, the device needs to be provisioned over the backbone and not through the PhL, since in this case there are no advertising routers that can forward their join request to the system manager.

To talk to the system manager on the backbone without the help of an advertising router, the backbone router sends a join request to the system manager directly over the backbone; the backbone device can form the network header necessary to send this message. It can do so because it has been provisioned with the IPv6Address of the system manager (DPO.Target_System_Manager_Address). The remaining procedure at the system manager is same as that in 13.11.2.3.

Annex A
(informative)

User layer / application profiles

A.1 Overview

Annex A describes what is meant by the terms “user layer” and “application profile”, and also describes how these terms relate to each other and to this standard.

A.2 User layer

The user layer is the term often applied to a non-existent eighth layer located atop the OSI seven-layer computer networking model. The intent of the user layer is to perform purpose-specific functions not related to network communications. With respect to industrial automation, the term user layer is sometimes applied to describe non-network communication-related hardware and/or software, such as a field sensor or a process control function block. It is possible that such a user layer has information that is to be communicated over another network that conforms to ISO/IEC 7498, the OSI Basic Reference Model.

The network communication to support the user layer function is initiated by the user layer employing the methods and protocols defined by the 7th layer of the OSI model, which is called the AL.

This standard is intended to support a variety of industrial automation industry functions that are not directly related to network communication. As such, it defines a general purpose communication stack compatible with the OSI computer networking model and includes the definition of AL standard services.

This standard also defines generic extensible standard objects, which may be used by industrial automation applications. This standard permits specialization of those standard objects, as well as definition of both new industry-specific standard objects and vendor-defined objects. The definition of industry-specific standard objects is outside the scope of this standard; that is left to each industry organization that promotes use of this standard for their industry. This standard does not limit the scope of user-layer functionality relative to any non-network-related communication need.

NOTE The ISA100 Wireless Compliance Institute (WCI) is an example of such an organization for the process automation industries.

A.3 Application profile

An application profile defines application-specific properties to be implemented in a manner that fosters inter-operability among communicating entities. An application profile may also define implementation policies, and may suggest implementation guidelines. Any user layer within a device may implement one or more application profiles.

Some application-profile-specific properties may be mandatory for all instances of applications compliant with the particular application profile. Other application-profile-specific properties may be common practice properties that are construction options. All of these properties are represented as object attributes, so that communication of their values can occur through use of the basic application-layer services of this standard.

The scope of an application profile is often deliberately limited, in order to promote greater adoption and use of the particular application profile. An example of such a limited application profile is an application profile for temperature sensors.

In a loosely coupled system, the binding of devices that support application profiles with host system applications that employ those profiles usually is accomplished via the use of a device characterization file provided by the device vendor, the content of which often is based on a standard descriptive technology.

An example of a standard that may be used to describe profile content is IEC 61804-3,*)*, which may be used by industrial automation industry device vendors to create a file that may be used with appropriate host system companion tools, enabling the host to represent device functions, parameters (attributes) and their dependencies, graphical representations appropriate to data representation, as well as supported interactions with other devices.

A device may implement an application profile or set of profiles and may use the native AL methods and protocols of this standard to communicate over wireless networks conforming to this standard.

Because this standard is intended to support a variety of non-network communication-related industrial automation industry functions, this standard does not define or limit the definition or use of application profiles, languages or files that represent such devices, or tools used to represent such devices. The definition of industry-specific standard application profiles is therefore outside the scope of this standard. Instead it is delegated to those organizations that promote use of this standard in a particular automation industry.

NOTE ISO and IEC mechanisms exist for proposing such industry-specific application profiles.

Annex B
(normative)

Role profiles

B.1 Introduction

B.1.1 General

A role profile is defined as the baseline capabilities, including any settings and configurations, that are required of a device to perform that role adequately. The roles are defined in 5.2.6.2, but are listed for reference here as system manager, security manager, backbone router, router, I/O, gateway, system time source, and provisioning device.

Annex B provides the role profile pro forma for compliance to this standard.

B.1.2 Purpose

The role profile will define those device capabilities, such as settings and configurations, necessary to fulfill each specific role defined in 5.2.6.2. The purpose for this is to ensure that devices complying with this standard, including Annex B, can be interworkable or interoperable, as appropriate, within the domain covered by the role profile.

B.1.3 System size

The capabilities required of a device to implement a role may be dependent upon the number of devices in the intended system. The minimum system size is defined in Clause 5, but there is no maximum system size. To allow the requirements of Annex B to serve a broad range of system sizes, those requirements dependent upon system size shall use a formula to specify the minimum capability. For the purposes of Annex B, the number of system devices is referred to as NSD.

B.1.4 Abbreviations and special symbols

Abbreviations and symbols used include:

- Notations for requirement status:
  - M: mandatory;
  - O: optional;
  - O.n: optional, but support of at least one of the group of options labeled O.n is required;
  - N/A: not applicable;
  - X: prohibited.
- Item: Conditional, status dependent upon the support marked for the item.

For example, a status of FD1:O.1 and FD2:O.1 indicates that the status is optional but at least one of the features described in FD1 and FD2 is required**.**

B.1.5 Role profiles

Table B.1 describes the protocol layers and media requirements for all role profiles. Should a device be declared to support more than one role, that device shall fulfill minimum capabilities for each role declared.

Table B.1 – Protocol layer device roles

Item number	Device role	Status					Reference	Support		
		Protocol layers			Medium					
		AL	TL	NL	Type A	Backbone		N/A	Yes	No
DR1	I/O	M	M	M	M	N/A	5.2.6.6			
DR2	Router	M	M	M	M	N/A	5.2.6.7			
DR3	Backbone router	M	M	M	M DR4: O DR7: O	M	5.2.6.9 5.2.6.9 5.2.6.9			
DR4	Gateway	M	M	M	DR2:O.1	DR3:O.1	5.2.6.10			
DR5	System time source	N/A	N/A	N/A	N/A	N/A	5.2.6.13			
DR6	Provisioning	M	M	M	M	N/A	5.2.6.8			
DR7	System manager	M	M	M	DR2:O.2	DR3:O.2	5.2.6.11			
DR8	Security manager	N/A	N/A	N/A	N/A	N/A	5.2.6.12			

B.2 System

The protocol of WISN supports the ability to upgrade devices over-the-air, as shown in Table B.2.

Table B.2 – Over-the-air upgrades

Item number	Role types affected	Reference	Status	Support		
				N/A	Yes	No
OTAR1	I/O		M			
OTAR2	Router		M			
OTAR3	Backbone router		N/A			
OTAR4	Gateway		N/A			
OTAR5	System manager		N/A			
OTAR6	Provisioning device		O			

B.3 System manager

The system manager allocates the ability for devices to communicate by generating, distributing, and maintaining contracts that define the resources necessary for that communication need. Since each device is required to store its contracts, the capacity of a device for contract storage is critical. While the necessary capacities of the I/O, router, and backbone router devices are dependent upon the number of application objects within those devices, the gateway and system manager are dependent upon the number of devices in the system, defined in Annex B as NSD. NSD does not include the system manager in its device count.

Contracts require communication sessions for communication, established by the security manager in conjunction with the system manager. Multiple contracts, communicating to the same endpoints, may share a single session. Minimum capacities described here assume that each session is matched with a single contract, recognizing that more contracts may be needed depending on the nature of the device's applications.

B.4 Security manager

The security manager establishes sessions between application processes. For example, when a device joins the network it needs a DMAP-SMAP session. The number of sessions that a device implementing a role shall be able to maintain is defined in Table B.3. The number of sessions supported by a system manager is dependent on NSD. The number of keys supported by a gateway is dependent on the number of Gateway-UAP connections that the gateway is designed to support, referred to as GUC in Table B.3.

An I/O device is presumed to require capacity to support the following sessions:

- A session between the device’s DMAP and the SMAP, established when the device joins the network.
- A session between the device’s UAP and a first device such as a gateway.
- A session between the device’s DMAP and the first device, for reporting process alerts.
- A session between the device’s UAP and a second device’s UAP, such as for peer-to-peer communication.

Table B.3 – Session support profiles

Item number	Role types affected	Minimum number sessions supported	Comments	Status	Support		
					N/A	Yes	No
NCS1	I/O	4	DMAP-SMAP UAP-Gateway DMAP-Gateway UAP-Peer	M			
NCS2	Router	1	DMAP-SMAP	M			
NCS3	Backbone router	1	DMAP-SMAP	M			
NCS4	Gateway	(2 x GUC) + 1	DMAP-SMAP GUC x (Gateway-UAP) GUC x (Gateway-DMAP)	M			
NCS5	System manager	NSD	NSD x (SMAP-DMAP)	M			

The security manager assigns the security keys that are required for communication between devices. The number of keys that a device implementing a role shall be able to maintain is defined in Table B.4. The number of keys supported for a device depends on the number of sessions supported, with minimum capacities shown in Table B.3. In addition, each device needs capacity for a join key, a master key, and a D-key if a DL is included on the device. Key counts need to be doubled, because all keys except for the join key may be in the process of change-over.

Table B.4 – Baseline profiles

Item number	Role types affected	Minimum number keys supported	Comments	Reference	Status	Support		
						N/A	Yes	No
NKS1	I/O	1+((NCS1+2)×2)		7.2.2	M			
NKS2	Router	1+((NCS2+2) ×2)		7.2.2	M			
NKS3	Backbone router	1+((NCS3+2) ×2)		7.2.2	M			
NKS4	Gateway	1+((NCS4+1) ×2)	Add 2 if gateway has a DL	7.2.2	M			
NKS5	System manager	(NCS5+1) ×2	Add 2 if SM has a DL					
NKS7	Security manager	–N/A		7.2.2	N/A			

B.5 Physical layer

Since the PhL cites the specifications from IEEE 802.15.4:2011, the role capabilities for the PhL are referenced in IEEE 802.15.4:2011.

Table B.5 describes the physical layer roles.

Table B.5 – PhL roles

Item number	Item description		IEEE 802.15.4:2011 reference	Status	Support		
					N/A	Yes	No
PLR1	I/O	The device is a reduced function device	5.1	O.1			
		The device is a full function device	5.1	O.1			
PLR2	Router	The device is a full function device	5.1	M			
PLR3	Backbone router	The device is a full function device	5.1	M			
PLR4	Provisioning device	The device is a full function device	5.1	M			
O.1: at least one option shall be selected.							

B.6 Data-link layer

B.6.1 General

The DL affects four role profiles, as indicated in Table B.6.

Table B.6 – DL required for listed roles

Item number	Role types	Reference	Status	Support		
				N/A	Yes	No
DLR1	I/O	5.2.6.6	M			
DLR2	Router	5.2.6.7	M			
DLR3	Backbone router	5.2.6.9	M			
DLR4	Provisioning	5.2.6.8	M			

B.6.2 Role profiles

B.6.2.1 General

A DL role profile describes a set of minimum capabilities that shall be supported by every compliant device that implements the Type A field medium. For example, a device filling the router role shall support 8 neighbors. If a device meets all of the other requirements of a router, but supports only 4 neighbors, it is not compliant in its role as router. A device may exceed any of the requirements of its role, as long as all of the roles' minimum requirements are met.

The DL is configured through settings to the DL management object (DLMO) attributes, and the various roles are described as ranges of DLMO settings that a device can support.

B.6.2.2 DL management object attributes

A device's level of support for a capability can be expressed in relation to a set of DLMO attributes and elements of those attributes. Each attribute and/or element whose support varies by role is included.

If a number or range of numbers is listed, then a device filling this role shall support that number. If a single number is listed, it shall be interpreted as a minimum value unless indicated otherwise. For example, if a device shall support 3 neighbors, then it may support 4 neighbors, but is non-compliant if it supports only 2 neighbors. An I/O device may be capable of routing even if it isn't fully compliant with the router role; hence some capabilities related to routing are shown as optional (not prohibited) for an I/O device.

Table B.7 describes simple DLMO attributes with a single element. (The remaining tables in Annex B address DLMO attributes containing multiple elements.)

Table B.7 – Role profiles: General DLMO attributes

Attribute	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
ActScanHostFract	O	M	M	A non-mains device will not necessarily have the energy to act as an active scanning host for an extended period of time. See Table B.8			
AdvJoinInfo AdvSuperframe	O	M	M	All routers and backbone routers can be configured to send advertisements			
TaiTime TaiAdjust	M	M	M	The DL is not necessarily the source of TaiTime for a particular device, and there are cases where a device's DL might not be involved in time propagation as a source or recipient. For example, a BBR might remain time synchronized through a backbone mechanism, and not be involved in DL time propagation			
ClockTimeout	M	M	M	A BBR may be configured as a clock recipient, but this is not intended as typical			

Table B.8 describes baseline role profiles for the dlmo.Device_Capability attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.8 – Role profiles: dlmo.Device_Capability

Element	Status			Notes	Support		
	I/O	Router	BBR		N/A	Yes	No
QueueCapacity	0	10	20	(Note 1)			
ClockStability	100	10	10	(Note 2)			
DLRoles	0000 xxx1	0000 xx1x	0000 x11x	a)			
AdvRate	0 (X)	6	6	b)			
ListenRate	0 (X)	36	36	c)			
TransmitRate	0 (X)	30	60	d)			

NOTE 1 A system manager configures the DL queue only to the extent that the device is forwarding messages on behalf of other devices. The DL queue in a BBR is an internal device matter for graphs that originate or terminate in the device's DL.

NOTE 2 ClockStability values, as multiples of 1 x 10^{-6} are maximum allowed values over any continuous 30 s interval under industrial operating conditions. While low-cost I/O devices may have clocks with a short-term stability of only 100 x 10^{-6}, industrial I/O devices in general should have better stability. This standard was designed assuming that I/O devices have clocks with a short-term stability of 25 x 10^{-6} or better, and it is anticipated that most application profiles will be constrained accordingly.

a) Bits indicate all of the DL roles that are supported by the device. Note that BBR is required to act as a router, such as for peer-to-peer messaging within a D-subnet.

b) All devices serving router and backbone router roles shall have sufficient resources to transmit an advertisement every 10 s (6 DPDUs per minute), on average. See 9.1.17.

c) All devices serving router and backbone roles shall have sufficient resources to operate their receivers for 36 s per hour (1%), on average. A mains powered BBR will normally be capable of running its receiver continuously, but some BBR classes (such as wireless bridges) might be energy constrained

d) All devices serving router and backbone roles shall have sufficient resources to transmit the specified number of DSDUs per minute. See 9.1.17.

Table B.9 describes baseline role profiles for the dlmo.Ch attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.9 – Role profiles: dlmo.Ch (channel-hopping)

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	10	10	10	Five default channel-hopping sequences, numbered 1..5, are defined by this standard. A device can be provisioned or configured with up to 5 additional channel-hopping sequences			
MaxRowID (metadata)	127	127	127	One octet			

Table B.10 describes baseline role profiles for the dlmo.TsTemplate attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.10 – Role profiles: dlmo.TsTemplate

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	8	10	10	Three default timeslot templates, numbered 1..3, are defined by this standard. These are included in the capacity			
MaxRowID (metadata)	127	127	127	One octet			

Table B.11 describes baseline role profiles for the dlmo.Neighbor attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.11 – Role profiles: dlmo.Neighbor

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2	8	32	An I/O shall support at least two neighbors, so that it can maintain two active DL routes for reporting. A router adds additional capacity to support routing on behalf of neighbors			
MaxRowID (metadata)	2^{15}	2^{15}	2^{15}	6LoWPAN unicast address limited to 2^{15}			
GroupCode	O	M	M	GroupCode enables links to be used for multiple neighbors			
ExtendGraph	O	O	O	Automatic extension of graphs is required for all devices. Support for the ExtendGraph field is a construction option that provides a finer degree of control over graph extensions			

Table B.12 describes baseline role profiles for the dlmo.Diagnostic attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.12 – Role profiles: dlmo.NeighborDiag

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2×15 +1×9	3×15 +2×9	3×15 +2×9	Diagnostic capacity (metadata) is measured in octets. Summary diagnostics, in Table 188, involve 15 octets of storage in the worst case. Actual storage and transmission may be more compact. Summary diagnostics are intended to be maintained on the “publication” side of a given link, to collect diagnostics from the direction where more traffic flows. Summary diagnostics include a baseline clock diagnostic (ClockSigma). More detailed clock diagnostics (Table 190) involve 9 octets of storage in the worst case. A summary clock diagnostic is provided along with the general diagnostic. Capacity is provided to collect these detailed clock diagnostics on an as-needed basis			
MaxRowID (metadata)	2^{15}	2^{15}	2^{15}	6LoWPAN unicast address limited to 2^{15}			

Table B.13 describes baseline role profiles for the dlmo.Superframe attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.13 – Role profiles: dlmo.Superframe

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	3	5	10	Default superframes for discovery of provisioning device are included in this count			
MaxRowID (metadata)	127	127	127	One octet			
AlwaysHop	O	O	O	Support for this feature is a construction option			

Table B.14 describes baseline role profiles for the dlmo.Graph attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.14 – Role profiles: dlmo.Graph

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2	8	16				
MaxRowID (metadata)	127	127	127	One octet			

Table B.15 describes baseline role profiles for the dlmo.Link attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.15 – Role profiles: dlmo.Link

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	9	15	30	Default links for discovery of provisioning device are included in this count			
MaxRowID (metadata)	127	127	127	One octet			
Discovery	0, 3	0, 1, 2, 3	0, 1, 2	Discovery refers to bits 3/2 in Table 182. A system manager may be configured to discover routing-capable neighbors through active or passive scanning for advertisements			
JoinResponse	O	M	M				
NeighborType=2	O	M	M	Support of neighbor groups is mandatory for routing devices			

Table B.16 describes baseline role profiles for the dlmo.Route attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.16 – Role profiles: dlmo.Route

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	3	1	64	I/O device has capacity for routing to the system manager, a first device, and a second device. Router needs only a route to the system manager. BBR needs at least one route (outbound route lookup) for each device in its sphere of influence, even if those routes are identical to each other			
MaxRowID (metadata)	127	127	127	One octet			

Table B.17 describes baseline role profiles for the dlmo.Queue_Priority attribute. Those device elements not mentioned in Annex B shall be supported as described in Clause 9.

Table B.17 – Role profiles: dlmo.Queue_Priority

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	O	2	2				
MaxRowID (metadata)	127	127	127	One octet			

B.7 Network layer

Table B.18 describes role profiles for routing table sizes.

Table B.18 – Routing table size

Item number	Role types affected	Minimum number entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
RTS1	I/O	0			M			
RTS2	Router	0			M			
RTS3	Backbone router	15			M			

Table B.19 describes role profiles for address table sizes.

Table B.19 – Address table size

Item number	Role types affected	Minimum number entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
ATS1	I/O	4			M			
ATS2	Router	3			M			
ATS3	Backbone router	15			M			

B.7.1 Transport layer

Table B.20 describes role profiles for port support sizes.

Table B.20 – Port support size

Item number	Role types affected	Minimum number entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
PSS1	I/O	2			M			
PSS2	Router	1			M			
PSS3	Backbone router	1			M			

B.8 Application layer

Table B.21 describes the minimum number of APs per role.

Table B.21 – APs

Item number	Role types affected	Minimum number APs supported	Comments	Reference	Status	Support		
						N/A	Yes	No
UAPO1	I/O	2		Clause 6,12.17	M			
UAP02	Router	1		Clause 6	M			
UAP03	Backbone router	1		Clause 6	M			
UAP04	Gateway	2		Clause 6, Annex U	M			
NOTE The maximum number of contained objects supported includes the UAPMO.								

B.9 Provisioning

Table B.22 provides the role profile devices implementing the I/O, router, gateway, or backbone router roles, all devices with a Type A field medium.

Table B.22 – Role profiles: I/O, routers, gateways, and backbone routers

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
DBPR-1	Joining a provisioning network using K_global	13.6	M		See 7.2.2.2			
DBPR -2	Joining a provisioning network using K_open	13.6	O		Default value of K_join = K_open. Disabled once S is overwritten. Enabled again only if device reset to factory defaults			

B.10 Gateway (informative)

Table B.23 provides a notional role profile for a gateway.

Table B.23 – Role profile: Gateway

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP1	Native access	U.3.1.5	O.1	Allows native service access only			
GWRP2	Interworkable tunnel mechanism	U.3.1.5	O.1	Allows tunneled access only			

Table B.24 provides the notional role profile for gateway native access.

Table B.24 – Role profile: Gateway native access

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP1.1	Min IFOs supported	U.3.1.5	1				
GWRP1.2	Buffered message behavior	U.3.4		Constant, static, dynamic, non-cacheable.			
GWRP1.3	Min devices	Table 373	NSD	NSD ≥ 5			
GWRP1.4	Min leases	Table 373	2 x NSD - 3	NSD ≥ 5			

Table B.25 provides the notional role profile for a gateway interworkable tunnel mechanism.

Table B.25 – Role profile: Gateway interworkable tunnel mechanism

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP2.1	Min TUNs supported	U.3.1.5	GD x AD + 1	GD ≥ 1 AD ≥ 5			
GWRP2.1.1	Supports a foreign protocol	U.3.1.5	Annex O				
GWRP2.1.2	2-part tunneling	U.3.1.5					
GWRP2.1.3	TUN objects with Array of Tunnel endpoints attributes with multiple address elements	U.3.1.5	1				
GWRP2.1.3.1	Number of elements in TUN with multiple address elements	U.3.1.5	A	A ≥ 5			
GWRP2.2	Min devices	Table 373	A	A ≥ 5			
GWRP2.3	Min leases	Table 373	2 x A	A ≥ 5			

Key:

FNG = number of foreign nodes behind gateway.

FNA = number of foreign nodes behind adapter(s).

A = number of adapters.

Annex C
(informative)

Background information

C.1 Industrial needs

The wireless needs for industrial applications are significantly different than those required for residential, commercial, or military applications. These differences stem from the unique industrial ranking of priorities of characteristics such as device cost, system cost, lifecycle cost, reliability, maintainability, consistency, robustness, extensibility, security, coexistence, regulatory restrictions, interconnectability, and (within the relevant domains) interworkability or interoperability.

ISA100 committee members collected and analyzed more than 500 use cases to define more completely the wireless communication needs of the industrial sector. The major conclusions of this effort were:

- Opportunity: Non-existent wireless sensing is an opportunity for end users, vendors, and emerging standards.
- Interworkable: Since multi-instrument-vendor facilities dominate the industrial environment, wireless standards should be of high value.
- Interoperable: Devices that target the same broad application domain (e.g., process control or asset management) should be interoperable with respect to basic functionality needed for cooperative action in that application domain.
- Integration: Multiple communication paths between devices are needed, especially to distributed control system (DCS) instruments.
- Applications: Applications such as monitoring/alerting are of greatest immediate interest since they constitute the largest potential use of wireless devices.
- Reliability and security: Critical factors for emerging standards and vendors.
- Power: Battery life expectations will vary due to application, environment, cost constraints, etc. Some devices will have mains power, while others will be powered by batteries or will scavenge energy from the environment.

C.2 Usage classes

C.2.1 General

While there are many techniques that may be used to categorize the communications needs of industrial applications, this standard uses classes based upon usage. Analysis of the patterns of intended use of inter-device industrial wireless communications resulted in a partitioning of such communications into six classes. These classes are summarized in Table C.1.

Table C.1 – Usage classes

Safety	Class 0: Emergency action	Always critical
Control	Class 1: Closed loop regulatory control	Often critical
	Class 2: Closed loop supervisory control	Usually non-critical
	Class 3: Open loop control	Human in the loop
Monitoring	Class 4: Alerting	Short-term operational consequence (e.g., event-based maintenance)
	Class 5: Logging and downloading / uploading	No immediate operational consequence (e.g., history collection, sequence-of-events, preventive maintenance)
NOTE Batch levels 3 and 4 could be class 2, class 1 or even class 0, depending on function. Batch levels are defined in IEC 61512-1, where L3 = unit and L4 = process cell.		

C.2.2 Class examples

- Class 0: Emergency action (always critical)

  Examples include:
  - safety interlock;
  - emergency shutdown;
  - automatic fire control.
- Class 1: Closed loop regulatory control (often critical)

  Examples include:
  - direct control of primary actuators (e.g., field device to host connection availability on demand of at least 99,99%, with link outages > 500 ms intolerable, with demand rates of 0,2 Hz or greater);
  - high-frequency cascade loops.
- Class 2: Closed loop supervisory control (usually non-critical)

  Examples include:
  - low-frequency cascade loops;
  - multivariable controls;
  - optimizers.
- Class 3: Open loop control (human in the loop)

  Examples include:
  - operator manually initiates a flare and watches the flare;
  - guard remotely opens a security gate;
  - operator performs manual pump/valve adjustment.
- Class 4: Alerting – Short-term operational consequence

  Examples include:
  - event-based maintenance;
  - marginal bearing temp results in technician sent to field;
  - battery low indicator for a device results in technician sent to change battery;
  - asset tracking.
- Class 5: Logging – data/messages with no immediate operational consequence

  Examples include:
  - history collection;
  - preventive maintenance rounds;

– sequence of events (SOE) uploading.

NOTE SOE uses lossless communication, such as file transfer, rather than timely communication such as used by control messaging.

C.2.3 Other uploading and downloading alarms (human or automated action)

Alarm examples include:

- Class 0: leak detector for radiation or fatally toxic gas, automated response (e.g., automated containment response).
- Class 1: high-impact process condition, automated response (e.g., automated shutdown of reaction).
- Class 2: automated response to process condition (e.g., automated flow diversion).
- Class 3: process condition with manually-initiated operational response (e.g., decide whether to divert flow to a parallel reactor).
- Class 4: equipment condition with short-time-scale maintenance response (e.g., send technician to field).
- Class 5: equipment condition with long-time-scale maintenance action (e.g., order spare parts).

C.3 The Open Systems Interconnection Basic Reference Model

C.3.1 Overview

This standard defines the protocol suite of the wireless network. A protocol suite is a particular software implementation of a networking protocol suite. In practical implementation, protocol suites are often divided into layers such as those defined by the Open Systems Interconnection Basic Reference Model ISO/IEC 7498-1. The format in this standard is based upon this reference model (see Figure C.1), implementing five of the basic reference model's seven layers.

NOTE It is useful to realize that this is a virtual model, which therefore imposes no actual requirements on implementations, or even specifications.

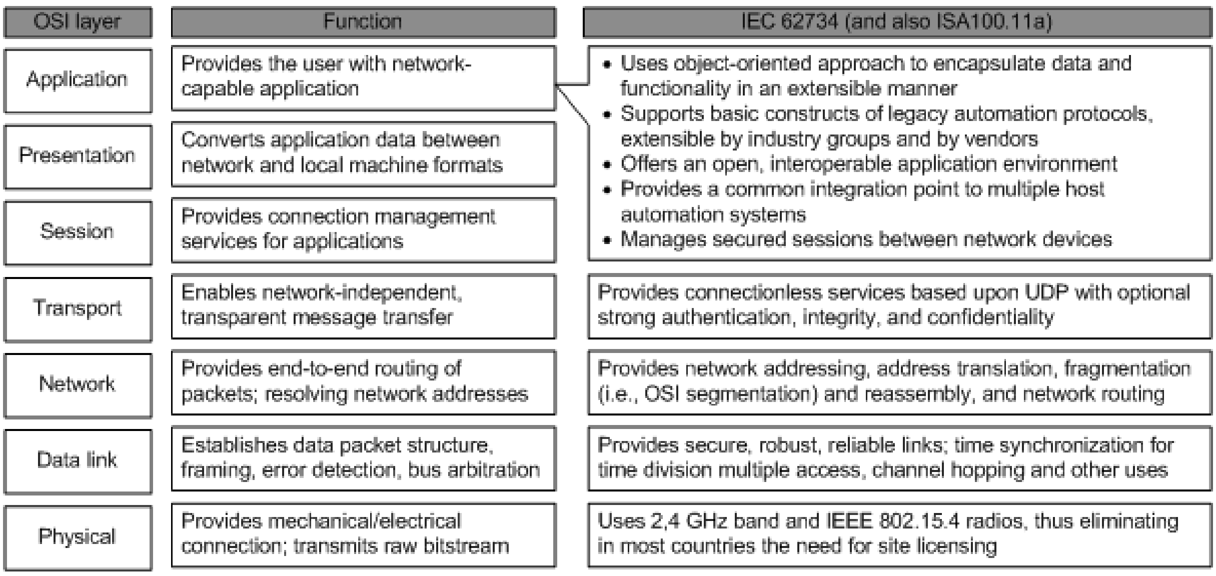


Figure C.1 – OSI Basic Reference Model

The upper layer, application (AL), of the Basic Reference Model of this standard provides local functionality for one or more associated UAPs.

The four lower layers, transport (TL), network (NL), data-link (DL), and physical (PhL), are devoted to data communication. Each has the capability of multiplexing and demultiplexing, and of splitting and merging information flows from adjacent layers. In other words, the messaging relationships between an AL entity and a TL entity, or between a TL entity and an NL entity, or between an NL entity and a DL entity, or between a DL entity and a PhL entity, do not have to be one-to-one.

These lower layers also have the following abilities to:

- to sequence service data units (SDUs) to maintain the order of original presentation;
- to do one or more of the following
  - segment or reassemble SDUs into protocol data units (PDUs),
  - block or deblock SDUs into protocol data units (PDUs), and
  - concatenate or separate PDUs,

  so that they are sized more appropriately for the conveyance capabilities of the lower layer;
- to split PDUs for conveyance over multiple lower layer routes, or to recombine such PDUs on receipt before forwarding on a higher-layer route; and
- to acknowledge receipt of PDUs as a form of error control.

C.3.2 Application layer

The AL is the layer that interfaces directly to (and conceptually includes) UAPs, managing communications with other UAPs under the guidance of the local management UAP. A UAP may perform an individual function or any combination of functions. UAPs may be used, for example, to:

- handle input and/or output hardware;
- distribute communications to a set of co-resident UAPs within a device (proxy function);
- support tunneling of a non-native (e.g., control system legacy) protocol compatible with the network environment described in this standard; and/or
- perform a computational function.

The AL is usually composed of one or more UAPs that share common service elements.

The primary tasks of an AL entity are to provide:

- a place in the architecture of this standard for UAPs;
- the means by which UAPs manage communications with UAPs for other devices through the protocol suite, including:
  - identification of intended communications partners (e.g., by name, by address, by description, etc.),
  - agreement on security aspects (e.g., authentication, data integrity),
  - determination of acceptable quality of service (e.g., priority, time windows for control messaging, acceptability of out-of-order message delivery, acceptability of message delivery in partial increments, etc.),
  - agreement on responsibility for error recovery,
  - identification of abstract syntaxes, and
  - synchronization of cooperating UAPs;
- the means by which UAPs can inform the associated application entity of needed resource requirements, including those applicable to message buffering:
  - expected and maximum message sizes, and

- maximum expected burstiness of message transmission and reception or how many messages can be sent or arrive within a short amount of time as compared to the average periodicity of messages; and

- any necessary communication functions that are not already performed by the lower layers.

C.3.3 Transport layer

The TL is the highest layer at which communicating applications are addressable. The primary tasks of a TL entity are:

- to provide addressing of UAPs via selection of a specific associated AL entity;
- to establish end-to-end messaging paths from one UAP to one or more other UAPs via their associated AL entities, where those processes are usually in separate devices;
- to convey and regulate the flow of messages between or among those UAPs; and
- to terminate those messaging paths when appropriate.

C.3.4 Network layer

The NL is the highest layer at which communicating devices are addressable. It is the lowest layer with more than local scope, which forwards messages between one entity group and others, or discards the messages. The primary tasks of an NL entity are:

- to provide network-wide addressing of devices;
- to relay messages (NPDUs) between entities (e.g., a router) via D-subnets, usually changing source and destination DL entity addresses associated with the message envelopes (DPDUs) in the process, or to discard the NPDUs; and
- to provide segmentation and reassembly of messages, as appropriate, to match the capabilities of the D-subnets on which messages are being forwarded.

NOTE The NL is the OSI layer where endpoint device addressing and routing occur. Lower layer relays are able to forward messages within a single addressing domain without message modification, but are unable to readdress messages or span addressing domains. Network-wide device addresses are IPv6Addresses.

C.3.5 Data-link layer

The DL is the lowest information-centric layer, which coordinates interacting PhL entities and provides basic low-level messaging among DL entities. The primary tasks of a DL entity are:

- to provide link-local addressing of peer-DL entities;
- to convey messages (DPDUs) from one DL entity to all others whose PhL entities are correspondents (e.g., to all PhL entities of the local link), or to discard the DPDUs;
- to manage use of the PhL;
- to provide low-level message addressing, message timing and message integrity checks;
- to provide low-level detection of and recovery from message loss (e.g., immediate acknowledgment; retry if no acknowledgment); and
- optionally, to relay DPDUs between DL entities (e.g., a bridge).

NOTE The DL is the OSI layer that manages and compensates for the specific characteristics of the selected physical communications technology. It provides only local addressing, and forwards messages within the local addressing domain without readdressing. It does not modify message addresses. DL16Addresses have only local scope, so it is possible that the same DL16Addresses are duplicated in other local links.

C.3.6 Physical layer

The PhL is the lowest layer of the OSI model and the only layer that deals with real-world physics. All other layers deal with abstract information, ultimately represented as bits; the PhL is concerned with physical signals (sometimes referred to as baud or chips). The primary tasks of a PhLE are:

- to code bits, either singly or in multi-bit groups, into physical signals;
- to convey those signals from one physical location to another;
- to decode those signals into single-bit or multi-bit groups, possibly with error correction;
- to take direction from the associated DLE with respect to physical channel setup, physical receiver addressing and other aspects of the communications channel and coding;
- to convey to the locally-associated DLE information about the state of the PhLE, the channel and the last set of received signals; and
- optionally, to relay PhPDUs between PhLEs (e.g., a repeater).

Annex D
(normative)

Configuration defaults

D.1 General

Annex D summarizes the default settings for configuration.

D.2 System management

Table D.1 lists the system management configuration defaults.

Table D.1 – System management configuration defaults

Name	Initial default value	Reference
Confirmation_Timeout_Device_Diagnostics	10	Table 7
Alerts_Disable_Device_Diagnostics	0	Table 7
Confirmation_Timeout_Comm_Diagnostics	10	Table 7
Alerts_Disable_Comm_Diagnostics	0	Table 7
Confirmation_Timeout_Security	10	Table 7
Alerts_Disable_Security	0	Table 7
Confirmation_Timeout_Process	10	Table 7
Alerts_Disable_Process	0	Table 7
Comm_Diagnostics_Alarm_Recovery_AlertDescriptor	Default value: [FALSE, 3]	Table 7
Security_Alarm_Recovery_AlertDescriptor	Default value: [FALSE, 3]	Table 7
Device_Diagnostics_Alarm_Recovery_AlertDescriptor	Default value: [FALSE, 3]	Table 7
Process_Alarm_Recovery_AlertDescriptor	Default value: [FALSE, 3]	Table 7
DL_Alias_16_Bit	0	Table 10
Network_Address_128_Bit	0	Table 10
Device_Power_Status_Check_AlertDescriptor	Default value: [FALSE, 8]	Table 10
DMAP_State	1	Table 10
Join_Command	0	Table 10
Static_Revision_Level	0	Table 10
Restart_Count	0	Table 10
Uptime	0	Table 10
TAI_Time	0	Table 10
Comm_SW_Major_Version	0	Table 10
Comm_SW_Minor_Version	0	Table 10
System_Manager_128_Bit_Address	0	Table 10
System_Manager_EUI64	0	Table 10
System_Manager_DL_Alias_16_Bit	0	Table 10
Contract_Request_Timeout	30	Table 10
Max_ClientServer_Retries	3	Table 10
Max_Retry_Timeout_Interval	30	Table 10
DMAP_Objects_Count	1	Table 10
Warm_Restart_Attempts_Timeout	60	Table 10

Name	Initial default value	Reference
Current_UTC_Adjustment	35	Table 25
Next_UTC_Adjustment_Time	See Table 25	Table 25
Next_UTC_Adjustment	35	Table 25

D.3 Security

Table D.2 lists the security configuration defaults.

Table D.2 – Security configuration defaults

Name	Initial default value	Reference
Security_Level	1	Table 87
Protocol_Version	1	Table 92
DL_Security_Level	1	Table 92
Transport_Security_Level	1	Table 92
Join_Timeout	60 s	Table 92
MPDU_MIC_Failure_Limit	5	Table 92
MPDU_MIC_Failure_Time_Unit	60 s	Table 92
TPDU_MIC_Failure_Limit	5	Table 92
TPDU_MIC_Failure_Time_Unit	5	Table 92
DSMO_KEY_Failure_Limit	1	Table 92
DSMO_KEY_Failure_Time_Unit	1	Table 92
Security_MPDU_Fail_Rate_Exceeded_AlertDescriptor	[FALSE, 6]	Table 92
Security_TPDU_Fail_Rate_Exceeded_AlertDescriptor	[FALSE, 6]	Table 92
Security_Key_Update_Fail_Rate_Exceeded_AlertDescriptor	[FALSE, 6]	Table 92
pduMaxAge	510	Table 92
SoftLifeTime	50	Table 93
DSMO alert type 0 = Security_MPDU_Fail_Rate_Exceeded	0	Table 97
DSMO alert type 1 = Security_TPDU_Fail_Rate_Exceeded	0	Table 97
DSMO alert type 2 = Security_Key_Update_Fail_Rate_Exceeded	0	Table 97

D.4 Data-link layer

Table D.3 lists the DLE configuration defaults.

18259 **Table D.3 – DLE configuration defaults**

Name	Initial default value	Reference
ActScanHostFract	0	Table 141
AdvJoinInfo	Null	Table 141
AdvSuperframe	0	Table 141
SubnetID	0	Table 141
SolicTemplate	Null	Table 141
AdvFilter	See 9.4.2.20	Table 141
SolicFilter	See 9.4.2.20	Table 141
TaiAdjust	Null	Table 141
MaxBackoffExp	5	Table 141
MaxDsduSize	96	Table 141
MaxLifetime	120 (30 s)	Table 141
NackBackoffDur	60 (15 s)	Table 141
LinkPriorityXmit	8	Table 141
LinkPriorityRcv	0	Table 141
EnergyDesign	See 9.4.2.22	Table 141
DeviceCapability	See 9.4.2.23	Table 141
IdleChannels	0	Table 141
ClockExpire	See 9.4.2.1	Table 141
ClockStale	45	Table 141
RadioSilence	600	Table 141
RadioSleep	0	Table 141
RadioTransmitPower	See 9.4.2.1	Table 141
CountryCode	0x3C00	Table 141
Candidates	Null	Table 141
DiscoveryAlert	60	Table 141
SmoothFactors	See Table 153	Table 141
QueuePriority	N=0	Table 141
Ch	See 9.4.3.2	Table 141
TsTemplate	See 9.4.3.3	Table 141
Neighbor	Empty	Table 141
Superframe	Empty	Table 141
Graph	Empty	Table 141
Link	Empty	Table 141
Route	Empty	Table 141
NeighborDiag	Empty	Table 141
ChannelDiag	See 9.4.2.27	Table 141
Transaction receiver template parameters	See Table 165	Table 165
Transaction initiator template parameters	See Table 166	Table 166
Transaction receiver template for scanning parameters	See Table 167	Table 167

18260

D.5 Network layer

Table D.4 lists the NLE configuration defaults.

Table D.4 – NLE configuration defaults

Name	Initial default value	Reference
Enable_Default_Route	FALSE	Table 206
Max_NSDU_size	70	Table 206
Frag_Reassembly_Timeout	60	Table 206
Frag_Datagram_Tag	uniform random	Table 206
DroppedNPDUAlertDescriptor	[TRUE, 7]	Table 206
Source_Address*	0	Table 207
Destination_Address	0	Table 207
Contract_Priority	00	Table 207
Include_Contract_Flag	FALSE	Table 207
NWK_HopLimit	64	Table 208
Outgoing_Interface	0	Table 208

D.6 Transport layer

Table D.5 lists the TLE configuration defaults.

Table D.5 – TLE configuration defaults

Name	Initial default value	Reference
MaxNbOfPorts	15	Table 229
TPDUin	0	Table 229
TPDUinRejected	0	Table 229
TSDUout	0	Table 229
TSDUin	0	Table 229
TSDUinRejected	0	Table 229
TPDUout	0	Table 229
IllegalUseOfPortAlertDescriptor	[TRUE, 8] -- medium)	Table 229
TPDUonUnregisteredPortAlertDescriptor	[TRUE, 4] -- low	Table 229
TPDUoutOfSecurityPoliciesAlertDescriptor	[TRUE, 2] -- journal	Table 229

D.7 Application layer

Table D.6 lists the ALE configuration defaults.

Table D.6 – ALE configuration defaults

Name	Initial default value	Reference
ObjectIdentifier	0	Table 240
UAP_ID	0=N/A	Table 240
UAP_TL_Port	0=N/A	Table 240
State	Active	Table 240
Command	0=None	Table 240
MaxRetries	3	Table 240
Number of unscheduled communication correspondents	0=N/A	Table 240
Number of objects in the UAP including this UAPMO	1	Table 240
Static_Revision_Level	0	Table 240
Categories	0	Table 243
Errors	0	Table 243
State	0=Idle	Table 246
MaxBlockSize	1..(MaxNPDUsize + Max TL header size - max(sizeof (additional coding of AL UploadData service request), additional coding of sizeof(AL DownloadData service response))	Table 246
MaxDownloadSize	0	Table 246
MaxUploadSize	0	Table 246
DownloadPrepTime	0	Table 246
DownloadActivationTime	0	Table 246
UploadPrepTime	0	Table 246
UploadProcessingTime	0	Table 246
DownloadProcessingTime	0	Table 246
CutoverTime	0	Table 246
LastBlockDownloaded	0	Table 246
LastBlockUploaded	0	Table 246
ErrorCode	0 =(noError)	Table 246
Revision	0	Table 256
CommunicationEndpoint	The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	Table 256
MaximumItemsPublishable	Local matter	Table 256
NumberItemsPublishing	0	Table 256
Array of ObjectAttributeIndexAndSize	Element size is 0	Table 256
Concentrator ContentRevision	0	Table 258
CommunicationEndpoint	The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	Table 258
MaximumItemsSubscribing	Local matter	Table 258
NumItemsSubscribing	0	Table 258

Name	Initial default value	Reference
Array of ObjectAttributeIndexAndSize	Element size is 0	Table 258
Protocol	Local matter (protocol-specific)	Table 260
Status (Configuration status)	0	Table 260
Period (Data publication period)	0	Table 260
Max_Peer_Tunnels	0	Table 260
Num_Peer_Tunnels	0	Table 260
ObjectIdentifier	7	Table 283
MalformedAPDUsAdvise	FALSE	Table 283
TimeIntervalForCountingMalformedAPDUs	0	Table 283
MalformedAPDUsThreshold	0	Table 283
MalformedAPDUAlertDescriptor	[TRUE, 7]	Table 283
PV	NaN	Table 287
Mode	OOS	Table 287
Scale	Engineering units values for 0% and for 100% BOTH indicate 0	Table 287
OP	NaN	Table 290
Mode	OOS	Table 290
Readback	NaN	Table 290
Scale	Engineering units values for 0% and for 100% BOTH indicate 0	Table 290
PV_B	0	Table 293
Mode	Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access	Table 293
OP_B	0	Table 296
Mode	Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access	Table 296
Readback_B	0	Table 296
Target	OOS	Table 302
Actual	OOS	Table 302
Permitted	OOS	Table 302
Normal	OOS	Table 302
Engineering units at 100%	0	Table 304
Engineering units at 0%	0	Table 304
Decimal point location	0	Table 304

D.8 Provisioning

Table D.7 lists the provisioning configuration defaults.

Table D.7 – Provisioning configuration defaults

Name	Initial default value	Reference
Default_NWK_ID	0x0001	Table 368
Default_SYM_Join_Key	K_global	Table 368
Open_SYM_Join_Key	K_open	Table 368
Default_Channel_List	0x7FFF	Table 368
Join_Method_Capability	00	Table 368
Allow_Provisioning	TRUE (1)	Table 368
Allow_Over_The_Air_Provisioning	TRUE (1)	Table 368
Allow_OOB_Provisioning	TRUE (1)	Table 368
Allow_Reset_to_Factory_defaults	TRUE (1)	Table 368
Allow_Default_Join	TRUE (1)	Table 368
Target_NWK_ID	0	Table 368
Target_NWK_BitMask	0xFFFF	Table 368
Target_Join_Method	1 (Asymmetric key)	Table 368
Number of PKI_Certificates	1	Table 368
Current_UTC_Adjustment	35	Table 368
White_List	[ ]	Table 371
Symmetric_Key_List	{K_global}	Table 371
Symmetric_Key_Expiry_Times	{0xFFFF FFFF}	Table 371
Target_NWK_ID	0	Table 371
Target_Join_Method	1 (Asymmetric key)	Table 371
Target_Join_Time	0	Table 371
Allow_Provisioning	TRUE (1)	Table 371
Allow_Default_Join	TRUE (1)	Table 371
Device_Specific_Provisioning_Flag	disabled (0)	Table 371
DPSO_Alerts_AlertDescriptor	[FALSE, 6]	Table 371
Current_UTC_Adjustment	35	Table 371
Device_EUI	0x0000 0000 0000 0001	Table 372
Device_Symmetric_Key	K_global	Table 372
Device_Symmetric_Key_Expiry_Time	{0xFFFF FFFF}	Table 372
Target_NWK_ID	0	Table 372
Target_Join_Method	1 (Asymmetric key)	Table 372
Allow_Provisioning	TRUE	Table 372
Allow_Default_Join	TRUE	Table 372

D.9 **Gateway** (informative)

Table D.8 lists the gateway configuration defaults.

Table D.8 – Gateway configuration defaults

Name	Initial default value	Reference
Max_Devices	0	Table U.41

Annex E
(informative)

Use of backbone networks

E.1 General

Use of a backbone network can be advantageous to the system designer, since it takes the message off of the Type A field medium, allowing additional bandwidth and higher QoS for other messages.

E.2 Recommended characteristics

Although the backbone itself is not specified within this standard, it is assumed and recommended that the backbone will have the following characteristics:

- Throughput equal to or greater than the throughput of the Type A field medium (≥ 250 kbit/s).
- Capability of supporting two-way unsolicited message traffic.
- Quality of service of a sufficient level such that time synchronization can be maintained across the network. This may place specific time synchronization methods on the backbone.
- High reliability. Operation should not burden the network with frequent lost messages and retries.
- Security sufficient not to present a security threat to the end users application or the network.
- Capability of either encapsulating (tunneling) protocol TPDUs or TSDUs defined by this standard or translating them in a such a way that they may traverse the backbone without being modified when emerging at the backbone devices. In general, the backbone shall be able to take a standard-compliant TSDU from the point of ingress and deliver it across the backbone to the point of egress unmodified.
- Capability of preserving the end-to-end application security mechanisms.
- Support for multipoint networking between devices.

It is recognized that many standard fieldbuses may not have these characteristics and therefore may not be suitable for use as a backbone network. In many cases, a backbone network will be an IP network such as ISO/IEC 8802-3 (IEEE 802.3) or ISO/IEC 8802-11 (IEEE 802.11), but there is no requirement for this. There are many other alternatives in the marketplace that exist and are well-suited for the purposes of a backbone network. These might include simple point-to-point or point-to-multipoint wireless networks.

E.3 Internet protocol backbones

E.3.1 Methods of IPv6 protocol data unit transmission

In many cases, an available backbone will use an Internet protocol (IP) NL. In this case there are many different ways to transport the wireless industrial sensor network (WISN) TPDUs using standardized protocol behavior:

- Encapsulate wireless industrial sensor network transport protocol data units within IPv4 NPDUs.

  The mechanism used to encapsulate WISN TPDUs within IPv4 NPDUs is formally known as IPv6 over IPv4 or 6over4 and is sometimes called virtual Ethernet. This method is documented in IETF RFC 2529. A backbone router – IETF RFC 2529 refers to them as

IPv6 hosts – located on a physical link that has no directly connected IPv6 router may become a fully functional IPv6 host by using an IPv4 multicast domain as its virtual local link. Backbone routers connected using this method do not require IPv4-compatible addresses or configured tunnels.

- Tunnel wireless industrial sensor network transport protocol data units over IPv4 network.

  Following IETF RFC 4213, this method encapsulates IPv6 protocol data units (PDUs) within IPv4 headers to carry them over IPv4 routing infrastructures. Two types of tunneling are possible, configured and automatic. In configured tunneling, the IPv4 tunnel endpoint address is determined by configuration information on the encapsulating node. In automatic tunneling, the IPv4 tunnel endpoint address is determined from the IPv4 address embedded in the IPv4-compatible destination address of the IPv6 PDU.

- Encapsulate wireless industrial sensor network transport protocol data units within raw Ethernet DPDUs.

  This method specifies the NPDU format for transmission of IPv6 PDUs following IETF RFC 2464. Furthermore, this method dictates the formation of link-local IPv6Addresses and statelessly-autoconfigured addresses on Ethernet networks. Finally, this approach specifies the content of the source/target link-layer addresses used in router solicitation, router advertisement, neighbor solicitation, neighbor advertisement, and redirect messages when those messages are transmitted on an Ethernet network.

- Use native IPv6 backbone without any encapsulation or tunnelling.

  If the backbone uses an IPv6 NL, neither encapsulation nor tunneling is necessary, since the backbone native mode is to transport IPv6 PDUs.

E.3.2 Backbone router peer device discovery

For the backbone router (BBR) to function properly and to connect WISN devices on the backbone, it needs to know the backbone addresses of the other BBRs or peers in the backbone network. Within each BBR, the addressing information of its peers should be stored in a backbone router peer table (BRPT). There are two basic methods of generating the BRPT, configuration and discovery.

NOTE The BRPT and the mechanism for discovering peers are beyond the scope of this standard.

Configuration occurs when the addresses of peer BBRs are inserted into the BRPT by the system manager or an operator. The advantages of this method are that it is straightforward and prevents the BBR from accessing inappropriate devices on the backbone.

Discovery occurs when the BBR automatically searches the backbone for peer devices. There are multiple discovery techniques, such as those mentioned in IETF RFC 2529 and others. The advantages of this method are that it is automatic, requires no operator involvement, and can be easily and often updated.

E.3.3 Security

E.3.3.1 Security of transport protocol data units

The security mechanisms of the backbone are beyond the scope of this standard. Typical IP security methods include IPSec, SSL, and others. In addition to any security mechanisms on the backbone, the WISN TL security mechanism protects the TPDU within the backbone.

E.3.3.2 Security of the backbone

There is a perception by some that allowing a WISN to access an IP backbone could degrade the security of the backbone. This concern may be mitigated by restricting the BBR access to only peer BBRs via an access control list or by the use of firewalls set up to restrict access properly to specific devices.

Annex F
(normative)

Basic security concepts – Notation and representation

F.1 Strings and string operations

A string is a sequence of symbols over a specific set (e.g., the binary alphabet (0, 1) or the set of all octets).

The size of a string is the number of symbols it contains (over the same alphabet).

The right-concatenation of two strings x and y of size m and n respectively (notation $x \,||\, y$) is the string z of size m+n that coincides with x on its leftmost m symbols and with y on its rightmost n symbols.

An octet is a symbol string of size 8. In this context, all octets are strings over the binary alphabet.

F.2 Integers, octets, and their representation

Throughout this standard, the representation of integers as octet strings and of octet strings as binary strings shall be fixed.

All integers shall be represented as octet strings in most-significant-octet-first order. This representation conforms to the convention in ANSI X9.63:2011, 4.3.

All octets shall be represented as binary strings in most-significant-bit-first order.

F.3 Entities

Throughout this standard, each entity shall be a DEV and shall be uniquely identified by its EUI64Address. The parameter entityIdSize shall have the value 64.

Annex G
(informative)

Using certificate chains for over-the-air provisioning

This standard uses implicit certificate called the PublicReconstrKey (see Annex H for details) for the asymmetric key-based cryptography. Given the identity of a device A (ID_A) and the implicit certificate γ_A of the device, the public key of the device A can be computed using the following equation:

$Q_A = Hash(\gamma_A || ID_A)\gamma_A + Q_{CA}$

where Q_{CA} is the public key of the certificate authority (CA).

With this background, the following steps outline the process for OTA provisioning using asymmetric-key cryptography as outlined in Figure 137.

1) The CA publishes Q_{CA}, its public key, on the web.
2) The device manufacturer (DM) gets a certificate from the CA:

   C_DM = PublicReconstrKey(DM) || Subject(DM) || Issuer(CA) || Text

   where:

   - Subject = ID of the DM
   - Issuer = ID of the CA
   - PublicReconstrKey_DM = γ_DM is used to calculate the public key of the DM using the equation:

     $Q_{DM} = HASH(\gamma\_DM || Subject)\gamma\_DM + Q_{CA}$
3) The individual device gets a certificate from the DM:

   C_DEV = PublicReconstrKey(DEV) || Subject(DEV) || Issuer(DM) || Text

   where:

   - Subject = ID of the device
   - Issuer =ID of the DM
   - PublicReconstrKey_DEV = γ_DEV is used to calculate the public key of the device using the equation:

     $Q_{DEV} = HASH(\gamma\_DEV || Subject)\gamma\_DEV + Q_{DM}$
   - C_DEV and C_DM are populated in the DBP by the DM.
4) The DBP joins the PD in a provisioning network. The PD has QCA.
5) The DBP sends a random number, C_DEV, and C_DM to the PD. The PD calculates Q_DEV as explained in steps 2) and 3).
6) A challenge/response mechanism is used to authenticate the device, and the security manager should validate the manufacturer's implicit certificate at this point.
7) If the challenge/response is passed, the PD sends K_join encrypted with Q_DEV.

Annex H
(normative)

Security building blocks

H.1 Symmetric key cryptographic building blocks

H.1.1 Overview

The following symmetric key cryptographic primitives and data elements are defined for use with all security processing operations specified in this standard.

H.1.2 Symmetric key domain parameters

The symmetric key shall have key size keylen=128 (in bits).

H.1.3 Block cipher

The block cipher used in this standard shall be AES-128, as specified in ISO/IEC 18033-3. This block-cipher shall be used with symmetric keys as specified in H.1.2. In this case the key size is equal to the block size of the block-cipher, 128 bits.

H.1.4 Mode of operation

The block-cipher mode of operation used in this standard shall be the CCM* mode of operation, as specified in IEEE 802.15.4:2011, B.3.2.

H.1.5 Cryptographic hash function

The cryptographic hash function used in this standard shall be the block cipher-based cryptographic hash function specified in Clause H.9, with the following instantiations:

- each entity shall use the block-cipher E as specified in H.1.3;
- all integers and octets shall be represented as specified in Clause F.2.

The Matyas-Meyer-Oseas hash function (see Clause H.9) has a message digest size hashlen that is equal to the block size, in bits, of the established block cipher.

H.1.6 Keyed hash function for message authentication

The keyed hash message authentication code (HMAC) used in this standard shall be HMAC, as specified in the FIPS 198, with the following instantiations:

- each entity shall use the cryptographic hash H function as specified in H.1.5;
- the block size B shall have the integer value B=keylen/8, where keylen is as specified in H.1.2 (i.e., B is equal to the size of the symmetric key, in octets, that is used by the keyed hash function);
- the output size HMAClen of the HMAC function shall have the same integer value as the message digest parameter hashlen, as specified in H.1.5.

H.1.7 Specialized keyed hash function for message authentication

The specialized[12] keyed hash message authentication code used in this standard shall be the keyed hash message authentication code, as specified in H.1.6.

H.1.8 Challenge domain parameters

The challenge domain parameters used in this standard shall be as specified in H.6.2, with the instantiation (minchallengelen, maxchallengelen)=(keylen, keylen), where keylen is as specified in H.1.2.

All challenges shall be validated using the challenge validation primitive as specified in Clause H.7.

H.2 Asymmetric-key cryptographic building blocks

H.2.1 General

The following asymmetric-key cryptographic primitives and data elements are defined for use with all security processing operations specified in this standard.

NOTE See also ISO/IEC 18033-2 for more information on asymmetric cryptography.

H.2.2 Elliptic curve domain parameters

The elliptic curve domain parameters used in this specification shall be those for the curve ansit283k1 as specified in ANSI X9.63:2011, Appendix J4.5, example 1.

All elliptic curve points shall be validated using the public key validation primitive as specified in ANSI X9.63:2011, 5.2.2.

H.2.3 Elliptic curve point representation

All elliptic curve points shall be represented in polynomial notation as specified in ANSI X9.63:2011, 4.1.2.1. All elliptic curve points shall be transmitted in compressed form, as specified in ANSI X9.63:2011, 4.2.2.

H.2.4 Elliptic curve public-key pair

An elliptic curve-key pair consists of an integer q and a point Q on the curve determined by multiplying the generating point G of the curve by this integer (i.e., Q=qG) as specified in ANSI X9.63:2011. Here, Q is called the public key, whereas q is called the private key; the pair (q, Q) is called the public-key pair. Each private key shall be represented as specified in ANSI X9.63:2011, 4.3.1. Each public key shall be on the curve as specified in H.2.2 and shall be represented as specified in H.2.3.

H.3 Keying information

H.3.1 General

The following specifies the format of asymmetric-key keying information used in this standard.

---

[12] This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also remark 9.8 of Menezes et al.). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

H.3.2 Elliptic curve cryptography implicit certificates

Implicit certificates IC_U shall be specified as

IC_U = PublicKeyReconstrData || Subject || Issuer || Usage_Serial || KeyValidityInfo || Text

where:

- The parameter PublicKeyReconstrData shall be the public-key reconstruction data BEU as specified in the implicit certificate generation scheme (see H.5.1).
- The parameter Subject shall be the entity U that is bound to the public key reconstruction data BEU during execution of the implicit certificate generation scheme, i.e., the entity that purportedly owns the private key corresponding to the public key that can be reconstructed from PublicReconstrKey.
- The parameter Issuer shall be the entity of the certificate authority (CA) that creates the implicit certificate during the execution of the implicit certificate generation scheme.
- The parameter Usage_Serial is defined in Table 68.
- The parameter KeyValidityInfo shall indicate the validity period of the keying material as indicated by the parameters ValidNotBefore and ValidNotAfter, which indicate the beginning and the end of the validity period, respectively. The KeyValidityInfo shall be formatted as

  KeyValidityInfo = ValidNotBefore || ValidNotAfter

  where ValidNotBefore and ValidNotAfter shall be represented as specified in □.
- The parameter Text shall be the representation of additional information, as specified in H.3.4.
- The string I_U as specified in the implicit certificate generation scheme (see H.5.1) shall be the octet string consisting of the octet strings Subject, Issuer, and Text, as follows:

  I_U = Subject || Issuer || Text

H.3.3 Elliptic curve cryptography manual certificates

Manual certificates MC_U shall be specified as MC_U = PublicKey || Subject || Issuer || Text, where:

- The parameter PublicKey shall be the octet representation of the public key W_U as specified in the manual certificate generation transformation.
- The parameter Subject shall be the entity U of the purported owner of the private key corresponding to the public key represented by PublicKey.
- The parameter Issuer shall be the entity of the CA that creates the manual certificate during the execution of the manual certificate generation transformation (the so-called certificate authority).
- The parameter Usage_Serial is defined in Table 68.
- The parameter KeyValidityInfo shall indicate the validity period of the keying material as indicated by the parameters ValidNotBefore and ValidNotAfter, which indicate the beginning and the end of the validity period, respectively. The KeyValidityInfo shall be formatted as

  KeyValidityInfo = ValidNotBefore || ValidNotAfter

- where ValidNotBefore and ValidNotAfter shall be represented as specified in □.
- The parameter Text shall be the representation of additional information, as specified in H.3.4.
- The string I_U as specified in the manual certificate scheme (see Clause H.10) shall be the octet string consisting of the octet strings Subject, Issuer, and Text, as follows:

  I_U = Subject || Issuer || Text.

NOTE A manual certificate is not a real digital certificate, since the binding between the PublicKey and the Subject is established and verified by non-cryptographic means.

H.3.4 Additional information

Additional information Text shall be specified as follows:

Text = Reserved

where the parameter Reserved allows for future extensions of the additional information and shall be set to the all-zero bit string for this version of the standard.

H.4 Key agreement schemes

H.4.1 Symmetric-key key agreement scheme

The symmetric-key key agreement scheme used in this standard shall be the full symmetric-key with key confirmation scheme as specified with the following instantiations:

- Each entity shall be identified as specified in Clause F.3.
- Each entity shall use the HMAC-scheme as specified in H.1.5.
- Each entity shall use the cryptographic hash function as specified in H.1.5.
- The parameter keydatalen shall have the same integer value as the key size parameter keylen as specified in H.1.2.
- Each entity shall use the challenge domain parameters as specified in H.1.8.
- All octets shall be represented as specified in Clause F.2.

H.4.2 Asymmetric-key key agreement scheme

The asymmetric-key key agreement scheme used in this standard shall be the full MQV with key confirmation scheme as specified in ANSI X9.63:2011, 6.11, with the following instantiations:

- Each entity shall be identified as specified in Clause F.3.
- Each entity shall use the HMAC-scheme as specified in H.1.5.
- Each entity shall use the cryptographic hash function as specified in H.1.5.
- The parameter keydatalen shall have the same integer value as the key size parameter keylen as specified in H.1.2.
- The parameter SharedData shall be the empty string; parameter shareddatalen shall have the integer value 0.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.
- All elliptic curve points shall be represented as specified in H.2.3.
- All octets shall be represented as specified in Clause F.2.

H.5 Keying information schemes

H.5.1 Implicit certificate scheme

The implicit certificate scheme used in this standard shall be the ECQV implicit certificate scheme as specified in SEC 4, with the following instantiations:

- Each entity shall be identified as specified in Clause F.3.
- Each entity shall use the cryptographic hash function as specified in H.1.5.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.

- All elliptic curve points shall be represented as specified in H.2.3.
- All implicit certificates shall be represented as specified in H.3.2.
- The implicit certificate infrastructure shall be one of the schemes as specified in H.3.2.
- All octets shall be represented as specified in Clause F.2.

H.5.2 Manual certificate scheme

The manual certificate scheme used in this standard shall be the manual certificate scheme as specified in Clause H.10, with the following instantiations:

- Each entity shall be identified as specified in Clause F.3.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.
- All elliptic curve points shall be represented as specified in H.2.3.
- All manual certificates shall be represented as specified in H.3.2.
- The manual certificate infrastructure shall be one of the schemes as specified in H.3.2.
- All octets shall be represented as specified in Clause F.2.

H.6 Challenge domain parameter generation and validation

H.6.1 Overview

Challenge domain parameters impose constraints on the size(s) of bit challenges that a scheme expects. As such, this determine a bound on the entropy of challenges and, thereby, on the security of the cryptographic schemes in which these challenges are used. In most schemes, the challenge domain parameters will be such that only challenges of a fixed size will be accepted (e.g., 128-bit challenges). However, one may define the challenge domain parameters such that challenges of varying size might be accepted. The latter is useful in contexts wherein entities that wish to engage in cryptographic schemes might have a defective or low-quality random bit generator. Allowing both entities that engage in a scheme to contribute sufficiently long inputs enables each of these to contribute sufficient entropy to the scheme at hand.

In this standard, challenge domain parameters will be shared by a number of entities using a scheme of this standard. The challenge domain parameters may be public; the security of the system does not rely on these parameters being secret.

H.6.2 Challenge domain parameter generation

Challenge domain parameters shall be generated using the following routine:

- Input: This routine does not take any input.
- Actions: The following actions are taken:
  - Choose two nonnegative integers minchallengelen and maxchallengelen, such that minchallengelen ≤ maxchallengelen.
- Output: Challenge domain parameters D = (minchallengelen, maxchallengelen).

H.6.3 Challenge domain parameter verification

Challenge domain parameters shall be verified using the following routine:

- Input: Purported set of challenge domain parameters D=(minchallengelen, maxchallengelen).
- Actions: The following checks are made:
  - Check that minchallengelen and maxchallengelen are nonnegative integers.

   - Check that minchallengelen ≤ maxchallengelen.
- Output: If any of the above verifications has failed, then output invalid and reject the challenge domain parameters. Otherwise, output valid and accept the challenge domain parameters.

H.7 Challenge validation primitive

Challenge validation refers to the process of checking the size properties of a challenge. It is used to check whether a challenge to be used by a scheme in this standard has sufficient size (e.g., messages that are too short are discarded, due to insufficient entropy).

The challenge validation primitive is used in Clause H.7 and uses the following routine:

- Input: The input of the validation transformation is a valid set of challenge domain parameters D = (minchallengelen, maxchallengelen), together with the bit string Challenge.
- Actions: The following actions are taken:
   - Compute the bit-length challengelen of the bit string Challenge.
   - Verify that challengelen ∈ [minchallengelen, maxchallengelen]. (That is, verify that the challenge has an appropriate size.)
- Output: If the above verification fails, then output invalid and reject the challenge. Otherwise, output valid and accept the challenge.

H.8 Secret key generation (SKG) primitive

The SKG primitive derives a shared secret value from a challenge owned by an entity U_1 and a challenge owned by an entity U_2 when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

- Prerequisites: The following are the prerequisites for the use of the SKG primitive:
   - Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same size, entityIdSize. Entity U_1s identifier will be denoted by the bit string U_1. Entity U_2s identifier will be denoted by the bit string U_2.
   - A specialized[13] MAC scheme shall have been chosen, with tagging transformation as specified in ANSI X9.63:2011, 5.7.1. The size in bits of the keys used by the specialized MAC scheme is denoted by macKeySize.
- Input: The SKG primitive takes as input:
   - A bit string MACKey of size macKeySize bits to be used as the key of the established specialized MAC scheme.
   - A bit string QEU_1 owned by U_1.
   - A bit string QEU_2 owned by U_2.
- Actions: The following actions are taken:
   - Form the bit string consisting of U_1's identifier, U_2's identifier, the bit string QEU_1 corresponding to U_1's challenge, and the bit string QEU_2 corresponding to U_2's challenge:
   - MacData = $U_1 \| U_2 \| QEU_1 \| QEU_2$.

---

[13] This refers to a MAC scheme wherein the MAC function has the additional property that it is also pre-image- and collision-resistant for parties knowing the key (see also remark 9.8 of Menezes et al.). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

  - Calculate the tag MacTag for MacData under the key MacKey using the tagging transformation of the established specialized MAC scheme:
  - $MacTag = MAC_{MacKey}(MacData)$.
  - If the tagging transformation outputs invalid, output invalid and stop.
  - Set Z=MacTag.
- Output: The bit string Z as the shared secret value.

H.9 Block-cipher-based cryptographic hash function

The Matyas-Meyer-Oseas hash function is a cryptographic hash function based on block-ciphers. This hash function is defined for block-ciphers with a key size that is equal to the block size, such as AES-128, and with a particular choice for the fixed initialization vector IV (which here is defined to be IV=0).

NOTE For a more general definition of the Matyas-Meyer-Oseas hash function, see *Handbook of applied cryptography*, 9.4.1, listed in the Bibliography.

The hash function is defined as follows:

- Prerequisites: The following are the prerequisites for the operation of Matyas-Meyer-Oseas hash function:
  - A block-cipher encryption function E shall have been chosen, with a key size that is equal to the block size. The Matyas-Meyer-Oseas hash function has a message digest size that is equal to the block size of the established encryption function. It operates on bit strings of size less than 2^n, where n is the block size, in octets, of the established block-cipher.
  - A fixed representation of integers as binary strings or octet strings shall have been chosen.
- Input: The input to the Matyas-Meyer-Oseas hash function is as follows:
  - A bit string M of size l bits, where $0 \leq l < 2^n$.
- Actions: The hash value shall be derived as follows:
  - Pad the message M according to the following method:
  - Right-concatenate to the message M the binary value consisting of one bit of 1 followed by k bits of 0, where k is the smallest non-negative solution to the equation

    $l+1+k \equiv 7n \pmod{8n}$.
  - Form the padded message M by right-concatenating to the resulting string the n-bit string that is equal to the binary representation of the integer l.
  - Parse the padded message M as $M_1 \| M_2 \| \ldots \| M_t$ where each message block M_i is an n-octet string.
  - The output $Hash_t$ is defined by

    $Hash_0 = 0^{8n}$; $Hash_j = E(Hash_{j-1}, M_j) \oplus M_j$ for j=1,…,t.

Here, E(K, x) is the ciphertext that results from encryption of the plaintext x, using the established block-cipher encryption function E with key K; the string 0^{8n} is the n-octet all-zero bit string.

- Output: The bit string $Hash_t$ as the hash value.

The cryptographic hash function operates on bit strength of size less than 2^n bits, where n is the block size (or key size) of the established block cipher, in octets. For example, the Matyas-Meyer-Oseas hash function with AES-128 operates on bit strings of size less than 2^{16} bits. It is assumed that all hash function calls are on bit strings of size less than 2^n bits. Any scheme attempting to call the hash function on a bit string exceeding 2^n bits shall output invalid and stop.

H.10 Elliptic curve cryptography manual certificate scheme

H.10.1 Overview

A manual certificate scheme based on elliptic curve cryptography (ECC) that is used in this standard is described.

The manual certificate scheme is used by three entities: a certificate authority CA, a certificate requester U, and a certificate processor V, where U wishes to obtain a manual certificate from CA in order to convey U's associated public key to V.

The manual certificate scheme is described in terms of a certificate generation transformation and a certificate processing transformation. CA, U, and V use these schemes when they wish to communicate.

Prior to use of the scheme, U, V, and CA agree on the parameters with which the scheme shall be used. In particular, this includes U and V obtaining an authentic copy of CA's unique identifier.

CA executes the manual certificate generation transformation to compute an elliptic curve public-key pair for U and a manual certificate MC for this public key provided by CA. V executes the manual certificate processing transformation, to obtain U's purported static public key from U's purported manual certificate MC presented to V.

The manual certificate generation transformation yields a public-key pair and a certificate for this public key. This public-key pair shall be communicated to the purported holder in a secure and authentic way. The mechanism by which this public-key pair is communicated is outside the scope of this standard.

The manual certificate processing transformation yields a static public key (and associated keying information) purportedly bound to the claimed holder; evidence that this public key is genuinely bound to this entity can, however, not be corroborated via processing of the manual certificate. Thus, with manual certificates, the binding of an entity and its public or private key cannot be verified, although one may obtain evidence that some entity that claims to be bound to the public key has indeed access to the corresponding private key, during cryptographic usage of the public key (e.g., via execution of an authenticated key agreement scheme or a signing transformation involving this public-key pair).

The manual certificate generation transformation is specified in H.10.2 and the manual certificate processing transformation is specified in H.10.3.

The prerequisites for the use of the scheme are:

- An infrastructure shall have been established for the operation of the scheme, including a certificate format, certificate processing rules, and unique identifiers. For an example of such an infrastructure, see IETF RFC 3280.
- Each entity has an authentic copy of the system's elliptic curve domain parameters D=(p,a,b,G,n,h) or D=(m,f(x),a,b,G,n,h). These parameters shall have been generated using the parameter generation primitives in SEC 1:2009, 3.1.1.1 or 3.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in SEC1:2009, 3.1.1.2 or 3.1.2.2.
- Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same size, entityIdSize. Entity U's identifier will be denoted by the bit string U. Entity V's identifier will be denoted by the bit string V. Entity CA's identifier will be denoted by the bit string CA.
- A cryptographic hash function Hash shall have been chosen for use with the ECQV implicit certificate generation scheme. Let hashlen denote the size in bits of the output value of this hash function.

- Each entity shall have decided how to represent elliptic curve points as octet strings (i.e., compressed form, uncompressed form, or hybrid form).
- A fixed representation of octets as binary strings shall have been chosen (e.g., most-significant-bit-first order or least-significant-bit-first order).

H.10.2 Elliptic curve cryptography manual certificate generation transformation

A CA shall execute the following transformation to provide a manual certificate for the user, U. The CA shall obtain an authentic copy of U's identifier.

- Inputs: This routine does not take any inputs.
- Ingredients: The certificate generation transformation employs the key pair generation primitive in SEC 1:2009, 3.2.1, and the manual certificate generation primitive of the established infrastructure.
- Actions: The CA shall proceed as follows:
  - The key pair generation primitive specified in SEC 1:2009, 3.2.1, shall be used to generate an ephemeral key pair (w_U, W_U) for the parameters D.
  - The elliptic curve point W_U shall be converted to the octet string WE_U as specified in SEC 1:2009, 2.3.3.
  - The octet string I_U, which is the to-be-conveyed-manual-certificate data. I_U shall be constructed to contain identification information according to the procedures of the established infrastructure and may also contain other information, such as the intended use of the public key, the serial number of the manual certificate, and the validity period of the manual certificate. The exact form of I_U depends on the manual certificate format specified during the setup procedure.
  - The octet string MC_U, which is U's manual certificate, shall be constructed according to the procedures of the established infrastructure. MC_U shall contain the octet strings I_U and WEU encoded in a reversible manner. The exact form of MC_U depends on the manual certificate format specified during the setup procedure.
- Output: MC_U, which shall serve as U's manual certificate provided by CA.

H.10.3 Elliptic curve cryptography manual certificate processing transformation

V shall execute the following transformation to obtain U's purported static public key from U's purported manual certificate provided by CA. V shall obtain an authentic copy of U's and CA's identifier.

- Input: U's purported manual certificate MC_U provided by CA.
- Ingredients: The manual certificate processing transformation employs the public key validation primitive in SEC 1:2009, 3.2.2, and the manual certificate validation primitive of the established infrastructure.
- Actions: V proceeds as follows:
  - Verify the content of MCU according to the established infrastructure. This includes verifying the contents of the certificate, such as the subject's name and the validity period. If the subject's name is not U, output invalid and stop.
  - Derive I_U from MCU, according to the manual certificate format specified during the setup procedure.
  - Derive CA's identifier from I_U, according to the certificate format specified during the setup procedure. If CA's identifier is unknown to V, output invalid and stop.
  - Derive WE_U from MCU, according to the manual certificate format specified during the setup procedure.
  - Convert the octet string WE_U to the elliptic curve point W_U as specified in SEC 1:2009, 2.3.4
  - Verify that W_U is a valid key for the parameters D as specified in SEC 1:2009, 3.2.2. If the validation primitive rejects the key, output invalid and stop.

- Output: If any of the above verifications has failed, then output invalid and stop; otherwise, output valid and accept W_U as U's purported static public key. (V may accept W_U as U's genuine static public key provided U evidences knowledge to V of the corresponding private key w_U and provided V accepts U to be the only party that may have access to this private key.)

Annex I
(informative)

Definition templates

I.1 Object type template

It is recommended that standard objects and their associated standard object identifiers be identified in a table for quick reference, as shown in Table I.1. This indicates the information needed to define standard object types defined by this standard.

Table I.1 – Table of standard object types

Defining organization:			
Standard object type name **(not expected to be transmitted, size not specified – check DD limits)**	**Standard object type identifier (non-negative)**	**Standard object identifier (non-negative), if applicable** **Used for mandatory objects with exactly one instance per device**	**Object description** **(not expected to be transmitted, size not specified – check DD limits)**
…	…	…	…

Elements of the table include:

- Standard object type name defines the name of the object.
- Standard object type identifier is the standard non-negative numeric identifier of the object type; uniquely identifies this object type.
- Standard object identifier, for standard object types that are required by a device and that may only be instantiated once, represents the standard non-negative numeric identifier for the object instance. This identifier is common to all devices. If 7 bits do not suffice, the high-order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Object description is a description of the purpose and intent of this object.

I.2 Standard object attributes template

The template shown in Table I.2 indicates the information needed to define the attributes of a standard object.

Table I.2 – Template for standard object attributes

<table>
<tr><th colspan="5">Standard object type name:</th></tr>
<tr><th colspan="5">Standard object type identifier:</th></tr>
<tr><th colspan="5">Defining organization:</th></tr>
<tr><th>Attribute name</th><th>Attribute identifier</th><th>Attribute description</th><th>Attribute data information</th><th>Description of behavior of attribute</th></tr>
<tr><td rowspan="3">ObjectIdentifier</td><td rowspan="3">Key identifier</td><td rowspan="3">Unique identifier for the object</td><td>Type: Unsigned16.</td><td rowspan="3">N/A</td></tr>
<tr><td>Classification: Static</td></tr>
<tr><td>Valid range: 1..32 767</td></tr>
<tr><td rowspan="5">…</td><td rowspan="5">…</td><td rowspan="5">…</td><td>Type: …</td><td rowspan="5">…</td></tr>
<tr><td>Classification: …</td></tr>
<tr><td>Accessibility: …</td></tr>
<tr><td>Default value: …</td></tr>
<tr><td>Valid range: …</td></tr>
<tr><td>Reserved for future use</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

Elements of the table include:

- Standard object type name defines the name of the object type.
- Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this object type. The value of this identifier fits into at most two octets.
- Defining organization is the organization defining this object (e.g., base standard, standard defined extension to the base standard object, industry specific profile (and which industry), special interest group (and which interest group)), or device vendor.
- Attribute name defines the name of the attribute.
- Attribute ID is the standard numeric identifier of the attribute. All attributes of an object are uniquely identified. If 7 bits does not suffice, the high-order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Description is the description of the attribute.
- Type is the data type of the attribute. If the data may vary in size (such as for a variable size OctetString or a variable size VisibleString), then the maximum number of octets of data is indicated.
- Classification is the data classification (constant, static, static-volatile, dynamic, non-cacheable) of the attribute.
- Accessibility is how the attribute may be accessed remotely (e.g., Read only, or Read/write)
- Initial default value specifies the initial default value.
- Valid value set specifies the valid set of values for this attribute.

I.3 Standard object methods

The template shown in Table I.3 indicates the information needed to describe the methods of a standard object.

Table I.3 – Template for standard object methods

Standard object type name:				
Standard object type identifier:				
Defining organization:				
Method name	**Method ID**	**Method description**		
<name of method>	**Input arguments**			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	…	…	…	…
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	…	…	…	…

Elements of the table include:

- Standard object type name defines the name of the object.
- Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this object type. The value of this identifier fits into at most two octets.
- Defining organization is the organization defining this object (e.g., base standard, standard defined extension to the base standard object, industry specific profile (and which industry), special interest group (and which interest group)).
- Method name is the name of the method.
- Method ID is the numeric identification of the method. All methods of an object will have unique method identifiers. If 7 bits does not suffice, the high-order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Method description is the description of the method.
- List of input parameters and their data types is a list of input parameters, their type and size (if not explicitly discernable from the type), and a description of use (how they are used when sent on a method invoke). These should be listed in order of transmission.

NOTE 1 For simplicity, all parameters are specified. If there are situations where parameters vary, separate methods are appropriate to accommodate each class of variance.

- List of output parameters and their data types is a list of output parameters, their type and size (if not explicitly discernable from the type), and a description of use (how they are used when sent on a method invoke). These should be listed in order of transmission.

NOTE 2 See NOTE 1.

- Description of behavior describes the behavior of the object when this method is invoked.

I.4 Standard object alert reporting template

The template shown in Table I.4 indicates the information needed to describe the alert reporting behavior of a standard object.

Table I.4 – Template for standard object alert reporting

Standard object type name(s):					
Standard object type identifier:					
Defining organization:					
Description of the alert:					
Alert class (Enumerated: alarm or event)	**Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)**	**Alert type (Enumerated: based on alert category)**	**Alert priority**	**Value data type**	**Description of value included with alert**
<name of alert>	…	…	…	Type: …	…
				Default value: …	…
				Valid range: …	…

Elements of the table include:

- Standard object type name defines the name of the object.
- Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this(these) object type(s) that may report this alert. The value of this identifier fits into at most two octets.
- Defining organization is the organization defining this object (e.g., base standard, industry specific profile (and which industry), special interest group (and which interest group)).
- Description of the alert describes the semantic meaning of the alert.
- Alert class indicates if this is an event (stateless) or alarm (state-oriented) type of alert.
- Alert category indicates if this is a device related (e.g., a device specific diagnostic), communication related, security related, or process related alert. Only one category applies. Selection of the best fit for an alert may need to be discussed in order to be best established.
- Alert type is dependent on the alert category. See the alert reporting model in 12.8 for further details.
- Alert priority is the priority of the alert.
- Value and size is the size and value included in the alert report.
- Description of value included in alert report is the description of the value, if a value is included in the alert report (e.g., for a process alarm that is a high alarm, this may be the process variable (PV)).
- Accessibility defines if the attribute is readable, writeable, or both.
- Initial default value indicates the initial default value of the attribute.
- Description of value set describes the set of values that may be taken on by this attribute.
- Description of behavior describes the behavior of this attribute (e.g., when a particular value is written, or error conditions). Restrictions on use (e.g., operators should not write to this attribute) may be noted here.

I.5 Data structure definition

The template for describing data structures that are used to define special data types is given in Table I.5.

Table I.5 – Template for data structures

Standard data type name:		
Defining organization:		
Element name	**Element identifier**	**Element scalar type**
…	…	Type: …
		Size: …
		Classification: …
		Accessibility: …
		Default value: …
		Valid range: …

Annex J
(informative)

Operations on attributes

J.1 Operations on attributes

J.1.1 General

Attribute classification and accessibility dictate the operations permitted on a given attribute. Attribute classification and accessibility are described in 12.6.

J.1.2 Attribute classification

For a discussion of attribute classification, see 12.6.3.

J.1.3 Retrieving, setting, and resetting attributes

J.1.3.1 General

Attributes defined in the management objects can be accessed using the standard ASL-provided read or write services. Such operations enable configuration of the layers, as well as monitoring of their status. They can be used to retrieve, set / modify, and reset the values of attributes. The service primitives for these services, as well as the enumerated service feedback codes, are given in Clause 12.

Attributes can be reset using the write service by writing the default value to the relevant attribute. If a reset attribute is defined for a management object, it can be used to reset all the attributes in that management object that belong to certain classes of attributes.

More complex methods may be defined if necessary, but only if the equivalent results cannot be achieved using the more direct read / write services. A complex method may be warranted, for example, to replace a sequence of communication transactions in order to save energy. A complex method may also be warranted when synchronization issues may result if individual actions are used, rather than an atomic transaction set.

J.1.3.2 Scheduled operations to enable synchronized cutover

The generic method template Scheduled_Write provided in Table J.1 can be used to define a method for writing a value to an attribute at a scheduled TAI time. It can also be used to reset an attribute to its default value at a scheduled TAI time.

Table J.1 – Scheduled_Write method template

Method name	Method ID	Method description		
Scheduled_Write	<given in management object definition>	Method to write a value to an indicated attribute at an indicated TAI time		
	Input Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the value should be written to the attribute
	3	Value	Data type: <given in management object definition>	The value that needs to be written to the attribute
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

J.1.4 Retrieving and setting structured attributes

The generic method templates Read_Row and Write_Row given in Table J.2 and Table J.3 can be used for defining methods that retrieve and set / modify the values of structured attributes. When the structured attribute is visualized as an information table, these methods allow access to a particular row based on one or more unique index field values. It is assumed that each table has at least one unique index field. The index field may either be a single element or the concatenation of a few elements in the row.

The input argument Scheduled_TAI_Time in the Write_Row method template allows scheduled operation for a particular row of the structured attribute. A value of 0 for this argument indicates an immediate write operation.

Table J.2 – Read_Row method template

Method name	Method ID	Method description		
Read_Row	<given in management object definition>	Method to read the value of a single row of a structured attribute whose data is visualized as an information table		
	Input Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	n+1	Index_n	Data type of n^{th} index field of the structured attribute <given in management object definition>	The n^{th} index field in the structured attribute to access a particular row
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Data_Value	Data type: <given in management object definition>	An octet string that contains the data value

Table J.3 – Write_Row method template

Method name	Method ID	Method description		
Write_Row	<given in management object definition>	Method to set / modify the value of a single row of a structured attribute whose data is visualized as an information table		
	Input Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_ Time	Data type: TAITimeRounded	TAI time at which the value should be written to the row of the structured attribute
	3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	N+2	Index_n	Data type of n^{th} index field of the structured attribute <given in management object definition>	The n^{th} index field in the structured attribute to access a particular row
	N+3	Data_Value	Data type: <given in management object definition>	An octet string that contains the data value
	Output Arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

A method based on the Write_Row template can also be used to create a new row in the structured attribute if the index field(s) provided in the input argument(s) does(do) not exist.

J.1.5 Resetting structured attribute values

For a structured attribute, the generic method template Reset_Row given in Table J.4 can be used for defining methods that reset / clear certain values in the structured attribute. The input argument Scheduled_TAI_Time in this method allows a scheduled reset operation. A value of 0 for this argument indicates an immediate reset operation.

Table J.4 – Reset_Row method template

Method name	**Method ID**	**Method description**		
Reset_Row	<given in management object definition>	Method to reset a single row of a structured attribute whose data is visualized as an information table		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the row of the structured attribute should be reset
	3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	n+2	Index_n	Data type of n^{th} index field of the structured attribute <given in management object definition>	The n^{th} index field in the structured attribute to access a particular row
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

J.1.6 Deleting structured attribute values

The generic method template Delete_Row described in Table J.5 can be used for defining methods that delete the values of structured attributes. The input argument Scheduled_TAI_Time in this method allows a scheduled delete operation. A value of 0 for this argument indicates an immediate delete operation.

Table J.5 – Delete_Row method template

Method name	Method ID	Method description		
Delete_Row	<given in management object definition>	Method to delete a single row of a structured attribute whose data is visualized as an information table		
	Input arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	1	Attribute_ID	Data type: Unsigned16 given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the row of the structured attribute should be deleted
	3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	n+2	Index_n	Data type of nth index field of the structured attribute <given in management object definition>	The nth index field in the structured attribute to access a particular row
	Output arguments			
	Argument number	**Argument name**	**Argument type (data type and size)**	**Argument description**
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

J.2 Synchronized cutover

A synchronized cutover capability is needed for some attributes and structured attributes that represent management information. For such an attribute, updates for the attribute value may be scheduled by indicating the TAI cutover time information; this operation may be accomplished by using one of the methods defined above. Such updates are sent to the management object for which this attribute is defined. The management object immediately validates whether the cutover is feasible, and, if feasible, arranges for the cutover to occur on schedule.

Annex K
(normative)

Standard object types

Annex K specifies the standard object types defined by this standard. Each object type has three pieces of information to identify it:

- a corresponding standard object type identifier that identifies the standard defined base object type (example: analog input);
- a corresponding object standard subtype identifier that identifies the standard subtype of a standard base type (example: analog input specialized for temperature); and
- a corresponding vendor subtype identifier that identifies a vendor specific subtype of either a standard base object or standard subtype.

Standard base objects shall have their object subtype identifier value equal to zero (0) and their vendor subtype identifier equal to zero (0).

A newer version of this standard that finds it necessary to extend the base object type definition of this standard may maintain the standard object identifier value and the subtype value of zero (0). This is permitted since the DMO contains an attribute to represent the version of the standard in use by the device, which can be thus be used to establish the base object type structure in use.

IEC62734 industry profiles may define a standard object subtype as a standard object. Doing so creates a standard profile subtype. This standard provides a range of 1..255 to represent all such standard object subtype across all profiles.

Vendors may also subtype either a standard base object or a standard subtype object. This standard provides a range of 1..255 for vendor specific subtyping.

Object subtyping occurs when:

- one or more attributes is/are added to the base type;
- one or more methods is/are added to the base type;
- one or more alerts is/are added to the base type; or
- any combination of the above.

Examples of object identification with subtyping follow:

- Analog input standard base object:
  - object type identifier = 99,
  - object standard subtype identifier = 0,
  - vendor subtype identifier = 0.
- Analog input temperature subtype object:
  - object type identifier = 99,
  - object standard subtype identifier = (this standard defines (this standard’s profile team), range 1..255),
  - vendor subtype identifier = 0.
- Vendor-specific analog input object:
  - object standard type identifier = 99,
  - object standard subtype identifier = 0,

  – vendor-specific subtype identifier = (vendor defines, range 1..255).

- Vendor-specific analog input temperature object:
  – object standard type identifier = 99,
  – object standard subtype identifier = n,
  – vendor specific subtype identifier = (vendor defines, range 1..255).

Table K.1 specifies standard object types.

Table K.1 – Standard object types

Object type	Standard object type identifier (1 octet)	Standard object industry subtype identifier (1 octet)	Object identifier restrictions
Object types available to all UAPs			
Null object	0	0	Reserved
UAP management object	1	0	This object is required for all UAPs, but is not required for the DMAP.
AlertReceiving object	2	0	
UploadDownload object	3	0	
Concentrator object	4	0	
Dispersion object	5	0	
Tunnel object	6	0	
Interface object	7	0	
Reserved for use by this standard for standard UAP objects	8..50	0	Reserved for future standard object definitions for profile independent objects
Reserved for use by this standard	51..95	0	Industry-specific types
Process control industry object types			
Analog input	99	0	Analog input
Analog output	98	0	Analog output
Binary input	97	0	Binary input
Binary output	96	0	Binary output
DMAP object types			
DMAP: Device management object (DMO)	127	0	This object facilitates the management of the device’s general device-wide functions
DMAP: Alert reporting management object (ARMO)	126	0	This object facilitates the management of the device’s alert reporting functions
DMAP: Device security management object (DSMO)	125	0	This object facilitates the management of the device’s security functions
DMAP: DL management object (DLMO)	124	0	This object facilitates the management of the device’s DL
DMAP: NL management object (NLMO)	123	0	This object facilitates the management of the device’s NL

Object type	Standard object type identifier (1 octet)	Standard object industry subtype identifier (1 octet)	Object identifier restrictions
DMAP: TL management object (TLMO)	122	0	This object facilitates the management of the device's TL
DMAP: Application sublayer management object (ASLMO)	121	0	This object facilitates the management of the device's application sublayer
DMAP: Device provisioning object (DPO)	120	0	This object facilitates the provisioning of the device before it joins the network
DMAP: Health reports concentrator object (HRCO)	128	0	This object facilitates the periodic publication of device health reports to the system manager
Standard management objects	119..114	0	
System time service object (STSO)	100	0	This object facilitates the management of system-wide time information
Directory service object (DSO)	101	0	This object facilitates the management of addresses for all existing devices in the network
System configuration object (SCO)	102	0	This object facilitates the configuration of the system including contract establishment, modification and termination
Device management service object (DMSO)	103	0	This object facilitates device joining, device leaving, and device configuration
System monitoring object (SMO)	104	0	This object facilitates the monitoring of system performance
Proxy security management object (PSMO)	105	0	This object acts as a proxy for the security manager
Device provisioning service object (DPSO)	106	0	This object facilitates device provisioning
Standard system management objects	107..113	0	Reserved for standard management object type definitions. See Clause 6 for details
Vendor-defined types			
Vendor-defined objects	129..255	0	Reserved for use by implementers

Table K.2 specifies standard object instances.

19050 **Table K.2 – Standard object instances**

Object type	Standard object type identifier (1 octet)	Standard object industry subtype identifier (1 octet)	Standard object identifier (1 octet)	Object identifier restrictions
Object types available to all UAPs				
Null object	0	0	0	Reserved
UAP management object	1	0	1	This object is required for all UAPs, but is not required for the DMAP
UploadDownload object	3	0	2	For UAP upgrade use only
Process control industry object types				
N/A				
DMAP object types				
DMAP: Device management object (DMO)	127	0	1	This object facilitates the management of the device's general device-wide functions
DMAP: Alert reporting management object (ARMO)	126	0	2	This object facilitates the management of the device's alert reporting functions
DMAP: Device security management object (DSMO)	125	0	3	This object facilitates the management of the device's security functions
DMAP: DL management object (DLMO)	124	0	4	This object facilitates the management of the device's DL
DMAP: NL management object (NLMO)	123	0	5	This object facilitates the management of the device's NL
DMAP: TL management object (TLMO)	122	0	6	This object facilitates the management of the device's TL
DMAP: Application sublayer management object (ASLMO)	121	0	7	This object facilitates the management of the device's application sublayer
DMAP: Upload/download object (UDO)	3	0	8	This object facilitates the management of the device's upload/download functions
DMAP: Device provisioning object (DPO)	120	0	9	This object facilitates the provisioning of the device before it joins the network

Object type	Standard object type identifier (1 octet)	Standard object industry subtype identifier (1 octet)	Standard object identifier (1 octet)	Object identifier restrictions
DMAP: Health reports concentrator object (HRCO)	128	0	10	This object facilitates the periodic publication of device health reports to the system manager
System management AP standard types				
System time service object (STSO)	100	0	1	This object facilitates the management of system-wide time information
Directory service object (DSO)	101	0	2	This object facilitates the management of addresses for all existing devices in the network
System configuration object (SCO)	102	0	3	This object facilitates the configuration of the system including contract establishment, modification and termination
Device management service object (DMSO)	103	0	4	This object facilitates device joining, device leaving, and device configuration
System monitoring object (SMO)	104	0	5	This object facilitates the monitoring of system performance
Proxy security management object (PSMO)	105	0	6	This object acts as a proxy for the security manager
Upload/download object (UDO)	3	0	7	This object facilitates downloading firmware/data to devices and uploading data from devices
Alert-receiving object (ARO)	2	0	8	This object receives all the alerts destined for the system manager
Device provisioning service object (DPSO)	106	0	9	This object facilitates device provisioning
Health reports concentrator object (HRCO)	4	0	10	This object facilitates the periodic publication of device health reports to the system manager
Vendor-defined types				
…	…	…	…	…

Annex L
(informative)

Standard data types

Table L.1 specifies the standard data type identifiers for the standard data types. Standard data types are defined for constructs that are accessible using ASL services, such as read and write.

NOTE 1 It is possible for data structures to not be directly accessible using ASL services, e.g., a data structure that is used as a parameter of a method, but which is not exposed as an ASL-accessible object attribute.

NOTE 2 Many of the type identifiers in this table are based on type identifiers used in an existing IEC standard.

Table L.1 – Standard data types

Data type	Type identifier (Unsigned16)	Size (octets)
Reserved types		
Invalid (type not specified)	0	0
AP standard data structure types		
Communication association endpoint	468	See Table 265
ObjectAttributeIndexAndSize	469	See Table 264
Communication contract data	470	See Table 266
Alert communication endpoint	471	See Table 267
ObjectIDandType	472	See Table 271
Unscheduled correspondent	473	See Table 272
Process control types		
Process control value and status for analog value	65	See Table 300
Process control value and status for binary value	66	See Table 301
Process control scaling	68	See Table 304
Process control mode	69	See Table 302
Alert descriptor types		
Process control alarm report descriptor for analog with single reference condition	498	See Table 270
Alert report descriptor (also used for process control binary alarms)	499	See Table 269

Data type	Type identifier (Unsigned16)	Size (octets)
General communication / management types		
Contract_Data	401	See Table 30
Address_Translation_Row	402	See Table 14
New_Device_Contract_Response	405	See Table 31
Metadata_attribute	406	See Table 2
Security_Sym_Join_Request	410	See Table 62
Security_Sym_Join_Response	411	See Table 63
Security_Sym_Confirm	412	See Table 66
Security_Pub_Join_Request	415	See Table 70
Security_Pub_Join_Response	416	See Table 70
Security_Pub_Confirm_Request	417	See Table 72
Security_Pub_Confirm_Response	418	See Table 72
Security_New_Session_Request	420	See Table 81
Security_New_Session_Response	421	See Table 82
Security_Key_and_Policies	422	See Table 84
Security_Key_Update_Status	423	See Table 85
DPSOWhiteListTbl	440	See Table 372
NLContractTbl	441	See Table 207
NLRouteTbl	442	See Table 208
NLATTbl	443	See Table 209

Annex M
(normative)

Identification of tunneled legacy fieldbus protocols

Table M.1 lists the Unsigned8 protocol identification values currently defined to tunnel legacy wired fieldbus protocols via the tunnel object.

Table M.1 – Identification of tunneled legacy fieldbus protocols

Value	Protocol
0	None (required)
1	HART (see IEC 61158)
2	FF-H1 (see IEC 61158)
3	Modbus/RTU (see IEC 61158)
4	PROFIBUS PA (see IEC 61158)
5	CIP (see IEC 61158)
6..255	<reserved>

NOTE These protocol identification values have been isolated into Annex M in order to facilitate ease of maintenance.

Value 0 for None should be preserved or tunnel functionality will be impaired.

Annex N
(informative)

Tunneling and native object mapping

N.1 Overview

Tunneling involves the exchange of PDUs of one protocol by using a second protocol. Most often these PDUs are application PDUs, but lower layer PDUs may also be exchanged. The PDU is encapsulated in the second protocol at an origination node and sent through the network to a termination node. With tunneling, what goes in one end comes out the other end, no more, no less.

Foreign protocol application communication (FPAC) is a more sophisticated PDU exchange mechanism. It involves the usage of additional mechanisms, including caching, compression, address translation, and proxy. As far as the application is concerned, the same PDUs are still exchanged between the origination node and the termination node as with tunneling. The difference is that the additional mechanisms act to improve energy efficiency and host system responsiveness.

N.2 Tunneling

Tunneling carries messages verbatim between endpoints of a tunnel. This standard provides tunneling that uses un-buffered client/server exchange of foreign PDUs between exactly two pre-configured tunnel endpoints. No interpretation of the PDU content is required. For most legacy protocols, this method will not be energy efficient, and some protocols may not operate properly due to variable or lengthy response times associated with sleeping devices. Regardless of the shortcomings, in many cases this will be the most expedient method for adapting existing devices and systems to this standard.

An extension of tunneling interprets the addressing within foreign PDUs to allow dynamic foreign PDU exchange with multiple endpoints.

N.3 Foreign protocol application communication

Tunneling is not an appropriate mechanism for most low-power wireless link applications. It is usually necessary to minimize PhPDU overhead and the number of transactions in order to conserve energy stored in batteries or to operate within the power budget of scavenging and harvesting techniques. In addition, foreign protocols often have a need for fast response in order to avoid built-in timeouts. Devices in low-power wireless operation are most often in a sleep mode and thus cannot respond immediately.

FPAC increases energy efficiency and addresses potential timing issues by using change-of-state transfer and caching to eliminate redundant transfer. Improvements in energy efficiency and performance are achieved by caching the information in the gateway, transferring information to the gateway only when it changes, and providing a heartbeat mechanism for integrity. This minimizes transfers initiated by the end devices (i.e., periodic publications), as well as minimizing transfers initiated by the foreign communication link (i.e., multi-master access through the gateway). In addition, this method can address foreign protocol timing requirements. Compared to tunneling, additional effort is necessary to translate the foreign protocol.

This standard provides support for FPAC that minimizes PhPDU overhead using a combination of techniques:

- Encapsulation is limited to a single encapsulation. Protocol translators provide additional encapsulation across foreign links as necessary.
- Encapsulation is achieved through configuration agreement by carrying the foreign protocol within the protocol defined by this standard, rather than by carrying additional protocol headers. Mapping occurs as follows:
  - Transport supported relationships (publish/subscribe and client/server).
  - Foreign addresses and native addresses.
  - Size fields and integrity fields.
- This standard provides a native application service format for message exchange. Foreign protocols have their own service formats and message exchange protocols. The tunnel object allows the transfer of foreign APDUs with no extraneous overhead imposed by the native application service format.

This standard provides support for FPAC that minimizes transaction overhead using the following techniques:

- Distributed buffer caching mechanisms to minimize redundant transfer of unchanged data between gateways and end devices.
- Periodic, change-of-state (CoSt), and aperiodic transfer mechanisms.
- Watchdog timers to monitor endpoint and communication channel availability and assure data quality.

This standard provides support for FPAC that improves foreign protocol device access timing performance (and minimizes unnecessary transactions) by the provision of buffered device information through a gateway high side interface.

NOTE Change of state (CoSt) is distinct from class of service (CoS) as defined by IEEE 802.1Q.

N.4 Native object mapping

This standard supports a native object format and messaging services. Automation-specific objects can be used to support protocol translation by using these objects to perform a mapping of the foreign protocol into these objects and their messaging. Compared to the tunneling and FPAC methods, additional effort is necessary to translate the foreign protocol.

N.5 Tunneling and native object mapping tradeoffs

Native object mapping has a unique advantage in the ability to build a single standard-compliant end device for use with multiple foreign protocols. This is especially attractive for new devices.

Tunneling and FPAC have an advantage in simplicity for adapting wired automation devices through an adapter. Little, if any, translation may be required on either end.

Using tunneling in conjunction with native object mapping is also useful. This allows common legacy functions to use native object mapping, while rarely used functions can be tunneled. This can lead to less total effort in protocol translation.

Annex O
(informative)

Generic protocol translation

O.1 Overview

This standard does not include protocol translators. It does include features to support the construction of protocol translators (generally located within gateways) for common fieldbus protocols, where such a translator would also be sensitive to the constraints of low-power wireless automation networks. Since specific protocol translators are not defined in this standard, all support for protocol translation is thus generic.

Annex O provides an example of how to use the tunnel object and a conceptual GIAP to support common protocol translation interactions. The tunnel object includes the normative features to support protocol translation.

Specific protocol translators (for specific fieldbuses) could include Annex O, potentially in modified form. They could also use a different approach. Such choices are not specified by this standard.

O.2 Publish

A portion of a generic gateway is depicted in Figure O.1, which relates to the usage of publication. A generic protocol translator interacts with a gateway UAP through the GIAP. The gateway UAP uses the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.

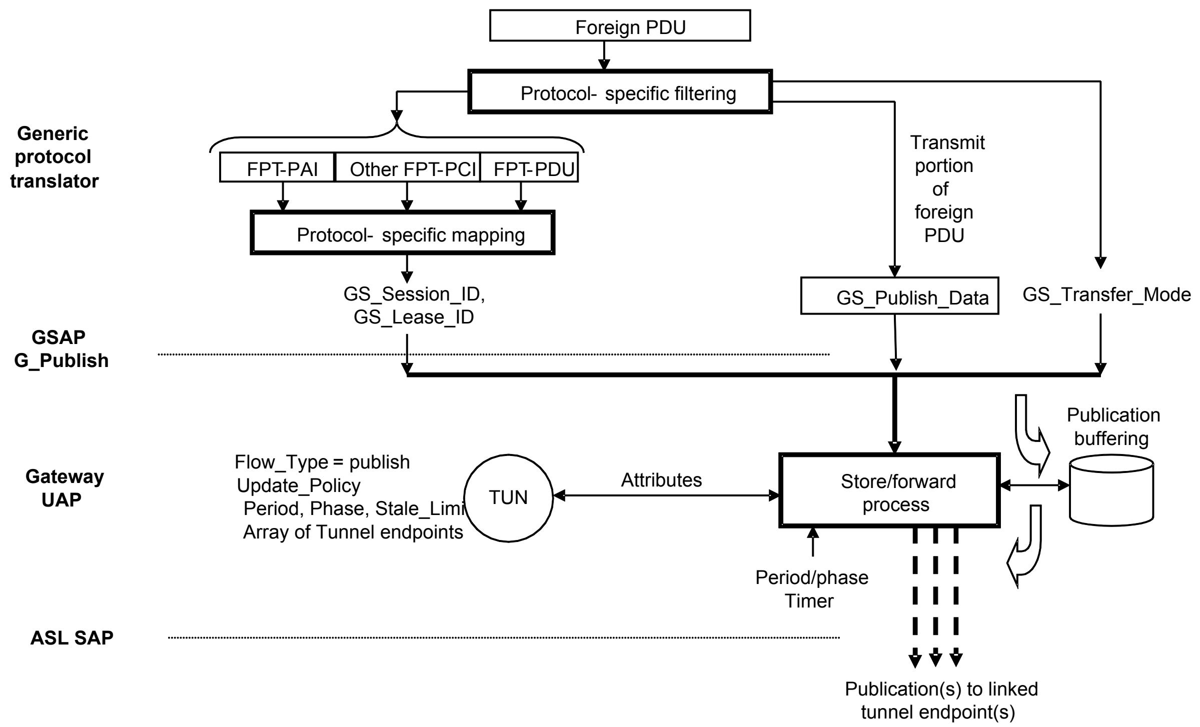


Figure O.1 – Generic protocol translation publish diagram

A foreign PDU is received by the protocol translator, and protocol-specific filtering is applied. Depending on the protocol, a combination of FPT-PAI, other FPT-PCI, and FPT-PDU may be necessary in order to determine the proper GS_Session_ID and GS_Lease_ID for GIAP usage in linking to a subscriber. The protocol-specific filtering determines the portion of the foreign PDU that needs to be transmitted (GS_Publish_Data) and foreign protocol-specific transport parameters such as priority (GS_Transfer_Mode). The parameters are then used to invoke GIAP services.

The GS_Session_ID and GS_Lease_ID are used by the gateway UAP to identify the TUN object and to retrieve the necessary parameters for store and forward processing decisions. GS_Publish_Data is buffered and forwarded at the appropriate time based on the Update_Policy, the period, the phase, the Stale_Limit, and the prior and current data content. Store and forward decisions are also driven by timer events based on the period and the phase. The ASL SAP is used to forward any messages.

A publication may be sent to one or more endpoints depending on the number of elements contained by the array of tunnel endpoints.

O.3 Subscribe

A portion of a generic gateway is depicted in Figure O.2, which relates to the usage of subscription. A generic protocol translator interacts with a gateway UAP through the GIAP. The gateway UAP uses the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.

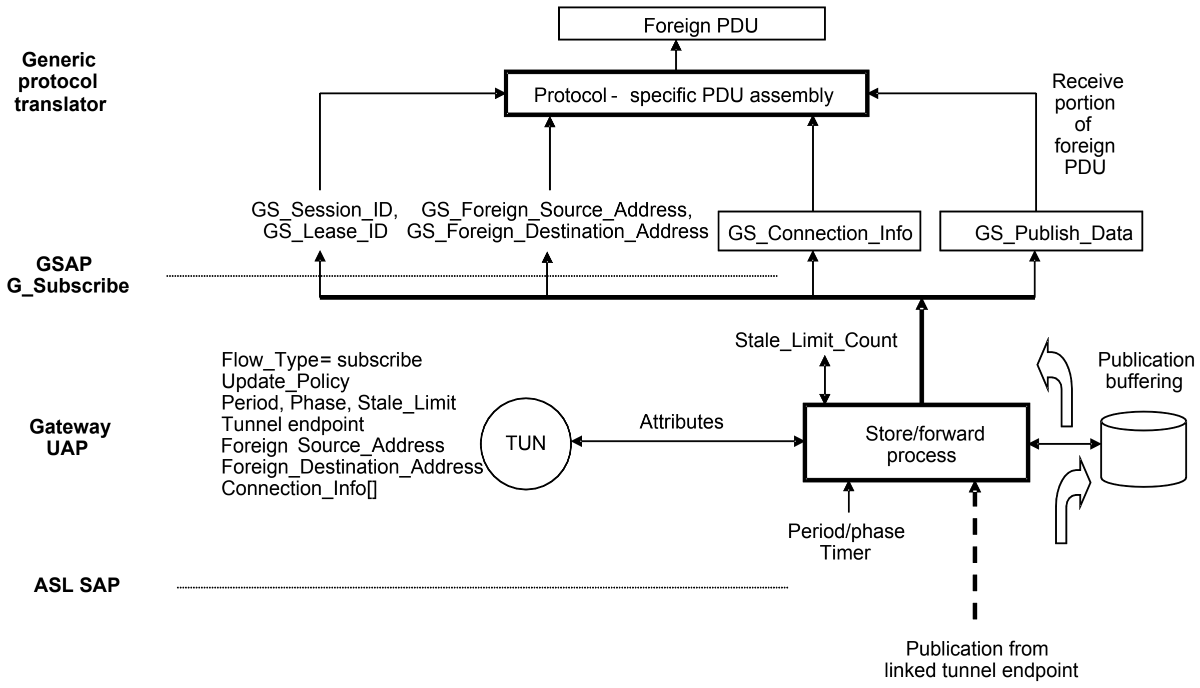


Figure O.2 – Generic protocol translation subscribe diagram

A publication APDU arrives at the gateway UAP through the ASL SAP. The addressing indicates a local TUN object that is linked to the remote publisher TUN object. The necessary attributes are retrieved from the TUN object for store and forward decisions.

Publication data includes the GS_Publish_Data from the publisher. Publication buffering is based on Update_Policy, the Period, the Phase, the Stale_Limit, and Period/Phase based timer events. Forwarding occurs to the protocol translator through the GIAP based on polled and event driven interaction with the protocol translator. The gateway UAP also stores and includes the GS_Session_ID and GS_Lease_ID for the protocol translator to identify the publication. Publication specific information may be stored locally and used to reduce unnecessary transmission of the information. This information includes addressing information

(GS_Foreign_Source_Address and GS_Foreign_Destination_Address) and connection specific information (GS_Connection_Info).

The protocol translator performs a protocol-specific assembly to generate the foreign PDU.

O.4 Client

A portion of a generic gateway is depicted in Figure O.3, which relates to the transmission of client/server tunneled messages. A generic protocol translator interacts with a gateway UAP through the GIAP. The gateway UAP uses the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.

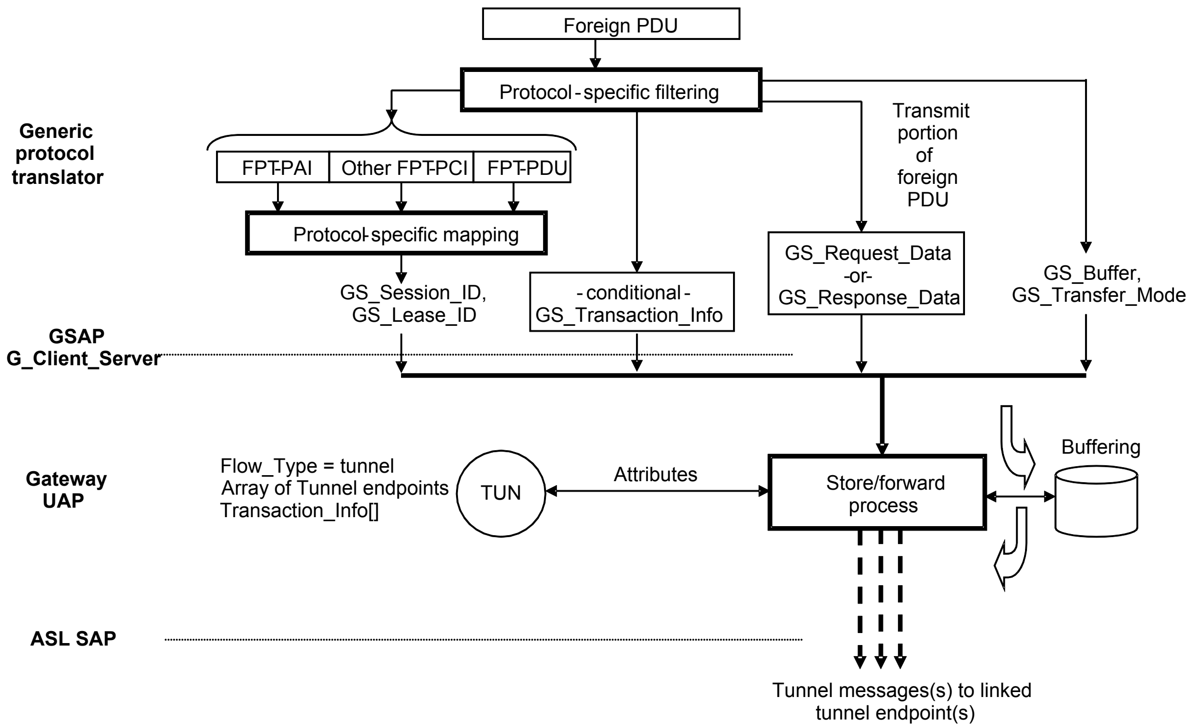


Figure O.3 – Generic protocol translation client/server transmission diagram

A foreign PDU is received by the protocol translator, and protocol-specific filtering is applied. Depending on the protocol, a combination of FPT-PAI, other FPT-PCI, and FPT-PDU may be necessary in order to determine the proper GS_Session_ID and GS_Lease_ID for GIAP usage. Protocol-specific filtering determines the portion of the foreign PDU that needs to be transmitted (GS_Request_Data or GS_Response_Data) and the appropriate transport parameters such as priority (GS_Transfer_Mode). For requests, the SDU may also specify GS_Transaction_Info that is to be returned at the GIAP when a matching response arrives. The parameters are then used to invoke GIAP services.

The GS_Session_ID and GS_Lease_ID are used by the gateway UAP to identify the TUN object and to retrieve the necessary parameters for store and forward processing decisions. The GIAP information (GS_Request_Data or GS_Response_Data) may be buffered before forwarding, depending on whether buffering is requested (GS_Buffer), depending on the prior buffer content, and depending on whether a request or response is specified. The ASL SAP is used to forward any messages.

A tunnel request message may be sent to one or more endpoints depending on the number of elements contained by the array of tunnel endpoints. A tunnel response message can be sent to a single endpoint, but multiple responses can be sent to the same endpoint over time.

O.5 Server

A portion of a generic gateway is depicted in Figure O.4, which relates to the reception of client/server tunneled messages. A generic protocol translator interacts with a gateway UAP through the GIAP. The gateway UAP uses the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.

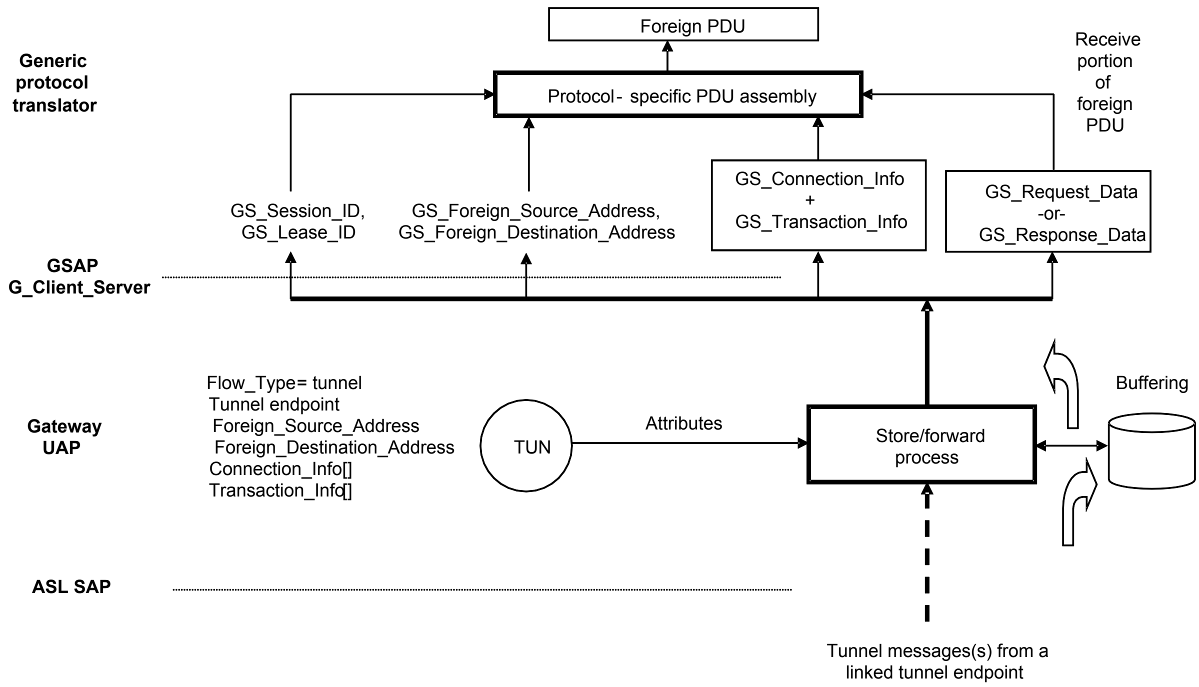


Figure O.4 – Generic protocol translation client/server reception diagram

A tunnel request or response APDU arrives at the gateway UAP through the ASL SAP. The addressing indicates a local TUN object that is linked to a remote TUN object. The necessary attributes are retrieved from the TUN for store and forward decisions.

Tunnel APDU data includes either GS_Request_Data or GS_Response_Data. Depending on the tunnel mode, the response data may be buffered to answer subsequent requests from local buffers. Forwarding occurs to the protocol translator through the GIAP based on polled and event driven interaction with the protocol translator. The gateway UAP also stores and includes the GS_Session_ID and GS_Lease_ID for the protocol translator to identify the tunnel data. Tunnel message specific information may be stored and used to reduce unnecessary duplicated transmission of the information. This includes addressing information (GS_Foreign_Source_Address and GS_Foreign_Destination_Address), connection specific information (GS_Connection_Info) and transaction specific information (GS_Transaction_Info) to be conveyed in responses.

The protocol translator performs a protocol-specific assembly to generate the foreign PDU.

Annex P
(informative)

Exemplary GIAP adaptations for this standard

P.1 General

This standard does not define functionality for a complete gateway. It does include supporting examples that allow gateway construction by the addition of a protocol translator, and a hardware interface and protocol stack for a foreign network. Annex P does not define a protocol translator; that might be a subject for future standardization.

Annex P provides an example of a conceptual interface that would be internal to a gateway – the GIAP, which is intended to be an abstraction of the underlying wireless system. In particular, it is intended to provide an abstraction for the wireless system described in this standard.

Annex P describes one way to implement the informative GIAP by using this standard's normative objects and services. It is not a complete design, but a reference to aid understanding.

Specific gateways (for specific fieldbuses) could include Annex P, thus making it normative. They could also determine a different approach that was compliant.

In this exemplary gateway, the GIAP services are implemented as a specialized UAP that uses native objects as defined in this standard.

P.2 Parameters

GS_Network_Address is the IPv6Address.

P.3 Session

The GIAP session service tracks resources and releases the resource when the session is closed or expires. Resources include communication contracts, bulk transfers in progress, buffered information, publication/subscribe/Client/server resources in objects, and alert subscriptions.

P.4 Lease

The GIAP lease service allows allocation of resources and individual release when the lease is closed or expires. Resources include communication contracts and object resources for: bulk transfer, publish/subscribe, client/server, and alerts.

A lease differs from a communication contract in that a lease allocates resources both within a gateway entity and, when needed, the resources corresponding to a related communication contract.

The specification of multiple IPv6Addresses within the GS_Network_Address_List represents a multicast group. Specifying multiple addresses will result in a simulated multicast via multiple unicast operations. Even though this is a single lease, simulated multicast requires the allocation of multiple point-to-point contracts and simultaneous management of this contract set within the gateway.

GS_Resource specifies the bulk transfer item for a lease (Destination_Port and OID). GS_Resource is also used in the linkage of matching sets of TUN objects and matching CON and DIS objects. A matched publisher and subscriber(s) specify related values in lease creation. These values, along with the GS_Network_Address_List, allow the Array of Tunnel endpoint to be filled on linked TUN objects and CON and DIS objects to be allocated and linked.

Subscriber leases specify GS_Update_Policy, GS_Period, GS_Phase, and GS_Stale_Limit.

P.5 Device list report

There is no specific adaptation information for this item.

P.6 Topology report

There is no specific adaptation information for this item.

P.7 Schedule report

There is no specific adaptation information for this item.

P.8 Device health report

There is no specific adaptation information for this item.

P.9 Neighbor health report

GS_Signal_Strength maps to ED and GS_Signal_Quality maps to LQI as defined in 9.1.15.2.

P.10 Network health report

There is no specific adaptation information for this item.

P.11 Time

There is no specific adaptation information for this item.

P.12 Client/server

P.12.1 General

The GIAP client/server service uses the TUN object or the IFO, depending on the lease establishment.

P.12.2 Native access

Where the lease establishment specifies GS_Protocol_Type = 0, the native protocol is configured through an IFO, the GS_Network_Address_List is empty, the GS_Lease_Parameters specify only GS_Transfer_Mode, which in turn specifies both priority and discard eligibility, as defined in Clause 12 for the read, write and execute services.

The payloads (GS_Request_Data and GS_Response_Data) conform to the native APDU formats and use only the ASL service types: read, write, and execute. The IFO objects in gateways transfer these payloads via the read, write, and execute services. GS_Transfer_Mode is used with each transfer to indicate the quality of service, including priority, associated with the transfer.

GS_Buffer is used to request buffered and unbuffered behavior as appropriate to the ASL service and attribute classifications.

GS_Transaction_Info is empty.

The native client/server service is used to address native objects in the gateway.

P.12.3 Foreign access

Where the lease establishment specifies a GS_Protocol_Type, a foreign protocol is configured through a TUN object. GS_Network_Address_List is supplied to establish the remote TUN endpoints. GS_Resource is used to determine whether 2-part or 4-part tunnel services apply and to match the TUN endpoints within devices. A lone client or server lease establishes a 2-part tunnel. A pair of client and server leases with the same GS_Resource establishes a 4-part tunnel. GS_Lease_Parameters supply GS_Connection_Info on Server services as appropriate for the foreign protocol and GS_Transfer_Mode in order to set default transfer quality of service and priority.

The payloads (GS_Request_Data and GS_Response_Data) conform to the foreign APDU formats, including specification of foreign service types and service-specific fields. The TUN objects in gateways transfer these payloads by using the 2-part and 4-part tunnel services. GS_Cache is used to request buffered and unbuffered behavior as appropriate to the TUN object configuration and the foreign protocol requirements. GS_Transfer_Mode is used with each transfer to indicate the quality of service, including priority, associated with the transfer.

The GS_Transfer_Mode specifies priority and discard eligibility, as defined in Clause 12 for the tunnel service.

GS_Transaction_Info is supplied on client services and returned on server services as appropriate for the foreign protocol.

P.13 Publish/subscribe

P.13.1 General

The GIAP publish/subscribe service uses the TUN object or CON and DIS objects, depending on the lease establishment.

P.13.2 Native access

Where the lease establishment specifies GS_Protocol_Type = 0, the native application protocol will be published through the CON object and subscribed through the DIS object. GS_Network_Address_List is empty. GS_Lease_Parameters contain only GS_Transfer_Mode in order to set default transfer quality of service and priority.

GS_Network_Address_List is used to establish the publish and subscribe endpoints. GS_Network_Address determines the remote device address. GS_Resource is used to determine the DIS object within this device. A local CON object is selected to be linked with the remote DIS object. GS_Lease_Parameters supply GS_Update_Policy, GS_Period, GS_Phase, and GS_Stale_Limit to establish the periodic or changes of state behavior for the CON and DIS objects. GS_Connection_Info is empty.

The publication payload (GS_Publish_Data) is sent and received in NativeIndividualValue or NativeValueList format. The CON and DIS objects in gateways transfer these payloads by using the publish service. GS_Transfer_Mode is provided with each transfer in order to indicate the quality of service, including priority, associated with the transfer.

The GS_Transfer_Mode specifies priority and discard eligibility, as defined in Clause 12 for the publish service.

P.13.3 Foreign access

Where the lease establishment specifies GS_Protocol_Type not equal to 0, GS_Protocol_Type is used to specify the foreign application protocol that will be published through the TUN objects. GS_Network_Address_List is used to establish the remote TUN endpoints. GS_Resource is used to match the TUN endpoints within devices. GS_Lease_Parameters supply GS_Update_Policy, GS_Period, GS_Phase, and GS_Stale_Limit to establish the periodic or changes of state behavior for Publish and Subscribe services. GS_Lease_Parameters supply GS_Connection_Info on Subscribe services as appropriate for the foreign protocol and GS_Transfer_Mode in order to set default transfer quality of service and priority.

The publication payload (GS_Publish_Data) is sent and received in non-native format. The TUN objects in gateways transfer these payloads by using the publish service. GS_Transfer_Mode is provided with each transfer in order to indicate the service, including priority, associated with the transfer.

The GS_Transfer_Mode specify priority and discard eligibility, as defined in Clause 12 for the tunnel service.

P.14 Bulk transfer

The GIAP bulk transfer service is implemented through the bulk transfer protocol and IFO and UDO objects.

Bulk transfer is used for upload/download in half-duplex mode. An IFO acts as a client. UDOs act as servers. The UDO object identifier represents the target resource for the operation. A series of AL block transfers are controlled by the end objects to provide ordered, error-free delivery of complete blocks of a negotiated size. There is no reliance on reliable transfer in lower layers. A multi-phase transfer protocol (open, transfer and close) is employed. A series of separate requests and responses track the total transfer size. Timing attributes are defined for the UDO to assist the client in determining timeout and retry policies and to avoid congestion errors. An upload or download operation may be closed due to errors on either end.

Lease establishment for bulk transfers establishes the necessary communication resources via a communication contract prior to bulk transfer.

The G_Bulk_Open request primitive is used to initiate a bulk transfer. The target device for a bulk transfer is addressed by the GS_Network_Address, which is aIPv6Address. The target item for a bulk transfer is identified by GS_Resource, which contains the Transport_Port and the OID pointing to a specific UDO.

P.15 Alert

The GIAP alert service is implemented through the alert (alarms and events) services.

Lease establishment for alerts establishes the necessary communication resources via a communication contract to enable alert receipt. GS_Alert_Source_ID specifies Transport_Port, OID, and alert type.

P.16 Gateway configuration

There is no specific adaptation information for this item.

P.17 Device configuration

There is no specific adaptation information for this item.

Annex Q
(informative)

Exemplary GIAP adaptations for IEC 62591

NOTE The following information was derived by analysis of IEC 62591 and may contain errors. See the actual IEC standard for a full and correct understanding.

Q.1 General

Q.1.1 Overview

This standard does not define functionality for a complete gateway. It does include supporting examples that allow gateway construction by the addition of a protocol translator and a hardware interface and stack for a foreign network. Such an addition requires a separate effort to define the protocol translator.

Annex Q describes an exemplary gateway interface, called the GIAP, which is intended to be an abstraction of an underlying wireless system. In particular, it is intended to provide an abstraction for the wireless system described in this standard, and also of the wireless system described in IEC 62591.

Annex Q describes one way to implement the informative GIAP by using the IEC 62591 command set. It is not a complete design, but a reference to aid understanding.

Specific gateways (for specific fieldbuses) could include Annex Q, thus making it normative. They might also adopt a different approach.

Q.1.2 Reference

Annex Q references IEC 62591, IEC 61158-5-20, IEC 61158-6-20, and HCF_SPEC-183, which specify some of the HART commands and field encodings used by IEC 62591.

Q.1.3 Addressing

IEC 62591 device addressing and identification information includes:

- Nickname: a 2-octet short identifier for a device;
- Unique ID: an 8-octet globally unique identifier formed by HCF OUI = 0x00 1B1E + 5 octet HART Unique ID, together conforming to EUI64Address requirements;
- Long Tag: a 32-octet human-readable string.

The GIAP interface uses logical IPv6Addresses. Most IEC 62591 commands use nicknames. IEC 62591 gateways are required to implement command 841 (Read Network Device Identity) using nickname that returns a unique ID and a long tag for a nickname. Command 832 (Read Network Device Identity) converts the unique ID to the nickname and long tag of a device.

It is recommended to map the unique ID into the low octets of the longer GIAP address.

Q.1.4 Stack Interface

IEC 62591 describes its highest interface as an interface to the NL. The NL interface description receives parameters that it uses to invoke a TL. Regardless of the interface description, the over-the-air packet encapsulates the TL header within a NL payload.

The TL payload encapsulates one or more HART or IEC 62591 commands, both requests and responses. Annex Q describes the mapping of the GIAP services to commands that are carried by the TL.

Q.1.5 Tunneling

IEC 62591 gateways are required to tunnel HART commands. This means that a gateway includes a foreign network (the host interface) connected to the gateway and the gateway will tunnel HART commands through the foreign network.

Q.1.6 Entities

The virtual gateway, network manager, host interface (host applications) and network interface (network devices) are all IEC 62591 entities that implement (issue and respond to) HART and IEC 62591 commands. The network manager has exclusive communication to a security manager. All communication between the network manager and the network devices and all communication between the host applications and the network devices is routed through the virtual gateway, which acts as a command routing hub. The virtual gateway itself also implements certain commands. The virtual gateway communicates to the network devices through one or more network access points as well as interposing network devices that perform routing.

Q.1.7 Delayed response

HART incorporates a delayed response mechanism, where a first response indicates that the command was received but that the actual response is delayed due to extended processing requirements. The GIAP services require handling of delayed responses within the gateway. An error is returned if a command that expects an acknowledgment is not acknowledged.

Q.2 Parameters

GS_Network_Address is a logical IPv6Address used to identify a specific IEC 62591 device within a network.

GS_Unique_Device_ID is a device-unique identifier in EUI64Address format, used to identify a unique IEC 62591 device. All gateways share a unique ID of 0xF9 8100 0002.

GS_Network_ID indicates an IEC 62591 network that is accessible through the gateway. IEC 62591 defines a 16-bit ID. IEC 62591 specifies a single gateway per network. A multi-mode gateway specifies multiple networks per gateway and uses the network ID to identify the specific network associated with an IEC 62591 virtual gateway.

Q.3 Session

Multiple sessions may be established through a gateway. Each session is used to communicate with a specific network as indicated by the GS_Network_ID that is provided when the session is invoked.

IEC 62591 includes a different concept that is also called a session. This session refers to an end to end security session. Annex Q does not refer to the security session, but the GIAP session.

The session service releases IEC 62591 virtual gateway resources when a session ends explicitly or by timer expiration by using the following commands:

- release all leases;
- release unused communication resources;
- release unused cache.

Q.4 Lease

A lease is used to allocate and release specific communication resources within the context of a session.

NOTE IEC 62591 "services" are allocated communication path resources from a requesting device (including the gateway) to a destination. Services are requested from the network manager and identified by a service ID. Services have independent bandwidth and latency guarantees, based on service allocation requests. The network manager handles establishment and management of intermediate resources, such as common (shared) routes, based on requests.

A lease is established with command 799 (request service). This command is used to request from the network manager a connection to another device (a service) with specified bandwidth and latency.

The service is identified by a service ID (maps to GS_Lease_ID).

GS_Lease_Period is set by the protocol translator.

GS_Lease_Type is defined by the service request flags and the service application domain.

GS_Protocol_Type is defined in Annex M.

The nickname specifies the address of the gateway peer for the service (maps to GS_Network_Address_List which includes a single GS_Network_Address). IEC 62591 includes multicast mechanisms, but not for services. Device level peer-to-peer is possible within the protocol, but not recommended due to security concerns.

GS_Resource is unused in this context, so is set to 0.

The period/latency maps to GS_Lease_Parameters (GS_Period, GS_Phase, and GS_Stale_Limit).

Command 801 (delete service) is used to notify a device of the deletion of a specific service (based on the service ID) due to peer request or network manager decision.

Q.5 Device list report

An IEC 62591 gateway is required to implement command 814 (read device list entities). This command retrieves a list of the unique IDs for the devices known to the gateway.

All devices returned are on the active device list. Whitelist and blacklist indication are maintained in the network manager and within the gateway.

GS_Network_Address, GS_Unique_Device_ID, GS_Manufacturer, GS_Model, and GS_Revision are returned for each device.

Q.6 Topology report

The topology report returns a list of devices (GS_Device_List), their address (GS_Network_Address), and related information. The device list report identifies the devices in a system.

An IEC 62591 gateway is required to implement command 834 (read network topology information). This command is used to retrieve the graph information (GS_Graph_List) for a specific device. Retrieved information includes a list of Graph IDs (GS_Graph_ID) for the graphs that the device participates in and a list of nicknames for the neighbors in the graph (associated to GS_Network_Address).

An IEC 62591 gateway is required to implement command 833 (read network device's neighbor health), which returns the set of neighbors of a specific device. Each element in the list returns the neighbor nickname (which maps to GS_Network_Address within GS_Neighbor_List).

Q.7 Schedule report

The schedule report service returns schedule information for a specific device identified by GS_Network_Address. The device list report may be used to identify the devices in the system.

Command 783 (read superframe list, normally used by the network manager) is used to retrieve the list of superframes and their related information from a specific device. Retrieved information includes the superframe ID (GS_Superframe_ID), the number of slots (GS_Num_Time_Slots) and superframe mode flags (HCF_SPEC-183, table 47).

GS_Slot_Size is fixed to 10 ms. GS_Start_Time is calculated from SuperframeSlot = (Absolute Slot Number) % Superframe.NumSlots.

Command 784 (read link list; normally used by the network manager) is used to retrieve information about the link entries from a specific device. Link entries are related to slot usage within superframes. Retrieved information includes the Superframe ID (GS_Superframe_ID), the slot number in the superframe, the channel (GS_Channel), linkOptions (HCF_SPEC-183, table 46), linkType (HCF_SPEC-183, table 45), and nickname (associated to GS_Network_Address) of the link neighbor to build GS_Link_List.

GS_Channel_List contains a list of whitelist and blacklist channels as defined by GS_Channel_Status to reach GS_Channel_Number. GS_Channel_Number maps to Index = 0 for IEEE 802.15.4 channel = 11, 2,405 MHz … Index = 14 for IEEE 802.15.4 channel = 25, 2,475 MHz. Command 817 (read channel blacklist) is used to identify the GS_Channel_Status for each channel.

Q.8 Device health report

The device health report returns device health information for a list of devices (GS_Device_List) each identified by GS_Network_Address.

All IEC 62591 devices implement and periodically publish command 779 (report device neighbor health) to make information available to the network manager and applications.

An IEC 62591 gateway is required to implement command 840 (read network device's statistics), which reports most of the command 779 information (no power status). This command uses a Unique ID to retrieve a variety of information related to a specific device, including:

- number of DPDUs generated by this device (GS_DPDUs_Transmitted);
- number of DPDUs terminated by this device (GS_DPDUs_Failed_Transmission);
- number of DL MIC failures (GS_DPDUs_Received, GS_DPDUs_Failed_Reception);
- number of NL MIC failures (GS_DPDUs_Received, GS_DPDUs_Failed_Reception);
- number of CRC errors (GS_DPDUs_Received, GS_DPDUs_Failed_Reception).

Command 840 is used multiple times to gather information for each device in the list.

Q.9 Neighbor health report

Neighbor health is periodically published to the network manager by command 780 (report neighbor health list). Neighbor signal strength is periodically published to the network manager by command 787 (report neighbor signal levels), which duplicates information in command 780.

G_Neighbor_Health_Report returns a list of link-level connection quality information for the set of neighbors of a specific device. The service is primarily implemented by command 833.

A list of devices known to the gateway (and each device address GS_Network_Address) may be retrieved by using the GIAP device list report service (G_Device_List_Report).

An IEC 62591 gateway is required to implement command 833 (read network device's neighbor health) which returns a list of link-level connection quality information for the set of neighbors of a specific device. Each element in the list returns the neighbor nickname (which maps to GS_Network_Address), the receive signal level in dB (GS_Signal_Strength), the number of packets transmitted to the neighbor (GS_DPDUs_Transmitted), the number of failed transmissions to the neighbor where no ACK/NAK DPDU was received (GS_DPDUs_Failed_Transmission), and the packets received from neighbor (GS_DPDUs_Received).

GS_Link_Status = 1 indicates that the neighbor is available for communication. GS_Link_Status = 0 indicates that the neighbor is unavailable for communication.

An IEC 62591 gateway is required to implement command 840 (read network device's statistics), which reports GS_DPDUs_Failed_Reception as described in the device health report clause.

GS_Signal_Quality is not available, so is set to the maximum quality value.

Q.10 Network health report

The device health report and neighbor health report are used to determine GS_Device_Health_List and GS_Network_Health.

An IEC 62591 gateway is required to implement command 840 (read network device's statistics). This command uses a Unique ID to retrieve a variety of information related to a specific device, including:

- number of joins (GS_Join_Count);
- date of most recent join and time of join (GS_Start_Date);
- average latency from the gateway to this node (GS_GPDU_Latency).

ASN is a count of all slots that have occurred since forming the network. It always increments and is never reset. ASN is 5-octets long. ASN 0 is when the network is born. GS_Start_Date and GS_Current_Date are derived from ASN.

Q.11 Time

IEC 62591 network time is measured relative to the absolute slot number 0 (ASN 0), which is the time when the network was last restarted. Time advances in 10 ms increments per slot.

Time distribution is configured by the network manager by using command 971 (write neighbor property flag) to specify a neighbor with the neighbor flags (0x01 time source, HCF_SPEC-183, table 59) indicating a specific neighbor as a time source. The IEC 62591 gateway is always configured as the source of network time.

Slot time is updated through neighbors by synchronization via time errors seen in packet exchanges (i.e., an ACK/NAK DPDU's TsError field).

The virtual gateway is required to synchronize with an external time source at least once per hour. UTC time is mapped to slot time from an external reference through the gateway. The mapping of ASN 0 to UTC is broadcast from the gateway. Command 793 (write UTC time mapping) is a gateway command that allows the network manager to set the mapping of the start of ASN 0 to UTC time on a device.

GS_Time is based on TAI time. UTC time is based on TAI time with leap seconds added at irregular intervals. This service applies time updates through the GIAP. TAI and UTC time updates occur due to drift. UTC adds additional updates due to leap seconds. A conversion is necessary to the internal HART time format from and to GS_Time: HART date 3 octets, time of day, 3 octets.

Command 794 (read UTC time mapping) is a gateway command that allows a device or the network manager to set and read the mapping of the start of ASN 0 to UTC time. GS_Command is used to set and read GS_Time within the gateway for synchronization purposes. Command 89 (Set Real-Time Clock) is used to set the time. Command 90 (read real-time clock) is used to read the current time.

Q.12 Client/server

Unless specified elsewhere in Annex Q, the gateway tunnels all HART commands through the GIAP client/server service. These commands are issued from a master to a slave (field device). The master assumes the client role and the slave assumes the server role.

The commands follow a request/response format. Request data octets are sent from the client to the server in GS_Request_Data. Response data octets are returned from the server to the client in GS_Response_Data. The command-specific response codes are mapped into GS_Status.

The GS_Buffer flag is set or cleared to indicate whether a command is to be buffered. The following commands are buffered:

- 0: read unique id
- 11: read unique id associated with tag
- 13: read tag, descriptor, date
- 20: read long tag
- 21: read unique id associated with long tag?
- 48: read additional status
- 50: read dynamic variable assignments
- 18: write tag, descriptor, date
- 22: write long tag
- 25: write primary variable range values
- 44: write primary variable units

Multiple server responses may be received with the same GS_Transaction_ID in the case of a delayed response.

Client/server priority is established via the GS_Transfer_Mode.

IEC 62591 priority falls into one of four levels, command (highest priority), process data, normal, and alarm (lowest priority). Command priority is reserved for packets containing network control, configuration and diagnostics. Process-data priority packets contain process data and are refused when three-quarters of a device's packet buffers are full. Alarm priority packets contain alarms and events. Only a single alarm priority packet is buffered. Normal priority packets are all other packets and are refused when one-half of a device's packet buffers are full.

GS_Transaction_Info is not required.

Q.13 Publish/subscribe

Q.13.1 General

The GIAP publish/subscribe service is implemented through publication of commands by the IEC 62591 devices using burst mode. Adapters are able to publish on behalf of non-native sub-devices. IEC 62591 natively aggregates published commands where the time aligns and command 78 (read aggregated commands) is not required.

Normally, a gateway subscribes to a device publication. Within G_Subscribe, GS_Publish_Data returns the published data.

It is required that a lease be acquired for the subscription (obtain GS_Lease_ID). Lease establishment allocates resources between the gateway and the device using command 799.

The G_Publish_Watchdog indication is received if the publication is not received by the GS_Stale_Limit.

Q.13.2 Lease establishment

A subscription lease is established through the lease service. GS_Resource specifies the subscription information (command number and process variable list) to the lease service.

Command 108 (write publish data mode command number) is used to select the command to be published.

If command 108 specifies universal command 9 (read device variables) or common practice command 33 (read device variables), process variables will be assigned to slots for publication. Command 107 (write publish data device variables) is used to assign the slots.

Command 103 (write publish data period) selects the minimum (GS_Period, GS_Phase) and maximum update period (GS_Stale_Limit) for a publication (in 1/32 ms increments up to 3 600 s; requested and actual values may differ).

Command 104 (write publish data trigger) sets a trigger condition (GS_Update_Policy) for publication (continuous/windowed/rising and a level) resulting in dynamic changes to publication time. Publication occurs at least as often as when the maximum period is reached.

Command 109 (publish data mode control) turns publishing on and off. The publication source device contacts the network manager to request bandwidth.

Q.13.3 Buffering

The following commands are buffered:

- 1: read primary variable
- 2: read current & percent

- 3: read all variables
- 9: device variables and status
- 33: read device variables
- 123: read trend
- Device-specific

Q.14 Bulk transfer

The GIAP bulk transfer service corresponds to the AL provided block transfer. Operation permits upload/download (GS_Mode) in either half or full duplex modes, and relies on the TL to provide a series of application level block transfers. The transport segments and reassembles based on limited MTU in lower layers and provides error free delivery of complete blocks (all pieces are in order).

The operation uses several phases, including open (G_Bulk_Open), transfer (G_Bulk_Transfer), reset, and close (G_Bulk_Close). New commands were created to execute these phases. A master opens a session (command 111) with a slave to initiate the operation (GS_Transfer_ID links the phases of this operation). The master proposes the block sizes (GS_Block_Size), and the slave may reduce the size. A port (an octet) identifies the target resource (GS_Resource) for the operation (firmware, parameters, and log file). The total size is not stated (GS_Item_Size = 0) and may not be known even to the application (such as a continuous stream of samples organized in blocks). There is an octet counter selected by each end to track progress. Command 112 is organized such that the request contains download data (GS_Bulk_Data) and the response has upload data (GS_Bulk_Data). The request creates an indication in the slave; the response contains an indication in the master. The session is closed on errors. No rule exists on how to deal with the partial data set. The delayed response mechanism is mentioned in status, but is not described further.

Q.15 Alert

The GIAP alert service is implemented through several mechanisms. Locally buffered changes include burst mode updates (process changes), event notification (general alarms and events), device status changes, device configuration changes, network topology changes, and network schedule changes.

Change notification simply indicates a change, and further action is required to retrieve altered information from the gateway buffers. The gateway entity acknowledges event arrival to devices. Publications and alerts are stored in the gateway entity. The gateway entity acknowledges alert arrival to devices.

For example, the gateway often internally uses HART command 115..118 to set up change notification and HART command 119 to indicate that changes have occurred.

Events are configured with assigned event numbers on a per-device basis.

Command 116 (write event notification bit mask) configures the event mask that is used to trigger an event notification for a specific event. The event mask corresponds to command 48 (read additional device status), which refers to common tables 14, 17, 29, 30, 27, 31, 32, 28 and device specific status.

Command 117 controls the timing of event notifications. Event notification uses burst mode for delivery when an event is triggered. A de-bounce period is specified to prevent events that are too short from triggering a burst message. A retry time (desired burst period) and a maximum update time (maximum burst period) set the burst transfer timing if an event triggers a message.

Command 118 (event notification control) is used to enable or disable an event notification for a specific event.

Command 119 (acknowledge event notification) is used to acknowledge the event notification and clear the event from being sent in the burst updates. Other events may be in queue.

Command 115 (read event notification summary) is used to determine the configuration of an event based on a specific event number.

The following commands are buffered:

- 119 read event notification status (time stamp + device status + command 48).
- Command 788 (alarm path down), command 789 (alarm source route failed), command 790 (alarm graph route failed), and command 791 (alarm TL failed) report communication failures to the network manager.

IEC 62591 gateway command 836 (write update notification bit mask for a device) registers a client for notification updates. The device is addressed by the unique ID and given a set of change notification flags. Codes exist for BurstMode, EventNotification, DeviceStatus, DeviceConfiguration, NetworkTopology (gateway or NM), and NetworkSchedule (gateway or NM). This is used in G_Alert_Subscription to subscribe (by providing a GS_Subscription_List with GS_Alert_Source ID, GS_Subscribe, and GS_Enable for a specific device GS_Network_Address and a specific category GS_Category).

IEC 62591 gateway command 838 (read update notification bit mask for a device) returns a list of the update notifications for a device. This is used in G_Alert_Subscription to identify the subscriptions.

IEC 62591 gateway command 839 (change notification) is sent by the gateway to a client and returns a list of up to 10 change notifications (cached commands) for a device. Each change results in a single G_Alert_Notification.

Q.16 Gateway configuration

There is no specific adaptation information for this item.

Q.17 Device configuration

There is no specific adaptation information for this item.

Annex R
(informative)

Host system interface to standard-compliant devices via a gateway

R.1 Background

R.1.1 Host system integration reference model

A simplified reference model for a standard-compliant device/host system integration is depicted in Figure R.1.

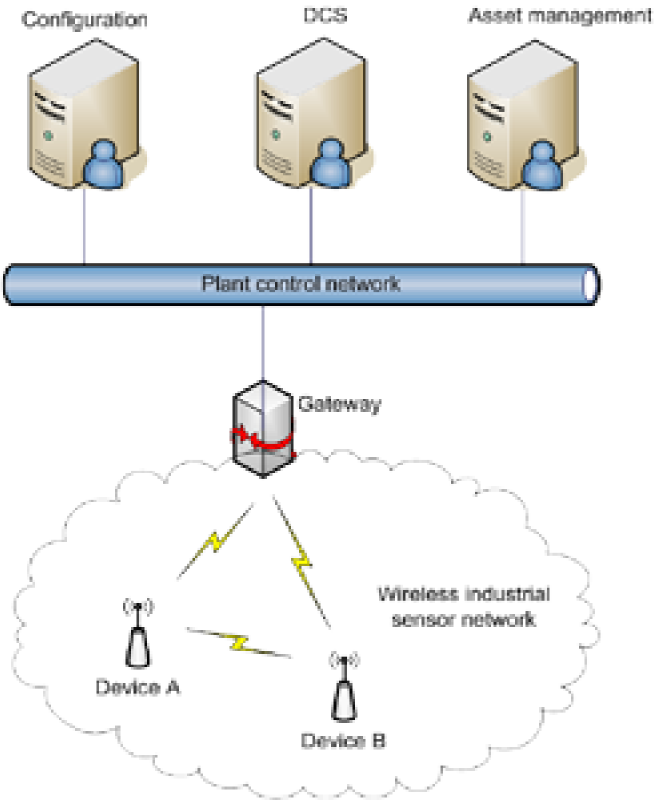


Figure R.1 – Host integration reference model

R.1.2 Asset management tools

Asset management involves overseeing the health of the system's assets by monitoring health related conditions in order to identify a potential problem before the process or plant operation is affected. Host systems provide an asset management tool or set of tools to fulfill the asset management function, with goals of lowering maintenance costs, reducing down-time, and ensuring that appropriate product quality levels are met.

R.1.3 Configuration tools

Once the system design has been established, and the system components identified, the operation of the components in the overall system needs to be configured. Host systems provide a configuration tool or set of tools that support system component configuration and define component operation in the system.

R.1.4 Distributed control system

A distributed control system (DCS) is a control system that supports a process wherein the control elements are geographically distributed. These distributed elements are connected by communication networks, which are used for communicating with the distributed elements.

R.1.5 Gateway

A gateway connects the host systems with the network. See Annex U for more information regarding the gateway.

R.2 Device application data integration with host systems

R.2.1 General

There are two generic means for host systems to integrate application data from connected devices:

- integration via protocol mapping; and
- integration via protocol tunneling.

R.2.2 Native protocol integration via mapping

Existing host systems may integrate device application data by mapping the relationship between the devices and data to the information handling performed by the existing host system. This mapping function is usually performed by a gateway between the existing host system and the wireless industrial sensor network (WISN).

R.2.3 Legacy device protocol integration via tunneling

Existing host systems may integrate application data from existing legacy devices that are using the WISN application tunneling capability in the same manner by which it presently integrates the application data from the legacy devices.

R.3 Host system configuration tool

R.3.1 General

Host systems usually support either one or both of two generic integration methods for configuring field devices:

- electronic device description language (EDDL);
- field device tool / device type manager (FDT/DTM).

R.3.2 Host configuration using electronic device description language

IEC 61804-3, which deals with EDDL, describes a generic language for describing automation device properties. EDDL can describe device functions, interactions supported by a device, device-supported objects, and other properties.

EDDL is used by a device vendor to create an electronic device definition (EDD) file that corresponds to a particular device. An EDD file is an operating system and automation system independent structured ASCII text file that describes the capabilities of a device to allow integration of the device with a host DCS system. This independence enables vendors to describe their devices in a manner that enables vendor independent interworkability and constrained interoperability of the device across host systems. EDD files describe device data, device vendor desired user interface characteristics, and device command handling, such as command ordering and timing.

Host DCSs provide tools to interpret EDD files in order to configure and handle the device, such as for monitoring or parameter handling, to support control applications.

EDDs are defined by device vendors and tested by the appropriate fieldbus supporting organization.

Figure R.2 represents configuration using a DD file.

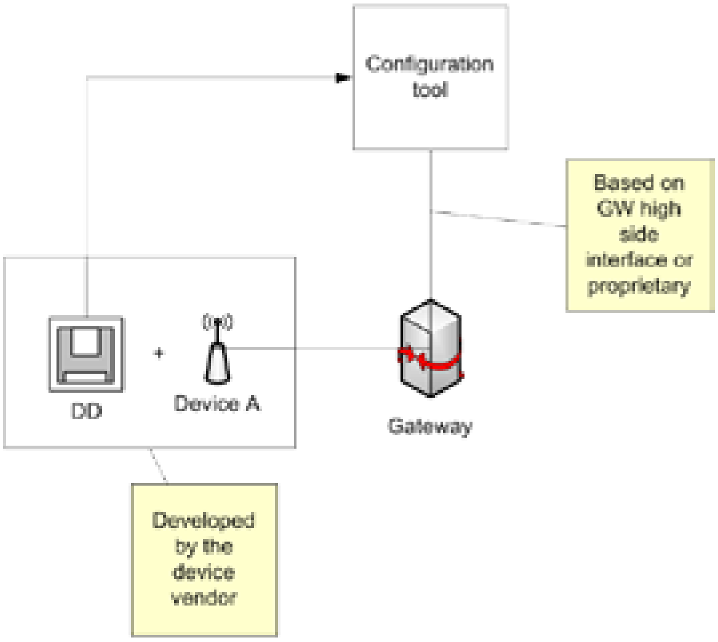


Figure R.2 – Configuration using an electronic device definition

R.3.3 Host configuration using field device tool/device type manager

The device functionality described by EDD is limited by IEC 61804-3. Additional device functionality (if any) that cannot be described via EDD can be supported via proprietary plug-ins or snap-ons. To provide this greater support, field device tool / device type manager (FDT/DTM) technology may be used. FDT/DTM technology requires, for example, FDT PDU application support in the DCS. For further information on FDT/DTM, consult the FDT Group.

Figure R.3 represents a configuration using the FDT/DTM approach.

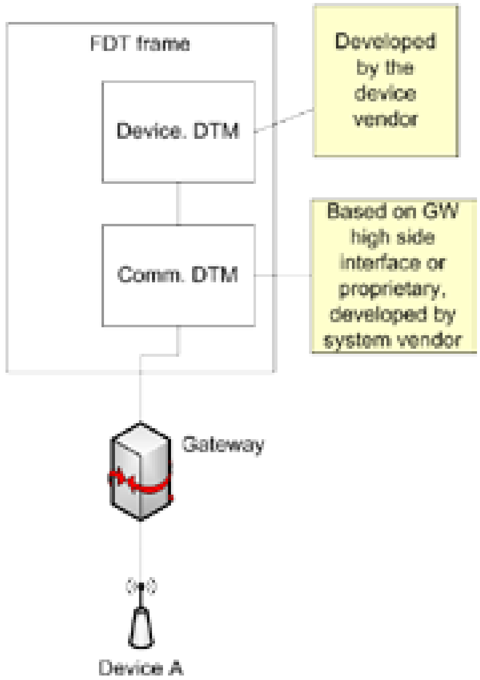


Figure R.3 – Configuration using FDT/DTM approach

R.4 Field device / distributed control systems integration

R.4.1 General

Distributed control systems usually consist of devices such as controllers, human-machine-interface (HMI) stations, data historian servers, advanced applications, etc. HMI stations, historian servers, and advanced applications often employ interfaces with rich data semantics, such as OPC. Communication with controllers usually employs simpler protocols, such as Modbus, or Foundation Fieldbus High Speed Ethernet (FF-HSE).

R.4.2 Foundation Fieldbus High Speed Ethernet

Application data integration with FF-HSE can, for example, be accomplished by mapping the native application data to FF transducer blocks. Application objects map to FF blocks, while object attributes map directly to the FF block parameters.

R.4.3 Modbus

Application data can integrate with Modbus by assigning a Modbus address to the gateway. The gateway then may present a set of register tables to Modbus masters. Each object attribute may be mapped to a specific register. The host system may provide automated support for the mapping, or mapping may be performed manually by the user.

R.4.4 Open connectivity for industrial automation

Open connectivity for industrial automation (OPC) allows client applications to access data in a consistent manner via an OPC server by referencing the data using a Tag.Parameter construct.

An OPC client may be supported by an OPC server in the host system or by a high-side OPC interface provided by a gateway to a standard-compliant system.

For example, this standard provides value, quality, and timestamp information in data publications, in support of OPC server access to online data. Native alarms and events also provide support for OPC client notification.

The OPC client may specify Tag.Parameter using the device name for the Tag, and a unique object name and attribute to represent the parameter (e.g., TI101.AITB1.PV). In the OPC server, the Tag is mapped to the device, the object instance maps to a particular object instance of a particular UAP, and the attribute name maps to the particular attribute identifier of the referenced object instance.

R.5 Gateway

R.5.1 General

Host system configuration of applications residing within the gateway itself, including data mapping (if necessary), is defined by the plant control network, which is the high side interface of the gateway that couples the WISN into a higher level control system. This includes, for example, configuration of a system management application or a tunneling application. Therefore, Annex R describes in generalities the type of information that needs to be configured for gateway support.

R.5.2 Devices supported

A host system configuration tool may need to establish the complement of standard-compliant devices with which the gateway will communicate.

R.5.3 Data subscription

A host system configuration tool may need to establish the configuration of the dispersion objects in the gateway for the data the gateway will receive via publication.

R.5.4 Data publication

A host system configuration tool may need to establish the configuration of the concentrator objects in the gateway for the data the gateway will itself publish.

R.5.5 Client/server access

Non-management related client/server communications may, for example, be established by the gateway on an as-needed basis through interface objects.

R.5.6 Alerts reception

A host system configuration tool may need to establish the alert categories associated with gateway-resident alert-receiving object(s) (AROs).

R.6 Asset management application support

R.6.1 General

An asset management tool may access information about a device that is either stored in or accessed via the gateway by using plant control network services.

A gateway may access information directly from a field device to satisfy asset management requests. The gateway may, for example, employ client/server services to read data, to write data, or to execute a particular method on a particular object instance within the wireless device.

A gateway may act as a pass-through for asset information directly from an asset to an asset management application via a plant control network tunnel if the plant control network supports such a tunneling capability.

R.6.2 Field device tool / device type manager

A DTM may be provided by a device vendor to provide process and device information to an asset management tool. A host system supporting an FDT PDU can employ the device DTM and a communication DTM for the gateway to acquire the information necessary to manage the device via the gateway.

R.6.3 HART

A standard-compliant device may be made to appear as a HART[14] native device on a HART asset management application (ASM) in several ways:

- Manually or using automation along with either explicitly coded or data-driven conversion rules provide a HART DD source file for the device. The HART DD file can be passed through a HART tokenizer to produce binary files representing the DD content. Most HART clients use the binary format of the of the DD files.
- Standard commands may be defined in HART to integrate ASM with this standard, such as a HART commands for READ_IEC62734_ATTRIBUTE, WRITE_IEC62734_ATTRIBUTE, and EXECUTE_IEC62734_METHOD.
- Mapping tables in the gateway may be employed to define attribute value mapping that differs between this standard and HART, such as for engineering unit indices.

R.6.4 OPC

Open connectivity for industrial automation (OPC) allows client applications to access data in a consistent manner via an OPC server. An OPC client may be supported by an OPC server in the host system or by a high-side OPC interface provided by a gateway to a standard-compliant system.

For example, device health information may be provided by the OPC server to an OPC client.

[14] HART is a registered trademark of HCF. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

Annex S
(informative)

Symmetric-key operation test vectors

S.1 DPDU samples

S.1.1 General

[INGREDIENTS]

- TsDur: 10464 [2^-20sec]
- Data DPDU Source EUI64: 0x00 00 00 00 00 00 00 01
- Data DPDU Key: 0xC0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
- Data DPDU Sequence Number: 0x04
- TAI Time[TAINetworkTimeValue]: 0x00 01 02 03 04 05
- Channel: 0x02
- Data DPDU Headers: 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28
- Data DPDU Payload: 0x30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B
- ACK DPDU Source EUI64: 0x00 00 00 00 00 00 00 02
- ACK DPDU Sequence Number: 0x05
- ACK DPDU Headers: 0x10 11 12 13 14 15 16 17 18

S.1.2 DPDU with expected DMIC32

[PRE-PROCESSED MATERIAL]

- Data DPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 14
- Data DPDU MIC: 0xBF 5A BB 7C
- ACK DPDU Nonce: 0x00 00 00 00 00 00 00 02 04 08 0C 10 15
- ACK DPDU authentication vector: 0x10 11 12 13 14 15 16 17 18 BF 5A BB 7C
- ACK DPDU MIC: 0x74 F0 41 B3

[DELIVERABLE]

- Data DPDU: 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B BF 5A BB 7C
- ACK DPDU: 0x10 11 12 13 14 15 16 17 18 74 F0 41 B3

S.1.3 DPDU with expected ENC-DMIC32

[PRE-PROCESSED MATERIAL]

- Data DPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 14
- Encrypted Data DPDU Payload: 0x23 F4 C4 3F BA 9B E4 3E D8 9B FD 36 A8 76 C7 99 27 14 E0 42 94 94 DE 64 B2 6B 14 18 51 9F 8D 11 36 F4 09 17 6B D6 A6 75 07 B1 D2 90
- Data DPDU MIC: 0xD0 F6 B2 65
- ACK DPDU Nonce: 0x00 00 00 00 00 00 00 02 04 08 0C 10 15

- ACK DPDU authentication vector: 0x10 11 12 13 14 15 16 17 18 D0 F6 B2 65
- • ACK DPDU MIC: 0x26 AB 87 D2

[DELIVERABLE]

- Data DPDU: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 23 F4 C4 3F BA 9B E4 3E D8 9B FD 36 A8 76 C7 99 27 14 E0 42 94 94 DE 64 B2 6B 14 18 51 9F 8D 11 36 F4 09 17 6B D6 A6 75 07 B1 D2 90 D0 F6 B2 65
- ACK DPDU: 0x10 11 12 13 14 15 16 17 18 26 AB 87 D2

S.2 TPDU samples

S.2.1 General

[INGREDIENTS]

- TPDU time creation[TAINetworkTimeValue]: 0x00 01 02 03 04 05
- Key: 0xC0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
- Crypto Key Identifier Mode: 0x00
- Crypto Key Identifier = 0x10
- Source EUI64Address: 0x00 00 00 00 00 00 00 01
- Source IPv6Address: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01
- Dest IPv6Address: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 02
- Source Port: 0x00 01
- Dest port: 0x00 02
- TSDU (Application Layer Payload): 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B

S.2.2 TPDU with expected ENC-TMIC-32:

[PRE-PROCESSED MATERIAL]

- TPDU Pseudo header: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 2B 00 11 00 01 00 02
- TPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 FF
- TPDU Security header: 0xA0 0C 10

[DELIVERABLE]

- TPDU: 0x 00 01 00 02 00 23 00 00 A0 0C 10 8E 7C 0B B9 8B CD 15 7E 59 CE 71 18 14 B7 05 FE C2 6A F1 C3 9D 05 B9 FD E6 5F 16 C9 DE 37 DE BE

S.2.3 TPDU with expected TMIC-32:

[PRE-PROCESSED MATERIAL]

- TPDU Pseudo header: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 2B 00 11 00 01 00 02
- TPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 FF
- TPDU Security header: 0x20 0C 10

[DELIVERABLE]

- TPDU: 0x 00 01 00 02 00 23 00 00 20 0C 10 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 7E 8C 35 57

Annex T
(informative)

Data-link and network headers for join requests

T.1 Overview

Annex T illustrates the DL header and NL header for a typical join request.

T.2 MAC header (MHR)

MAC header for join messages is shown in Table T.1. IEEE convention shows bit 0 on the right, which is the nominal order of transmission. Per IEEE 802.15.4 convention, the Sequence Number and Addressing fields of the MHR, when considered as unsigned integers, are transmitted lowest-weight octet (LSB) first.

NOTE IEEE 802.15.4 2.4 GHz DSSS actually transmits quartets of four bits simultaneously as 32-chip spread-spectrum signaling, so there is no “first” or “last” bit transmitted within the quartet. However, the lower-bit-weight quartet in an octet, when interpreted as an Unsigned8, is transmitted before the higher-bit-weight quartet of that same octet, and the lower-weight octet is transmitted before the higher-weight octet.

Table T.1 follows the convention of this standard, showing bit 7 on the left.

Table T.1 – Sample MHR for join request

Subfield	Number of octets	bits							
		7	6	5	4	3	2	1	0
Frame Control	2	Reserved=0	PAN ID Compress =1 (yes)	ACK Request = 0 (no)	Frame Pending =0 (no)	Security Enabled =0 (no)	Frame Type =1 (Data)		
		Source Addressing Mode =3 (64-bit)		Frame Version=1		Dest Addressing Mode =2 (16-bit)		Reserved=0	
Sequence Number	1	(determined by DLE at time of transmission)							
Addressing	2	PAN ID (LSB), from advertisement							
	2	Destination Address (16-bit, LSB), from advertisement							
	8	Source Address (64-bit, LSB), device’s EUI64Address							

T.3 DL header (DHR)

DL header for join messages is shown in Table T.2. This example assumes that the advertisement does not specify slow-channel-hopping.

Table T.2 – Sample DHR for join request

<table>
<tr><th rowspan="2">Sub-header</th><th rowspan="2">octets</th><th colspan="8">bits</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>DHDR</td><td>1</td><td>ACK/NAK DPDU needed =1 (yes)</td><td>Signal quality in ACK/NAK DPDU =0 (no)</td><td>Request EUI64Address =0 (no)</td><td>Include DAUX = 0 (no)</td><td>Include slow channel hopping-offset =0 (no)</td><td>Clock recipient =1(yes)</td><td colspan="2">DL version = 00</td></tr>
<tr><td rowspan="2">DMXHR</td><td>1</td><td colspan="3">Reserved=0</td><td colspan="2">Key identifier mode =1</td><td colspan="3">Security level=1 (MIC-32)</td></tr>
<tr><td>1</td><td colspan="8">Crypto key identifier = 0: K_global</td></tr>
<tr><td>DAUX</td><td>0</td><td colspan="8">(absent by DHDR setting)</td></tr>
<tr><td rowspan="2">DROUT</td><td>1</td><td>Compress=1</td><td colspan="4">Priority =0 (irrelevant)</td><td colspan="3">DlForwardLimit =1</td></tr>
<tr><td>1</td><td colspan="8">GraphID (Unsigned8) =0 (Single hop source routing)</td></tr>
<tr><td rowspan="3">DADDR</td><td>1</td><td>DE=0</td><td>LH=0</td><td colspan="2">ECN=0</td><td colspan="4">Reserved=0</td></tr>
<tr><td>1</td><td colspan="8">SrcAddr = 0 (Use EUI64Address in MHR)</td></tr>
<tr><td>1</td><td colspan="8">DestAddr = 0 (Use DL16Address in MHR)</td></tr>
</table>

T.4 NL header

Network header for join messages is shown in Table T.3.

Table T.3 – Network header for join messages

<table>
<tr><th rowspan="2">octets</th><th colspan="8">bits</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>1</td><td colspan="3">LOWPAN_IPHC dispatch = 011</td><td colspan="5">LOWPAN_IPHC encoding (bits 8..12) = 11 101</td></tr>
<tr><td>2</td><td colspan="8">LOWPAN_IPHC encoding (bits 0..7) = 0111 0111</td></tr>
</table>

Annex U
(informative)

Gateway role

U.1 General

U.1.1 Overview

The primary purpose of a gateway as described by this standard is to enable host-level applications to interact with wireless field devices. A large installed base of applications exists, including automation devices, controllers, and supervisory systems, which together use numerous legacy protocols, thus requiring protocol translation when interacting with wireless field devices. Such protocol translation may be present in the gateway and also in adapters to legacy wired field devices. Within this standard, the term adapter is used to identify devices that convert from a wired fieldbus protocol to a wireless fieldbus protocol on behalf of one or more field devices.[15] Such protocol translation generally serves to tunnel a foreign protocol across a wireless network as described in this standard, or to convert a legacy protocol to and from this standard's native format. The term native field device refers to a field device that functions exclusively through the usage of the native objects, native interfaces, and native message content as defined in this standard.[16] It is also possible to write or modify host-level applications to use the native application protocol directly, reducing or eliminating the need for protocol translation within a gateway.

NOTE The examples provided in Annex U are symmetric, potentially applicable without modification to both gateways and adapters. In practice, the specific foreign protocol features and the usage of a gateway or adapter relative to host-level applications and field devices will dictate the subset of the protocols that apply to each.

The description of the gateway role relates to the following capabilities:

- Interfacing foreign host-level applications:
  - directly to "native" field devices (i.e., ones conforming to this standard); and
  - indirectly to legacy wired field devices through legacy adapters.
- Interfacing host-level applications to multiple wireless systems, including a combination of one or more wireless systems as described in this standard and one or more foreign wireless systems, through a single (conceptual) device with a common high-side interface.

This standard provides supporting functionality for the construction of gateways. It does not provide complete details on how to construct any particular gateway. Annex U is strictly informative, since no gateways are specified. As such it provides a suggested basis for future construction of gateway specification, but is itself not one. No validation of the content of Annex U has occurred.

Annex U describes support functionality for foreign protocol translation needs, but does not describe details on how to perform any specific protocol translation or how to interface to any specific plant network.

[15] Usage of the term adapter is not uniform. Technically, an adapter is an interface from a CPU to a communication channel. Technically, a gateway is an interface from one communication channel to another communication channel, where protocol translation is used at one or more layers of the protocol suite. There is a precedent set in the automation industry to (incorrectly) use the term adapter to identify devices that convert from a wired fieldbus protocol to a wireless fieldbus protocol on behalf of one or more field devices. There is also a precedent in the automation industry to (correctly) use the term gateway to identify devices that convert from a wireless fieldbus protocol to a wired fieldbus protocol for attachment to a control system. This document adheres to this automation industry usage in an attempt to minimize confusion.

[16] The native tunnel object uses the native tunnel and publication services to carry foreign message content. A field device that requires foreign message content to perform its function cannot be considered a native device.

Legacy protocols were not designed to operate over wireless networks. They do not access information in a manner that conserves energy, and they often are intolerant of delayed access to quiescent devices that are conserving energy. The gateway support functionality of this standard is, in large part, intended to enable the construction of gateways that adapt legacy protocols to the requirements of low-energy-consumption wireless devices.

The gateway role includes a specialized UAP. Functionality is provided by AL objects and gateway internal operation. The objects use the interfaces of the communications protocol suite to support gateway high-side interface functions.

U.1.2 Notional gateway protocol suite diagrams for native devices and adapters

The diagram in Figure 17 depicts a notional gateway interfacing a host-level application (the example control system) to a wireless field device. In this case, the field device is a native device. Protocol translation is performed in the gateway to convert between the plant network protocol and this standard's native protocol. Routers may exist between the gateway and the field device, as depicted in Figure 17, but from the gateway's perspective their operation is transparent.

The diagram in Figure 19 depicts a notional gateway interfacing a host-level application (the example control system) to a wired I/O device through a wireless system that conforms to this standard. In this specific case, the interfaced I/O device is a legacy device, not a native wireless device, and thus an adapter is required. Protocol translation is performed in both the gateway and the adapter. The gateway and the adapter each convert between legacy protocols and the communication protocols specified by this standard.

NOTE 1 It is often possible to implement an adapter to a single legacy wired I/O device by performing a protocol translation to and from native formats, without carrying any foreign message content. To a gateway, such a combination of an adapter and a connected legacy device is indistinguishable from a native I/O device. No special gateway provisions are made for such devices. Additionally, there are no special gateway provisions to facilitate multiplexing of such an adapter to multiple legacy wired I/O devices.

As seen in Figure 19, a notional gateway and a notional adapter share a common structure. Both have an interface and a protocol suite for a foreign network. Both have an interface and a protocol suite as described in this standard. Both have protocol translators. The common structure extends even further – they may share common objects and a common high-side interface structure. For this reason, no separate role was described for an adapter.

NOTE 2 The differences between a gateway and an adapter relate mostly to the implementation. For example, certain legacy protocols only publish from the field, thus requiring support for producer functionality but not consumer functionality in the adapter. Other legacy protocols also support publishing to the field, so require both producer and consumer functionality. In another example, legacy engineering tools carried into the field and plugged into the legacy network behind the adapter sometimes need to use the same functions as if they were behind the gateway.

For a gateway and an adapter to be interworkable, they require common protocol translation. If the adapter converts to and from native format, the gateway may do the same. If the adapter tunnels a legacy protocol, the gateway may tunnel the same protocol.

U.1.3 Gateway scenarios

Common gateway scenarios are depicted in Figure U.1. This figure does not attempt to provide an exhaustive description of all variations; rather, it is included to illustrate the bounds of this standard.

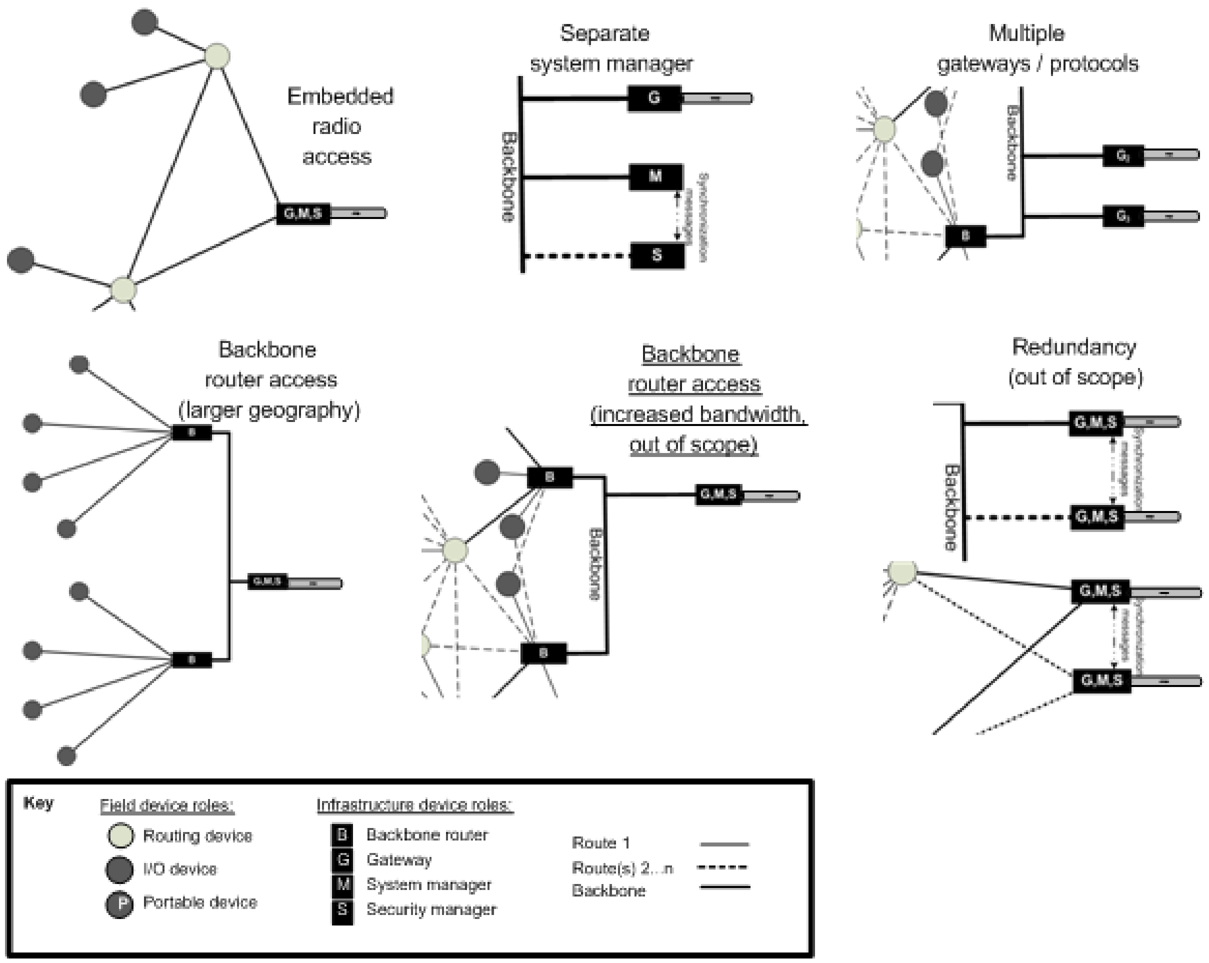


Figure U.1 – Gateway scenarios

As described in Clause 5, a gateway implements a role within the system. A variety of physical implementations are possible.

Some device implementations may have a Class A wireless interface and protocol stack embedded in the same packaging as the gateway, providing direct access to the wireless network. Independently, some device implementations may have system management and security management roles co-resident with the gateway role.

The system manager and security manager roles need not be co-resident with the gateway role. The gateway does not interact directly with the security manager, but indirectly through the system manager. The gateway functionality requires a communication path with a system manager to function as part of an operational system.

A device implementing a gateway role may use backbone routers to communicate with wireless field devices conforming to this standard.

Gateway communication with field devices through backbone routers is transparent in operation. NL extensions exist in other portions of this standard to support this transparency. It is, however, necessary to configure this routing within the gateway and the backbone routers. The backbone routers may be used to extend the geographical scope of gateway-connected devices. Backbone routers may also be used to increase bandwidth or to add redundant paths between a gateway and a mesh.

Multiple independent gateways may exist within a system. This is facilitated by independent addressing and independent communication relationships between devices. One use for independent gateways is to support multiple independent protocols. No special provisions are made in this standard for inter-dependent operation between gateways, such as for redundancy or load sharing.

U.1.4 Basic gateway model

Gateways may follow the general model depicted in Figure U.2.

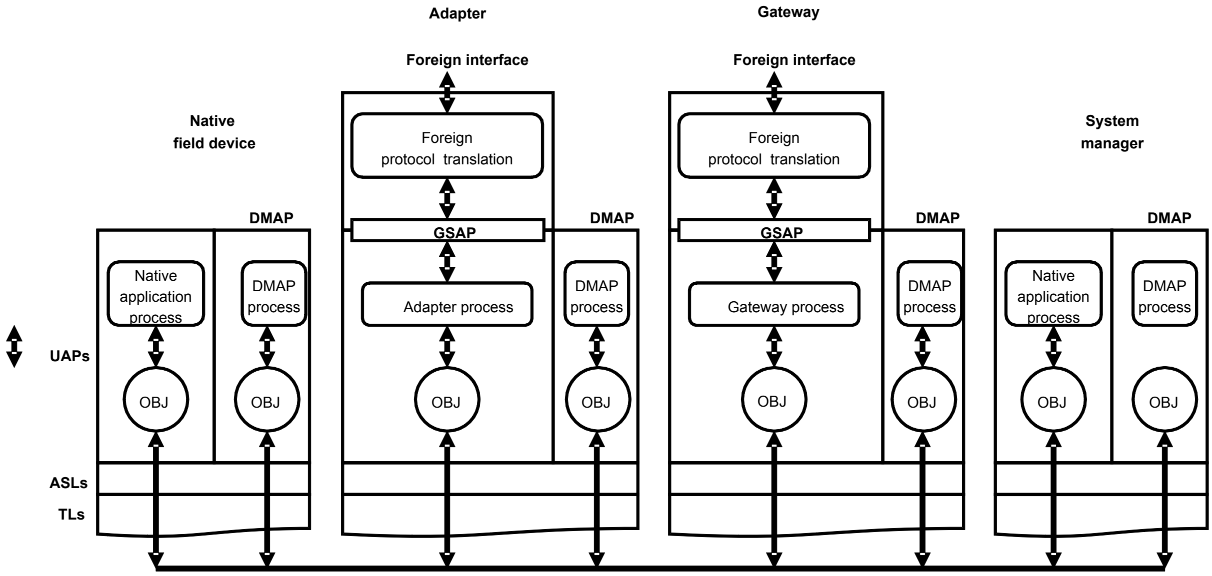


Figure U.2 – Basic gateway model

In this example, gateways and adapters host foreign protocol translators that receive and transmit foreign interface messages (usually from a control system, an asset management system, or an engineering system) and use gateway interfaces to interact with wireless devices. Gateway interfaces are provided via a high side interface that is accessed at a GIAP. Device-local protocol translators use these interfaces through the GIAP.

Protocol translation conveys application information for control, monitoring, configuration, and management. Foreign protocol messages contain this information. Tunneling, foreign protocol application communication (FPAC), and native object access methods are provided within the objects described in this standard to support protocol translation as described in Annex N. Each method entails specific tradeoffs of translation effort, energy efficiency, and performance. Practical protocol translators are likely to use a combination of these methods.

Gateways and adapters may each have application processes that interface to the protocol translators through the GIAP. Each process provides the high side interfaces by using application objects (OBJ). Inter-object communication uses the messaging methods provided through the application sublayer (ASL).

The GIAP interfaces are used for network management, protocol tunneling, upload and download, alerts, time management, and access to native application and management objects. The GIAP is described in detail in U.2.

For wired automation devices, an adapter performs a symmetric function and converts foreign interface messages to an adapter high side interface (also a GIAP) via a gateway peer foreign protocol translator.

Gateway access to a native field device (from a gateway) is enabled through the same GIAP; however, the object interactions use native messaging exclusively. A native application process interacts with the gateway or adapter process in a manner that depends on the type of the object.

Gateways, adapters, and native field devices can all be managed through the same symmetric method. The DMAP process is the peer process in this instance. Such management is specific to this standard and not necessarily to foreign protocols. Foreign protocols may provide additional device and wired fieldbus management methods that are outside the scope of this standard.

The basic gateway model includes interfacing to the system manager, which permits the system manager to be accessed via the gateway.

Backbone routers and routing devices are not shown in Figure U.2, because gateway functionality occurs within the application layer, not within the lower communication layers.

U.2 Notional GIAP

U.2.1 Summary of interfaces and primitives

The gateway portion of this standard describes a notional GIAP that can serve as a high side interface above a wireless communication protocol suite for conveying wireless information and managing wireless behavior. This notional GIAP is generic and could be used as a common interface above the AL of this standard and above other functionally in similar communication protocol suites. Annex P describes one potential implementation of the GIAP interfaces for the wireless protocol suite of this standard, using the defined AL objects and interfaces. Annex Q describes another notional GIAP interface implementation for an alternative wireless protocol suite.

NOTE 1 A primary intent of the example GIAP interfaces is to allow multimode access where a number of wireless interfaces are available to the gateway. In configurations where the path from the system manager to the plant network is only via the gateway role the GIAP supports consistent reporting information on each underlying wireless interface to promote improved coexistence. System management information can be included in a report on communication performance to identify potential interference problems, a report on topology to identify collocated devices, and a report on channel and schedule information to identify potential usage conflicts.

NOTE 2 Another intent of the example GIAP is to provide a model for the configuration and access of multiple underlying wireless networks. This potentially reduces the effort for a gateway developer if they have multiple fieldbus protocols to support. The GIAP interfaces may or may not be applicable to specialized gateway developers, such as those serving a single foreign protocol; specialized gateways may prefer to use customized gateway internal interfaces.

This notional GIAP interface is usable by a variety of protocol translators to interface to wireless communication protocol suites for conveying wireless information and managing wireless behavior. A protocol translator exists in a gateway. Depending on the implementation, a protocol translator and a GIAP may also exist in an adapter. Protocol translators will vary in complexity depending on the protocol that exists above the gateway and below the adapters. Certain protocols will use a subset of these notional GIAP interfaces. For example, a protocol may only require client/server interaction and not require publish/subscribe interfaces. Functionally, an adapter is considered a subset of a gateway and would only be expected to support a subset of these notional GIAP interfaces related to conveying wireless information.

NOTE 3 This gateway discussion does not describe protocol translation for specific fieldbus protocols.

GIAP interfaces are summarized in Table U.1.

Table U.1 – Summary of notional gateway high-side interface examples

Interface example	Interface subtype	Primitive	Description
Session	—	G_Session request G_Session confirm	A foreign protocol translator within the gateway may establish sessions on behalf of remote clients
Lease	—	G_Lease request G_Lease confirm	Leases allow the gateway to internally manage its internal communication resources on a per session basis
Device_List_Report	—	G_Device_List_Report request G_Device_List_Report confirm	Determines the devices associated with the gateway role
Topology_Report	—	G_Topology_Report request G_Topology_Report confirm	Provides a topology report related to devices in a wireless mesh. This interface may be useful if, for example, the gateway role is operating in a system in which the system management interface to the plant network is via the gateway role
Schedule_Report	—	G_Schedule_Report request G_Schedule_Report confirm	Provides detailed time slot and channel allocations on a per-device basis. This interface may be useful if, for example, the gateway role is operating in a system in which the system management interface to the plant network is via the gateway role
Device_Health_Report	—	G_Device_Health_Report request G_Device_Health_Report confirm	Device health report for devices associated with the gateway. This interface may be useful if, for example, the gateway role is operating in a system in which the system management interface to the plant network is via the gateway role
Neighbor_Health_Report	—	G_Neighbor_Health_Report request G_Neighbor_Health_Report confirm	Communication health report for the set of neighbor devices associated with a specific device that is associated with the gateway. This interface may be useful if, for example, the gateway role is operating in a system in which the system management interface to the plant network is via the gateway role

Interface example	Interface subtype	Primitive	Description
Network_Health_Report	—	G_Network_Health_Report request	Summary of communication health report for the wireless network. This interface may be useful if, for example, the gateway role is operating in a system in which the system management interface to the plant network is via the gateway role
		G_Network_Health_Report confirm	
Time	—	G_Time request	Retrieval and setting of time for the wireless network associated with the gateway
		G_Time confirm	
Client/server	—	G_Client_Server request	Provides client/server communication
		G_Client_Server indication	
		G_Client_Server response	
		G_Client_Server confirm	
Publish/subscribe	Publish	G_Publish request	Provides publish/subscribe communication
		G_Publish indication	
		G_Publish confirm	
	Subscribe	G_Subscribe request	
		G_Subscribe confirm	
	Publish_Timer	G_Publish_Timer indication	
	Subscribe_Timer	G_Subscribe_Timer indication	
	Watchdog_Timer	G_Watchdog_Timer indication	
Bulk_Transfer[1]	Open	G_Bulk_Open request	Allows upload and download of large items such as firmware images and sample buffers
		G_Bulk_Open confirm	
	Transfer	G_Bulk_Transfer request	
		G_Bulk_Transfer confirm	
	Close	G_Bulk_Close request	
		G_Bulk_Close confirm	
Alert	Subscribe	G_Alert_Subscription request	Allows subscription and receipt of specific alerts
		G_Alert_Subscription confirm	
	Notify	G_Alert_Notification indication	
Gateway_Configuration	Read	G_Read_Gateway_Configuration request	Provides read and write access to configuration attributes of the gateway
		G_Read_Gateway_Configuration confirm	
	Write	G_Write_Gateway_Configuration request	
		G_Write_Gateway_Configuration confirm	
Device_Configuration	Read	G_Read_Device_Configuration request	Allows the gateway to determine which devices are associated with it
		G_Read_Device_Configuration confirm	
	Write	G_Write_Device_Configuration request	
		G_Write_Device_Configuration confirm	
NOTE The interface primitives are common to both upload and download operations.			

U.2.2 Sequence of primitives

Figure U.3, Figure U.4, Figure U.5, Figure U.6, Figure U.7, Figure U.8, Figure U.9, Figure U.10, Figure U.11, Figure U.12, Figure U.13, Figure U.14, and Figure U.15 show the sequences of primitives for gateway high side interfaces. The figures are described in terms of a gateway-internal client, a gateway entity, a device client, and a device entity. A gateway-internal client is a user of the GIAP interfaces within a gateway. A gateway entity is a provider of GIAP interfaces within the gateway. The provision of the interfaces entails additional interactions across the wireless network to one or more devices. A device client is a user of GIAP interfaces within a device. A device entity is a provider of GIAP interfaces within the device.

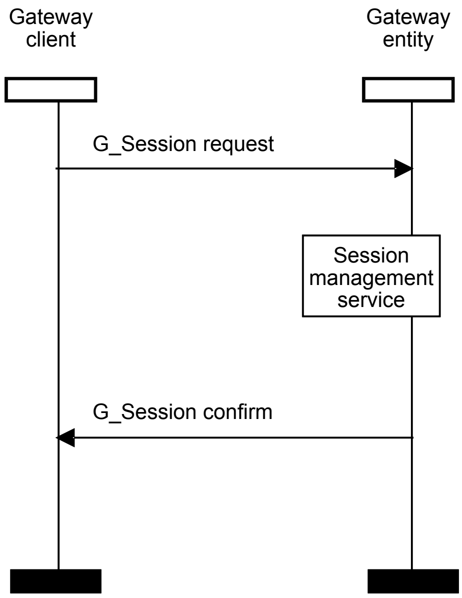


Figure U.3 – Internal sequence of primitives for session interface

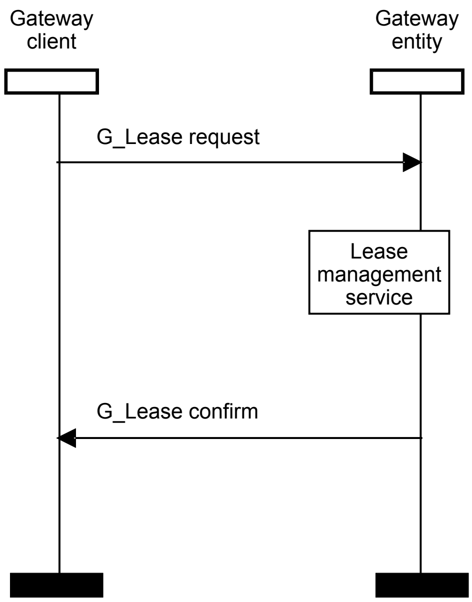


Figure U.4 – Internal sequence of primitives for lease management interface

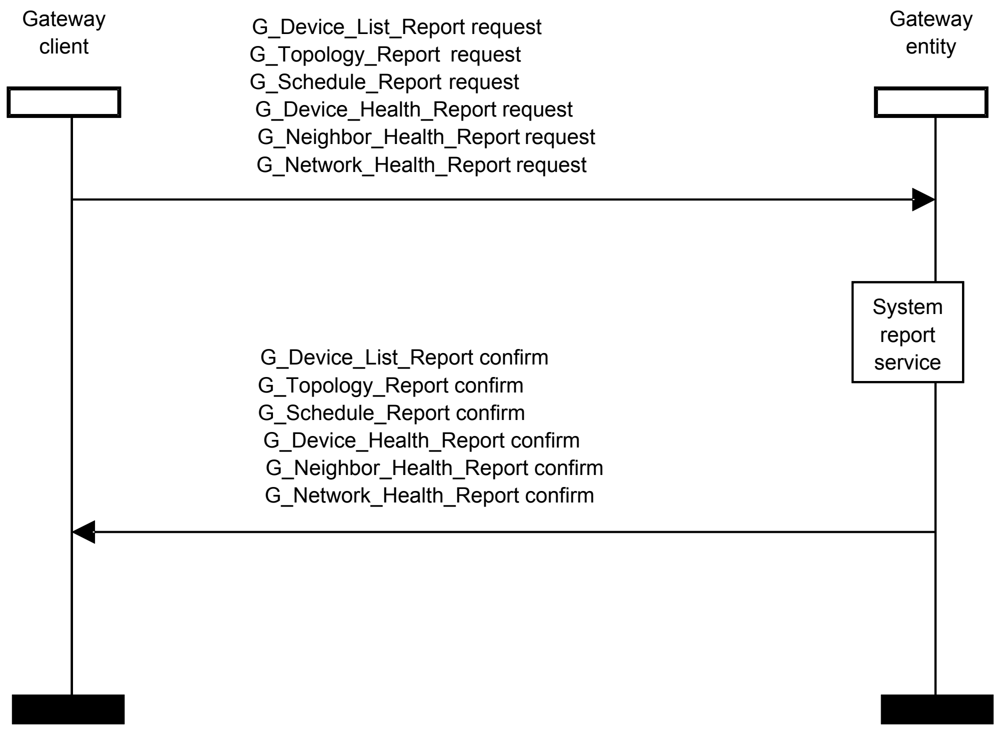


Figure U.5 – Internal sequence of primitives for system report interfaces

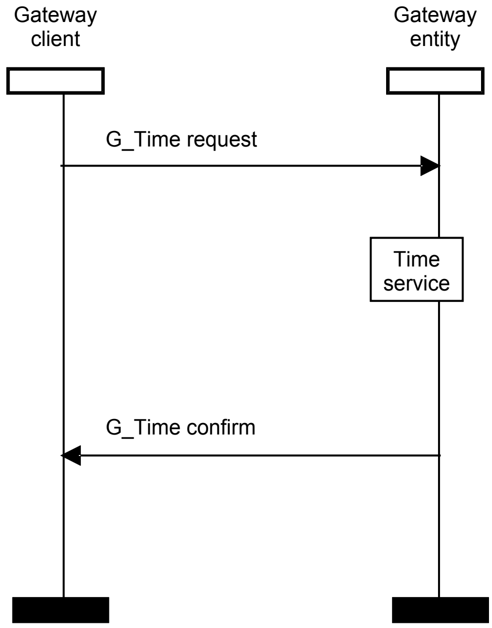


Figure U.6 – Internal sequence of primitives for time interface

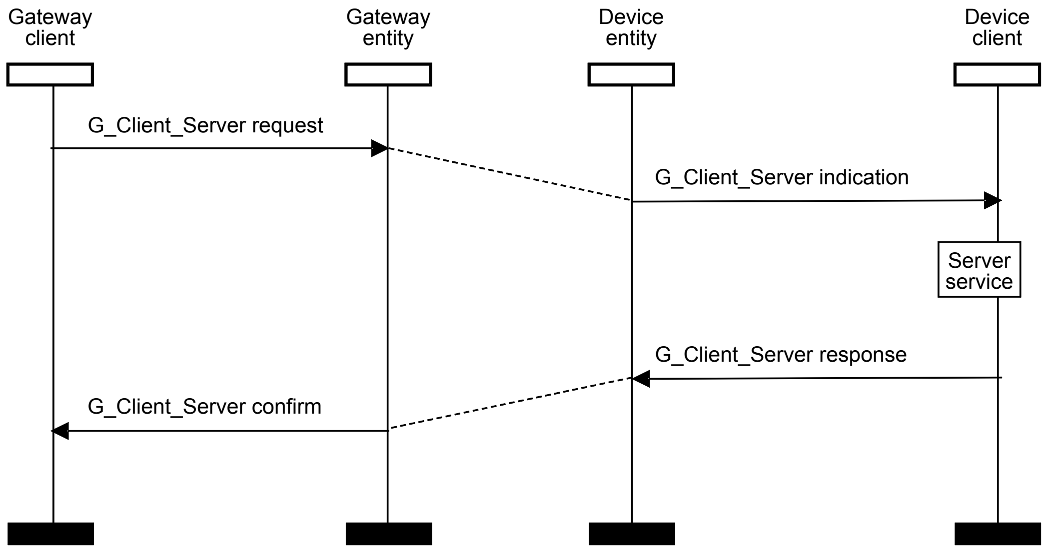


Figure U.7 – Internal sequence of primitives for Client/server interface initiated from gateway to an adapter device

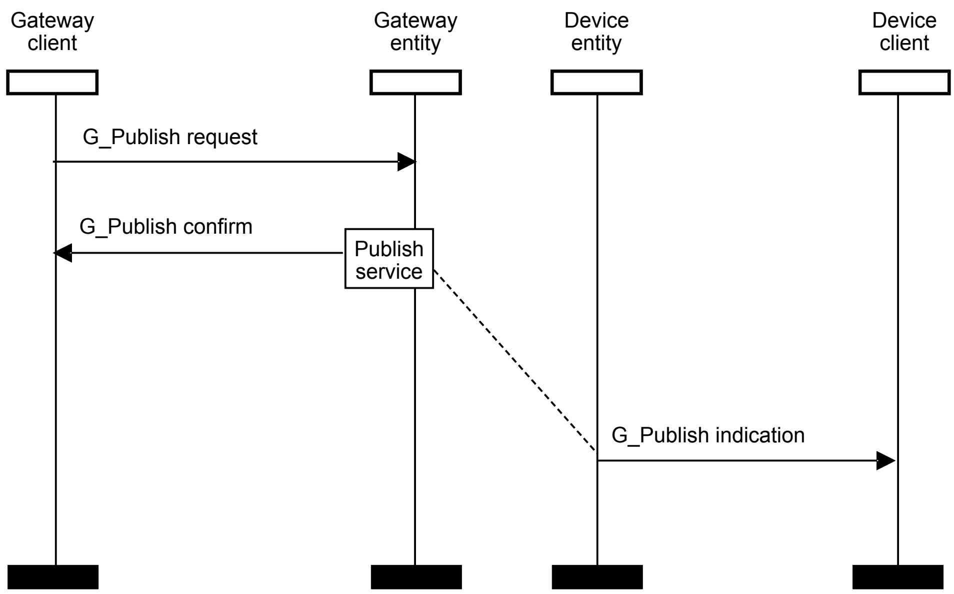


Figure U.8 – Internal sequence of primitives for publish interface initiated from gateway to an adapter device

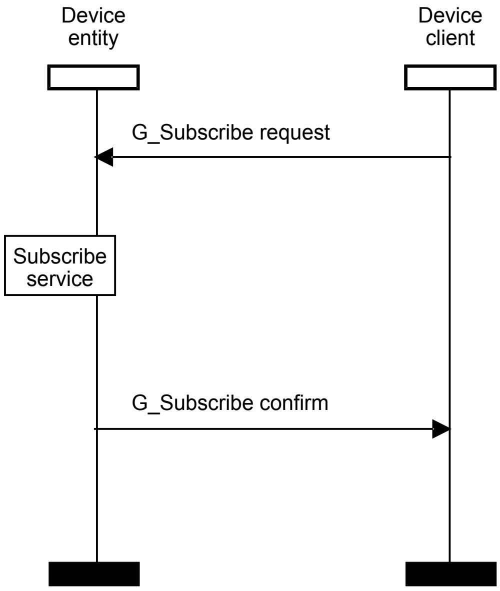


Figure U.9 – Internal sequence of primitives for subscribe interface initiated from an adapter device

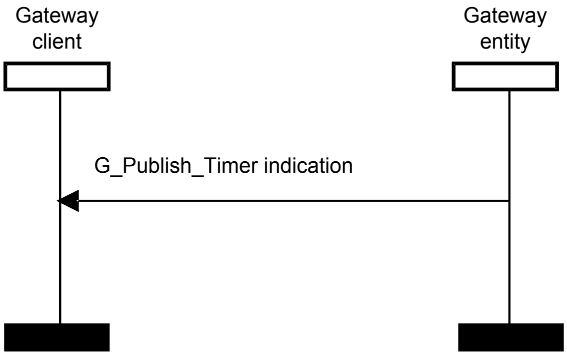


Figure U.10 – Internal sequence of primitives for publisher timer initiated from gateway to an adapter device

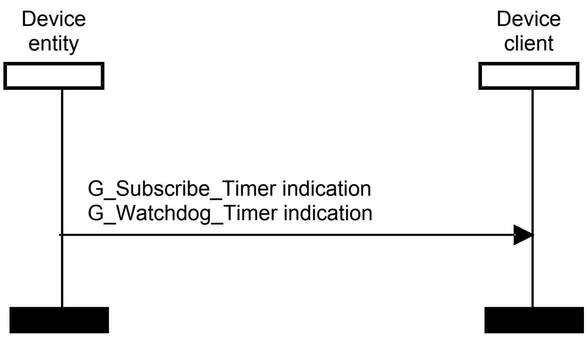


Figure U.11 – Internal sequence of primitives for subscriber timers initiated from an adapter device

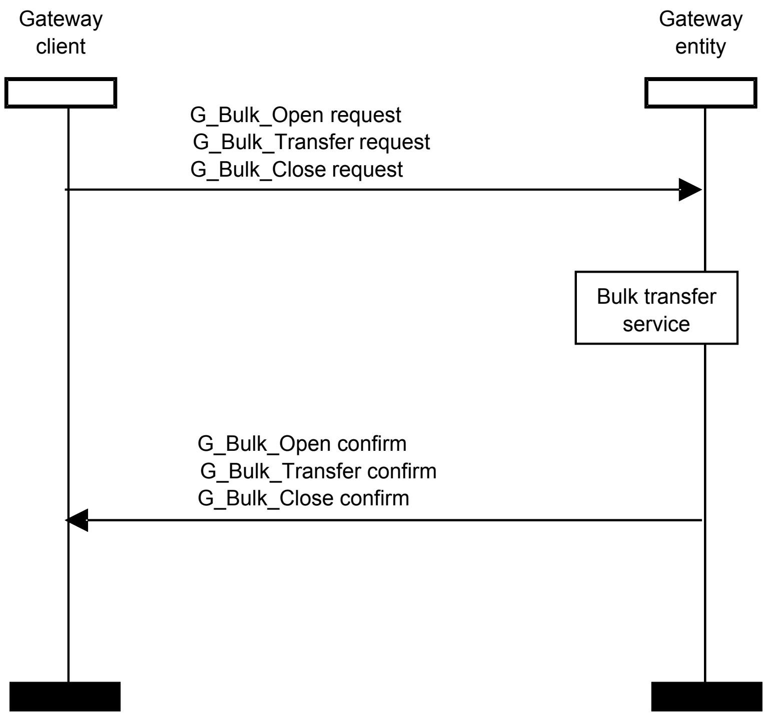


Figure U.12 – Internal sequence of primitives for the bulk transfer interface

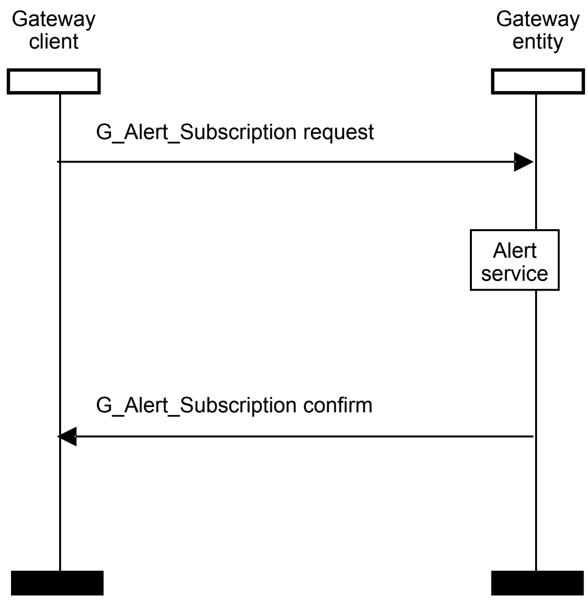


Figure U.13 – Internal sequence of primitives for the alert subscription interface

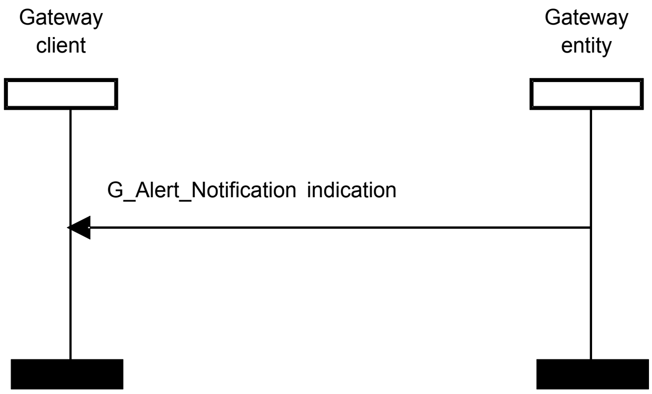


Figure U.14 – Internal sequence of primitives for the alert notification interface

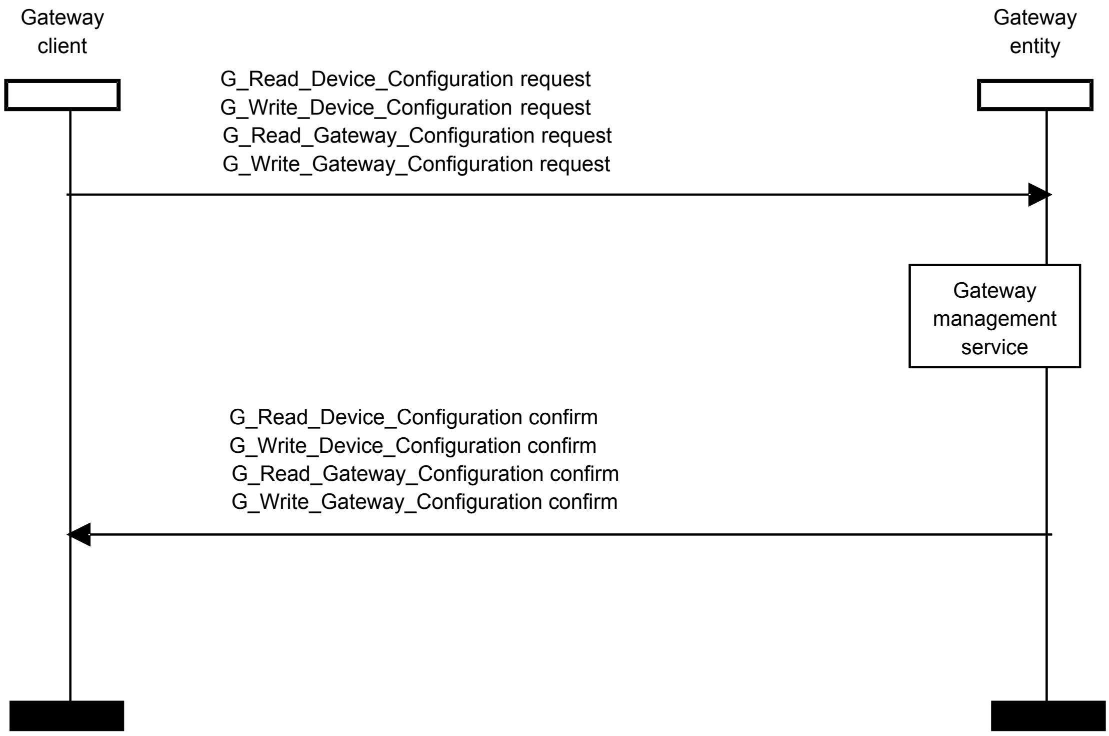


Figure U.15 – Internal sequence of primitives for gateway management interfaces

U.2.3 Detailed description of parameters

U.2.3.1 General

Parameters that are common to multiple interfaces are described in U.2.3. Parameters that are unique to an interface are described within that interface.

NOTE Since this standard does not define any gateways, and because the GIAP discussed is notional, all of these parameters are also strictly notional, as are all statements about them.

U.2.3.2 Parameter GS_Session_ID

The parameter GS_Session_ID uniquely identifies a specific session.

A valid session identifier that has not expired is provided in order to invoke all interfaces except the session interfaces.

U.2.3.3 Parameter GS_Transaction_ID

The parameter GS_Transaction_ID uniquely identifies request and response portions of a transaction within the context of a specific session.

The transaction identifier is used to match a GIAP interface request with the corresponding GIAP interface response.

U.2.3.4 Parameter GS_Lease_ID

GS_Lease_ID identifies gateway entity resources and communication resources that are allocated to a particular session.

The lease identifier is provided by the GIAP interface user when an interface is invoked to identify the particular communication resources used to support the interface.

U.2.3.5 Parameter GS_Status

GS_Status is returned by a confirm primitive. It may represent either the status resulting from handling a local request or the status corresponding to a response received from a remote entity.

The status indicates the success or failure of the interface call and, when applicable, the reason for failure.

U.2.3.6 Parameter GS_Network_Address

GS_Network_Address is an IPv6Address used to identify a logical device that is unique across all networks.

This parameter uniquely identifies a specific NLE.

U.2.3.7 Parameter GS_Unique_Device_ID

GS_Unique_Device_ID is a an EUI64Address.

This parameter uniquely identifies a specific physical device for asset management purposes.

U.2.3.8 Parameter GS_Network_ID

GS_Network_ID is a unique identifier for one of several networks that may be accessible through a single gateway.

This parameter uniquely identifies a specific network.

U.2.3.9 Parameter GS_Time

GS_Time is a 48-bit TAI time field.

The time parameter is used to describe time-related fields such as timestamp, start time, and stop time.

U.2.3.10 Parameter GS_Transfer_Mode

GS_Transfer_Mode identifies GPDU-level transfer variations.

GS_Transfer_Mode is provided with each GPDU provided for transfer in order to indicate the desired quality of service and the priority associated with the transfer of the PDUs generated to support the notional interface primitive.

U.2.4 Detailed description of interfaces

U.2.4.1 Session management interface

U.2.4.1.1 General

A gateway entity is a process within a gateway that provides gateway interfaces through the GIAP. A gateway-internal client is a user of gateway entity-provided interfaces. Typical gateway-internal clients include host systems, asset management systems, and engineering tools.

Gateway entities provide interfaces to gateway-internal clients within the context of a session. The session management interface is used to establish and manage these sessions. All other gateway entity-provided interfaces are used within the context of an established session.

A foreign protocol translator within the gateway may establish sessions on behalf of remote clients and perform protocol translation on the communication flows that correspond to gateway entity-provided interfaces.

The primary purpose of a session is to allow resource allocation and bulk reclamation of gateway and communication resources on a per-gateway entity client basis.

A session may be established by a local process or remotely (such as through a TCP/IP remote session).

One or more sessions may exist concurrently between a gateway entity and one or more gateway-internal clients. Each session is uniquely identified.

NOTE The number of concurrent sessions supported is implementation-dependent. It is possible that some implementations provide a fixed function gateway with a single session, while other implementations provide a number of sessions that are allocated on demand to a variety of applications, including host systems, historians, asset management tools, and engineering tools.

The gateway-internal client uses the G_Session primitive to create, renew or delete a session.

U.2.4.1.2 G_Session primitive

U.2.4.1.3 Primitives and their parameters

Table U.2 describes parameter usage for the primitive G_Session.

Table U.2 – Primitive G_Session parameter usage

Parameter name	G_Session	
	Request	Confirm
GS_Session_ID	M	M
GS_Session_Period	M	M
GS_Network_ID	M	—
GS_Status	—	M

U.2.4.1.4 Use of G_Session request

The gateway-internal client uses the primitive G_Session request to create, renew or delete a session.

A session is created by providing a null session identifier (GS_Session_ID = 0) and a requested session duration. Submitting GS_Session_Period > 0 requests a limited duration session, specified in seconds. Submitting GS_Session_Period = -1 requests an indefinite session duration.

A limited duration session is renewed by providing an existing (non-null) session identifier and session duration greater than 0 s. An indefinite duration session does not need to be renewed.

Changing a limited duration session to an indefinite duration session by attempting to renew it with a specified duration of -1 s is not permitted.

NOTE The upper bound of session duration is implementation-dependent. For instance, implementations are able to dedicate resources to specific applications, such as a host system, never releasing those resources.

A session is deleted by providing an existing (non-null) session identifier and session duration of 0 s.

A gateway may connect to multiple networks. Each session is associated with a specific network. A network identifier (GS_Network_ID) is specified to establish a particular network for the session. The scope of further identifiers used within a session is limited to the particular network.

U.2.4.1.5 Use of G_Session confirm

The gateway entity uses the G_Session confirm primitive to complete the G_Session request to the gateway-internal client.

For a successful session creation request, the gateway entity returns a unique, non-null session identifier. This identifier is used in subsequent session renew and delete operations. GS_Session_Period is returned with the actual session duration allocated by the gateway entity.

The GS_Session_Period value returned may not be the same as the value requested.

For a session renew request, the request session identifier is echoed, and a new session duration is returned.

For a session deletion request, the session identifier is echoed, and the session duration is set to 0 s.

GS_Status is returned to indicate the success or failure of the operation, as described in Table U.3.

Table U.3 – GS_Status for G_Session confirm

Value	Meaning
0	Success; new session created, renewed or deleted
1	Success; new session created or renewed with reduced period
2	Failure; session does not exist to renew or delete
3	Failure; session cannot be created (no additional sessions available) This may occur, for example, if sessions have expired, but have not explicitly been deleted
4	Failure; other

U.2.4.2 Lease management interface

U.2.4.2.1 General

Gateway entities allocate communication resources to gateway-internal clients via leases. The lease management interface is used to establish and manage leases.

The primary purpose of a lease is to allow fine-grained communication resource allocation and reclamation on a per-session basis.

Resources may be separately allocated depending on communication needs. For example, client/server, publish/subscribe, bulk transfer, and alert subscription resources may be separately allocated.

One or more leases may exist concurrently between a gateway entity and one or more gateway-internal clients. Each lease is uniquely identified within a session.

U.2.4.2.2 Use of the interface

The gateway-internal client uses the G_Lease primitive to create, renew or delete a lease.

U.2.4.2.3 G_Lease primitive

U.2.4.2.4 Primitives and their parameters

Table U.4 describes parameter usage for the primitive G_Lease.

Table U.4 – Primitive G_Lease parameter usage

Parameter name	G_Lease	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	M
GS_Lease_Period	M	M
GS_Lease_Type	M	—
GS_Protocol_Type	M	—
GS_Network_Address_List	C	—
GS_Network_Address	C	—
GS_Resource	C	—
GS_Lease_Parameters	C	—
GS_Transfer_Mode	C	—
GS_Update_Policy	C	—
GS_Period	C	—
GS_Phase	C	—
GS_Stale_Limit	C	—
GS_Connection_Info	C	—
GS_Wireless_Parameters	C	—
GS_Status	—	M

U.2.4.2.5 Use of G_Lease request

The gateway-internal client uses the primitive G_Lease request to create, renew, or delete a lease.

A session identifier (GS_Session_ID) is included in the G_Lease request primitives.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

A lease is created by providing a null lease identifier (GS_Lease_ID = 0) and a requested lease duration. Submitting GS_Lease_Period > 0 requests a limited duration lease, specified in seconds. Submitting GS_Lease_Period = 0 requests an indefinite lease duration.

A limited duration lease is renewed by providing an existing (non-null) lease identifier and lease duration greater than 0 s. An indefinite duration lease does not need to be renewed. A limited duration lease cannot be changed to an indefinite duration lease by renewal with duration of 0 s.

The maximum supported value is implementation-dependent. Implementations can choose to dedicate resources to specific applications, such as a host system, never releasing those resources.

A lease is deleted by providing an existing lease identifier and a requested lease duration of 0 s.

Different types of leases are available, as specified by GS_Lease_Type and as shown in Table U.5. Each lease type allocates lease-specific gateway entity resources and communication resources on behalf of the gateway-internal client.

Table U.5 – GS_Lease_Type for G_Lease request

Value	Meaning
0	Client
1	Server
2	Publisher
3	Subscriber
4	Bulk transfer client
5	Bulk transfer server
6	Alert subscription

GS_Protocol_Type identifies the protocol that is associated with the lease, as indicated in Annex M. Specification of the protocol type allows special processing for particular protocols within the gateway entity.

All leases relate to establishing communication interfaces between the gateway entity and one or more device specified by one or more elements in GS_Network_Address_List. Alert subscription leases do not allocate communication resources during lease establishment, but dynamically as alert subscriptions are modified. GS_Network_Address_List is not used with the alert subscription lease type.

Client, server, subscriber, bulk transfer client, and bulk transfer server describe only a single element within GS_Network_Address_List. The publisher lease type may describe multiple elements.

The specification of multiple IPv6Addresses within the GS_Network_Address_List represents a multicast group. Elements within the G_Network_Address_List include GS_Network_Address.

GS_Lease_Parameters is a parameter structure for the specification of parameters necessary for the establishment of certain lease types. Usage of the GS_Lease_Parameters is conditioned on the specific lease type as follows.

NOTE Annex P provides additional information on detailed GS_Lease_Parameters usage.

Client, server, publish, and subscribe leases describe a unique GS_Resource. This value identifies matching client and server connection endpoints, and also identifies matching publisher and subscriber endpoints. GS_Resource also is specified by the bulk transfer client to identify the upload/download item.

Publisher leases require specification of GS_Update_Policy, GS_Period, GS_Phase, and GS_Stale_Limit to control timing and buffered behavior. GS_Transfer_Mode also is specified in order to set the default transfer quality of interface and priority.

Subscriber leases require specification of GS_Update_Policy, GS_Period, GS_Phase, and GS_Stale_Limit to control timing and buffered behavior.

A publisher and subscriber may agree to describe GS_Connection_Info in the subscriber lease for provision on each publication receipt.

Client and server leases describe GS_Transfer_Mode in order to set the default transfer quality of interface and priority.

An additional GS_Wireless_Parameters field usage depends on gateway construction. This allows access to all exposed, requestable communication features.

U.2.4.2.6 Use of G_Lease confirm

The gateway entity uses the primitive G_Lease confirm to complete the G_Lease request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm primitive with the original request primitive.

For a successful lease create request, the gateway entity returns a session unique lease identifier. This lease identifier is used in subsequent lease renew and delete operations. GS_Lease_Period is returned with the actual lease duration allocated by the gateway entity.

For a lease renew request, the request lease identifier is echoed, and the actual lease duration is given.

For a lease delete request, the lease identifier is echoed, and the lease duration is set to 0 s.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.6.

Table U.6 – GS_Status for G_Lease confirm

Value	Meaning
0	Success; new lease created, renewed or deleted
1	Success; new lease created or renewed with reduced period
2	Failure; lease does not exist to renew or delete
3	Failure; no additional leases available
4	Failure; no device exists at IPv6Address
5	Failure; invalid lease type
6	Failure; invalid lease type information
7	Failure; other

U.2.4.3 Device list report interface

U.2.4.3.1 General

The device list report interface provides a report of the devices that are associated with a gateway. This is useful for mapping wireless devices to host systems and network browsers.

The gateway-internal client uses the G_Device_List_Report primitive to retrieve a report on the devices associated with a gateway entity.

U.2.4.3.2 G_Device_List_Report primitive

U.2.4.3.3 Primitives and their parameters

Table U.7 describes parameter usage for the primitive G_Device_List_Report.

Table U.7 – Primitive G_Device_List_Report parameter usage

Parameter name	G_Device_List_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	—	M
GS_Network_Address	—	M
GS_Device_Type	—	M
GS_Unique_Device_ID	—	M
GS_Manufacturer	—	M
GS_Model	—	M
GS_Revision	—	M
GS_Status	—	M

The gateway-internal client uses the primitive G_Device_List_Report request to retrieve a report on the devices associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

U.2.4.3.4 Use of G_Device_List_Report confirm

The gateway entity uses the primitive G_Device_List_Report confirm to complete the G_Device_List_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A list of devices associated with the gateway entity (GS_Device_List) is returned. For each device, the list includes the IPv6Address (GS_Network_Address), the type of the device (GS_Device_Type), and the unique device identifier (GS_Unique_Device_ID).

The list also includes additional manufacturer related information (GS_Manufacturer, GS_Model, and GS_Revision).

Where the gateway includes the role of the provisioning device, the IPv6Address may be a default address. The unique identifier and the manufacturer information are used within the host-level applications to control device commissioning through the device configuration interface.

For example, a browser may display a list of devices available for provisioning along with identification information. A select set of the devices are picked from the display and are commissioned with the IPv6Address and other information to join the system. The browsing display is then refreshed with devices that are available for linkage to a control strategy.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

Table U.8 – GS_Status for G_Device_List_Report confirm

Value	Meaning
0	Success
1	Failure

U.2.4.4 Topology report interface

U.2.4.4.1 General

In system configurations where access to system management information is via the gateway, the topology report interface provides a topology report that relates devices within a wireless mesh.

The gateway-internal client uses the G_Topology_Report primitive to retrieve a report on the devices associated with a gateway entity.

U.2.4.4.2 G_Topology_Report primitive

U.2.4.4.3 Primitives and their parameters

Table U.9 describes parameter usage for the primitive G_Topology_Report.

Table U.9 – Primitive G_Topology_Report parameter usage

Parameter name	G_Topology_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	—	M
GS_Network_Address	—	M
GS_Neighbor_List	—	M
GS_Network_Address	—	M
GS_Graph_List	—	M
GS_Graph_ID	—	M
GS_Network_Address	—	M
GS_Status	—	M

U.2.4.4.4 Use of G_Topology_Report request

The gateway-internal client uses the primitive G_Topology_Report request to retrieve a report on the topology of the devices associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

U.2.4.4.5 Use of G_Topology_Report confirm

The gateway entity uses the primitive G_Topology_Report confirm to complete the G_Topology_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of a confirm with the original request.

A list of devices associated with the gateway entity (GS_Device_List) is returned. The list includes the IPv6Address (GS_Network_Address) for each device.

Also included within the list is a second list (GS_Neighbor_List) of the neighbor devices associated with each device. The list includes the IPv6Address (GS_Network_Address) of all neighbors (as described in the neighbor health report).

Also included within the list is a third list (GS_Graph_List) of the graph connections associated with each device. For each graph connection, the list includes the graph identified (GS_Graph_ID) and an associated IPv6Address list (GS_Network_Address) of the neighbors on the graph.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.5 Schedule report interface

U.2.4.5.1 General

In system configurations where access to system management information is via the gateway, the schedule report interface provides a schedule report detailing time slot and channel allocations on a per-device basis.

The gateway-internal client uses the G_Schedule_Report primitive to retrieve a report on the schedule of the devices associated with a gateway entity.

U.2.4.5.2 G_Schedule_Report primitive

U.2.4.5.3 Primitives and their parameters

Table U.10 describes parameter usage for the primitive G_Schedule_Report.

Table U.10 – Primitive G_Schedule_Report parameter usage

Parameter name	G_Schedule_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Address	M	—
GS_Channel_List	—	M
GS_Channel_Number	—	M
GS_Channel_Status	—	M
GS_Device_Schedule	—	M
GS_Network_Address	—	M
GS_Superframe_List	—	C
GS_Superframe_ID	—	C
GS_Num_Time_Slots	—	C
GS_Start_Time	—	C
GS_Link_List	—	C
GS_Network_Address	—	C
GS_Slot_Size	—	C
GS_Channel	—	C
GS_Direction	—	C
GS_Link_Type	—	C
GS_Status	—	M

U.2.4.5.4 Use of G_Schedule_Report request

The gateway-internal client uses the primitive G_Schedule_Report request to retrieve a schedule report for a specific device associated with a gateway entity. The particular device is identified by its IPv6Address (GS_Network_Address).

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

U.2.4.5.5 Use of G_Schedule_Report confirm

The gateway entity uses the primitive G_Schedule_Report confirm to complete the G_Schedule_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A list of channels is returned. Each element of the list includes the channel number (GS_Channel_Number) and the status of the channel (GS_Channel_Status). Channel status is set to 0 to indicate a disabled channel or to 1 to indicate an enabled channel.

A device schedule (GS_Device_Schedule) is also returned. The schedule includes device identification information (GS_Network_Address) and a list of superframes (GS_Superframe_List) that are used by the device for communication.

If a device does not use superframes, GS_Superframe_List is not returned.

The superframe list includes general superframe information, including a superframe identifier (GS_Superframe_ID), the number of time slots in the superframe (GS_Num_Time_Slots), and the start time of the superframe (GS_Start_Time). Only active superframes are reported.

GS_Start_Time is an offset relative to the beginning of TAI time. This number has significance only relative to the current network time, unless the communication is synchronized to an external source. GS_Start_Time is set to -1 to indicate that the superframe has no known synchronization.

The superframe list also includes an ordered list (GS_Link_List) with one element per timeslot in the superframe. The link list elements are used to describe communication relationships related to the superframe. Each link list element describes the timeslot duration in microseconds (GS_Slot_Size) and the channel (GS_Channel) for communication within the superframe. The link (GS_Direction) parameter describes the direction of communications. A value of 0 describes reception, and a value of 1 describes transmission. The IPv6Address (GS_Network_Address) describes the logical IPv6Address of a communication partner for the slot.

The link type (GS_Link_Type) describes the purpose of the communication:

- A value of 0 describes aperiodic data communication.
- A value of 1 describes aperiodic management communication.
- A value of 2 describes periodic data communication.
- A value of 3 describes periodic management communication.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.6 Device health report interface

U.2.4.6.1 General

The device health report interface provides a communication health report for each device's view of its own health.

The gateway-internal client uses the G_Device_Health_Report primitive to retrieve a device health report for a specified set of devices that are associated with a gateway entity.

U.2.4.6.2 G_Device_Health_Report primitive

U.2.4.6.3 Primitives and their parameters

Table U.11 describes parameter usage for the primitive G_Device_Health_Report.

Table U.11 – Primitive G_Device_Health_Report parameter usage

Parameter name	G_Device_Health_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	M	M
GS_Network_Address	M	M(=)
GS_DPDUs_Transmitted	—	M
GS_DPDUs_Received	—	M
GS_DPDUs_Failed_Transmission	—	M
GS_DPDUs_Failed_Reception	—	M
GS_Status	—	M

U.2.4.6.4 Use of G_Device_Health_Report request

The gateway-internal client uses the primitive G_Device_Health_Report request to retrieve a device health report for a specified set of devices that are associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

A health report is requested for a specific list of devices (GS_Device_List). The IPv6Address of each device (GS_Network_Address) is required.

U.2.4.6.5 Use of G_Device_Health_Report confirm

The gateway entity uses the primitive G_Device_Health_Report confirm to complete the G_Device_Health_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A device list (GS_Device_List) is returned. The list includes device identification information (GS_Network_Address) and communication health information. The communication health information includes the total number of DPDUs transmitted (GS_DPDUs_Transmitted) from the device to all neighbors, the total number of DPDUs received (GS_DPDUs_Received) from the device by all neighbors, the total number of DPDUs to all neighbors that failed transmission (GS_DPDUs_Failed_Transmission), and the total number of DPDUs from all neighbors that failed reception (GS_DPDUs_Failed_Reception). Failed receptions include identifiable DPDUs that are discarded due to transmission-related corruption.

NOTE Failed receptions will likely be less than failed transmissions, since many failed DPDUs will not have enough uncorrupted information to determine the addressing. Failed reception does not include protocol-related errors.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.7 Neighbor health report interface

U.2.4.7.1 General

In system configurations where access to system management information is via the gateway, the neighbor health report interface provides a communication health report for each device's view of its neighbors.

A neighbor device is a link level wireless communication partner that is configured for direct exchange of DPDUs (RF transmission without hops). The neighbor health report interfaces provide information on these physical neighbors. Neighbor devices are able to collect DPDU exchange statistics that indicate local RF conditions.

The gateway-internal client uses the G_Neighbor_Health_Report primitive to retrieve a communication health report for the set of neighbor devices associated with a specific device that is associated with a gateway entity.

U.2.4.7.2 G_Neighbor_Health_Report primitive

U.2.4.7.3 Primitives and their parameters

Table U.12 describes parameter usage for the primitive G_Neighbor_Health_Report.

Table U.12 – Primitive G_Neighbor_Health_Report parameter usage

Parameter name	G_Neighbor_Health_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Address	M	—
GS_Neighbor_Health_List	—	M
GS_Network_Address	—	M
GS_Link_Status	—	M
GS_DPDUs_Transmitted	—	M
GS_DPDUs_Received	—	M
GS_DPDUs_Failed_Transmission	—	M
GS_DPDUs_Failed_Reception	—	M
GS_Signal_Strength	—	M
GS_Signal_Quality	—	M
GS_Status	—	M

U.2.4.7.4 Use of G_Neighbor_Health_Report request

The gateway-internal client uses the primitive G_Neighbor_Health_Report request to retrieve a communication health report for the set of neighbor devices associated with a specific device that is associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

A neighbor health report is requested for a device at a specific IPv6Address (GS_Network_Address).

U.2.4.7.5 Use of G_Neighbor_Health_Report confirm

The gateway entity uses the primitive G_Neighbor_Health_Report confirm to complete the G_Neighbor_Health_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A neighbor health list (GS_Neighbor_Health_List) is returned. The list includes the neighbor device identification information (GS_Network_Address) and communication health information. The communication health information includes a general status (GS_Link_Status). GS_Link_Status = 1 indicates that the neighbor is available for communication. GS_Link_Status = 0 indicates that the neighbor is unavailable for communication.

Health information also includes the number of DPDUs transmitted to the neighbor (GS_DPDUs_Transmitted), the number of DPDUs received from the neighbor (GS_DPDUs_Received), the number of failed transmission attempts (GS_DPDUs_Failed_Transmission), and the number of failed receptions (GS_DPDUs_Failed_Reception) from the neighbor. Failed receptions include identifiable DPDUs that are discarded due to transmission related corruption.

NOTE Failed receptions will likely be less than failed transmissions, since many failed DPDUs will not have enough uncorrupted information to determine the addressing. Failed reception does not include protocol-related errors.

Health information also includes GS_Signal_Strength and GS_Signal_Quality. These parameters return values between 0 (worst signal) and 100 (best signal). GS_Signal_Strength indicates the average uncorrelated power level of the signals received from a specific neighbor relative to the range of the receiver. GS_Signal_Quality indicates the average correlated power level of the signals received from a specific neighbor relative to the range of the receiver.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.8 Network health report interface

U.2.4.8.1 General

In system configurations where access to system management information is via the gateway, the neighbor health report interface provides a communication health report for each device’s view of its neighbors.

The gateway-internal client uses the G_Network_Health_Report primitive to retrieve a summary communication health report for an entire network.

U.2.4.8.2 G_Network_Health_Report primitive

U.2.4.8.3 Primitives and their parameters

Table U.13 describes parameter usage for the primitive G_Network_Health_Report.

Table U.13 – Primitive G_Network_Health_Report parameter usage

Parameter name	G_Network_Health_Report	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Health	—	M
GS_Network_ID	—	M
GS_Network_Type	—	M
GS_Device_Count	—	M
GS_Start_Date	—	M
GS_Current_Date	—	M
GS_DPDUs_Sent	—	M
GS_DPDUs_Lost	—	M
GS_GPDU_Latency	—	M
GS_GPDU_Path_Reliability	—	M
GS_GPDU_Data_Reliability	—	M
GS_Join_Count	—	M
GS_Device_Health_List	—	M
GS_Network_Address	—	M
GS_Start_Date	—	M
GS_Current_Date	—	M
GS_DPDUs_Sent	—	M
GS_DPDUs_Lost	—	M
GS_GPDU_Latency	—	M
GS_GPDU_Path_Reliability	—	M
GS_GPDU_Data_Reliability	—	M
GS_Join_Count	—	M
GS_Status	—	M

U.2.4.8.4 Use of G_Network_Health_Report request

The gateway-internal client uses the primitive G_Network_Health_Report request to retrieve a summary communication health report for an entire network.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

U.2.4.8.5 Use of G_Network_Health_Report confirm

The gateway entity uses the primitive G_Network_Health_Report confirm to complete the G_Network_Health_Report request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A network health summary (GS_Network_Health) is returned. The summary includes network identification information (GS_Network_ID and GS_Network_Type) and network communication health summary information. The communication health information includes the number of devices in the network (GS_Device_Count), the start date and the current date for the network (GS_Start_Date and GS_Current_Date), transmission statistics (GS_DPDUs_Sent, GS_DPDUs_Lost, and GS_GPDU_Latency), reliability statistics (GS_GPDU_Path_Reliability and GS_GPDU_Data_Reliability), and join statistics (GS_Join_Count).

A device-specific health summary (GS_Device_Health_List) is also returned. The list includes device identification information (GS_Network_Address) and communication statistics that are an identical subset of those contained in the network health summary (GS_Start_Date, GS_Current_Date, GS_DPDUs_Sent, GS_DPDUs_Lost, GS_GPDU_Latency, GS_GPDU_Path_Reliability, GS_GPDU_Data_Reliability, and GS_Join_Count).

GS_Start_Date is a 48-bit TAI time field indicating the time when a device first started operating. This is useful for calculation of battery replacement schedules.

GS_Current_Date is a 48-bit TAI time field indicating the current time as viewed by the device. This is the time used by the device for timestamp purposes.

GS_GPDU_Latency is a number from 0..100 indicating the percentage of scheduled GPDUs that arrive later than expected. These GPDUs may be delayed due to delivery over secondary paths or due to congestion in intermediate devices.

GS_GPDU_Path_Reliability is a number from 0..100 indicating the percentage of first path success for acknowledged GPDU transmission. GPDUs that are transmitted on a secondary path may arrive successfully, but may reduce the path reliability.

GS_GPDU_Data_Reliability is a number from 0..100 indicating the percentage of total GPDUs that are successful GPDUs. The total GPDUs are the number of acknowledged transmit GPDUs that are attempted plus the number of received GPDUs. Successful GPDUs are acknowledged transmit GPDUs that are transferred correctly on the first attempt plus receive GPDUs that pass integrity checks.

GS_Join_Count is a positive integer that indicates the number of times a device has joined the system. Join count may rise if power is interrupted, a device is reset, the network is reformed, or a device is moved to a new network. Excessive joins may indicate device integrity or communication problems.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.9 Time interface

U.2.4.9.1 General

The time interface enables retrieval and setting of the time for a wireless network associated with a gateway. This is useful for time synchronization of a network of wireless devices with a host system and other host-level applications.

The gateway-internal client uses the G_Time primitive to retrieve a report on the devices associated with a gateway entity.

U.2.4.9.2 G_Time primitive

U.2.4.9.3 Primitives and their parameters

Table U.14 describes parameter usage for the primitive G_Time.

Table U.14 – Primitive G_Time parameter usage

Parameter name	G_Time	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Command	M	—
GS_Time	C	M
GS_Status	—	M

U.2.4.9.4 Use of G_Time request

The gateway-internal client uses the primitive G_Time request to read or set the network time.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

GS_Command = 0 reads the network time. GS_Time is not included.

GS_Command = 1 attempts to set the network time. A new time (GS_Time) is provided.

U.2.4.9.5 Use of G_Time confirm

The gateway entity uses the primitive G_Time confirm to complete the G_Time request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

If GS_Command = 0 in the request, the current network time (GS_Time) is returned.

If GS_Command = 1 in the request, the interface attempts to set the network time. The current network time (GS_Time) is returned. If the update is successful, the current time will reflect the change.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.15.

Table U.15 – GS_Status for G_Time confirm

Value	Meaning
0	Success
1	Failure; not allowed to set time in this configuration
2	Failure; other

U.2.4.10 Client/server interface

U.2.4.10.1 General

The client/server interface provides for client/server data transfer. The necessary communication resources to enable the transfer are allocated through the use of the lease interface. The client and server each perform separate but related roles. Linkage of the client and the server is accomplished through the establishment of leases with matching lease information. Communication resources include local buffer facilities in order to minimize energy consuming transactions. Clients and servers may exist either in the gateway or in devices.

The G_Client_Server primitive is used to send an internal client request data payload to a server and to initiate receipt of a corresponding server response data payload. Depending on implementation, the response payload may come from the internal client buffer within the gateway or from the field device.

U.2.4.10.2 G_Client_Server primitive

U.2.4.10.3 Primitives and their parameters

Table U.16 describes parameter usage for the primitive G_Client_Server.

Table U.16 – Primitive G_Client_Server parameter usage

Parameter name	G_Client_Server			
	Request	Indication	Response	Confirm
GS_Session_ID	M	—	—	M(=)
GS_Transaction_ID	M	—	—	M(=)
GS_Lease_ID	M	—	—	—
GS_Buffer	M	—	—	—
GS_Transfer_Mode	M	—	M	—
GS_Request_Data	M	C(=)	—	—
GS_Response_Data	—	—	C	M
GS_Transaction_Info	C	—	—	C(=)
GS_Status	—	—	M	M

U.2.4.10.4 Use of G_Client_Server request

The primitive G_Client_Server request is used to either attempt to acquire the requested data locally from the gateway (if GS_Buffer = 1), or to send a corresponding WISN native client application data request using the content of the data payload (GS_Request_Data) parameter to a WISN communicating device and to initiate receipt of a corresponding WISN server response. Whether the requested data is accessed locally or remotely, the data is returned via the response/confirm primitive parameter for the response payload (GS_Response_Data).

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

The server device is known through the lease identifier (GS_Lease_ID) that was obtained from the lease interface.

The response data will be requested from the server device if the buffer is disabled for the transaction (GS_Buffer = 0). The response data will be delivered from the buffer if the buffer is enabled for the transaction (GS_Buffer = 1) and the buffer contains a matching response that has not expired.

GS_Transfer_Mode is provided with the request in order to indicate the quality of interface and priority for the transfer of the data.

If GS_Transaction_Info is provided as part of a request, it is returned by the corresponding confirm primitive.

U.2.4.10.5 Use of G_Client_Server indication

The primitive G_Client_Server indication is used to signal the arrival of a client request data payload at the server for processing.

The indication is conditional on whether the server response data payload could be delivered from the client buffer.

U.2.4.10.6 Use of G_Client_Server response

The primitive G_Client_Server response is used to return a server response data payload to the client.

GS_Transfer_Mode is provided with the response in order to indicate the quality of interface and priority for the transfer of the data.

The response is conditional on whether the server response data payload could be delivered from the client buffer.

U.2.4.10.7 Use of G_Client_Server confirm

The primitive G_Client_Server confirm is used to complete the G_Client_Server request to the client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm primitive with the original request primitive.

A server response data payload is returned. The payload is either delivered from the client buffer or from the server.

If GS_Transaction_Info was provided in the request, it will be returned in the confirm.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.17.

Table U.17 – GS_Status for G_Client_Server confirm

Value	Meaning
0	Success
1	Failure; server is inaccessible for unbuffered request
2	Failure; server is inaccessible and client buffer is invalid for buffered request
3	Failure; lease has expired
4	Failure; other

U.2.4.11 Publish/subscribe interface

U.2.4.11.1 General

The publish/subscribe interface provides mechanisms for publish/subscribe data transfer. The necessary communication resources to enable message exchange are allocated through the use of the lease interface. The publisher and the subscriber each perform separate but related roles. Linkage of the publisher and the subscriber is accomplished through separate establishment of matching communication. Communication resources include local buffer facilities in both the publisher and the subscriber in order to minimize energy consuming transactions. Publishers and subscribers may exist in gateways, adapters, or native devices.

U.2.4.11.2 Lease establishment

The G_Lease interface is used prior to the use of the G_Publish interface in order to establish a GS_Lease_ID. The GS_Lease_Type is set to either publisher or subscriber to configure the respective side and to establish and reserve the underlying gateway entity and communication channel resources.

GS_Network_Address_List is used by the publisher and subscriber to establish the identity of the other endpoints. A publisher may describe multiple addresses within the list in order to configure multiple subscribers.

Within the lease, GS_Protocol_Type is used to describe the application protocol that will be tunneled through the interface. This allows protocol-specific processing to occur.

GS_Lease_Parameters is used to establish the expected protocol interaction between a publisher and a subscriber.

U.2.4.11.3 Publication

The G_Publish primitive is used by a publisher to initiate transfer of a publish data payload to one or more subscribers.

The publish data payload is stored in a local buffer and forwarded from the buffer to subscribers. The lease configuration parameters determine when forwarding will occur. Forwarding occurs in order to meet scheduled deadlines. Over a period of time, the same payload may be forwarded multiple times to indicate that the publisher still exists and to prevent timeout. Invocation with unchanged data may not result in forwarding.

U.2.4.11.4 Subscription

The G_Subscribe primitive is used by a subscriber to retrieve the most recent publication data from the local buffer.

The subscriber also receives the most recent publication associated with a subscribe lease via the G_Publish indication primitive.

The primitive G_Publish_Watchdog is used within a subscriber to signal the expiration of a watchdog timer. The timer expires in the absence of expected updates from a publisher. The timer is reset on the arrival of publication data payload at the subscriber. The watchdog timer is configured as part of the lease configuration parameters.

The primitive G_Publish_Timer is used within a publisher to signal the expiration of a publication timer. The publication timer is a periodic timer that expires prior to the deadline for forwarding the publish data payload. The indication may be used to publish fresh data. The publication timer is configured with the lease configuration parameters.

The primitive G_Subscribe_Timer is used within a subscriber to signal the expiration of a subscription timer. The subscription timer is a periodic timer that expires at the delivery deadline for receiving the publish data payload. The indication is used to process existing publication data. Arrival of fresh data will reset the timer and will result in a G_Publish indication. The subscription timer is configured with the lease configuration parameters.

U.2.4.11.5 Types of primitives and parameters

U.2.4.11.6 G_Publish primitive and its parameters

Table U.18 describes parameter usage for the primitive G_Publish.

Table U.18 – Primitive G_Publish parameter usage

Parameter name	G_Publish		
	Request	Indication	Confirm
GS_Session_ID	M	M	M
GS_Transaction_ID	M	—	M(=)
GS_Lease_ID	M	M	—
GS_Transfer_Mode	M	—	—
GS_Publish_Data	M	C(=)	—
GS_Status	—	—	M

U.2.4.11.7 Use of G_Publish request

The primitive G_Publish request is used to initiate transfer of a publish data payload (GS_Publish_Data) to one or more subscribers. The publish data payload is stored in a local buffer and forwarded from the buffer to subscribers.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

The subscriber addressing is known through the lease identifier (GS_Lease_ID) that was obtained from the lease interface.

Within the lease parameters, GS_Resource is an identical value specified for a publisher and one or more subscribers in order to facilitate establishment of linkage between the endpoints.

GS_Transfer_Mode is provided with the request in order to indicate the quality of interface and priority for the transfer of the data.

U.2.4.11.8 Use of G_Publish indication

The primitive G_Publish indication is used to signal the arrival of a publish data payload at a subscriber for processing.

The publish data payload (GS_Publish_Data) is delivered with the indication.

The subscriber session identifier (GS_Session_ID) and subscriber lease identifier (GS_Lease_ID) are returned to allow association of the indication primitive with a specific publish/subscribe relationship.

The indication is conditional on configuration-dependent timed delivery from the publisher and indicates fresh publication data.

U.2.4.11.9 Use of G_Publish confirm

The primitive G_Publish confirm is used to complete the G_Publish request.

The publisher session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm primitive with the original request primitive.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.19.

Table U.19 – GS_Status for G_Publish confirm

Value	Meaning
0	Success
1	Failure; lease has expired
2	Failure; other

U.2.4.11.10 G_Subscribe primitive and its parameters

Table U.20 describes parameter usage for the primitive G_Subscribe.

Table U.20 – Primitive G_Subscribe parameter usage

Parameter name	G_Subscribe	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	—
GS_Publish_Data	—	M
GS_Status	—	M

U.2.4.11.11 Use of G_Subscribe request

The primitive G_Subscribe request is used to retrieve the most recent publication data (GS_Publish_Data) from the local buffer.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

The publisher addressing is known through the lease identifier (GS_Lease_ID) that was obtained from the lease interface.

U.2.4.11.12 Use of G_Subscribe confirm

The primitive G_Subscribe confirm is used to complete the G_Subscribe request.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm primitive with the original request primitive.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.21.

Table U.21 – GS_Status for G_Subscribe confirm

Value	Meaning
0	Success; fresh data
1	Success; stale data
2	Failure; lease has expired
3	Failure; other

U.2.4.11.13 G_Publish_Timer primitive and its parameters

Table U.22 describes parameter usage for the primitive G_Publish_Timer.

Table U.22 – Primitive G_Publish_Timer parameter usage

Parameter name	G_PublishTimer
	Indication
GS_Session_ID	M
GS_Lease_ID	M

U.2.4.11.14 Use of G_Publish_Timer indication

The primitive G_Publish_Timer indication is used within a publisher to signal the expiration of a publication timer.

The publisher session identifier (GS_Session_ID) and publisher lease identifier (GS_Lease_ID) are returned to allow association of the indication primitive with a specific publish/subscribe relationship.

U.2.4.11.15 G_Subscribe_Timer primitive and its parameters

Table U.23 describes parameter usage for the primitive G_Subscribe_Timer.

Table U.23 – Primitive G_Subscribe_Timer parameter usage

Parameter name	G_SubscribeTimer
	Indication
GS_Session_ID	M
GS_Publish_Data	M
GS_Lease_ID	M

U.2.4.11.16 Use of G_Subscribe_Timer indication

The primitive G_Subscribe_Timer indication is used within a subscriber to signal the expiration of a subscription timer. The timer is reset by the G_Publish indication.

The publish data payload (GS_Publish_Data) is delivered from the subscriber buffer with the indication.

The subscriber session identifier (GS_Session_ID) and subscriber lease identifier (GS_Lease_ID) are returned to allow association of the indication primitive with a specific publish/subscribe relationship.

U.2.4.11.17 G_Publish_Watchdog primitive and its parameters

Table U.24 describes parameter usage for the primitive G_Publish_Watchdog.

Table U.24 – Primitive G_Publish_Watchdog parameter usage

Parameter name	G_Publish_Watchdog
	Indication
GS_Session_ID	M
GS_Publish_Data	M
GS_Lease_ID	M

U.2.4.11.18 Use of G_Publish_Watchdog indication

The primitive G_Publish_Watchdog indication is used within a subscriber to signal the expiration of a watchdog timer due to the absence of expected updates from a publisher. The timer is reset by the G_Publish indication.

The now-stale publish data payload (GS_Publish_Data) is delivered from the subscriber buffer with the indication.

The session identifier (GS_Session_ID) and lease identifier (GS_Lease_ID) are returned to allow association of the indication primitive with a specific publish/subscribe relationship.

U.2.4.12 Bulk transfer interface

U.2.4.12.1 General

The bulk transfer interface provides for bulk data transfer. Bulk data transfer is used to transfer large items between gateway-internal clients and wireless devices.

Bulk transfers operate in the context of a session between the GIAP interface provider and the GIAP interface user. All primitives supported by the gateway through a GIAP include the corresponding GS_Session_ID.

The client of the session manages the session-unique GS_Transaction_IDs for each primitive invoked by the client. This is necessary in order to maintain coordination between bulk transfer primitives.

The GS_Lease_ID, which represents the necessary communication resources allocated within the gateway, is supplied with each primitive.

Separate parallel bulk transfers are distinguished by a GS_Transfer_ID. A GS_Transfer_ID also is included in each GIAP interface primitive. The transfer state is maintained for each bulk transfer in progress. For example, the block number being transferred is maintained by the endpoints.

G_Bulk_Open is used to open a bulk transfer. G_Bulk_Close is used to close a bulk transfer. G_Bulk_Transfer is used to perform the actual transfer of data segments within a bulk transfer.

U.2.4.12.2 Types of primitives and parameters

U.2.4.12.3 G_Bulk_Open primitive and its parameters

Table U.25 describes parameter usage for the primitive G_Bulk_Open.

Table U.25 – Primitive G_Bulk_Open parameter usage

Parameter name	G_Bulk_Open	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	—
GS_Transfer_ID	M	—
GS_Resource	M	—
GS_Mode	M	—
GS_Block_Size	M	M
GS_Item_Size	C	C
GS_Status	—	M

U.2.4.12.4 Use of G_Bulk_Open request

The G_Bulk_Open request primitive is used to initiate a bulk transfer. The target device for a bulk transfer is implied by the GS_Lease_ID.

The target item for a bulk transfer is identified by GS_Resource.

A transfer is directional (upload or download) and GS_Mode describes the direction of the transfer. GS_Mode = 0 describes download and GS_Mode = 1 describes upload.

The GIAP interface user sets GS_Block_Size to request a block size for the subsequent transfer phase.

The GIAP interface user sets the GS_Item_Size to request download of an item of a particular size. The item may exceed the available download limits, resulting in an error response. GS_Item_Size = 0 requests the download of an item of indeterminate size.

U.2.4.12.5 Use of G_Bulk_Open confirm

The G_Bulk_Open confirm primitive is used in response to the G_Bulk_Open request.

The GS_Item_Size is set by the GIAP interface provider to indicate the item size. For a download, this is the maximum item size that will be accepted. For an upload, this is the actual item size. GS_Item_Size = 0 indicates that there is no limit imposed on the item size.

The GIAP interface provider determines and returns the GS_Block_Size that will be used for the subsequent transfer phase. The block size may be reduced in size (based on available resources) from the original size requested in the GS_Bulk_Open request.

GS_Status indicates success or failure of the G_Bulk_Open, as shown in Table U.26.

Table U.26 – GS_Status for G_Bulk_Open confirm

Value	Meaning
0	Success
1	Failure; item exceeds limits
2	Failure; unknown resource
3	Failure; invalid mode
4	Failure; other

U.2.4.12.6 G_Bulk_Transfer primitive and its parameters

Table U.27 describes parameter usage for the primitive G_Bulk_Transfer.

Table U.27 – Primitive G_Bulk_Transfer parameter usage

Parameter name	G_Bulk_Transfer	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	—
GS_Transfer_ID	M	—
GS_Bulk_Data	C	C
GS_Status	—	M

U.2.4.12.7 Use of G_Bulk_Transfer request

The G_Bulk_Transfer request primitive is used to move bulk data. GS_Bulk_Data is a transfer segment that is conditionally sent to the target in the case of a download.

G_Bulk_Transfer is used as many times as required to complete the transfer of a large item. G_Bulk_Close is used by the GIAP interface user to indicate the completion of the transfer.

U.2.4.12.8 Use of G_Bulk_Transfer confirm

The G_Bulk_Transfer confirm primitive is used in response to the G_Bulk_Transfer request. GS_Bulk_Data is a transfer segment that is conditionally received from the target in the case of an upload.

GS_Status indicates success or failure of the G_Bulk_Transfer, as indicated in Table U.28.

Table U.28 – GS_Status for G_Bulk_Transfer confirm

Value	Meaning
0	Success
1	Failure; communication failed
2	Failure; transfer aborted
3	Failure; other

U.2.4.12.9 G_Bulk_Close primitive and its parameters

Table U.29 describes parameter usage for the primitive G_Bulk_Close.

Table U.29 – Primitive G_Bulk_Close parameter usage

Parameter name	G_Bulk_Close	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	—
GS_Transfer_ID	M	—
GS_Status	—	M

U.2.4.12.10 Use of G_Bulk_Close request

The G_Bulk_Close request primitive is used to complete a bulk transfer, and to clean up any resources or state handling necessary in the GIAP interface provider.

U.2.4.12.11 Use of G_Bulk_Close confirm

The G_Bulk_Close confirm primitive is used in response to the G_Bulk_Close request.

U.2.4.13 Alert interface

U.2.4.13.1 General

The alert interface provides for the establishment of alert notification events for gateway-internal clients. Additional operations may be required to collect additional information related to the alert or to respond to the alert.

Alert interfaces operate in the context of a session between the GIAP interface provider and the GIAP interface user. All primitives supported by the gateway through a GIAP include the corresponding GS_Session_ID.

The client of the session manages the session-unique outstanding GS_Transaction_IDs for each primitive it invokes. This is necessary in order to maintain coordination between alert primitives.

The GS_Lease_ID, which represents the necessary gateway entity and communication resources, is supplied with each primitive.

G_Alert_Subscription is used to subscribe to alerts either by category or by specific alerts.

U.2.4.13.2 Types of primitives and parameters

U.2.4.13.3 G_Alert_Subscription primitive and its parameters

Table U.30 describes parameter usage for the primitive G_Alert_Subscription.

Table U.30 – Primitive G_Alert_Subscription parameter usage

Parameter name	G_Alert_Subscription	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	—
GS_Subscription_List	M	C
GS_Category	C	C
GS_Network_Address	C	C
GS_Alert_Source_ID	C	C
GS_Subscribe	M	C
GS_Enable	M	C
GS_Status	—	M

U.2.4.13.4 Use of G_Alert_Subscription request

The G_Alert_Subscription request primitive is used to manage an alert subscription list.

GS_Subscription_List contains one or more alert subscription modification requests. Each list element can be used to modify the subscription for a particular category of alerts or to modify the subscription for a specific alert from a specific source.

List elements may describe the GS_Category to indicate subscription modification for a particular category of alerts. Alert categories include:

- 0 = device;
- 1 = network;
- 2 = security; and
- 3 = process.

GS_Network_Address and GS_Alert_Source_ID are not supplied if GS_Category is supplied.

Alternatively, list elements may describe GS_Network_Address and GS_Alert_Source_ID instead of GS_Category to describe a specific device and an identifier of a specific alert from that device.

NOTE It is anticipated that some gateway-internal clients, such as full alarm management systems, will use complete categories, whereas other gateway-internal clients are able to restrict their usage to only a select subset of alerts.

GS_Subscribe and GS_Enable control the actions for each list element. GS_Subscribe is used to describe which alerts are to be received within the gateway entity and forwarded to the GIAP interface user in the form of G_Alert_Notification indications. GS_Enable is used to control the underlying generation of alerts at the source. GS_Subscribe = 1 subscribes to a specific alert or an alert category, while GS_Subscribe = 0 unsubscribes from the alert. GS_Enable = 1 enables a specific alert or an alert category, while GS_Enable = 0 disables the alert at the source.

In order to synchronize the alarm state between the alarm source and the gateway entity, alarm recovery is initiated on subscriptions.

U.2.4.13.5 Use of G_Alert_Subscription confirm

The G_Alert_Subscription confirm primitive is used in response to the G_Alert_Subscription request.

GS_Status indicates success or failure of the G_Alert_Subscription request, as indicated in Table U.31. A GS_Subscription_List with a single element is conditionally returned if the operation fails. The status code relates to the particular element. Processing of the list stops at the first failed element.

Table U.31 – GS_Status for G_Alert_Subscription confirm

Value	Meaning
0	Success
1	Failure; invalid category
2	Failure; invalid individual alert
3	Failure; other

U.2.4.13.6 G_Alert_Notification primitive and its parameters

Table U.32 describes parameter usage for the primitive G_Alert_Notification.

Table U.32 – Primitive G_Alert_Notification parameter usage

Parameter name	G_Alert_Notification
	Indication
GS_Session_ID	M
GS_Lease_ID	M
GS_Alert	M
GS_Network_Address	M
GS_Alert_Source_ID	M
GS_Time	M
GS_Class	M
GS_Direction	M
GS_Category	M
GS_Type	M
GS_Priority	M
GS_Alert_Data	C

U.2.4.13.7 Use of G_Alert_Notification indication

The G_Alert_Notification indication is generated by the GIAP interface provider and sent to the GIAP user in response to an alert received by the gateway. Notification is provided only for those alerts to which the GIAP client had subscribed, and for which notification has been enabled.

A GS_Alert structure is provided within the indication to provide alert specific details as follows:

- GS_Network_Address indicates the source device of the alert.
- GS_Alert_Source_ID indicates the specific alert within the source device.

- GS_Time is a timestamp that indicates when the alert was originally generated.
- GS_Class = 0 identifies the alert as an event; GS_Class = 1 identifies the alert as an alarm.
- GS_Direction further classifies alarms as follows:
  0: alarm condition ended;
  1: alarm condition began.
- GS_Category describes the alert category as follows:
  0: process-related;
  1: device-related;
  2: network-related;
  3: security-related.
- GS_Type describes sub-categories for alerts. The actual value is application-specific.
- GS_Priority describes a priority for the alert. Larger values indicate higher priority. The actual value is application-specific.
- GS_Alert_Data allows inclusion of alert-related information. This field is conditional on whether additional alert information is available. The actual value is application-specific.

The gateway entity acknowledges the alert receipt.

U.2.4.14 Gateway configuration interface

U.2.4.14.1 General

The gateway configuration interface provides for reading and writing the gateway configuration attributes.

The gateway-internal client uses the G_Read_Gateway_Configuration primitive to retrieve gateway configuration attributes.

U.2.4.14.2 Types of primitives and parameters

U.2.4.14.3 G_Read_Gateway_Configuration primitive and its parameters

Table U.33 describes parameter usage for the primitive G_Read_Gateway_Configuration.

Table U.33 – Primitive G_Read_Gateway_Configuration parameter usage

Parameter name	G_ReadGatewayConfiguration	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Attribute_Identifier	M	—
GS_Attribute_Value	—	C
GS_Status	—	M

U.2.4.14.4 Use of G_Read_Gateway_Configuration request

The gateway-internal client uses the primitive G_Read_Gateway_Configuration request to retrieve gateway configuration parameters.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

The requested attribute is specified by the attribute identifier (GS_Attribute_Identifier), as shown in Table U.34. The requested value is specified by the attribute value (GS_Attribute_Value).

Table U.34 – GS_Attribute_Identifier values for G_Read_Gateway_Configuration request

Value	Meaning
0	GS_GUID
1	GS_Max_Retries
2	GS_Max_Devices
3	GS_Actual_Devices

U.2.4.14.5 Use of G_Read_Gateway_Configuration confirm

The gateway entity uses the primitive G_Read_Gateway_Configuration confirm to complete the G_Read_Gateway_Configuration request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

If the operation succeeds, the value (GS_Attribute_Value) is returned for the requested attribute (GS_Attribute_Identifier).

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.2.4.14.6 G_Write_Gateway_Configuration primitive and its parameters

Table U.35 describes parameter usage for the primitive G_Write_Gateway_Configuration.

Table U.35 – Primitive G_Write_Gateway_Configuration parameter usage

Parameter name	G_Write_Gateway_Configuration	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Attribute_Identifier	M	—
GS_Attribute_Value	M	—
GS_Status	—	M

U.2.4.14.7 Use of G_Write_Gateway_Configuration request

The gateway-internal client uses the primitive G_Write_Gateway_Configuration request to alter gateway configuration attributes.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

The requested attribute is specified by the attribute identifier (GS_Attribute_Identifier), as shown in Table U.36. The requested value is specified by the attribute value (GS_Attribute_Value).

Table U.36 – GS_Attribute_Identifier values for G_Write_Gateway_Configuration request

Value	Meaning
0	GS_GUID
1	GS_Max_Retries

U.2.4.14.8 Use of G_Write_Gateway_Configuration confirm

The gateway entity uses the primitive G_Write_Gateway_Configuration confirm to complete the G_Write_Gateway_Configuration request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm primitive with the original request primitive.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.37.

Table U.37 – GS_Status for G_Write_Gateway_Configuration confirm

Value	Meaning
0	Success
1	Failure; invalid attribute value
2	Failure; other

U.2.4.15 Device configuration interface

U.2.4.15.1 General

The device configuration interface provides a method to manage the configuration of the devices that are associated with a gateway. This is useful for commissioning wireless devices for host systems and related applications.

The device configuration has interfaces to write and to read back the configuration for one or more devices.

A unique identifier is used to match the configuration to a specific device. An IPv6Address is specified for usage in configuration of the device, allowing subsequent logical access of the device.

The device list report interface is used to determine the devices associated with the gateway. This interface works in conjunction with the device list report interface by providing the ability to limit the devices that are associated with a gateway.

A configuration file may be provided for each device. The format of such a configuration file is gateway-implementation dependent. Information contained in this file is intended to allow gateways to automatically provision devices to join the network. Further configuration is accomplished one-on-one by client server and bulk transfer interfaces.

If the gateway is used to provision devices, the device list report will be empty until devices are provisioned and join the system.

The gateway-internal client uses the G_Write_Device_Configuration primitive to set the configuration for devices associated with a gateway entity.

U.2.4.15.2 Types of primitives and parameters

U.2.4.15.3 G_Write_Device_Configuration primitive and its parameters

Table U.38 describes parameter usage for the primitive G_Write_Device_Configuration.

Table U.38 – Primitive G_Write_Device_Configuration parameter usage

Parameter name	G_Write_Device_Configuration	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	M	—
GS_Configure	M	—
GS_Unique_Device_ID	M	—
GS_Network_Address	M	—
GS_Provisioning_Info	U	—
GS_Status	—	M

U.2.4.15.4 Use of G_Write_Device_Configuration request

The gateway-internal client uses the primitive G_Write_Device_Configuration request to configure the devices associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

A list of devices associated with the gateway entity (GS_Device_List) is supplied. For each device in the list, the unique device identifier (GS_Unique_Device_ID) indicates the device associated with the configuration. If GS_Configure = 1, the configuration is added for the specific device, while if GS_Configure = 0, the configuration is removed for the specific device. A matching IPv6Address (GS_Network_Address) indicates the logical address to associate with the device.

Device provisioning information (GS_Provisioning_Info) is supplied to the gateway for the gateway to control provisioning of the device.

U.2.4.15.5 Use of G_Write_Device_Configuration confirm

The gateway entity uses the primitive G_Write_Device_Configuration confirm to complete the G_Write_Device_Configuration request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.39.

Table U.39 – GS_Status for G_Write_Device_Configuration confirm

Value	Meaning
0	Success
1	Failure; invalid or duplicate IPv6Address
2	Failure; out of memory
3	Failure; maximum gateway devices exceeded
4	Failure; provisioning information invalid
5	Failure; other

U.2.4.15.6 G_Read_Device_Configuration primitive and its parameters

Table U.40 describes parameter usage for the primitive G_Read_Device_Configuration.

Table U.40 – Primitive G_Read_Device_Configuration parameter usage

Parameter name	G_Read_Device_Configuration	
	Request	Confirm
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	U	M
GS_Unique_Device_ID	U	M
GS_Network_Address	—	M
GS_Provisioning_Info	—	U
GS_Status	—	M

U.2.4.15.7 Use of G_Read_Device_Configuration request

The gateway-internal client uses the primitive G_Read_Device_Configuration request to retrieve the configuration of the devices associated with a gateway entity.

A session identifier (GS_Session_ID) is obtained from the G_Session interface and included in the request.

A session unique transaction identifier (GS_Transaction_ID) is specified for each invocation of the interface.

If a list of devices associated with the gateway entity (GS_Device_List) is supplied, the request is for those specific devices, and the unique device identifier (GS_Unique_Device_ID) indicates the device associated with the configurations to be read. If no list is supplied, the request is for all devices.

U.2.4.15.8 Use of G_Read_Device_Configuration confirm

The gateway entity uses the primitive G_Read_Device_Configuration confirm to complete the G_Read_Device_Configuration request to the gateway-internal client.

The session identifier (GS_Session_ID) and transaction identifier (GS_Transaction_ID) are returned to allow matching of the confirm with the original request.

A list of devices associated with the gateway entity (GS_Device_List) is returned. For each device in the list, the unique device identifier (GS_Unique_Device_ID) indicates the device associated with the configuration. A matching IPv6Address (GS_Network_Address) indicates the logical address associated with the device. The device configuration file (GS_Provisioning_Info), when present, provides provisioning information for the device.

GS_Status is returned to indicate success or failure of the operation, as described in Table U.8.

U.3 Example uses of WISN standard services and objects

U.3.1 Tunneling

U.3.1.1 General

The tunnel object (TUN) is a native object that acts as a communication endpoint for the following messaging:

- encapsulated foreign protocol content (shown in Figure U.16 as a dotted line); and
- native interface content (shown in Figure U.16 as a solid line) to configure and manage the tunnel object.

Gateway processes and adapter processes use tunnel objects to support foreign protocol translation. An important aspect of the TUN object is that it provides buffered message behavior for foreign content.

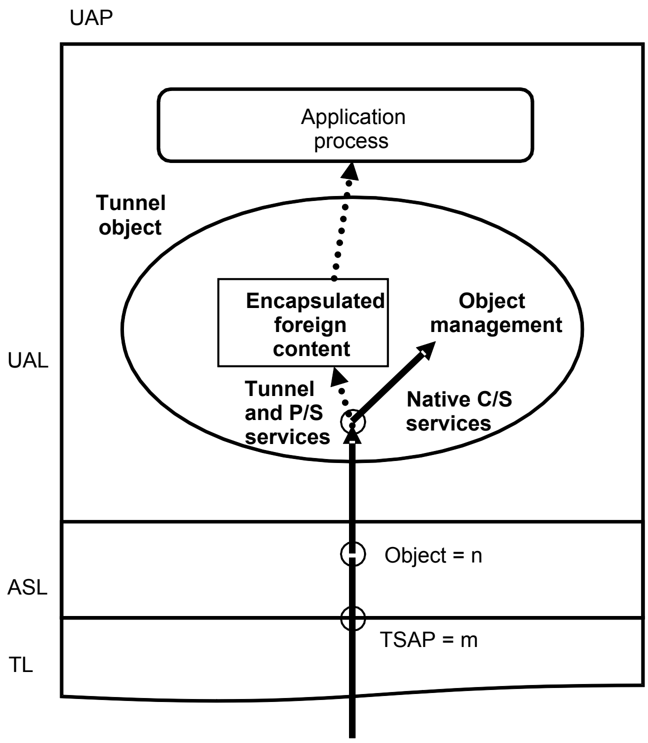


Figure U.16 – Tunnel object model

One or more TUNs may exist within a UAP.

Each TUN object can handle a complete foreign protocol or a portion thereof. Devices that handle multiple foreign protocols will need to implement multiple TUNs. The TUN object is independent of the foreign protocol.

The TUN object relies on the application sublayer (ASL) in order to route messages between peer TUNs and between a TUN object and other non-TUN objects.

U.3.1.2 Distributing tunnel objects

Each device may have one or more TUNs. Tunneling devices includes at least one TUN object as an endpoint for tunnels. Figure U.17 shows a group of related devices with tunnel endpoints interconnected between TUNs.

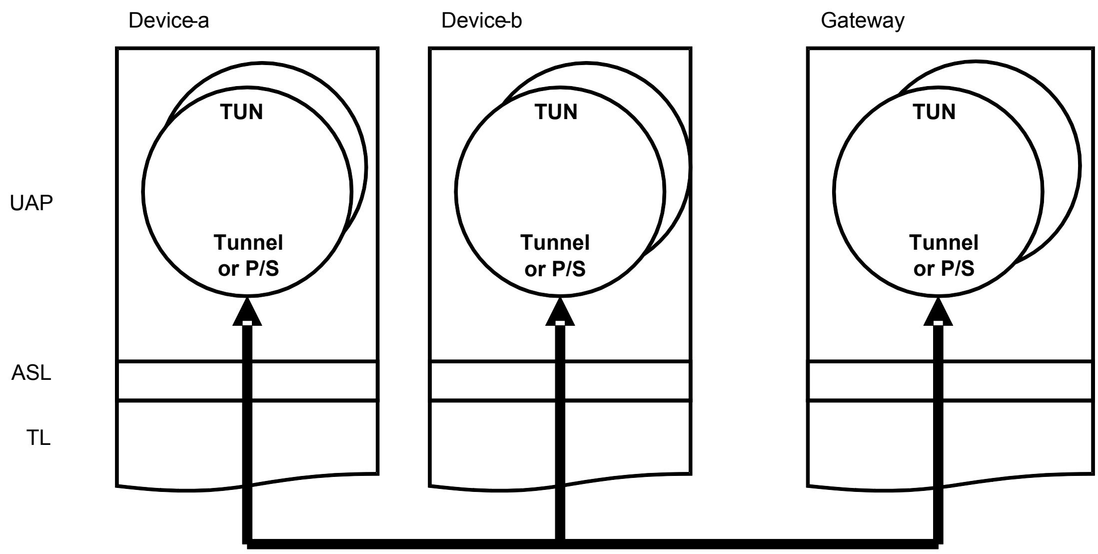


Figure U.17 – Distributed tunnel endpoints

Field devices and adapters may contain TUNs that cooperate with other TUNs in a gateway. A group of related TUNs communicate via a common foreign protocol. The typical usage of TUNs is to communicate between end devices and a host system via the gateway. Direct device-to-device tunneling is also supported within the object model.

TUN object communication is established by using TL interfaces invoked and augmented via the ASL. Communication relationships include publish, subscribe, 2-part tunnel, and 4-part tunnel. Multiple relationships may be established simultaneously.

U.3.1.3 Multicast, broadcast, and one-to-many messaging

As shown in Figure U.18, foreign protocols may require translation of broadcast/multicast messaging relationships when using interfaces such as publish/subscribe and alert distribution. This messaging requires translation support within this standard.

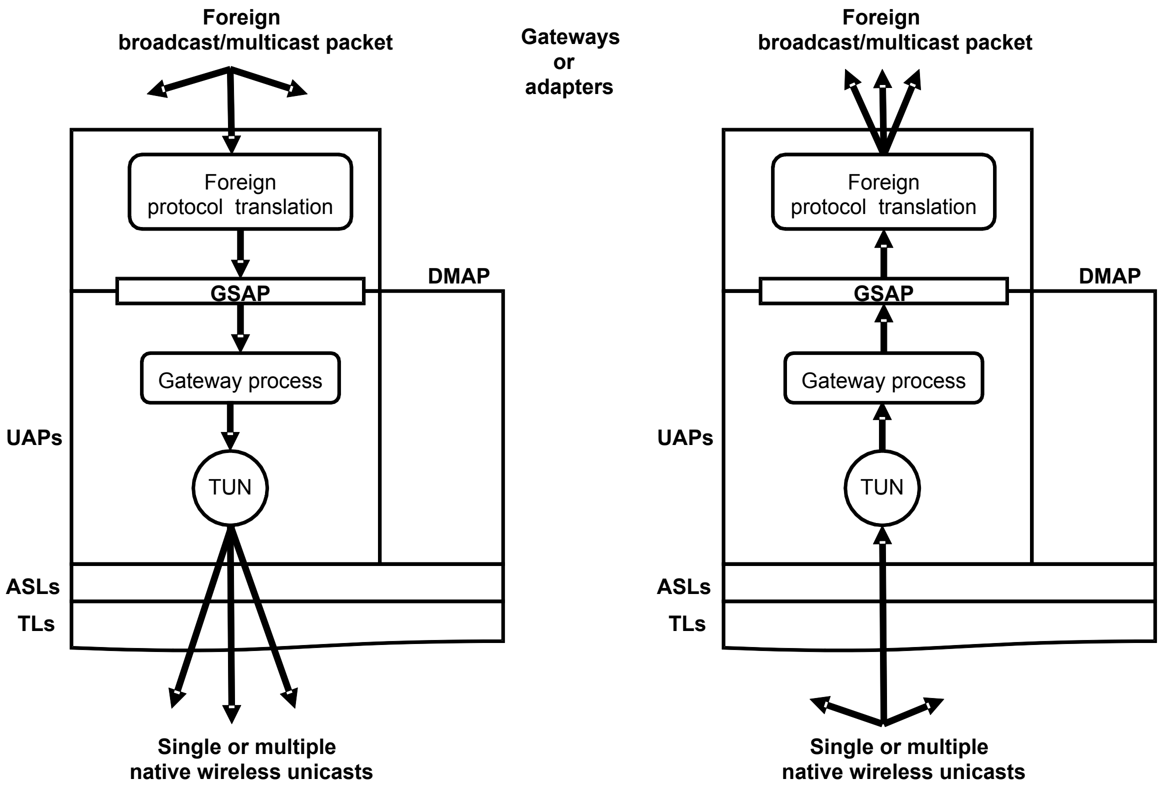


Figure U.18 – Multicast, broadcast, and one-to-many messaging

This standard provides one-to-many messaging support in the tunnel object in order to support translation of the foreign protocol multicast and broadcast requests. The underlying layers of the protocol suite do not provide broadcast or multicast interfaces to the AL. One-to-many messaging is achieved via a series of unicast operations. Protocol translation applications cannot rely on simultaneous delivery of unicast messages.

U.3.1.4 Tunnel buffered message behavior

TUN object communication may be implemented to provide capabilities for buffered and non-buffered behavior for publish/subscribe and tunnel-based message exchanges. TUNs are capable of cooperatively managing buffered behavior to reduce wireless transactions.

NOTE 1 Some legacy protocols use buffered messaging exchanges to support energy-efficient and high-performance protocol translation.

NOTE 2 Some applications are unable to tolerate buffered behavior, usually due to safety and synchronization requirements.

NOTE 3 Buffers are a single element deep. Nothing in this standard prevents implementation of caching and queuing enhancements.

As shown in Figure U.19, each endpoint of a communication flow has a different buffering responsibility, depending on the relationship.

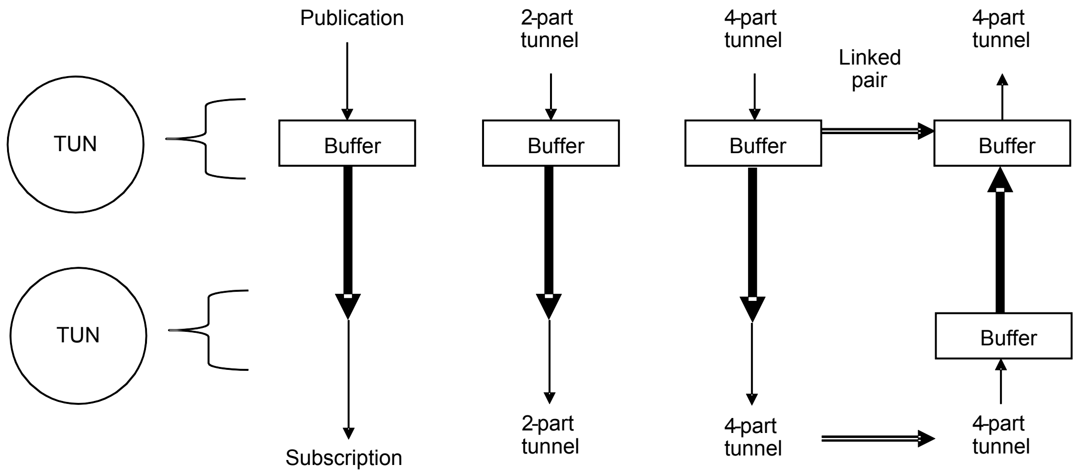


Figure U.19 – Tunnel object buffering

In Figure U.19, two TUN objects are shown. The thin arrows indicate interactions from tunnel applications that use the objects. The thick arrows indicate message flows across the network between the objects. Three types of transaction are shown, a publish/subscribe transaction, a 2-part tunnel transaction, and a 4-part tunnel transaction. Buffers are shown to illustrate the buffered messaging behavior between tunnel endpoints.

A publisher and a subscriber are linked from TUN object to TUN object for periodic updates. Publishers use change of state (CoSt) buffer publications to avoid sending repeated information; subscribers tolerate limited intervals with missing publications.

Messaging with the 2-part tunnel interface behaves in a similar manner, except that the messaging is aperiodic.

Messaging with the 4-part tunnel interface distinguishes a request and a response side via the request/response bit in the tunnel interface header. The request side buffers the first request and also forwards the request. A response is generated as indicated by the double arrow on the response side in Figure U.19. The response is stored in a second linked buffer on the request side, as indicated by the double arrow in Figure U.19. Change of state processing applies to subsequent duplicate requests, wherein the response is returned from the local buffer. Where change of state indicates an altered request, the request is forwarded and the local response buffer is updated. A final buffer is shown in Figure U.19 on the response side. This buffer supports change of state behavior wherein a single request results in multiple responses over time to update the request side.

U.3.1.5 Tunnel object attributes

TUN object attributes are described in Clause 12, but are further described herein.

The Protocol attribute is used to configure the protocol associated with the tunnel object and the associated remote tunnel objects. When the protocol is set to none, the tunnel can be configured. Once another protocol is set, the tunnel object configuration is applied and the status is updated to reflect the result.

The tunnel endpoint structure describes address information pointing to a single remote tunnel object. The array of tunnel endpoints allows specification of one or more tunnel endpoints representing remote tunnel objects. This allows a single communication relationship to span multiple tunnel objects where necessary. Max_Peer_tunnels indicates the maximum number of entries in the array. Num_Peer_Tunnels indicates the actual number of entries configured in the array.

One of several types of communication flow types is selected between the tunnel objects by configuration of the Flow_Type attribute. Flow types include 2-part tunnel, 4-part tunnel, publish, and subscribe.

For publish and subscribe Flow_Types, the Update_Policy allows configuration of periodic publication or change of state publication. Periodic publication occurs at every opportunity. CoSt publication occurs only when fresh publication data is available. The publication frequency is based on the Period attribute. The actual timing is based on a combination of the Period and the Phase attributes. The Stale_Limit is used in the subscriber to configure behavior for detection of excessive publication loss or delay. Stale_Limit is a multiplier that configures the number of periods that a subscriber will wait before considering lost publications to indicate a problem.

Foreign_Destination_Address and Foreign_Source_Address are the addresses associated with the tunnel endpoint by the foreign protocol. The format is dependent on the foreign protocol. These addresses are returned to protocol translator applications as tunnel object messages are received. They allow utilization of IPv6Addressing as defined in this standard in lieu of carrying the foreign addressing. Mapping via the tunnel object allows reconstruction of foreign PDUs containing address information.

NOTE Depending on the specific foreign protocol conversion, the foreign PDU will vary. Most fieldbus protocols will form DPDUs for direct delivery on a local link. In contrast, IP-based protocols usually form NPDUs, where a final encapsulation is achieved by an address resolution protocol.

Connection_Info[ ] and Transaction_Info[ ] are octet strings that are written by the protocol translator as required. Connection_Info[ ] is used to provide protocol specific static message content on message receipt in order to eliminate the repeated wireless message transfer of the content. Transaction_Info[ ] is used to provide protocol specific message content on receipt of a response, where the content would otherwise be echoed from the request in the response, eliminating the wireless transfer of the content. Further description is provided in U.3.1.9 and Annex O.

It is the responsibility of the TUN object implementation to maintain a related contract for each tunnel endpoint.

U.3.1.6 Tunnel object messaging

U.3.1.6.1 Application sublayer interface usage

TUN objects may be implemented to provide connection interfaces that include a publish/subscribe interface, a 2-part tunnel interface, and a 4-part tunnel interface. Each interface may be implemented in both a buffered and a non-buffered mode of operation.

An optional external interface for invoking the gateway connection interfaces is described in U.2.

The TUN object uses the ASL to deliver and receive interface content as described for the publish interface in 12.17.3.2 and the tunnel interface in 12.17.6.2. The ASL provides object-to-object delivery of publish/subscribe payloads in external formats through the publish request primitive. The ASL also provides a linked tunnel request and tunnel response primitive.

The header is described in 12.22.2.3. This header enables request and response specification, interface type specification (publish or tunnel), and object identifier addressing mode (4-bit, 8-bit, or 16-bit). A large number of tunnel objects will result in a larger address space and more overhead in the header.

The publish interface payload format is described in 12.22.2.12 by Table 348. There is no explicit size in the header. The size of the publication is supplied with the publish request and is known to the subscriber by information supplied with the indication.

The tunnel interface request and response payload formats are described in 12.22.2.9. The request allows 7-bit size (0..127 octet payloads) or 15-bit size (128..32 767 octet payloads). Inclusion of the size allows tunnel message to be concatenated by the ASL.

NOTE Most encapsulated messages from legacy protocols referenced by this standard fall into the range of less than a 127-octet payload, resulting in a 7-bit field.

U.3.1.6.2 Information classification and transfer rules

From a caching and buffering viewpoint, information may be classified as constant, static, dynamic, or non-cacheable. These classifications are described in 12.6.3 for native object attributes. The same guidance applies to the selection of buffering for publish and tunnel interfaces for foreign payloads.

Constant information should not be transferred more than once between TUNs, except where local copies are lost due to power cycling, reset, cache deletion, or elimination of references to the information.

Static information should not be transferred more than once between TUNs, except as indicated for constant information and where the static information has been modified.

Dynamic information should only be transferred between TUNs when its value has changed unless it is required more often to indicate that the source or destination is still active.

Non-cacheable information may be transferred between TUNs on each request.

U.3.1.6.3 Publish/subscribe interface

The tunnel object may be implemented to provide facilities to accomplish buffered and non-buffered publish/subscribe messaging for dynamic information update.

The flowcharts of Figure U.20, Figure U.21 and Figure U.22 describe the behavior of TUN object publishers and subscribers that use buffering. The behavior describes the base message transfer agreement between a publisher and a subscriber based on the TUN object attribute configurations.

NOTE The interpretation and actions for initial, stale, and repeat data are based on implementation, as is the CoSt algorithm.

The publish/subscribe publisher connection operates as shown in Figure U.20 when CoSt updates are configured.

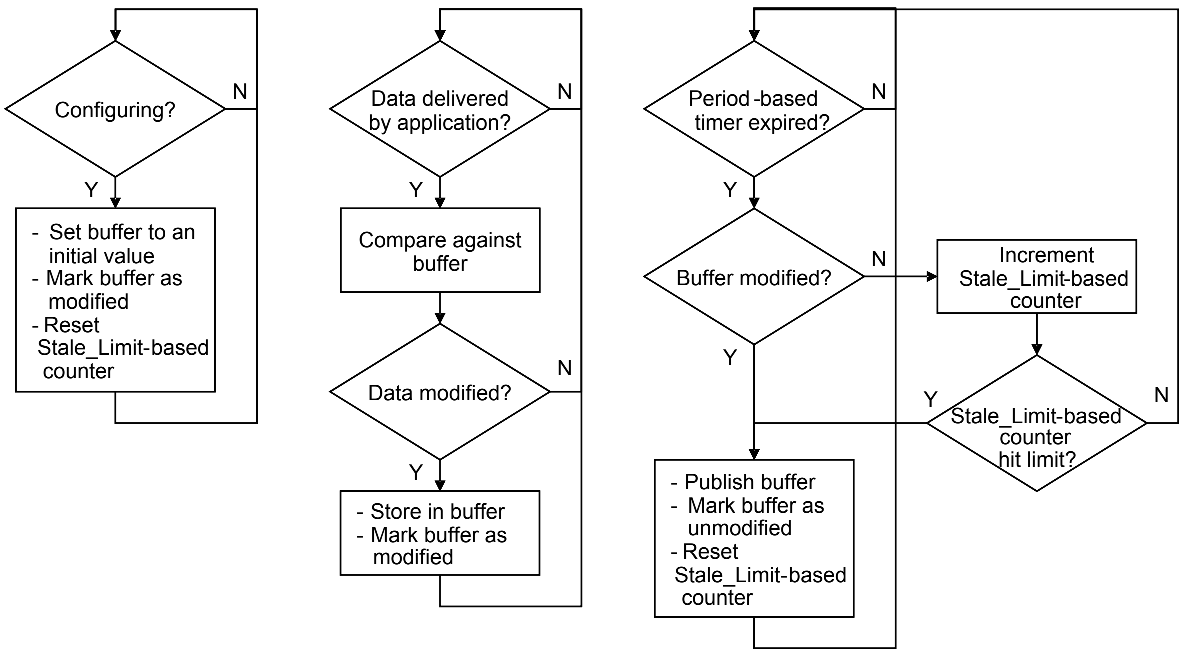


Figure U.20 – publish/subscribe publisher CoSt flowchart

The publish/subscribe publisher connection operates as shown in Figure U.21 when periodic updates are configured.

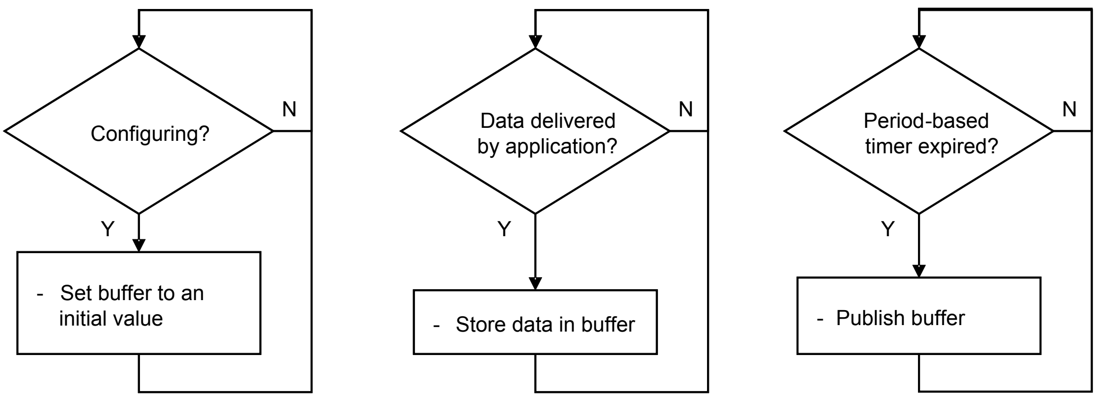


Figure U.21 – publish/subscribe publisher periodic flowchart

The publish/subscribe subscriber connection operates as shown in Figure U.22 when periodic or CoSt updates are configured.

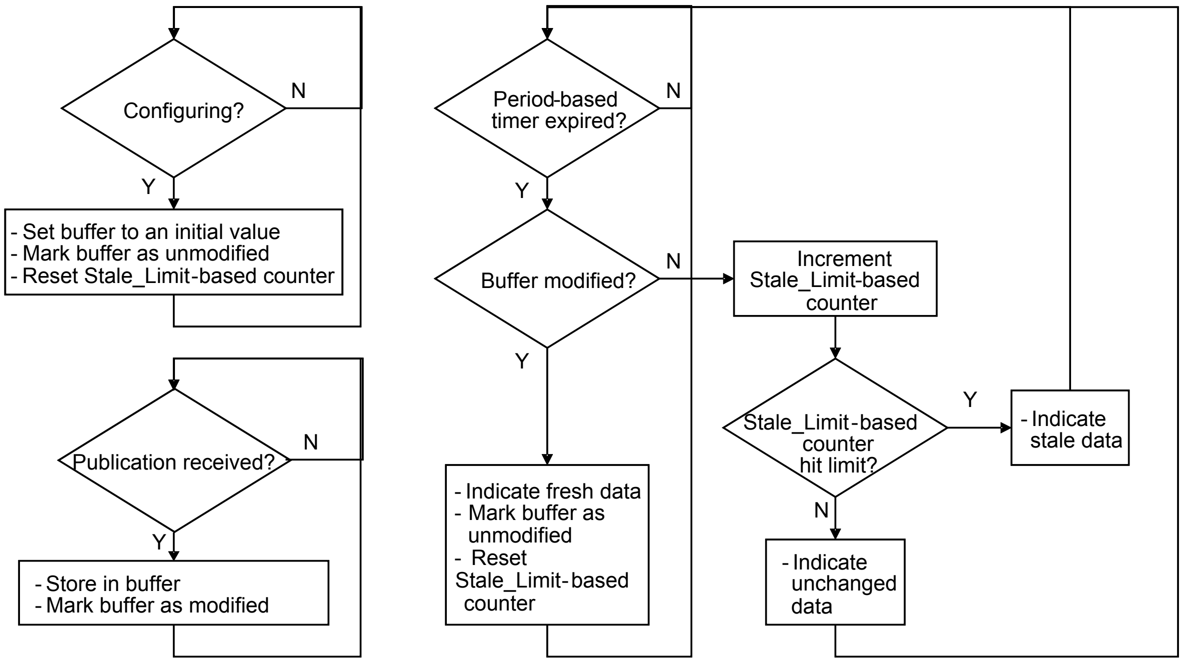


Figure U.22 – publish/subscribe subscriber common periodic and CoSt flowchart

U.3.1.6.4 Tunnel interface

The tunnel object may be implemented to provide facilities to accomplish buffered and non-buffered tunnel interface messaging. Non-buffered tunnel interface messaging provides support for unconditional transfer of non-cacheable and constant information. Buffered tunnel interface messaging provides support for buffering and contingent transfer of static and dynamic information.

U.3.1.7 Multiple server responses

Certain client/server requests receive multiple responses. One reason is that the request requires extended processing and an immediate response is sent which indicates that the request was received and that the real response will be sent after processing is complete. This is known in some protocols as a delayed response. In other cases, the server provides additional updates over time to satisfy the initial request. Certain protocols collect process variables or historian information in this manner.

The client/server buffered and unbuffered interfaces support multiple application responses for these purposes. In the case of the buffered response, the read buffer maintains the latest response. The client receives an indication on each response.

U.3.1.8 Tunnel object address mapping

The TUN object may be implemented to contain three address fields (Foreign_Destination_Address, Foreign_Source_Address, and the Array of Tunnel endpoints) that are used in the translation between foreign addresses and native addresses.

As shown in Figure U.23, foreign multicast and foreign broadcast addresses require translation to native addresses and messaging.

The first case is where the multicast or broadcast originates on the foreign network. Since multiple hosts, protocols, or applications may share a wireless network as described herein, sending a foreign broadcast to all wireless devices is inefficient. Thus, foreign broadcast into the wireless network uses simulated multicast to a limited group. The TUN object is used to simulate multicast delivery (one-to-many messaging) by maintaining a list of unicast addresses (array of tunnel endpoints) and by using a sequence of unicast deliveries.

The second case is where the multicast or broadcast originates on the wireless network and is destined for the foreign network. A single APDU is delivered from the wireless network and acted on by a protocol translator to generate a multicast or broadcast PDU on the foreign network.

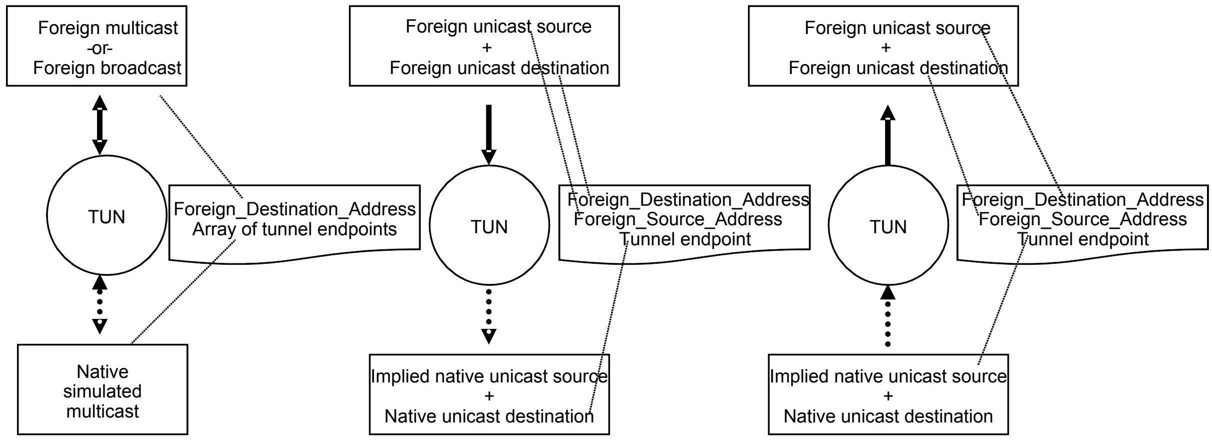


Figure U.23 – Network address mappings

Also shown in Figure U.23 is a pair of unicast address translations.

The first case translates from a foreign source/destination address pair to a native address pair. The second case translates from a native source/destination address pair to a foreign address pair. Both the Foreign_Destination_Address and the Foreign_Source_Address are used. Only the address information from a single tunnel endpoint is necessary, since the TUN object has access to its own native address for usage in source or destination fields. Foreign source and destination definition depend on the direction of the transfer.

U.3.1.9 Connection and transaction information

TUN objects function as initiator endpoints (publisher and tunnel request) and correspondent endpoints (subscriber and tunnel response). Protocol translation sends foreign content as TUN-DATA between the endpoints. Since most legacy protocols are not optimized for low-energy wireless communication, various mechanisms are available to increase efficiency.

When a protocol translator tunnels a foreign PDU, it is not efficient to repeatedly send static portions of the foreign PDU between the endpoints. Such static information includes preambles and secondary fixed addressing, such as logical unit identifiers. As shown in Figure U.24, TUN objects provide a generic mechanism (Connection_Info) for provision of static information on foreign PDU receipt without per-message wireless transfer of the static information.

NOTE Depending on the specific foreign protocol conversion, the foreign PDU will vary. Most fieldbus protocols will form DPDUs for direct delivery on a local link. In contrast, IP-based protocols usually form NPDUs, where a final encapsulation is achieved by an address resolution protocol.

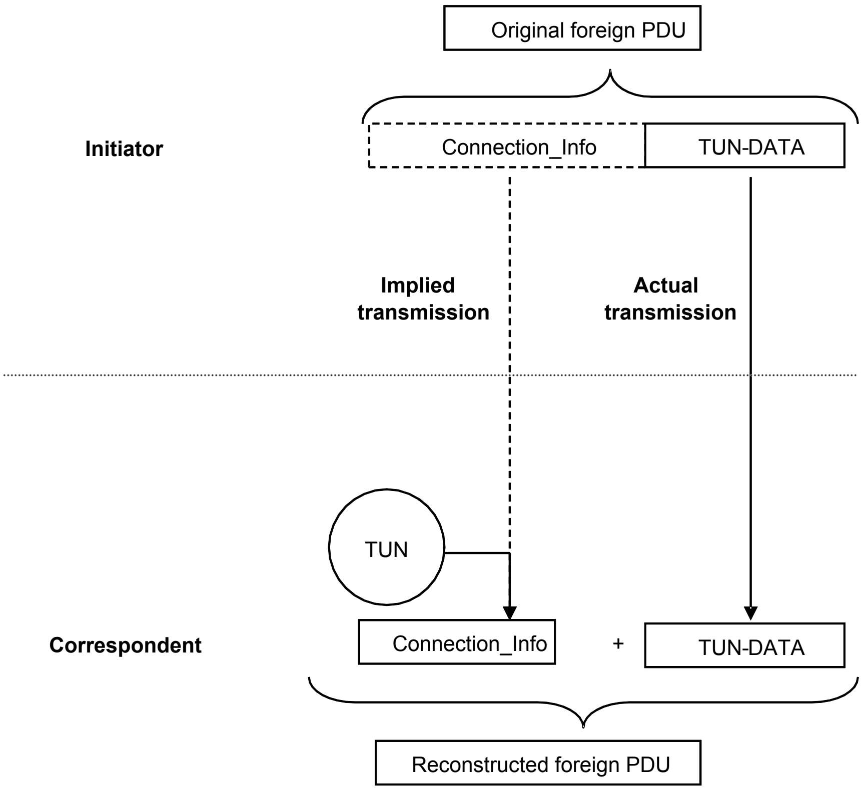


Figure U.24 – Connection_Info usage in protocol translation

When a protocol translator performs a transaction, it is not efficient to carry transaction-specific information that is only used to identify the transaction at the initiator. Such information includes information to link the original request to the response, where knowledge of the endpoint can be used. As shown in Figure U.25, TUN objects provide a generic mechanism (Transaction_Info) for provision of transaction-specific information without carrying the overhead in the wireless transfer.

Both Connection_Info and Transaction_Info can be used simultaneously.

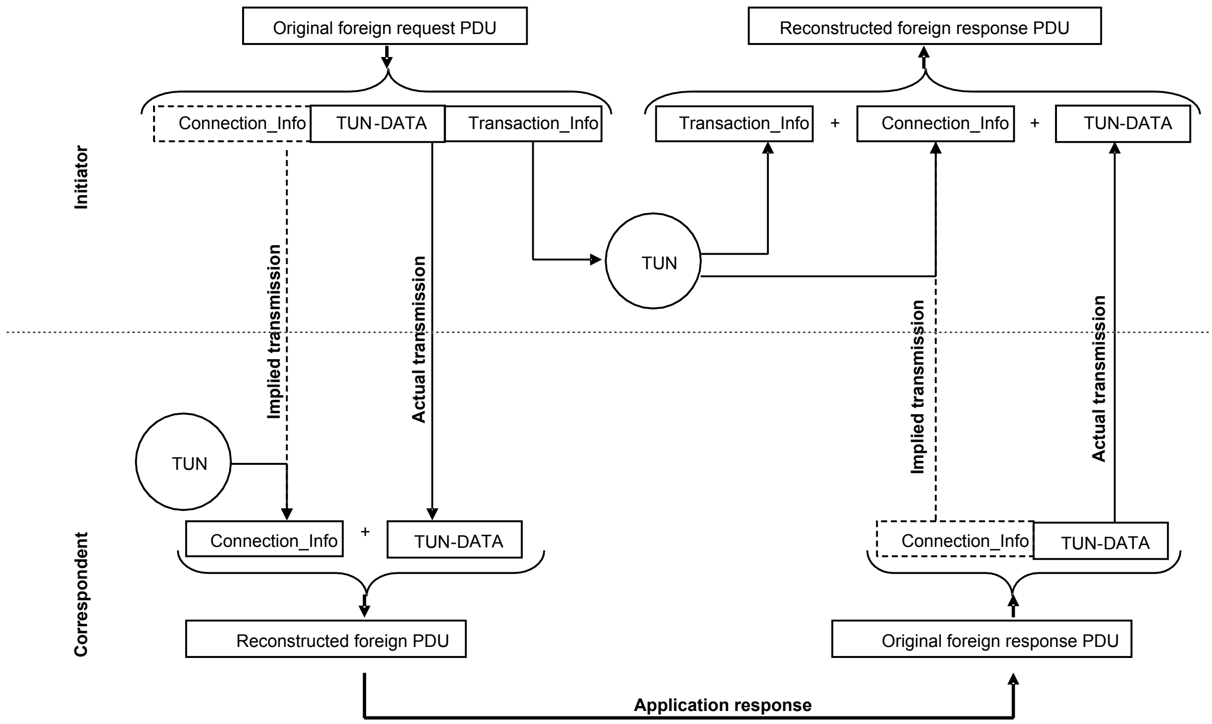


Figure U.25 – Transaction_Info usage in protocol translation

U.3.1.10 Interworkable tunneling mechanism

U.3.1.10.1 Overview

Annex U describes a communication mechanism for foreign network nodes to communicate across a wireless network via gateways and adapters. This mechanism enables vendor-independent development of interworkable gateways and adapters by implementing a restricted subset of the communication features defined within this standard. The interworking communication is achieved by the use of a constrained tunneling mechanism. The gateways and adapters serve to interconnect two or more foreign network segments by bridging foreign protocol DPDUs through the wireless network as depicted in Figure U.26. The gateway and adapter application processes use the AL tunnel objects and interfaces for the exchange of foreign unicast DPDUs and foreign broadcast/multicast DPDUs. The mechanism by which the gateway and adapter application processes exchange these DPDUs with foreign nodes is not specified by this standard.

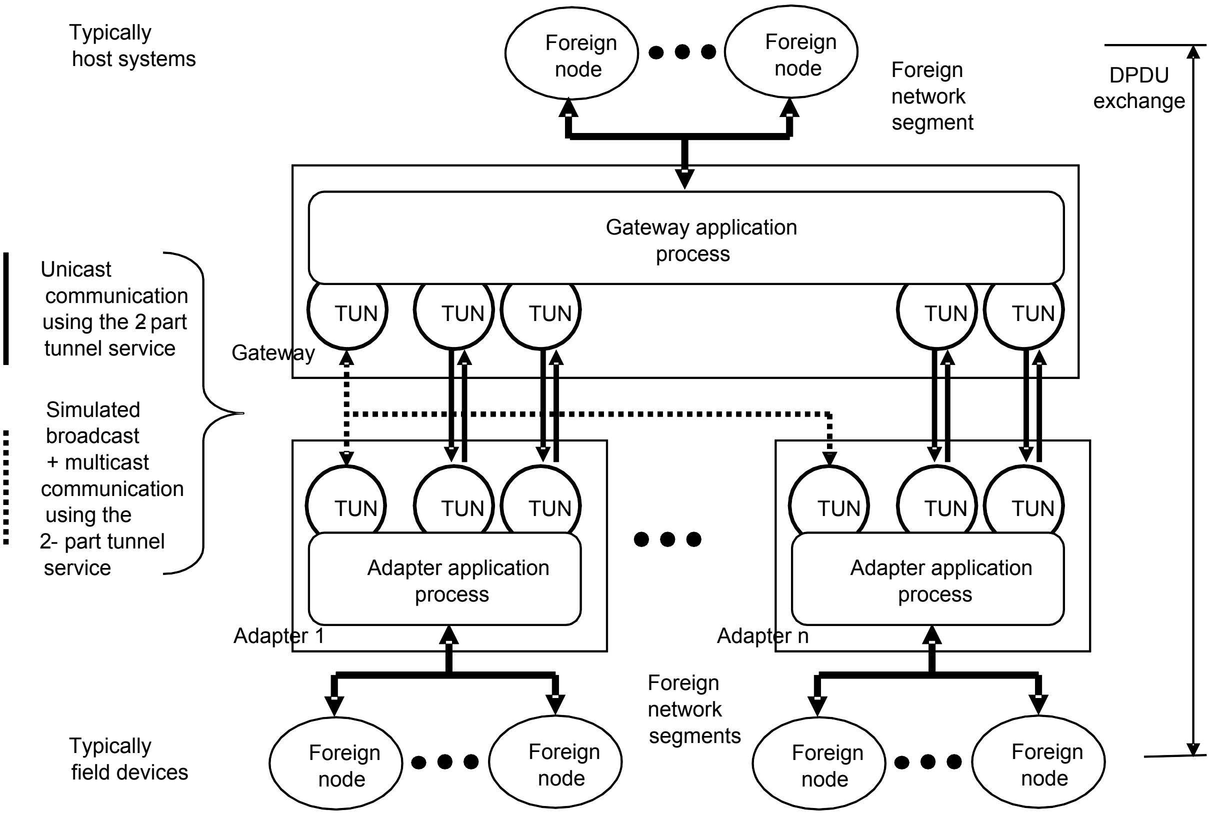


Figure U.26 – Interworkable tunneling mechanism overview diagram

U.3.1.10.2 Tunnel object placement

One or more foreign network nodes, individually addressable by a unicast DPDU address, may exist behind a gateway or an adapter. A foreign network node behind a gateway or adapter may require communication with an associated foreign network node behind another gateway or adapter.

For each associated gateway and adapter, a tunnel object is disposed and configured to carry foreign broadcast and multicast addressed DPDUs, one for a first associated foreign network segment and one for each additional associated foreign network segment.

For each associated foreign network node pair, a pair of tunnel objects is disposed and configured to carry unicast addressed DPDUs, one in the gateway or adapter for a first associated node and one in the gateway or adapter for a second associated node.

U.3.1.10.3 Tunnel object configuration

Tunnel operation is controlled as described in Clause 12. Tunnel objects are configured through attribute settings. Changes to the configuration are required to be correctly sequenced by setting the Protocol attribute and monitoring the Status attribute.

The unicast tunnel object pairs are configured as follows:

- The Flow_Type attribute is configured for a 2-part tunnel.
- The Array of Tunnel endpoints attributes are configured for a single address element, where each tunnel object in the pair addresses the other tunnel object in the pair.
- The Connection_Info[ ] and Transaction_Info[ ] attributes are not used.
- The Update_Policy, Phase, Period and Stale_Limit attributes are not used.

For unicast tunnel objects, the Foreign_Destination_Address attribute of each local tunnel object is set to the DPDU address of the associated foreign device behind the remote gateway or the adapter and the Foreign_Source_Address attribute of each local tunnel object are set to the DPDU address of the associated foreign device behind the local gateway or adapter.

The tunnel objects in the broadcast/multicast tunnel object set are configured as follows:

- The Flow_Type attribute is configured for a 2-part tunnel.
- The Array of Tunnel endpoints attributes are configured for one or more address elements, where each tunnel object in the set addresses all other tunnel objects in the set.
- The Connection_Info[ ] and Transaction_Info[ ] attributes are not used.
- The Update_Policy, Phase, Period and Stale_Limit attributes are not used.

For broadcast/multicast tunnel objects, the Foreign_Destination_Address attribute and the Foreign_Source_Address attribute are set to an equal value.

The usage of the Foreign_Source_Address attribute and the Foreign_Destination_Address attribute enables gateways and adapters using the interworkable tunneling mechanism to be configured strictly by configuration of the tunnel objects.

Associated gateways and adapters may send and receive foreign DPDUs from either identical versions or interworkable versions of the same foreign protocol. To enable the run state after the other attributes are configured, the Protocol attribute is configured last and is configured to the same value in all tunnel objects associated with all related foreign network segments on the D-subnet. The final Protocol attribute value is set as defined in Annex K. The gateway and adapter application processes may report tunnel object Status = 2 (configuration failed) if an attempt is made to configure a tunnel with an unsupported Protocol.

A compatible foreign protocol may be able to accommodate the timing imposed by the wireless mechanisms, either inherently or by configuration. Exchange of foreign DPDUs may not be the most efficient tunnel method, but it assures that sufficient information is available to process the packet within gateway and adapter application processes. It also assures multiple vendors convey the same information between gateway and adapter application processes. It is also assures that sufficient information is available within gateway and adapter application process to link client/server requests and responses. Addressing is also carried and enables multiple foreign network devices to sit behind each gateway or adapter.

U.3.1.10.4 Tunnel operation

Foreign network DPDUs may be delivered to the gateway and adapter application processes in one of two ways, either through a tunnel object or from a foreign source outside of the wireless network. The outside source will usually be a wired network (and its associated protocol stack) attached directly or indirectly to the gateway or adapter. Alternatively, the PDUs may be generated by software or firmware interacting with (or embedded in) the gateway or adapter application process directly. In either case, the tunneled PDU exchange between gateways and adapters may remain identical.

The gateway and adapter application processes may examine the foreign network DPDU destination address prior to forwarding the PDU over the wireless network. DPDUs without a known destination that is reachable through the tunnels are not forwarded.

Gateways and adapters application processes may forward foreign protocol unicast DPDUs to DPDU address destinations that are reachable through a linked pair of unicast tunnel objects.

Gateway and adapter application processes forward valid foreign protocol broadcast and multicast PDUs through the broadcast/multicast tunnel that exists within each associated gateway and adapter, distributing the same PDU to one or more destinations. The PDU is not echoed back to the source.

The gateway and adapters application processes may use multicast group establishment PDUs from within the foreign protocol, where such PDUs exist, in order to limit the distribution scope.

Since generation of multiple copies of the same message is almost certain to occur, the foreign protocol may tolerate timing skew.

U.3.1.10.5 Efficient operation

It is recommended that foreign protocols that are using the interworkable tunneling mechanism reduce PDU exchanges to the minimum that is acceptable to the foreign protocol and its applications. This is accomplished by extending update periods and timeouts for periodic update. This is also accomplished by elimination of redundant transfer of static information by maintaining local copies.

U.3.2 Bulk transfer

Large item transfer is accomplished through upload/download objects (UDOs), as shown in Figure U.27. Large item transfers are useful for firmware updates, transfer of large sample buffers such as captured waveforms, and general configuration. One UDO represents a single item that can be transferred in either direction (uploaded or downloaded) to/from another application. The item to be transferred exists at the location of the UDO. Interface objects (IFOs) act as clients to initiate transfers. The transfer protocol provides buffering, flow control, and guaranteed and in-order delivery. Protocol translators have access to the UDO through the GIAP.

Items are associated with a string that can be used to encode item specific identification and revision information. Asset management systems can be constructed to monitor revisions for regulated industries and to backup and restore items generically, without knowledge of the item content. Protocol translators may also transfer large items via foreign protocols through tunneling, but this precludes protocol independent asset management.

End applications are expected to understand the content of the transferred item and how to apply it. Provisions exist (depending on device capabilities) to request utilization of the item (possibly altering run-time behavior) and for storage of the item in non-volatile memory.

The UDO and the upload and download bulk transfer protocol are described in 12.15.2.4.

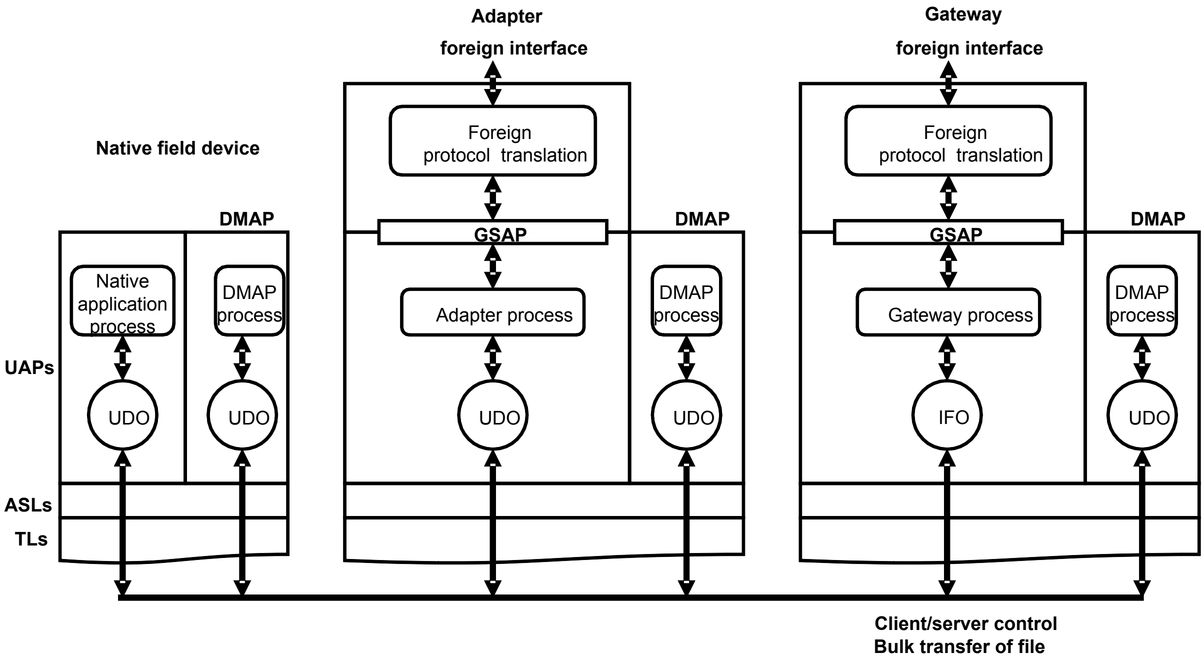


Figure U.27 – Bulk transfer model

U.3.3 Alerts

Alerts may be generated by many of the objects defined by this standard. Some objects reside within device UAPs, while others reside in the DMAP as management objects for each layer.

Alerts within a device are consolidated within the alert reporting management object (ARMO). Each device has a single ARMO that resides within the DMAP. All alerts within a device are conveniently consolidated in this single location.

The ARMO in each DMAP is responsible for reporting alerts through an AlertReport interface to an alert-receiving object (ARO). The ARO acknowledges alert receipt through the AlertAcknowledge interface. This transfer occurs independently of the actual processing of the alerts.

Alerts fall into four categories:

- process;
- device;
- network; and
- security.

Each category can be delivered by an ARMO to a different ARO. Thus, a single ARO might collect all process alerts across an entire network, or a set of AROs can be used, with each ARO only collecting a single category of alerts. If each ARO collects only one type of alert, then collection of all alerts requires four AROs.

The gateway contains one or more AROs that allow collection, reporting, and management of alerts.

Protocol translators have access to and can manage alerts through the GIAP, as shown in the alert model in Figure U.28. Subscription interfaces allow alert selection through the GIAP.

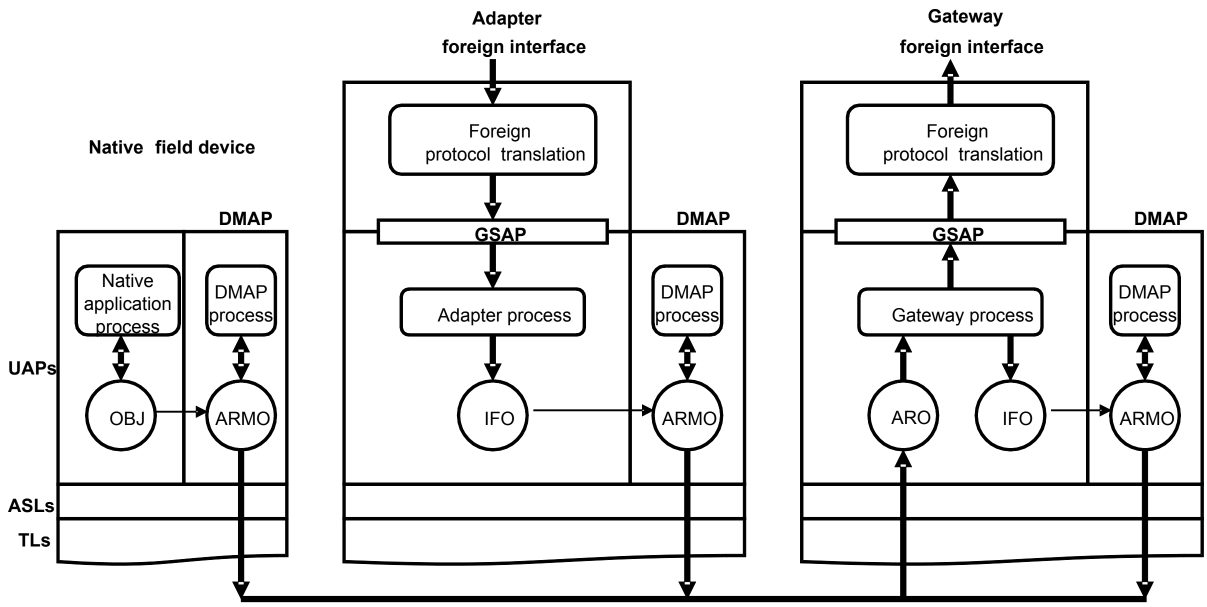


Figure U.28 – Alert model

Alerts fall into two classes, alarms and events. Events are informational and generate event messages through the GIAP. Alarms have states and require alarm-specific actions to clear the alarms. Usually, client/server messaging is used to perform these actions.

The gateway and the adapter applications are also able to generate native alerts from IFO instances. This allows protocol translators to generate alerts within the context of a standard alert management system.

In certain circumstances, the state of alerts may be lost at the ARO, such as when a gateway is reset or replaced. In such case, the original ARMOs will no longer contain information about events, but will maintain state information related to alarms. An alarm recovery procedure can be initiated in order to recover the system alarm state.

This standard does not support multicast alerts. As a result, the same alerts cannot be routed to both the gateway and the system manager if they are not physically co-located. Network and security alerts are currently sent to the system manger by default. Process and device alerts may be sent to the gateway role.

The alert model does not support multicast alerts. Network alerts and security alerts are potentially useful in a gateway for transformation into generic foreign protocol error messages. The system manager is the default destination for these alerts. In system configurations where the system manager is connected to the WISN via the gateway, the ARO in the gateway, when configured to collect alerts for network and security purposes, is capable of reposting the alerts through the local ARMO to the system manager. This is illustrated in Figure U.29.

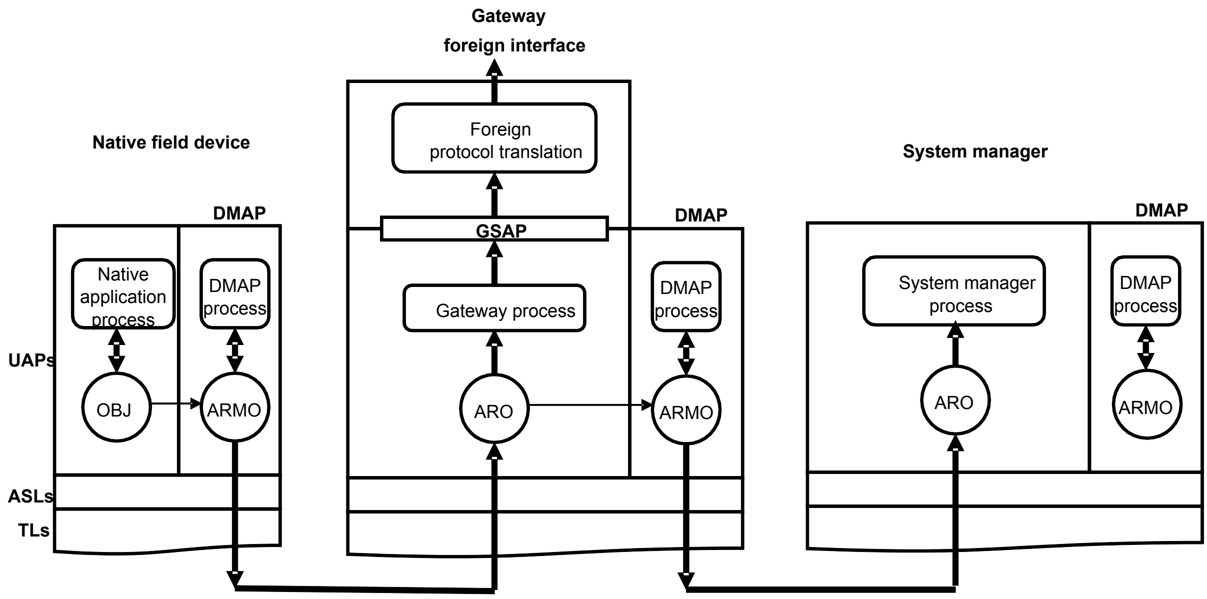


Figure U.29 – Alert cascading

U.3.4 Native publish/subscribe and client/server access

This standard provides publish/subscribe and client/server interfaces via the ASL that is used to interact with application-specific native objects.

For publish/subscribe, the concatenation (CON) and dispersion (DIS) objects are used as endpoints.

For client/server interfaces, two object endpoints are required in order to use these interfaces. The IFO may act as one endpoint for these interfaces within gateways. Any other application or management object within the system can act as the other endpoint.

As shown in Figure U.30, utilization of these objects allows protocol translators to integrate simple devices that do not include legacy protocols.

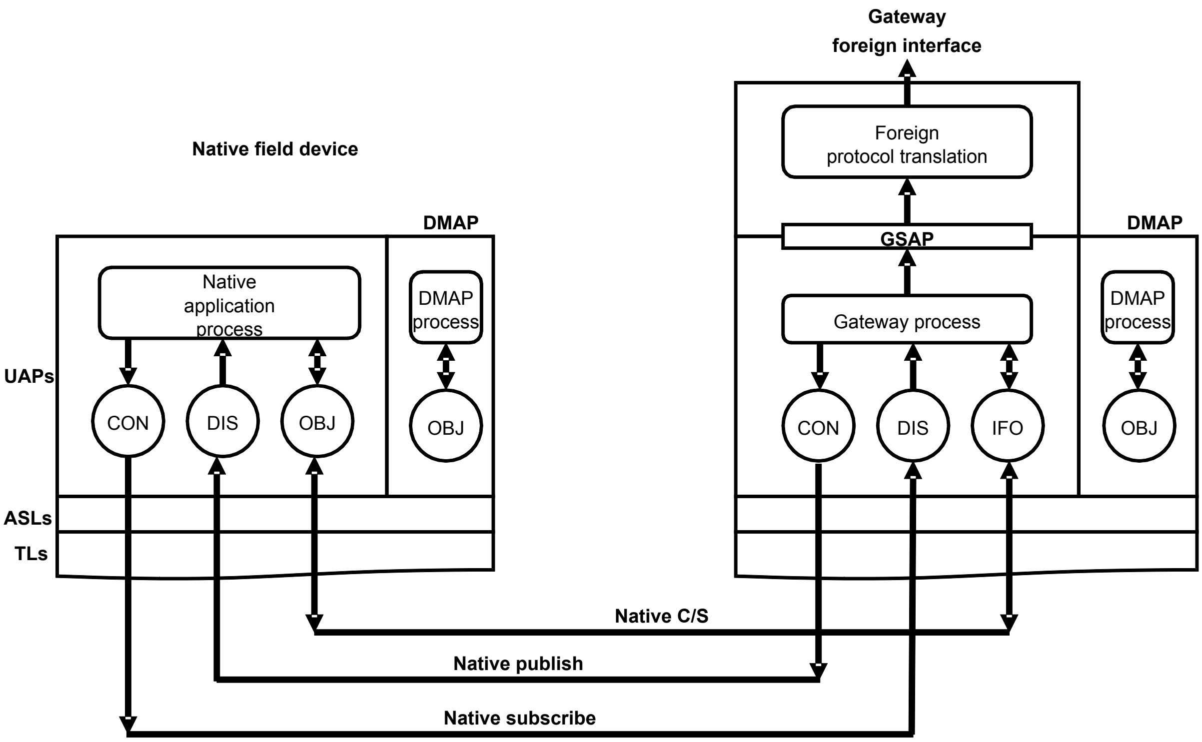


Figure U.30 – Native publish/subscribe and client/server access

Within a gateway, the CON and DIS objects may provide buffered message behavior for change of state operation.

Within a gateway, the IFO may provide buffered message behavior as described for the 4-part tunnel messaging between tunnel objects for client/server read interfaces. The IFO may use the attribute classification to determine buffering behavior. Non-cacheable attributes are not buffered. Constant attributes are buffered. Static and dynamic attribute buffering is determined by application requirements.

The protocol translator may use native addressing (Network_Address, Transport_Port, OID, and attribute identifier) to identify native messages.

NOTE Tunneling assumes that foreign protocol messages are transferred between endpoints. As such, foreign addresses are associated with the messages and used for teardown and reconstruction of the messages in order to avoid transfer. No such assumption is made for native messaging, where a one-to-one message flow is less likely to exist.

U.3.5 Time management

Host time may be propagated through a gateway to a wireless system, giving the host system and the field devices the same sense of time (within tolerances). This enables the host time to be used for purposes such as uniform alert timestamping and sequence of event determination that spans wireless and wired devices connected to the host. Without periodic synchronization to host time, the wireless system will drift, thus periodic adjustment capability is desirable. Both the host and wireless system may be synchronized to a common external source such as a GPS derived timesource.

To propagate host time, a gateway may perform periodic synchronization of time in an attached D-subnet time to an external source by requesting time changes through a DLMO.

Protocol translators within a gateway may access time management functions through the GIAP services. Protocol translators are responsible for accessing external time sources and converting protocols and time formats. Network time is represented in TAI format, as described in 5.6.

A DL configured as a clock master is used to propagate time synchronization information to an attached D-subnet, as described in 6.3.10.3. Each node contains a DMO within its DMAP. The DMO contains attributes DL_Subnet_Clock_Master_Role and DL_Subnet_Clock_Repeater_Role that control the ability of a node to be a clock master. Allocation of the clock master role is coordinated with the system manager. The device registers its ability to be a time source during the join process.

The DLMO contains an attribute called TaiTime that reports the current time and another attribute called TaiAdjust for adjusting the time. The DLMO is used to adjust the time of the D-subnet.

One or more DLs may be associated with a gateway. In one implementation, the DLs are integrated within the gateway. In another implementation, the DLs are within backbone routers and separated from the gateway, adding indeterminate delays. Each implementation may consider the implications of delay associated with access of DL objects to perform synchronization.

U.3.6 Security

Sets of wireless devices are related to a foreign host via a gateway. The gateway and the wireless devices are expected to belong to a common security group. Security for this group may be established by MAC or TL security configuration, or both as described in Clause 7. Establishment of common security settings is a prerequisite for communication between protocol translation communication endpoints.

Common foreign fieldbus protocols do not have security capabilities. This does not preclude extension of secured protocols into this standard's domain. It is the responsibility of foreign protocol translators in both gateways and adapters to act as trusted applications in the extension of foreign protocol security from end-to-end. This can be achieved by utilization of native security or through tunneled exchanges.

U.3.7 Configuration

For gateways which implement internal interfaces such as the example GSAP interfaces, it is recommended that the gateway entity configuration / capability be made internally available to gateway internal clients. How these gateway internal operational attributes are made available is a local matter. Examples attributes which the gateway GSAP entity may wish to present are described in Table U.41. For convenience the attribute describing conventions used are those used by other clauses of this standard. See also the G_Read_Gateway_Configuration interface example for an example of how an interface can be used to make this information available to a gateway-internal client.

Table U.41 – Example of gateway configuration management attributes

Standard object type name: not applicable				
Standard object type identifier: not applicable				
Attribute name	**Attribute identifier**	**Attribute description**	**Attribute data information**	**Description of attributes behavior**
Max_Devices	11	Maximum number of devices supported by gateway	Type: Unsigned16	Implementation dependent value set by gateway depending on resources
			Classification: Static	
			Accessibility: Read only	
			Default value: 0	
Actual_Devices	12	Current number of devices connected to gateway	Type: Unsigned16	Increases and decreases based on devices in communication with the gateway
			Classification: Dynamic	
			Accessibility: Read only	
Max_Leases	13	Maximum number of leases supported by the gateway	Type: Unsigned16	Implementation dependent value set by gateway depending on resources
			Classification: Static	
			Accessibility: Read only	
Actual_Leases	14	Current number of leases for devices connected to the gateway	Type: Unsigned16	Increases and decreases based on leases. Device complexity will determine the number of leases required
			Classification: Dynamic	
			Accessibility: Read only	

U.3.8 Provisioning and joining

A gateway is a network device as described in this standard and is provisioned using the generic methods described in this standard.

A gateway that communicates to D-subnets through backbone routers may provide a method to configure the gateway to communicate to a specific D-subnet and to specific devices within that D-subnet through a specific backbone router.

NOTE 1 Nothing precludes more dynamic implementations, such as a load-sharing algorithm that assigns devices to the best BBR found, or gateways and BBRs that discover each other, or support for redundancy that is provided automatically where D-subnets overlap.

A gateway that communicates to one or more D-subnets through backbone routers includes a method to configure the gateway to communicate with at least one system manager, where the system manager may reside:

- in the gateway;
- on a backbone that the gateway can use for communication; or
- within a D-subnet that the gateway can use for communication.

A gateway is a network device (containing an AL and IPv6Address) as described in this standard and joins the network following the join methods described in this standard.

NOTE 2 Several variations are possible, for example: a gateway that joins by sending an internal request to a co-resident system manager, or by sending a join request through a local PhL, or that uses a backbone router's PhL indirectly, or that sends the join request across the backbone to a system manager.

Annex V
(informative)

Country-specific and region-specific provisions

V.1 General

This standard is designed to support operation within a fixed geographic area that operates under uniform regulations. As such it is intended to support operation anywhere in the world, as discussed in 5.2.5 and 9.1.15.6.

This standard also is designed to support wireless automation systems operating on mobile platforms, such as marine vessels (e.g., container ships and petrochemical tanker ships) and trains, that can move between geographic regions (e.g., countries) where differing, and perhaps conflicting, regulations apply. For example, a container ship usually would be subject to local regulations when in port, and thus could have different compliance requirements when in Rotterdam than when in Tokyo Bay, because the regulations that apply to wireless systems when operating under EC jurisdiction differ from those that apply when operating under Japanese jurisdiction.

Radio regulations often require devices to operate at constrained power levels at all times, including during over-the-air provisioning. Some identified EIRP thresholds are: 10 mW/MHz (Japan); 10 dBm (China); 10 dBm, 10 mW/MHz and 20 dBm (EC ETSI); 36 dBm with at most 6 dBi antenna gain (US FCC).

In some countries, such as France, emission levels on certain channels may need to be attenuated. In other countries, such as Korea, the number and range of channels needs to be constrained.

V.2 Operation within a fixed regulatory regime

The dlmo.CountryCode field, described in 9.1.15.6, is used to specify the regulatory regime. It may also be used to specify some overriding regulatory constraints.

This field includes a “self-locking” mechanism, so that it is possible to set the value of this field to make the entire field unchageable while the device is operational. Once set, only reprovisioning the device, such as after a repair, is able to clear that lock.

This “set and forget” feature was included to support regulatory regimes, such as some within the EC, where no operational method may override the RF emission limits established under regulation. However, the feature is provided in a way that also supports device repair and resale into other regulatory jurisdictions, whose requirements might conflict with those of the jurisdiction into which the device was originally deployed.

V.3 Operation on a platform that moves between regulatory regimes

Some wireless automation systems may be located on a mobile platform such as a container ship or petrochemical tanker ship that moves between regulatory regimes, operating temporarily in each. That transition between regimes can occur rapidly, as when a train crosses a border, or slowly, as when a ship transits from national waters to international waters.

This standard is designed to support operation of wireless systems on such mobile platforms, by providing a means by which a single equipment parameter can be changed in each device, for example by a timed action downloaded in advance to each device, to cause all affected devices to change regulatory regimes, after which their operations are constrained by the regulations for the more-newly-adopted regime.

NOTE Such a change in wireless emission characteristics often will be accompanied by a change in link schedules, for example, to use more or fewer routers in a multi-hop path, or to have available more spare timeslots for retry of transactions that were aborted due to LBT-detected activity in the channel, to better match system operation to the more strict (or relaxed) requirements of the new regulatory regime.

V.4 Compliance with EN 300 328 [INFORMATIVE]

EN 300 328 is a complexly-interacting set of requirements which mandates specific declarations of operating behavior. Although only a certificate authority can determine whether conformance is actually achieved, it appears that there are seven different operating regimes under which a device conforming to this standard can meet the requirements of EN 300 328 v1.8.1, which within this annex is hereafter referred to as the “EN”.

Devices conforming to the EN are permitted to change their operating regime dynamically, at least within certain limits. Presumably each of these operating regimes would need to be tested independently for conformance and there would also need to be a submitted description of the conditions under which such dynamic changes in regime occur.

Under the EN, IEEE 802.15.4:2011 2,4 GHz DSSS qualifies as wideband modulation (WBM). As used in this standard it also qualifies as frequency-hopping spread-spectrum modulation (FHSSM) whenever the cyclic frequency-hopping schedule specifies at least 15 channels.

NOTE 1 Even with WBM, some frequency hopping is needed to avoid commonly-encountered narrow-band fading with a duration of more than a few ms. Thus frequency hopping will occur whether it is claimed for operation under FHSSM mode relative to conformance to the EN, or not.

NOTE 2 EN 4.3.1.3.2 permits blocking operation on some of the channels specified in the frequency-hopping schedule, but does not permit the number of channels in the cycle to be reduced to fewer than 15 channels. Therefore inclusion of fewer than 15 channels in a channel map that determines the frequency-hopping cycle of nominally-active channels means that the only possible remaining operating regimes are those under WBM.

In this standard, D-transaction initiators that enable CSMA/CA “listen before talk” (LBT) channel activity detection before sending each Data DPDU meet the EN requirements for “adaptive modulation”.

NOTE 3 These requirements are EN 4.3.1.6.1 (FHSSM) and EN 4.3.2.5.2.2.1 (WBM) and related text.

Under the EN,

- Tx-sequence-time is the transmitter-on time required to send a Data DPDU, which is ≤ 4,256 ms. In some cases it is also the the transmitter-on time to send an ACK/NAK DPDU,which is ≤ 1 ms;
- Tx-gap-time is the minimum required interval of non-transmission between the end of one transmission and the beginning of the next transmission by the same device; and
- “dwell time” (DT) is the nominal time that a D-transaction initiator using FHSSM keeps its transmitter tuned to a given channel before changing to another channel.

NOTE 4 Tx-sequence-time and Tx-gap-time are defined in EN 4.3.1.2 (FHSSM) and 4.3.2.3 (WBM). Dwell time, which applies to FHSSM, is defined to some extent in EN 3.1 under “frequency hopping spread spectrum” and in EN 4.3.1.3.1. Dwell time is necessarily at least as large as Tx-sequence-time.

EN 4.3.2.2.2 (WBM) imposes a power spectral density limit for WBM of 10 dBm/MHz. Due to the spectrum of the IEEE 802.15.4 2,4 GHz DSSS modulation, this constraint limits equipment operating under the EN’s WBM regulations to 20 mW (+13 dBm) maximum transmit power.

EN 4.3.1.1 (FHSSM) and EN 4.3.2.1 (WBM) limit maximum transmit power, after any antenna and beamforming gain, to 100 mW (+20 dBm).

EN 4.3.1.1 (FHSSM) and EN 4.3.2.1 (WBM) limit average transmit power of non-adaptive equipment, and of adaptive equipment operating in a non-adaptive mode, to 10 mW (+10 dBm). Use of adaptive modulation removes this restriction on average transmit power.

Equipment that always transmits at 10 mW or less has few special constraints.

When WBM without adaptive modulation is claimed, under EN 4.3.2.3 each D-transaction-respondent in one timeslot is not permitted to initiate a D-transaction in the immediately-following timeslot unless the intervening period of non-transmission meets the minimum Tx-gap-time requirement of 3,5 ms, which is inherently greater than the Tx-sequence-time for any just-sent ACK/NAK DPDU.

Similarly, when FHSSM without adaptive modulation is claimed, under EN 4.3.1.2 each D-transaction-respondent in one timeslot is not permitted to initiate a D-transaction in the immediately-following timeslot unless the intervening period of non-transmission meets the minimum Tx-gap-time requirement of 5 ms, which is inherently greater than the Tx-sequence-time for any just-sent DPDU.

When FHSSM with adaptive modulation is claimed, the ACK/NAK DPDUs that are sent by D-transaction respondents as immediate responses (within the same slot) to the Data DPDU sent by the D-transaction initiator can be considered “short control signaling” (SCS). While LBT is not required before transmitting SCS, under EN 4.3.1.6.3.2 SCS is constrained to occupy no more than 10% of the claimed dwell time. That restriction has an inverse impact on the minimum timeslot duration for the system, requiring the timeslot duration to be increased (and aggregate system throughput correspondingly decreased) relative to that otherwise required, just so that the channel occupancy of SCS (i.e., ACK/NAK DPDUs) in devices claiming conformance to FHSSM is never greater than 10% of the claimed nominal dwell time.

The recommended alternative approach to meeting EN 4.3.1.6.3.2 is to have each D-transaction-respondent dynamically mode-switch to operation in a non-adaptive mode while sending its ACK/NAK DPDU and for 5 ms thereafter (the mandated minimum Tx-gap-time), after which it reverts to the adaptive mode of operation. It appears that the only significant consequence of such a temporary non-adaptive operating mode is that the responding device is not permitted to initiate a D-transaction in the immediately-following time slot unless the intervening period of non-transmission meets the minimum Tx-gap-time requirement.

EN 4.3.1.3.2 (FHSSM) requires that each cyclic channel-hopping sequence contain a minimum of 15 channels, whether idle or active. In terms of this standard, this requirement means that only dlmo.Ch entries (Table 160) whose size field has a value of 15 or greater are suitable for use in FHSSM mode under the EN. Therefore, when channel-hopping sequences with cycle lengths less than 15 are used, operation under the EN shall necessarily conform to the EN’s WBM regulations.

It appears that a device conforming to this standard can comply with the requirements of the EN by being declaring to operate in any one of six categories and configuring its dlmo.CountryCode (9.1.15.6) attributes, particularly bits 10 and 12..14, appropriately:

1) low-power WBM equipment, with dlmo.CountryCode.mode=0b”x0011x”; or
2) non-adaptive WBM equipment, with dlmo.CountryCode.mode=0b”x0001x”; or
3) adaptive WBM equipment, with dlmo.CountryCode.mode=0b”x0101x”; or
4) low-power FHSSM hopping equipmen, with dlmo.CountryCode.mode=0b”x1011x”t; or
5) non-adaptive FHSSM equipment, with dlmo.CountryCode.mode=0b”x1001x”; or
6) adaptive FHSSM equipment that temporarily mode-switches to non-adaptive operation when operating as a D-transaction responder (i.e., to send an ACK/NAK DPDU) with dlmo.CountryCode.mode=0b”x1101x”.

NOTE 5 Although adaptive FHSSM equipment that does not temporarily mode-switch is possible, which is the seventh mode mentioned earlier in V.4, the constraints induced on declared dwell time and thus minimum timeslot duration required to operate under that set of constraints make such a hypothetical operating category inferior to 6), due to the massively reduced system throughput that such overly-extended timeslots necessarily induce.

NOTE 6 If regulators determine that equipment conforming to this standard does not meet the full regulatory intent for one or more of the above six possible categories, operation under any of the remaining categories is still possible.

Each of combinations 1) to 6) imposes a different set of contraints. Some are addressed automatically by all wireless devices that conform to this standard. Other constraints depend upon the claimed operating category. Whichever category is selected and configured via the device's dlmo.CountryCode attribute, the device shall operate in such a manner and take whatever action is required to conform to those constraints.

Summarizing the above, the regulatory constraints that require self-monitoring are:

a) for operation in categories 1 and 4, limiting the maximum transmitter output power, P_{out}Max, to less than 10 mW (+10 dBm);

b) for operation in categories 2 and 3, limiting the maximum transmitter output power, P_{out}Max, to 20 mW (+13 dBm), which is 10 mW/MHz for the signaling of IEEE 802.15.4 2.4 GHz DSSS;

c) for operation in categories 5 and 6, limiting the maximum transmitter output power, P_{out}Max, to 100 mW (+20 dBm);

d) for operation in categories 2 and 5, limiting the total number of transmissions, both of Data DPDUs and of ACK/NAK DPDUs, such that the mean transmitter output power, P_{out}Avg, is 10 mW (+10 dBm) or less over every 0,5 s measurement interval;

NOTE 7 Continuous averaging over shorter intervals is an acceptable way of meeting this requirement.

e) for operation in category 6, limiting the total number of transmissions of ACK/NAK DPDUs such that the mean transmitter output power, P_{out}Avg, used while transmitting ACK/NAK DPDUs is 10 mW (+10 dBm) or less over every 0,5 s measurement interval;

NOTE 8 The majority of channel occupancy by devices operating under category 6) occurs when sending Data DPDUs. Such transmissions qualify as adaptive modulation under EN 4.3.1.5, so constraint d) does not apply to them. However, devices operating under category 6) transmit ACK/NAK DPDUs in non-adaptive mode, to which constraint d) does apply per EN 4.3.1.5. Therefore it appears that only the totality of such ACK/NAK DPDUs transmitted by a device operating under category 6) are subject to the constraint d) power limit. If that interpretation of the EN is correct, then constraint d) will affect primarily backbone routers (BBRs), which in an automation WISN are largely receivers of process value-and-status publications and alert reports from WISN field devices. BBRs acting as transaction responders in duocast transaction may be impacted more than those not supporting duocast.

f) for operation in categories 2 and 5, meeting the required Tx-transmit-gap interval of non-transmission after transmission of a Data DPDU;

NOTE 9 This constraint is met automatically whenever the slot duration is ≥ 8,508 ms in mode 2, or ≥ 9,256 ms in mode 5.

g) for operation in categories 2, 5 and 6, meeting the required Tx-transmit-gap interval of non-transmission after transmission of an ACK/NAK DPDU.

For d), e), f) and g), the equipment shall dynamically monitor its recent activity to avoid transmitting whenever doing so would violate any of those three constraints.

NOTE 10 While a system manager can schedule device activity pessimistically to ensure that d), e), f) and g) are always met, it is the device's own responsibility to monitor its recent activity and inhibit transmission when doing so would violate regulatory constraints. Thus the ultimate responsibility for operation of a collection of devices rests with the individual devices themselves, not some remote manager that could be subverted by a successful cyberattack on a single device.

Bibliography

IEC 61158 (various parts), *Digital data communications for measurement and control – Fieldbus for use in industrial control systems*

IEC 61499-4:2005, *Function blocks – Part 4: Rules for compliance profiles*

IEC 61512-1, *Batch control – Part 1: Models and terminology*

IEC 61804-3, *Function blocks (FB) for process control - Part 3: Electronic device description language (EDDL)*

IEC 62264-1:2003, *Enterprise-control system integration – Part 1: Models and terminology*

IEC/TS 62351-2:2008, *Power systems management and associated information exchange – Data and communications security – Part 2: Glossary of terms*

IEC/TR 62390:2005, *Common automation device – Profile guideline*

IEC/TS 62443-1-1:2009, *Industrial communication networks – Network and system security – Part 1-1: Terminology, concepts and models*

IEC 62591, *Industrial communication networks – Wireless communication network and communication profiles – WirelessHART™*

IEC 62601, *Industrial communication networks – Fieldbus specifications – WIA-PA communication network and communication profile*

IEC/TS 62657-2, *Industrial communication networks – Wireless communication networks – Part 2: Coexistence management*

ISO/IEC 2375, *Information technology – Procedure for registration of escape sequences and coded character sets*

ISO/IEC 2382-14:1997, *Information technology – Vocabulary – Part 14: Reliability, maintainability and availability*

ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 7498-2, *Information processing systems – Open systems interconnection – Basic reference model – Part 2: Security architecture*

ISO/IEC 7498-3:1997, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 7498-4, *Information processing systems – Open systems interconnection – Basic reference model – Part 4: Management framework*

ISO/IEC 9646-7, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 7: Implementation Conformance Statements*

ISO/IEC 9796-2:2010, *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms*

ISO/IEC 9797-1:2011, *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*

ISO/IEC 9797-2:2011, *Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a dedicated hash-function*

ISO/IEC 9798-1:2010, *Information technology – Security techniques – Entity authentication – Part 1: General*

ISO/IEC 10116:2006, *Information technology – Security techniques – Modes of operation for an n-bit block cipher*

ISO/IEC 10118-2, *Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher*

ISO/IEC 10118-3, *Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*

ISO/IEC 10181-1:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Overview*

ISO/IEC 11770-1:2010, *Information technology – Security techniques – Key management – Part 1: Framework*

ISO/IEC 11770-2, *Information technology – Security techniques – Key management – Part 2: Mechanisms using symmetric techniques*

ISO/IEC 11770-3:2008, *Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques*

ISO/IEC 15408, *Information technology – Security techniques – Evaluation criteria for IT security*

ISO/IEC 18028-3:2005, *Information technology – Security techniques – IT network security – Part 3: Securing communications between networks using security gateways*

ISO/IEC 18031:2011, *Information technology – Security techniques – Random bit generation*

ISO/IEC 18033-1:2005, *Information technology – Security techniques – Encryption algorithms – Part 1: General*

ISO/IEC 18033-2, *Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*

ISO/IEC 19790:2012, *Information technology – Security techniques – Security requirements for cryptographic modules*

ISO/IEC 26907:2009, *Information technology – Telecommunications and information exchange between systems – High-rate ultra-wideband PHY and MAC standard*

ISO/IEC 27000:2009, *Information technology – Security techniques – Information security management systems – Overview and vocabulary*

ISO/IEC/IEEE 60559, *Binary floating-point arithmetic for microprocessor systems*

ISO 2382-12:1988, *Information processing systems – Vocabulary – Part 12: Peripheral equipment*

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*

ISO 11568-2:2012, *Financial services – Key management (retail) – Part 2: Symmetric ciphers, their key management and life cycle*

ISO 11568-4:2007, *Banking – Key management (retail) – Part 4: Asymmetric cryptosystems -- Key management and life cycle*

ISO 21188:2006, *Public key infrastructure for financial services – Practices and policy framework*

IEEE 802.1Q, Virtual bridged local area networks

IEEE 802.3, IEEE Standard for Information technology-Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications

NOTE 1 ISO/IEC 8802-3 is based on IEEE 802.3, usually with some delay in publication.

IEEE 802.11, IEEE standards for information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN medium access control (MAC) and physical layer (PhL) specifications

NOTE 2 ISO/IEC 8802-11 is based on IEEE 802.11, usually with some delay in publication.

IEEE 802.15.1, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)

NOTE 3 ISO/IEC 8802-15-1 is based on IEEE 802.15.1, usually with some delay in publication.

IEEE Std 802.15.4e-2012, (Amendment to IEEE Std 802.15.4-2011) IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer

IEEE 802.16, IEEE Standard for local and metropolitan area networks–Part 16: Air interface for fixed broadband wireless access systems

IERS conventions: IERS technical note 32

IETF RFC 1350, The TFTP protocol, rev. 2

IETF RFC 2347, TFTP option extension

IETF RFC 2348, TFTP blocksize option

IETF RFC 2349, TFTP timeout interval and transfer size options

IETF RFC 2525, Known TCP implementation problems

IETF RFC 3280, Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, available at

IETF RFC 5348, TCP-friendly rate control (TFRC): Protocol specification

IETF RFC 4949, Internet security glossary, rev. 2

ERC/REC 70-03, Relating to the use of short range devices (SRD), Annex 1, Band E

ETSI EN 300 220-1, Electromagnetic compatibility and radio spectrum matters (ERM) – Short range devices – Technical characteristics and test methods for radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mW – Part 1: Parameters intended for regulatory purposes

ETSI EN 300 328-1 v1.8.1, Radio equipment and systems (RES) – Wideband transmission systems – Technical characteristics and test conditions for data transmission equipment operating in the 2,4 GHz ISM band and using spread spectrum modulation techniques

ETSI EN 300 328-2 v1.8.1, Electromagnetic compatibility and radio spectrum matters (ERM) – Wideband transmission systems – Data transmission equipment operating in the 2,4 GHz ISM band and using spread spectrum modulation techniques – Part 2: Harmonized EN covering essential requirements under article 3.2 of the R&TTE Directive

ISC RSS 210, Radio standards specification 210 – Low-power license-exempt radio communication devices (all frequency bands): Category I equipment

ANSI X9.82-1, Random number generation – Part 1: Overview and basic principles

ANSI/ISA 100.11a:2011, Wireless Systems for Industrial Automation: Process Control and Related Applications

ISA TR100.00.01-2006, The Automation Engineer's Guide to Wireless Technology Part 1 – The Physics of Radio, a Tutorial

[US] FIPS 186-3, Digital Signature Standard (DSS)

[US] FIPS 197, Advanced encryption standard (AES)

[US] FIPS 198, The keyed-hash message authentication code (HMAC)

[US] NIST SP 800-22, A statistical test suite for random and pseudorandom number generators for cryptographic applications

[US] NIST SP 800-38C, Recommendation for block cipher modes of operation – The CCM mode for authentication confidentiality

[US] NIST SP 800-56A, Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography

[US] NIST SP 800-57, Recommendation for key management – Part 1: General

[US] NIST SP 800-57, Recommendation for key management – Part 2: Best practices for key management organization

[US] NIST SP 800-88, rev. 1, Guidelines for media sanitization

[US] Code of Federal Regulations (CFR) Title 47, Chapter I, Part 15 – *Telecommunication – Part 15: Radio frequency devices*

NAMUR Recommendation NE105, Specifications for integrating fieldbus devices

NAMUR Recommendation NE107, Self-monitoring and diagnostics of field devices

Guidelines for 64-bit Global Identifier (EUI-64™), available at http://standards.ieee.org/develop/regauth/tut/eui64.pdf.

HCF_SPEC-183, *Common Tables Specification*, available to members of the HART Communication Foundation, http://www.hartcomm.org

A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of applied cryptography*, ISBN 0-8493-8523-7

D. R. L. Brown, R. P. Gallant, S. A. Vanstone, *Provably secure implicit certificate schemes*, pp. 156-165 of ISBN 3-540-44079-8

F. Stajano, *The resurrecting duckling: What next?*, in *Proceedings of the 8th international workshop on security protocols*, B. Crispo, M. Roe, and B. Crispo, Eds., Lecture notes in computer science, Vol. 2133, Berlin: Springer-Verlag, April 2000.

F. Stajano, R. Anderson, *The resurrecting duckling: Security issues in ad-hoc wireless networks*, in Proceedings of the 7th international workshop on security protocols, B. Christianson, B. Crispo, J.A. Malcolm, and M. Roe, Eds., Lecture notes in computer science, Vol. 1796, Berlin: Springer-Verlag, 1999.

J. Jonsson, *On the security of CTR + CBC-MAC*, in *Proceedings of selected areas in cryptography – SAC 2002*, K. Nyberg, H. Heys, Eds. Lecture notes in computer science, Vol. 2595, pp. 76-93, Berlin: Springer, 2002.

J. Jonsson, *On the security of CTR + CBC-MAC, NIST mode of operation – Additional CCM documentation*, http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ccm/ccm-ad1.pdf

PKIX, L. Bassham, R. Housley, W. Polk, *Algorithms and identifiers for the internet X.509 Public key infrastructure certificate and CRL profile*, ftp://ftp.isi.edu/in-notes/rfc3279.txt

P. Rogaway, D. Wagner, *A critique of CCM*, IACR ePrint Archive 2003-070, April 13, 2003

R. Housley, D. Whiting, N. Ferguson, *Counter with CBC-MAC (CCM)*, submitted to NIST., June 3, 2002

____