IEC 61158-6-25

Edition 1.0 2019-06

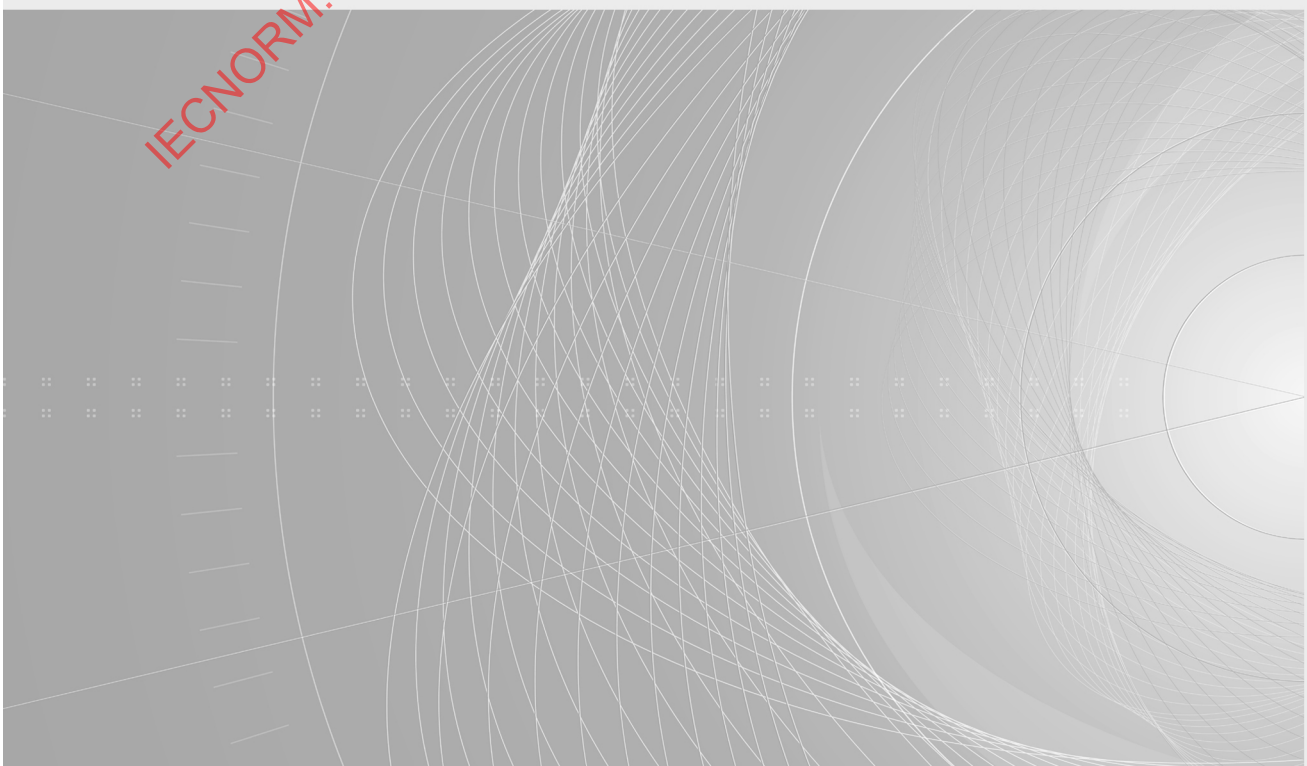
INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-25: Spécification du protocole de la couche application – Éléments de type 25



IEC 61158-6-25:2019-06(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

IEC 61158-6-25

Edition 1.0 2019-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-25: Spécification du protocole de la couche application – Éléments de type 25

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-9173-3

CONTENTS

FOREWORD 7
INTRODUCTION 9
1 Scope 10
1.1 General 10
1.2 Specification 11
1.3 Conformance 11
2 Normative references 11
3 Terms, definitions, symbols, abbreviations and conventions 12
3.1 Reference model terms and definitions 12
3.1.1 ISO/IEC 7498-1 terms 12
3.1.2 ISO/IEC 8822 terms 12
3.1.3 ISO/IEC 9545 terms 12
3.1.4 ISO/IEC 8824-1 terms 13
3.2 Additional Type 25 terms and definitions 13
3.3 Symbols and abbreviations 15
3.4 Conventions 16
3.4.1 General conventions 16
3.4.2 Conventions for class definitions 16
3.4.3 Conventions for bit description in octets 16
3.4.4 Conventions for state machine descriptions 17
4 FAL syntax description 18
4.1 FAL PDU type S abstract syntax 18
4.1.1 Basic abstract syntax 18
4.2 FAL PDU type N abstract syntax 22
4.2.1 Basic abstract syntax 22
4.2.2 CyclicData-PDU 22
4.2.3 MulticastData-PDU 23
4.2.4 PtoPData-PDU 23
4.2.5 Aliveinfo-PDU 23
4.2.6 Aliveinfo6-PDU 23
4.2.7 Inq-PDU 24
4.2.8 Ninq-PDU 24
4.2.9 Reply-PDU 24
4.2.10 RetransEnq-PDU 24
4.2.11 RetransConfirm-PDU 24
4.2.12 RetransNak-PDU 25
4.3 Data type assignments for type S 25
4.4 Data type assignments for type N 25
5 FAL transfer syntax 26
5.1 Encoding rules 26
5.1.1 Unsigned encoding 26
5.1.2 Octet string encoding 27
5.1.3 SEQUENCE encoding 27
5.2 FALPDU type S elements encoding 27
5.2.1 RCL_header 27
5.2.2 RHE-PDU 28

5.2.3 LCC-PDU 30
5.2.4 LCA-PDU 30
5.2.5 LCN-PDU 31
5.2.6 LNA-PDU 32
5.2.7 SCR-PDU 32
5.2.8 Cyclic_S-PDU 32
5.2.9 Cyclic_header 33
5.2.10 Control-PDU 33
5.2.11 RMTCTL-PDU 33
5.2.12 INFO-PDU 34
5.3 FALPDU type N elements encoding 35
5.3.1 General 35
5.3.2 FALAR-N Header 35
5.3.3 CyclicData-PDU 49
5.3.4 MulticastData-PDU 50
5.3.5 PtoP Data-PDU 50
5.3.6 Aliveinfo-PDU 50
5.3.7 Aliveinfo6-PDU 53
5.3.8 Inq-PDU 54
5.3.9 Ninq-PDU 55
5.3.10 Reply-PDU 55
5.3.11 RetransEnq-PDU 56
5.3.12 RetransConfirm-PDU 56
5.3.13 RetransNak-PDU 57
6 Structure of the FAL protocol state machine 58
7 FAL service protocol machine (FSPM) 58
7.1 Overview 58
7.2 FSPM type S 58
7.2.1 Overview 58
7.2.2 Interface of cyclic communication to FAL users 59
7.2.3 State machine of FSPM 61
7.3 FSPM type N 62
7.3.1 Overview 62
7.3.2 FSPM 63
8 Application relationship protocol machine (ARPM) 65
8.1 ARPM type S 65
8.1.1 Overview 65
8.1.2 Cyclic control 66
8.1.3 Remote control 70
8.1.4 RCL communication control 75
8.1.5 RT communication control 79
8.2 ARPM type N 83
8.2.1 Overview 83
8.2.2 General control 83
8.2.3 Cyclic transmission control 85
8.2.4 Acyclic transmission control 90
8.2.5 RT communication control 105
9 DLL mapping protocol machine (DMPM) 121

9.1 DMPM type S 121
9.2 DMPM type N 122
9.2.1 General 122
9.2.2 Communication port in transport layer 122
9.2.3 Quality of Service 122
Bibliography 124

Figure 1 – Bit description in octets 16
Figure 2 – hd_sa 35
Figure 3 – hd_da 36
Figure 4 – Valid sequence number for reception message 40
Figure 5 – hd_m_ctl 40
Figure 6 – Valid reception packet sequence number 44
Figure 7 – Node-list 55
Figure 8 – Relationships between protocol machines 58
Figure 9 – Structure of FSPM type S 59
Figure 10 – Shared memory allocation in type S network 61
Figure 11 – Structure of FSPM type N 63
Figure 12 – Structure of ARPM type S 66
Figure 13 – Sequence of cyclic communication 67
Figure 14 – The primitives for cyclic control 67
Figure 15 – The primitives for Remote control 70
Figure 16 – The primitives for RCL communication control 75
Figure 17 – The primitives for RT communication control 80
Figure 18 – Structure of ARPM type N 83
Figure 19 – Primitives of Cyclic transmission control 85
Figure 20 – Primitives of acyclic transmission control 91
Figure 21 – DSCP format 123
Figure 22 – IEEE 802.1Q tag frame format 123

Table 1 – State transition descriptions 17
Table 2 – Descriptions of state machine elements 17
Table 3 – Conventions used in state machine 17
Table 4 – Frame Class 27
Table 5 – DA_STaddress – DA_STaddress 27
Table 6 – DA_MACaddress 27
Table 7 – CMD field format 28
Table 8 – Send Direction 28
Table 9 – RHE ReceiveStatus 29
Table 10 – Physical Linkdown 29
Table 11 – RHE_pattern 1~4 30
Table 12 – LCC-Kind 30
Table 13 – RCL Status 31
Table 14 – hd_h_type 35

Table 15 – Usage of Mgn or Lnn 37
Table 16 – Detailed conditions for sequence number check of reception message 39
Table 17 – Valid bits of hd_m_ctl 40
Table 18 – Specified TCD 41
Table 19 – hd_pkind 42
Table 20 – PDU with an effective hd_pseq 42
Table 21 – Detailed conditions for sequence number check of reception packet (Multicast communication with retransmission) 44
Table 22 – Detailed conditions for packet sequence number check 45
Table 23 – Relation between message transmission/reception 46
Table 24 – hd_mode 46
Table 25 – Message priority level 46
Table 26 – Value of α 47
Table 27 – Example of header information for a UDP message fragmentation 47
Table 28 – Example of header information for a TCP message fragmentation 47
Table 29 – inqid_inq_sa value 48
Table 30 – inqid_tr_adr value 48
Table 31 – inqid_inq_seq value 49
Table 32 – Relationship between inqid_id_seq and inqid_tr_adr 49
Table 33 – Type of an alive-message 51
Table 34 – Type of an alive-message protocol 51
Table 35 – Time of each al_mode 52
Table 36 – Status change of tasks 52
Table 37 – Change of tasks content 52
Table 38 – The threshold of transmission factor 60
Table 39 – Example of the traffic control configuration menu 60
Table 40 – Cyclic data state table 61
Table 41 – Acyclic data state table 62
Table 42 – Cyclic data state table 64
Table 43 – Acyclic data state table 64
Table 44 – Cyclic control state table 68
Table 45 – Cyclic control functions 69
Table 46 – Cyclic control variables 70
Table 47 – Remote control state table 71
Table 48 – Remote control functions 74
Table 49 – Remote control variables 75
Table 50 – RCL communication control state table 76
Table 51 – RCL communication control functions 77
Table 52 – RCL communication control variables 79
Table 53 – RT communication control state table 80
Table 54 – RT communication control functions 82
Table 55 – RT communication control variables 82
Table 56 – Cyclic transmission control state table 86

Table 57 – Cyclic transmission control functions 88
Table 58 – Cyclic transmission control variables 90
Table 59 – Acyclic transmission control state table 91
Table 60 – Acyclic transmission control functions 101
Table 61 – Acyclic transmission control variables 104
Table 62 – RT communication control state table 105
Table 63 – RT communication control functions 116
Table 64 – RT communication control variables 120
Table 65 – ARPM to DL mapping 122
Table 66 – Assignment policy of communication ports 122
Table 67 – Default DSCP, IEEE 802.1D and IEEE 802.1Q priority mapping 123

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-25: Application layer protocol specification – Type 25 elements

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.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61158-6-25 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this International Standard is based on the following documents:

FDIS	Report on voting
65C/948/FDIS	65C/956/RVD

Full information on the voting for the approval of this International 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.

A list of all parts in the IEC 61158 series, published under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC website.

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.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

This document is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the “three-layer” fieldbus reference model described in IEC 61158-1.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this document is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer application entities (AEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- as a guide for implementers and designers;
- for use in the testing and procurement of equipment;
- as part of an agreement for the admittance of systems into the open systems environment;
- as a refinement to the understanding of time-critical communications within OSI.

This document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may work together in any combination.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-25: Application layer protocol specification – Type 25 elements

1 Scope

1.1 General

The Fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a “window between corresponding application programs.”

This International Standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 25 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This document defines in an abstract way the externally visible behavior provided by the different Types of the fieldbus Application Layer in terms of:

a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,

b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,

c) the application context state machine defining the application service behavior visible between communicating application entities; and

d) the application relationship state machines defining the communication behavior visible between communicating application entities.

The purpose of this document is to define the protocol provided to:

a) define the wire-representation of the service primitives defined in IEC 61158-5-25, and

b) define the externally visible behavior associated with their transfer.

This document specifies the protocol of the IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI Application Layer Structure (ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can

send/receive is specified. This permits greater flexibility to the FAL users in standardizing such object behavior. In addition to these services, some supporting services are also defined in this document to provide access to the FAL to control certain aspects of its operation.

1.2 Specification

The principal objective of this document is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-25. A secondary objective is to provide migration paths from previously-existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in subparts of IEC 61158-6.

1.3 Conformance

This document does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to the application layer service definition standard. Instead, conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-3-25:2019, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC 61158-5-25:2019, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO/IEC/IEEE 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet*

IEEE Std 802.1D, *IEEE Standard for Local and metropolitan area networks – Media access Control (MAC) Bridges*; available at http://www.ieee.org [viewed 2018-09-17]

IEEE Std 802.1Q, *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks*; available at http://www.ieee.org [viewed 2018-09-17]

IETF RFC 768, *User Datagram Protocol*; available at http://www.ietf.org> [viewed 2018-09-17]

IETF RFC 791, *Internet Protocol*; available at http://www.ietf.org> [viewed 2018-09-17]

3 Terms, definitions, symbols, abbreviations and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1 Reference model terms and definitions

3.1.1 ISO/IEC 7498-1 terms

For the purposes of this document, the following terms given in ISO/IEC 7498-1 apply:

a) application entity;
b) application process;
c) application protocol data unit;
d) application service element;
e) application entity invocation;
f) application process invocation;
g) application transaction;
h) real open system;
i) transfer syntax.

3.1.2 ISO/IEC 8822 terms

For the purposes of this document, the following terms given in ISO/IEC 8822 apply:

a) abstract syntax;
b) presentation context.

3.1.3 ISO/IEC 9545 terms

For the purposes of this document, the following terms given in ISO/IEC 9545 apply:

a) application-association;
b) application-context;
c) application context name;
d) application-entity-invocation;
e) application-entity-type;
f) application-process-invocation;
g) application-process-type;
h) application-service-element;

i) application control service element.

3.1.4 ISO/IEC 8824-1 terms

For the purposes of this document, the following terms given in ISO/IEC 8824-1 apply:

a) object identifier;

b) type.

3.2 Additional Type 25 terms and definitions

For the purpose of this document, the following definitions also apply.

NOTE Many definitions are common to more than one protocol Type; they are not necessarily used by all protocol Types.

3.2.1
ADP message
message conveyed by an autonomous decentralized system protocol

3.2.2
alive-message
message reporting own node state, periodically transmitted

3.2.3
block
basic unit of data transferred in a cyclic communication, each having a size of 64 octets

3.2.4
category N_ f
category of an autonomous decentralized system protocol (full specifications) in type N

3.2.5
category N_l
category of an autonomous decentralized system protocol (light weight specifications) in type N

3.2.6
control communication
acyclic data communication for higher time-critical applications in type S network

3.2.7
cyclic communication
periodic data communication for real-time communication

3.2.8
cyclic transfer memory
memory which is allocated to each node in the data field, which each node transmits periodically for the purpose of sharing of this memory area logically

3.2.9
data field
logical place through which specific data passes, corresponding to real networks

3.2.10
domain
administrative set consisting of multiple data fields

3.2.11
duplex LAN
two different network-paths between end nodes

3.2.12
equipment
physical hardware connected to the network

EXAMPLE Station, device, and server.

3.2.13
information communication
acyclic data communication for lower time-critical application in type S network

3.2.14
logical node, node
equipment in data fields, where an autonomous decentralized system's function is installed

3.2.15
message mode
identifier that indicates the uses of a message (whether an online message or a test message)

EXAMPLE Online message or a test message.

3.2.16
multicast group
group of nodes belonging to a data field to determine whether to receive same data from other node

3.2.17
node list
bit array structure of specifying a node to receive a request data

3.2.18
node mode
identifier that indicates the usage of a node

EXAMPLE Online node or a test node.

3.2.19
primary local area network
local area network (LAN) used in normal state in a duplex LAN system

3.2.20
priority control
transmission order control for the packets arrived from networks, according to the virtual local area network (VLAN)

3.2.21
ring control communication
control for the type S ring network using ring control (RCL) frames

3.2.22
remote control
control for other nodes connected to a network

3.2.23
secondary local area network
local area network (LAN) for backup of a primary LAN in a duplex LAN system

3.2.24
transaction code
information to identify the characteristics of the message within a PDU

3.3 Symbols and abbreviations

ADP	Autonomous Decentralized Protocol
ADS-net	Autonomous Decentralized System – network
AE	Application Entity
AL	Application Layer
AP	Application Process
APDU	Application Protocol Data Unit
APO	Application Process Object
AR	Application Relationship
AREP	Application Relationship Endpoint
ARPM	Application Relationship Protocol Machine
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
ASN.1	Abstract Syntax Notation 1
CRC	Cyclic Redundancy Check
DF	Data Field
Dfn	Data Field Number
DLL	Data-link Layer
Dmn	DoMain Number
DMPM	DLL Mapping Protocol Machine
DSCP	DiffServ Code Point
FAL	Fieldbus Application Layer
FSPM	FAL Service Protocol Machine
GMT	Greenwich Mean Time
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local area network
LCA	Loop condition alert (Type S frame type)
LCC	Loop condition check (Type S frame type)
LCN	Loop condition notify (Type S frame type)
LLD	Logical link down
LLU	Logical link up
LNA	Loop notify answer (Type S frame type)
Lnn	Logical Node Number
LSB	Least Significant Bit
MCG	MultiCast Group
Mgn	Multicast Group Number
MSB	Most Significant Bit
MTU	Maximum Transmission Unit

OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PLD	Physical link down
QoS	Quality-of-service
RCL	Ring control
RHE	Rapid hello (Type S frame type)
RMTCTL	Remote control
RT	Real time
SCR	Station condition report (Type S frame type)
TCD	Transaction CoDe
TCP	Transmission Control Protocol
TMID	Transfer Memory Identifier
UDP	User Datagram protocol

3.4 Conventions

3.4.1 General conventions

Conventions given in IEC 61158-3-25 are also applied in this document.

3.4.2 Conventions for class definitions

This description of FAL Type 25 uses a subset of ASN.1. The following structures are used. Selective type (CHOICE) – Represents a selection from candidate types.

Sequence type (SEQUENCE) – Represents a fixed-order list as in the following example:

```
DLPDU::= SEQUENCE {
preamble    Preamble,
sfd         SFD,
destaddr    DestAddr,
srcaddr     SrcAddr,
vlantag     VLANtag,
lt          LT,
dlsdu       FAL-PDU,
fcs         FCS
}
```

NOTE This example shows that the DLPDU which represents the Ethernet frame is defined as SEQUENCE. The DLPDU consists of Preamble, SFD, DestAddr, SrcAddr, VLANtag, LT, FALL-PDU and FCS.

3.4.3 Conventions for bit description in octets

When identifying each bit in an octet, each bit is identified by a number as shown in Figure 1 and described as Bit n.

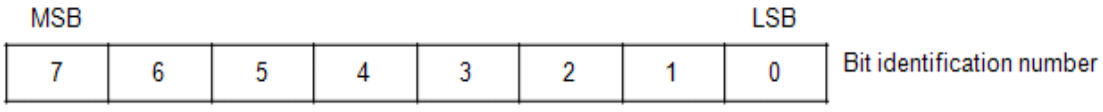


Figure 1 – Bit description in octets

When specifying multiple bits sequentially located, the range symbol (..) is used (e.g.: 7..0, specifies bits 7 through 0, inclusive).

When specifying multiple octets, the LSB of the lowest octet is assigned as 0, and bit identification numbers are assigned in an ascending order.

NOTE For example, when specifying 4 octets, the MSB of the highest octet is Bit 31, the MSB of the second octet is Bit 23, the MSB of the third octet is Bit 15, and the MSB of the lowest octet is Bit 7.

3.4.4 Conventions for state machine descriptions

The state machine description is defined in tabular form as shown in Table 1. The meanings of the elements are shown in Table 2. The conventions used in the state machines are shown in Table 3.

Each row of state table represents a state transition. The first column shows the state transition name or number. The second column shows the current state. The third column shows the events, conditions and actions. The fourth column shows the next state. When an event or condition is fulfilled, the action is performed and the state machine transits to the next state.

Table 1 – State transition descriptions

#	Current state	Event /condition => actions	Next state

Table 2 – Descriptions of state machine elements

Description element	Meaning
#	Number of the transition
Current state, Next state	Names of the originating state and the target state of transition
Event	Name or description of the trigger event that fire the transition
/conditions	Boolean expression, which shall be true for the transition to be fired
=>actions	List of assignments and service or function invocations. The action should be atomic. The preceding "=>" is not part of the action
"*/ conditions" may be omitted.	

Table 3 – Conventions used in state machine

Convention	Meaning
=	Substitution of the right side for the left side
==	A logical condition to indicate an item on the left is equal to an item on the right.
!=	A logical condition to indicate an item on the left is not equal to an item on the right
<	A logical condition to indicate an item on the left is less than the item on the right
>	A logical condition to indicate an item on the left is greater than the item on the right
&&	Logical "AND"
||	Logical "OR"
!	Negation operator
+ – * /	Arithmetic operator
;	Breakpoint
++	Increment operator
--	Decrement operator

4 FAL syntax description

4.1 FAL PDU type S abstract syntax

4.1.1 Basic abstract syntax

The definitions of FALPDU are shown below.

```
FAL-PDU::= CHOICE {
        RCL-PDU,
        RT-PDU
}

RCL-PDU::= CHOICE {
        RHE-PDU,
        LCC-PDU,
        LCA-PDU,
        LCN-PDU,
        LNA-PDU,
        SCR-PDU
}

RT-PDU::= CHOICE {
        Cyclic_S-PDU,
        CTL-PDU,
        INFO-PDU
}

CTL-PDU::= CHOICE {
        Control-PDU,
        RMTCTL-PDU
}
```

4.1.1.1 RCL-PDU

RCL_header to be used in each RCL-PDU is shown as follows.

```
RCL_header::= SEQUENCE {
        Frame Class             CLASS,
        Dst Address             SEQUENCE {
                Reserved1                   Unsigned8,
                DA_STaddress                ST_Address,
                DA_MACaddress               MAC_Address
        },
        Src Address             SEQUENCE {
                Reserved2                   Unsigned8,
                SA_STaddress                ST_Address,
                SA_MACaddress               MAC_Address
        },
        RCL Command             RCLCMD,
        Frame Sequence          SequenceNum32,
        Reserved3               OCTET STRING(SIZE 20)
}
```

4.1.1.1.1 RHE-PDU

```
RHE-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        RHE_header          SEQUENCE {
                RHE_Common          SEQUENCE {
                        Send Direction              PortNum,
                        RHE Sequence                SequenceNum32
                        },
                Neighborhood address        SEQUENCE {
                        Reserved1                   Unsigned32,
                        NB_STaddress                ST_Address,
                        NB_MACaddress               MAC_Address
                        },
                RHE ReceiveStatus           RHE_RxStatus,
                True Status                 SEQUENCE {
                        Reserved2                   Unsigned32,
                        Physical Linkdown           PLD,
                        Reserved3                   Unsigned32
                        },
                Blocking Status                     BLK_Status,
                Node Status                         Node_Status,
                Reserved4                           OCTET STRING(SIZE 20)
},
        Reserved5           OCTET STRING(SIZE 64),
        RHE_pattern         SEQUENCE {
            RHE_pattern1            Pattern32,
            RHE_pattern2            Pattern32,
            RHE_pattern3            Pattern32,
            RHE_pattern4            Pattern32,
                .
                .
                .
}
```

4.1.1.1.2 LCC-PDU

```
LCC-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCC_header          SEQUENCE {
                LCC Kind                RCLKIND,
                LCC Priority    SEQUENCE {
                        LCC_Priority            Priority,
                        LCC_STadd               ST_Address,
                        LCC_MACadd              MAC_Address
                },
                Reserved1               OCTET STRING(SIZE 52)
        }
}
```

4.1.1.1.3 LCA-PDU

```
LCA-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCA_header          SEQUENCE {
                Reserved1                   Unsigned32,
                RCL Status                  RCL_Status,
                LCA Priority            SEQUENCE {
                        LCA_Priority            Priority,
                        LCA_STadd               ST_Address,
                        LCA_MACadd              MAC_Address
                },
```

```
        Reserved2                   OCTET STRING(SIZE 48)
        }
}
```

4.1.1.1.4 LCN-PDU

```
LCN-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCN_header          SEQUENCE {
                Reserved1                   Unsigned32,
                RCL Status                  RCL_Status,
                Reserved2                   OCTET STRING(SIZE 56)
        }
}
```

4.1.1.1.5 LNA-PDU

```
LNA-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LNA_header          SEQUENCE {
                Reserved1           Unsigned32,
                RCL Status          RCL_Status,
                Reserved2           OCTET STRING(SIZE 56)
        }
}
```

4.1.1.1.6 SCR-PDU

```
SCR-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        SCR_header          SEQUENCE {
                Entry Number        EntryNum,
                SendStat Time       Send_Time,
                Reserved1           OCTET STRING(SIZE 52)
},
        Event_info          SEQUENCE {
                Event0              EventData,
                Event1              EventData,
                    .
                    .
                    .
                Event16             EventData
        }
}
```

4.1.1.2 RT-PDU

4.1.1.2.1 Cyclic_S-PDU

```
Cyclic_S-PDU::= SEQUENCE {
        IP_header           IP_HEADER,
        UDP_header          UDP_HEADER,
        Reserved1           Unsigned16,
        Cyclic_header       CYCLIC_HEADER,
        Block_num           SEQUENCE {
                Block number1               BlockNum,
                Block number2               BlockNum,
                        .
                        .
                        .
                Block number21              BlockNum,
                BLock_num pattern           Pattern16
```

```
        },
        Cyclic_data             SEQUENCE {
                Block data1             BlockData,
                Block data2             BlockData,
                        .
                        .
                        .
                Block data21            BlockData
        },
        Cyclic_FCS              CFCS
}

Cyclic_header::= SEQUENCE {
        Cyclic_pattern          CYCLIC_PATTERN,
        Total_Block             TOTAL_BLOCK,
        Src STaddress           ST_Address,
        Reserved1               Unsigned8
}
```

4.1.1.2.2 CTL-PDU

4.1.1.2.2.1 Control-PDU,

```
Control-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        CTL_data                CTLdata
}
```

4.1.1.2.2.2 RMTCTL-PDU

```
CTL_cyclic-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        RMTCTL_CMD              SEQUENCE {
                Reserved1               Unsigned16,
                Header Type             HEADER_TYPE,
                Module ID               MODULE_ID,
                Station number          ST_NUM,
                Command code            RMTCTL_CMD,
                Length                  LENGTH,
                RMTCTL Status           STATUS,
                RMTCTL Kind             RMTKIND,
                RMTCTL Length           RMT_LENGTH,
                RMTCTL Address          RMT_ADD,
                Send Number             SND_NUM,
                Reserved1               Unsigned16,
                Reserved2               Unsigned32
                },
        RMTCTL data             RMTCTL_DATA
}
```

4.1.1.2.3 INFO-PDU

```
INFO-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        INFO_data               INFOData
}
```

4.2 FAL PDU type N abstract syntax

4.2.1 Basic abstract syntax

The definitions of FALPDU are shown below.

```
FAL-PDU::= CHOICE   {
        CyclicData-PDU[0] CyclicData-PDU ,
        MulticastData-PDU       [1] MulticastData-PDU ,
        PtoP Data-PDU           [2] PtoP Data-PDU ,
        Aliveinfo-PDU           [3] Aliveinfo-PDU ,
        Aliveinfo6-PDU          [4] Aliveinfo6-PDU ,
        Inq-PDU                 [5] Inq-PDU ,
        Ninq-PDU                [6] Ninq-PDU ,
        Reply-PDU               [7] Reply-PDU ,
        RetransEnq-PDU          [8] RetransEnq-PDU,
        RetransConfirm-PDU      [9] RetransConfirm-PDU,
        RetransNak-PDU          [10] RetransNak-PDU
        }
```

FALAR-N Header to be used in each PDU is shown as follows.

```
FALAR-N Header::= SEQUENCE   {
        hd_h_type               Hd_h_type ,
        hd_ml                   Hd_ml ,
        hd_sa                   Hd_sa ,
        hd_da                   Hd_da ,
        hd_v_seq                Hd_v_seq ,
        hd_seq                  Hd_seq ,
        hd_m_ctl                Hd_m_ctl ,
        hd_inqid                Hd_inqid ,
        hd_tcd                  Hd_tcd ,
        hd_ver                  Hd_ver ,
        reserved1               OctString SIZE(3) ,
        hd_pkind                Hd_pkind ,
        hd_pseq                 Hd_pseq ,
        hd_mode                 Hd_mode ,
        hd_pver                 Hd_pver ,
        hd_pri                  Hd_pri ,
        hd_cbn                  Hd_cbn ,
        hd_tbn                  Hd_tbn ,
        hd_bsize                Hd_bsize ,
        reserved2               OctString SIZE(4)
        }

Hd_inqid::= SEQUENCE {
        inqid_inq_sa            Inqid_inq_sa ,
        inqid_tr_adr            Inqid_tr_adr ,
        inqid_id_seq            Inqid_id_seq
        }
```

4.2.2 CyclicData-PDU

```
CyclicData-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        tmid                    Tmid ,
        blockNumber             BlockNumber ,
        blockCount              BlockCount ,
        cyclicData              CyclicData
        }
```

4.2.3 MulticastData-PDU

```
MulticastData-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        multicastData           MulticastData
        }
```

4.2.4 PtoPData-PDU

```
PtoP Data-PDU ::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        pointToPointData        PointToPointData
        }
```

4.2.5 Aliveinfo-PDU

```
Aliveinfo-PDU ::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        alive4Header            ALIVE4 Header ,
        al_ExtentionList        SEQUENCE of Al_ExtentionList ,
        al_ExtentionInfo        Al_ExtenyionInfo
        }

ALIVE4 Header::= SEQUENCE {
        al_nd_name              Al_nd_name ,
        al_os_name              Al_os_name ,
        al_tm_out               Al_tm_out ,
        al_msgserno             Al_msgserno ,
        al_mode                 Al_mode ,
        al_protocol             Al_protocol ,
        al_tg_cmn_cnt           Al_tg_cmn_cnt ,
        al_tg_cflag             Al_tg_cflag ,
        reserved1               OctString SIZE(1) ,
        al_tg_max               Al_tg_max ,
        al_tg_usecnt            Al_tg_usecnt ,
        al_chg_time             Al_chg_time ,
        al_ipv4addr1            Al_ipv4addr1 ,
        al_ipv4addr2            Al_ipv4addr2 ,
        al_ver                  AL_ver ,
        reserved2               OctString SIZE(15)
        }

Al_ExtentionList ::= SEQUENCE {
        al_ta_chgalvstat        Al_ta_chgalvstat ,
        al_ta_chginfostat       Al_ta_chginfostat ,
        al_ta_tid               Al_ta_tid ,
        al_ta_data              Al_ta_data
        }
```

4.2.6 Aliveinfo6-PDU

```
Aliveinfo6-PDU ::= SEQUENCE {
        falArHeader             FALAR-N Header,
        alive6Header            ALIVE6 Header ,
        al_ExtentionList        SEQUENCE of Al_ExtentionList ,
        al_ExtentionInfo        Al_ExtentionInfo
        }

ALIVE6 Header::=  SEQUENCE {
        al_nd_name              Al_nd_name ,
        al_os_name              Al_os_name ,
        al_tm_out               Al_tm_out ,
        al_msgserno             Al_msgserno ,
```

```
        al_mode                 Al_mode ,
        al_protocol             Al_protocol ,
        al_tg_cmn_cnt           Al_tg_cmn_cnt ,
        al_tg_cflag             Al_tg_cflag ,
        reserved1               OctString SIZE(1) ,
        al_tg_max               Al_tg_max ,
        al_tg_usecnt            Al_tg_usecnt ,
        al_chg_time             Al_chg_time ,
        reserved2               Al_ipv4addr1 ,
        reserved3               Al_ipv4addr2 ,
        al_ver                  AL_ver ,
        reserved4               OctString SIZE(15) ,
        al_ipv6addr1            Al_ipv6addr1 ,
        al_ipv6addr2            Al_ipv6addr2
        }
```

4.2.7 Inq-PDU

```
Inq-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        inquirydata             InquiryData
        }
```

4.2.8 Ninq-PDU

```
Ninq-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        node-list               Node-List ,
        ninquirydata            NinquiryData
        }
```

4.2.9 Reply-PDU

```
Reply-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header
        }
```

4.2.10 RetransEnq-PDU

```
RetransEnq-PDU::= SEQUENCE {
        falArHeader                 FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransRequestNode          RetransRequestNode ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        reserved1                   OctString SIZE(4) ,
        retransInfo-List            SEQUENCE of RetransInfoList
        }

RetransInfoList ::= SEQUENCE {
        retransMcg                  RetransMcg ,
        retransPseqNo               RetransPseqNo
        }
```

4.2.11 RetransConfirm-PDU

```
RetransConfirm::= SEQUENCE {
        falArHeader                 FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        retransInfo-List            SEQUENCE  of RetransInfoList
        }
```

4.2.12 RetransNak-PDU

```
RetransNAK-PDU::= SEQUENCE {
        f alArHeader                FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        retransInfo-List            SEQUENCE  of RetransInfoList
        }
```

4.3 Data type assignments for type S

Data types used in FALPDU type S abstract syntax are shown as follows.

```
BLK_Status::= Unsigned32
BlockData::= OCTET STRING(SIZE=64)
BlockNum::= Unsigned16
CLASS::= Unsigned16
CLTData::= OCTET STRING(SIZE = 18..1472)
CYCLIC_FCS::= Unsighed32
EntryNum::= Unsigned32
ERR_ADD::= Unsigned32
HEADER_TYPE::= Unsigned16
INFOData::= OCTET STRING(SIZE = 18..1472)
IP_Header::= OCTET STRING(SIZE=20)
MODULE_ID::= Unsigned16
Node_Status::= Unsigned32
Pattern16::= Unsigned16
Pattern32::= Unsigned32
Pattern8::= Unsigned8
PLD::= Unsigned32
PortNum::= Unsigned32
Priority::= Unsigned8
RCL_Status::= Unsigned32
RCLCMD::= Unsigned32
RCLKIND::= Unsigned32
RMT_ADD::= Unsigned32
RMT_LENGTH::= Unsigned16
RMTCTL_CMD::= Unsigned16
RMTCTL_DATA::= OCTET STRING(SIZE = 0..1440)
RMTKIND::= Unsigned16
SendTime::= OCTET STRING(SIZE=8)
SequenceNum32::= Unsigned32
SND_NUM::= Unsigned16
ST_Address::= Unsigned8
ST_NUM::= Unsigned16
STATUS::= Unsigned32
TOTAL_BLOCK::= Unsigned8
UDP_Header::= OCTET STRING(SIZE=8)
```

4.4 Data type assignments for type N

Data types used in FALPDU type N abstract syntax are shown as follows.

```
AI_ExtentionInfo::= OctString SIZE(0..664)
AI_chg_time::= Unsigned32
AI_ipv4addr1::= OctString SIZE(4)
AI_ipv4addr2::= OctString SIZE(4)
AI_ipv6addr1::= OctString SIZE(16)
AI_ipv6addr2::= OctString SIZE(16)
AI_mode::= Unsigned8
AI_msgsernot::= Unsigned16
AI_nd_name::= OctString SIZE(10)
```

Al_os_name::= OctString SIZE(10)
Al_protocol::= Unsigned8
Al_ta_chgalvstat::= Unsigned8
Al_ta_chginfostat::= Unsigned8
Al_ta_data::= interger16
Al_ta_tid::= Unsigned16
Al_tg_cflag::= Unsigned8
Al_tg_cmn_cnt::= Unsigned16
Al_tg_max::= Unsigned16
Al_tg_usecnt::= Unsigned16
Al_tm_out::= Unsigned32
AL_ver::= Unsigned8
BlockCount::= Unsigned16
BlockNumber::= Unsigned16
CyclicData::= Octstring SIZE(64..16640)
Data type assignment
Hd_bsize::= unsigned16
Hd_cbn::= unsigned8
Hd_da::= unsigned32
Hd_h_type::= unsigned32
Hd_m_ctl::= unsigned32
Hd_ml::= unsigned32
Hd_mode::= integer16
Hd_pkind::= unsigned8
Hd_pri::= unsigned8
Hd_pseq::= unsigned32
Hd_pver::= unsigned8
Hd_sa::= unsigned32
Hd_seq::= unsigned32
Hd_tbn::= unsigned8
Hd_tcd::= unsigned16
Hd_v_seq::= unsigned32
Hd_ver::= unsigned16
Inqid_id_seq::= unsigned32
Inqid_inq_sa::= unsigned32
Inqid_tr_adr::= unsigned32
InquiryData::= OctString SIZE(0..262144)
MulticastData::= OctString SIZE(0..262144)
Node-List::= OctString SIZE(512)
PointToPointData::= OctString SIZE(0..262144)
RetransMcg::= unsigned32
RetransNumberOfRequests::= unsigned32
RetransPseqNo::= unsigned32
RetransRequest::= unsigned32
RetransRequestNode::= unsigned32
Tmid::= Unsigned32

5 FAL transfer syntax

5.1 Encoding rules

5.1.1 Unsigned encoding

The fixed-length, unsigned values Unsigned8, Unsigned16, Unsigned24, and Unsigned32 are encoded as unsigned integers of one octet, two octets, three octets, and four octets in length, respectively. Unsigned16, Unsigned24, and Unsigned32 are encoded as big endians, where the most significant octet is regarded as the first octet, the next octet is regarded as the second octet, and the least significant octet is regarded as the last octet.

5.1.2 Octet string encoding

OctetString which has variable number of octets is encoded octet by octet, in sequential order.

5.1.3 SEQUENCE encoding

SEQUENCE encoding is performed in sequence, starting from the initial element. The identifiers and length used in ASN.1 are not used.

5.2 FALPDU type S elements encoding

5.2.1 RCL_header

5.2.1.1 Frame Class

This field shows the PDU classes described in Table 4.

Table 4 – Frame Class

Value	Description
0x0001	Class 1 frame (RHE-PDU)
0x0002	Class 2 frame (LCC-PDU, LCA-PDU, LCN-PDU, LNA-PDU, SCR-PDU)

5.2.1.2 Dst Address

5.2.1.2.1 Reserved1

This field is reserved for future use. This value is 0x00.

5.2.1.2.2 DA_STaddress

This field indicates the destination station address described in Table 5.

Table 5 – DA_STaddress – DA_STaddress

Value	Description
0xFF	RHE-PDU, LCC-PDU, LCN-PDU, SCR-PDU
SA_STaddress in the received LCC-PDU	LCA-PDU
SA_STaddress in the received LCN-PDU	LNA-PDU

5.2.1.2.3 DA_MACaddress

This field indicates the destination MAC address described in Table 6.

Table 6 – DA_MACaddress

Value	Description
FF-FF-FF-FF-FF-FF	RHE-PDU, LCC-PDU, LCN-PDU, SCR-PDU
SA_MACaddress in the received LCC-PDU	LCA-PDU
SA_MACaddress in the received LCN-PDU	LNA-PDU

5.2.1.3 Src Address

5.2.1.3.1 Reserved2

This field is reserved for future use. This value is 0x00.

5.2.1.3.2 SA_STaddress

This field indicates the station address of the own node.

5.2.1.3.3 SA_MACaddress

This field indicates the MAC address of the own node.

5.2.1.4 RCL Command

Table 7 shows RCL Command (CMD) field format. This field is used to identify the RCL frame. Each bit in this field has the following meanings:

Table 7 – CMD field format

Bit	Value						Description
	RHE	LCC	LCA	LCN	LNA	SCR	
[3:0]	1						reserved field1
[7:4]	0	0	1	2	3	0	unique number 1
[11:8]	0	1	1	1	1	3	unique number 2
[15:12]	0						reserved field2
[23:16]	1	2	2	2	2	2	indicates the frame class
[31:24]	0						reserved field3

5.2.1.5 Frame Sequence

The frame sequence number is used to detect loss and duplication of received frames. The range of this value is from 0x00000000 to 0xFFFFFFFF with step size 1.

5.2.1.6 Reserved3

This field is reserved for future use. The value is 0.

5.2.2 RHE-PDU

5.2.2.1 RHE_Common

5.2.2.1.1 Send Direction

This field contains the transmission port of the RHE-PDU as described in Table 8.

Table 8 – Send Direction

Value	Description
0x01	Port A
0x02	Port B

5.2.2.1.2 RHE Sequence

This field contains the identification number of the RHE-PDU. The range of this value is from 0x0000 to 0xFFFF. The same value is used in port A and port B.

5.2.2.2 Neighborhood address

5.2.2.2.1 Reserved1

This field is reserved for future use. The value is 0x00.

5.2.2.2.2 NB_STaddress

This field indicates the station address of the neighborhood node.

5.2.2.2.3 NB_MACaddress

This field indicates the MAC address of the neighborhood node.

5.2.2.3 RHE RecieveStatus

This field indicates the status about receiving RHE-PDU as described in Table 9.

Table 9 – RHE ReceiveStatus

Value	Description
1	Receiving RHE-PDU
0	Not receiving RHE-PDU

5.2.2.4 True Status

5.2.2.4.1 Reserved2

This field is reserved for future use. The value is 0x00000000.

5.2.2.4.2 Physical Linkdown

This field indicates the link status at the port as described in Table 10.

Table 10 – Physical Linkdown

Value	Description
1	Physical Link up
0	Physical Link down

5.2.2.4.3 Reserved3

This field is reserved for future use. The value is 0x00000000.

5.2.2.5 Blocking Status

This field indicates the status code about the port.

5.2.2.6 Node Status

This field indicates the status code about the node.

5.2.2.7 Reserved4

This field is reserved for future use. This value is 0.

5.2.2.8 Reserved5

This field is reserved for future use. This value is 0.

5.2.2.9 RHE_pattern1~4

This field is used to check the pattern of RHE-frames as described in Table 11. The patterns are repeated until the maximum length of ISO/IEC 8802-3 standard's frame.

Table 11 – RHE_pattern 1~4

Pattern	Octets	Value
RHE_pattern1	4	0xFFFFFFFF
RHE_pattern2	4	0xAAAAAAAA
RHE_pattern3	4	0x55555555
RHE_pattern4	4	0x00000000

5.2.3 LCC-PDU

5.2.3.1 LCC Kind

This field indicates the reason of sending LCC-PDU as described in Table 12.

Table 12 – LCC-Kind

Value	Description
0x00000101	Logical link up port A of the node
0x00000201	Competitive of receiving other LCC-PDU

5.2.3.2 LCC Priority

5.2.3.2.1 LCC_Priority

This field indicates the priority of LCC-PDU. This value is 0x0C.

5.2.3.2.2 LCC_STadd

This field indicates the station address of the own node.

5.2.3.2.3 LCC_MACadd

This field indicates the MAC address of the own node.

5.2.3.3 Reserved1

This field is reserved for future use. This value is 0.

5.2.4 LCA-PDU

5.2.4.1 Reserved1

This field is reserved for future use. This value is 0x01.

5.2.4.2 RCL Status

This field indicates the status of the node sending LCA-PDU as described in Table 13.

Table 13 – RCL Status

Field name	Bit	Value	Description
-	31 .. 24	0x00	Reserved
Node status	23	0 or 1	1: The node is in “Edge-A” state. When the bit 23 and 22 are “1”, the node is in “Isolated” state. 0: The node is in other node status (Edge-B or Intermediate).
	22	0 or 1	1: The node is in “Edge-B” state. When the bit 23 and 22 are “1”, the node is in “Isolated” state. 0: The node is in other node status (Edge-A or Intermediate).
-	21	0	Reserved
Blocking status	20	0 or 1	1: Port A or B is logical link down port.
-	19 .. 8	0x000	Reserved
Port status	7	0 or 1	1: Port A is in “Port_Linkup” state.
	6	0 or 1	1: Port B is in “Port_Linkup” state.
	5	0 or 1	1: Port A is physical link up.
	4	0 or 1	1: Port B is physical link up.
-	3 .. 0	0x0	Reserved

5.2.4.3 LCA Priority

5.2.4.3.1 LCA_Priority

This field indicates the priority in the received LCC-PDU.

5.2.4.3.2 LCA_STadd

This field indicates the station address in the received LCC-PDU.

5.2.4.3.3 LCA_MACadd

This field indicates the MAC address in the received LCC-PDU.

5.2.4.4 Reserved2

This field is reserved for future use. This value is 0.

5.2.5 LCN-PDU

5.2.5.1 Reserved1

This field is reserved for future use. This value is 0x01.

5.2.5.2 RCL Status

This field indicates the status of the node sending LCN-PDU. It has the same format as “5.2.4.2 RCL Status” described in Table 13.

5.2.5.3 Reserved2

This field is reserved for future use. This value is 0.

5.2.6 LNA-PDU

5.2.6.1 Reserved1

This field is padding area. This value is 0x00.

5.2.6.2 RCL Status

This field indicates the status of the node sending LNA-PDU. It has the same format as "5.2.4.2 RCL Status" described in Table 13.

5.2.6.3 Reserved2

This field is reserved for future use. This value is 0.

5.2.7 SCR-PDU

5.2.7.1 Entry Number

This field indicates the number to contain the SCR_Event entry.

5.2.7.2 SendStat Time

This field contains the sending time of seconds.

5.2.7.3 Reserved1

This field is reserved for future use. This value is 0.

5.2.7.4 Event0~16

This field contains the event information and specifies each manufacture.

5.2.8 Cyclic_S-PDU

5.2.8.1 IP_header

The encoding of the fields shall be according to RFC 791.

5.2.8.2 UDP_header

The encoding of the fields shall be according to RFC 768.

5.2.8.3 Reserved1

This field is reserved for future use. This value is 0x0000.

5.2.8.4 Block_number1~21

This field indicates the identification number of the cyclic data block and specifies each manufacture.

5.2.8.5 Block_num pattern

This field is used to check the Cyclic-PDU and specifies each manufacture.

5.2.8.6 Block data1~21

This field contains the cyclic data.

5.2.8.7 Cyclic_FCS

This field is an error checking code from Cyclic-PDU.

The generating polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.9 Cyclic_header

5.2.9.1 Cyclic_pattern

This field is used to check the Cyclic_header and specifies each manufacture.

5.2.9.2 Total_Block

This field indicates the number of blocks in the Cyclic-PDU.

5.2.9.3 Src STaddress

This field indicates the MAC address of the own node.

5.2.9.4 Reserved1

This field is reserved for future use. This value is 0x00.

5.2.10 Control-PDU

5.2.10.1 IP_header

Refer to 5.2.8.1.

5.2.10.2 UDP_header

Refer to 5.2.8.2.

5.2.10.3 CTL_data

This field contains the data of the CTL-PDU.

5.2.11 RMTCTL-PDU

5.2.11.1 IP_header

Refer to 5.2.8.1.

5.2.11.2 UDP_header

Refer to 5.2.8.2.

5.2.11.3 RMTCTL_CMD

5.2.11.3.1 Reserved1

This field is reserved for future use. This value is 0x0000.

5.2.11.3.2 Header Type

This field is used to recognize the remote control frame. This value is “RC” in ASCII code (“RC” = 0x5243).

5.2.11.3.3 Module ID

This field indicates the module type.

5.2.11.3.4 Station number

This field indicates the station address of the destination node.

5.2.11.3.5 Command code

This field specifies the command type.

5.2.11.3.6 Length

This field indicates the length of RMTCTL-PDU.

5.2.11.3.7 RMTCTL Status

This field indicates the status code about the remote control operations.

5.2.11.3.8 RMTCTL Kind

This field indicates the module type for remote memory reading.

5.2.11.3.9 RMTCTL Length

This field indicates the data size of remote memory writing or reading.

5.2.11.3.10 RMTCTL Address

This filed indicate the address of remote memory writing or reading.

5.2.11.3.11 Send Number

This filed is use to manage of the remote control by the FAL-user.

5.2.11.3.12 Reserved2

This field is reserved for future use. This value is 0x0000.

5.2.11.3.13 Reserved3

This field is reserved for future use. This value is 0x00000000.

5.2.11.3.14 RMTCTL data

This field contains the data of the remote control command.

5.2.12 INFO-PDU

5.2.12.1 IP_header

Refer to 5.2.8.1.

5.2.12.2 UDP_header

Refer to 5.2.8.2.

5.2.12.3 INFO_data

This field contains the data of the INFO-PDU.

5.3 FALPDU type N elements encoding

5.3.1 General

FALPDU type N headers excluding user data are specified using the TCP/IP network octet ordering convention, which is known as big endian.

5.3.2 FALAR-N Header

5.3.2.1 hd_h_type

This field shall indicate the header type specification described in

Table 14. Specify "NUXM" or “NUV6” in ASCII. PDUs transmitted in IPv4 Data field specify “NUXM”. PDUs transmitted in IPv6 Data field specify “NUV6”.

Table 14 – hd_h_type

Value	PDU type
“NUXM”	PDUs transmitted in IPv4 Data field
“NUV6”	PDUs transmitted in IPv6 Data field

5.3.2.2 hd_ml

This field shall indicate the length of the total message including FALAR-N header length (= 64 octets) and message length. (hd_ml = 64 + message length).

5.3.2.3 hd_sa

This field shall contain the address of the source node (Logical Node Number) described in Figure 2.

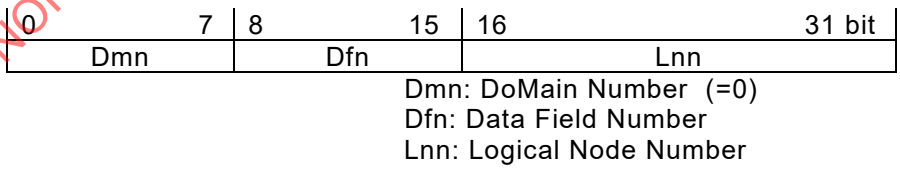


Figure 2 – hd_sa

Dmn:

A domain is defined as the concept above the data field. A domain consists of multiple data fields. Usually, one data field is defined as a local location and connected to other domains via wide area communication networks. Within a system, a number that uniquely identifies a domain is called a Domain Number (Dmn). Unique Dmn in the range of 1 to 64 is assigned to each of domains in the system. Either a domain number 0 (Dmn 0) or non-specified domain number indicate a local domain (default).

Dfn:

Data Field (DF) is a place where ADP messages consisting of information of specific characteristics can flow. Communication using ADP is available between nodes belonging to one data field. This means that any equipment in a system shall belong to one or more data fields. One data field is configured per network address or sub-network address of an IP address. Within a decentralized system, a number that uniquely identifies a data field is called a Data Field Number (Dfn). Unique Dfn is assigned to each of data fields within the system, in the range of 1 to 255. Dfn 0 is reserved for communication within the current node. An IPv4 network shall be assigned with IPv4 Dfn. An IPv6 network shall be assigned with IPv6 Dfn.

NOTE 1 No data field can be set up across multiple LAN segments. Every LAN segment is assigned with Dfn.

NOTE 2 One LAN segment can contain more than one data field.

Lnn:

A logical node is equipment in a data field. Within a data field, a number that uniquely identifies a logical node is called a Logical Node Number (Lnn). Unique Lnn in the range of 1 to 4095 is assigned to each of devices. Lnn 0 is reserved for communication within the current node.

5.3.2.4 hd_da

This field shall contain the destination multicast group address (Multicast Group Number) or the destination node address (Logical Node Number) described in Figure 3.

(Multicast communication)

0 7	8 15	16 31 bit
Dmn	Dfn	Mgn

(Peer to Peer communication)

0 7	8 15	16 31 bit
Dmn	Dfn	Lnn

Dmn: DoMain Number (=0)
Dfn: Data Field Number
Mgn: Multicast Group Number
Lnn: Logical Node Number

Figure 3 – hd_da

Dmn,

Refer to 5.3.2.3.

Dfn,

Refer to 5.3.2.3.

Mgn,

A multicast group (MCG) is a group of nodes belonging to a data field to determine whether to receive data multicast/ broadcast from a node. A multicast group shall be specified to perform multicast transmission. Any node shall belong to a multicast group to be used for multicast reception. One data field can contain more than one multicast group. Each node can be added to more than one multicast group as far as the groups are within a data field to which the node belongs. Multicast Group Number (Mgn) is a number that uniquely identifies a

multicast group within a data field. Unique Mgn in the range of 1 to 255 is assigned to each of multicast groups in the data field. Mgn 0 is reserved for the system to send/receive alive-messages.

A Mgn of MulticastData-PDUs shall be different from a Mgn of RetransX PDUs.

Here, RetransX PDU indicates any one of RetransEnq-PDU, RetransCinfirm-PDU or RetransNak-PDU.

A Mgn of Multicast communication with retransmission is also different from a Mgn of both Multicast communication (with no retransmission) and RetransX-PDU.

Lnn,

Refer to 5.3.2.3.

Table 15 shows either Mgn or Lnn used by each PDU.

Table 15 – Usage of Mgn or Lnn

Field(Mgn , Lnn)		Description(PDU)
Mgn	**Lnn**	
Applicable	N/A	CyclicData-PDU
Applicable	N/A	MulticastData-PDU
N/A	Applicable	PtoPData-PDU
Applicable (=0)	N/A	Aliveinfo-PDU
Applicable (=0)	N/A	Aliveinfo6-PDU
Applicable	Applicable	Inq-PDU
Applicable	N/A	Ninq-PDU
N/A	Applicable	Reply-PDU
Applicable	N/A	RetransEnq-PDU
Applicable	N/A	RetransConfirm-PDU
Applicable	N/A	RetransNak-PDU

5.3.2.5 hd_v_seq

This field shall indicate the version identification of the sequence number of the transmission message (hd_seq). This field normally specifies the time when the sequence number of the transmission message (hd_seq) is initialized. The objective of this field enables a receiving node to detect that a sending node has newly initialized the sequence number. New setting value of hd_v_seq shall be different from old value of it.

5.3.2.6 hd_seq

5.3.2.6.1 hd_seq

This field shall indicate the sequence number of the transmission message. The range of values is from 0x00000001 to 0x7FFFFFFF, and used cyclically.

5.3.2.6.2 Transmission sequence number and version number management

5.3.2.6.2.1 Management at a transmitting node

Assignment:

The version numbers (V_SEQ) shall be assigned to each node. The version numbers are assigned with time stamps. Non-zero value of the time stamp shall be different from the preceding V_SEQ value.

a) Multicast communication

Each node has transmission sequence numbers (S_SEQ). The transmission sequence numbers (S_SEQ) shall be managed by each Dfn (Data field number), by each Mgn (Multicast group number) and by each message priority level.

b) Peer to Peer communication

Each node has transmission sequence numbers (S_SEQ). The transmission sequence numbers (S_SEQ) shall be managed by each Dfn, by the connection for each node and by each message priority level.

Initialization:

a) Multicast communication

On opening a multicast communication port, V_SEQ corresponding to the port shall be set to the opening time and S_SEQ corresponding to the port shall be set to 0.

b) Peer to Peer communication

On establishing the TCP connection, V_SEQ corresponding to the connection shall be set to the establishing time and S_SEQ corresponding to the connection shall be set to 0.

Transmission:

On transmitting of a message, V_SEQ and updated S_SEQ are set to hd_v_seq and hd_seq in the FALAR_N Header, respectively. S_SEQ will be updated as follows:

– If S_SEQ is not 0x7FFFFFFF, S_SEQ is incremented by one.

– If S_SEQ is 0x7FFFFFFF, S_SEQ is reset to 1 without increment.

5.3.2.6.2.2 Management at a receiving node

Assignment:

a) Multicast communication

Each node has the reception sequence numbers (R_SEQ) lastly received and a reception version number (R_V_SEQ) lastly received. The R_SEQ shall be managed by each Dfn (Data Field Number), by each Mgn (Multicast Group Number), by each source node, and by each message priority level.

b) Peer to Peer communication

Each node has the reception sequence numbers (R_SEQ) lastly received and reception version number (R_V_SEQ) lastly received. The R_SEQ shall be managed by each Dfn (Data field number), by the connection for each node and by each message priority level.

Initialization:

a) Multicast communication

On opening a multicast communication port, R_V_SEQ and R_SEQ shall be initialized to 0.

b) Peer to Peer communication

On establishing the TCP connection, R_V_SEQ corresponding to the connection and R_SEQ corresponding to the connection shall be initialized to 0.

Reception:

On reception of a message, R_V_SEQ and R_SEQ are retrieved by hd_sa, by hd_seq, by hd_pri and by hd_v_seq in the FALAR_N Header.

NOTE Any message with hd_v_seq = 0, hd_seq = 1, hd_cbn = 1 and hd_tbn = 1 is passed to an application without checking the sequence number.

5.3.2.6.2.3 Sequence number check for reception message

1) If R_V_SEQ is equal to 0,

   the message is unconditionally received and hd_v_seq and hd_seq in the FALAR_N Header update R_V_SEQ and R_SEQ, respectively.

2) If R_V_SEQ is not 0 and hd_v_seq is equal to R_V_SEQ,

   R_SEQ and hd_seq are compared for detecting a duplicated message or lost messages. Table 16 and Figure 4 show conditions of sequence number check for reception message.

3) Communication

   a) Multicast communication

      If R_V_SEQ is not 0 and hd_v_seq is not equal to R_V_SEQ, hd_v_seq and hd_seq values update the R_V_SEQ and R_SEQ values, respectively. On detecting a duplicated message, a receiving node discards the received message.

   b) Peer to Peer communication

      If R_V_SEQ is not 0 and hd_v_seq is not equal to R_V_SEQ, a receiving node disconnects the TCP connection and discards the received message.

Table 16 – Detailed conditions for sequence number check of reception message

No.	Conditions	Result
1	R_SEQ != 0x7fffffff && R_SEQ + 1 == SEQ	Normal message reception
2	R_SEQ == 0x7fffffff && SEQ == 1	Normal message reception
3	R_SEQ > N1 && R_SEQ – N1 < SEQ <= R_SEQ	Duplicated message reception
4	R_SEQ <= N1 && (0 < SEQ <= R_SEQ || 0x7fffffff – (N1 – R_SEQ) < SEQ <= 0x7fffffff)	Duplicated message reception
5	Other conditions	Missing message
Key: SEQ: Value of hd_seq in the FALAR_N Header N1: Transmission sequence number assumed to be duplicated. This value is implemented matter.		

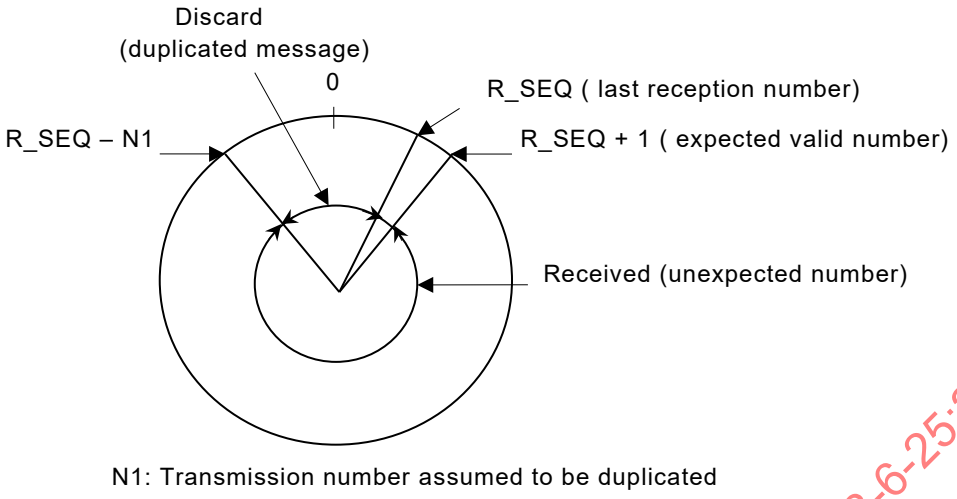


Figure 4 – Valid sequence number for reception message

5.3.2.7 hd_m_ctl

This field shall specify information on message transmission control described in Figure 5.

0	1	2	3	4	5	6 31 bit
MLT	ONE	INQ	RPL	NIQ	MCR	Rsv

Key:

MLT: Multicast communication (= 0x80000000)

ONE: Peer to Peer (One to One) communication (= 0x40000000)

INQ: Inquiry communication (= 0x20000000)

RPL: Reply response communication (= 0x10000000)

NIQ: N-inquiry communication (= 0x08000000)

MCR: Multicast communication with re-transmission control (= 0x04000000)

Rsv: Reserved

Figure 5 – hd_m_ctl

Table 17 shows the valid bits of hd_m_ctl field in each PDU.

Table 17 – Valid bits of hd_m_ctl

PDU	Valid bits of hd_m_ctl
CyclicData-PDU	0x80000000
MulticastData-PDU	0x80000000 (Multicast communication type)
	0x84000000 (Multicast communication with re-transmission type)
PtoPData-PDU	0x40000000
Aliveinfo-PDU	0x80000000
Aliveinfo6-PDU	0x80000000
Inq-PDU	0xA0000000 (Multicast communication type)
	0x60000000 (Peer to Peer communication type)

PDU	Valid bits of hd_m_ctl
Ninq-PDU	0x88000000
Reply-PDU	0x50000000
RetransEnq-PDU	0x84000000
RetransConfirm-PDU	0x84000000
RetransNak-PDU	0x84000000

5.3.2.8 hd_inqid

This field shall specify inquiry/response identifier, which consists of inqid_inq_sa, inqid_tr_adr and inqid_id_seq. These three fields are described in 5.3.2.20, 5.3.2.21 and 5.3.2.22 respectively. This hd_inqid field is valid in Inq-PDU, Ninq-PDU and Reply-PDU.

5.3.2.9 hd_tcd

This field shall specify transaction code. All transactions are assigned transaction codes (TCDs) to identify them. A TCD represents transaction data. Each application program transmits data with a specified TCD into a data field. On the other hand, each application program receives data with a specifying TCD from a data field. Unique TCDs shall be defined for each data field. TCDs are classified into those for user transactions and those for system transactions, as shown below.

- TCDs for user transactions: 1 to 59 999 (user-definable)
- TCDs for system transactions: 60 000 to 65 534 (reserved for the system)

A system transaction is created when the system detects an event (such as an error or internal status change) and notifies the user program. Table 18 shows specified TCD.

Table 18 – Specified TCD

Specified TCD	Description
60 003	Alive report
60 061	Retransmission control
60 027	File transfer
60 056	Cyclic memory transfer
60 058	Cyclic memory transfer (Fast type)

5.3.2.10 hd_ver

This field shall specify a program version identifier. A current value is 0.

5.3.2.11 hd_pkind

This field shall specify a transaction identifier as shown in Table 19. This field is valid for a one to one communication (PtoPData-PDU) using a duplex LAN.

Table 19 – hd_pkind

Value	Description
1	ACK Request (ACK.req); Request the destination node (Dfn, Lnn) to inform a reception packet sequence number (R-PSEQ)
2	ACK Response (ACK.inform); Inform the reception packet sequence number (R-PSEQ) to ACK Request node.
0	Others (NoACK)

5.3.2.12 hd_pseq

5.3.2.12.1 hd_pseq

This field shall specify a packet sequence number. The range of values is from 0x1 to 0x FFFFFFFF, and used cyclically. This field is effective for ”Multicast communication with retransmission” and for “One to one communication using a duplex LAN” as shown in Table 20.

Table 20 – PDU with an effective hd_pseq

PDU	Effectiveness	Description	Value of hd_pseq
MulticastData-PDU	Effective	Multicast communication with retransmission	0x1 – 0x FFFFFFFF
PtoPData-PDU	Effective	One to one communication (PtoPData-PDU) using a duplex LAN	0x1 – 0x FFFFFFFF
Other PDU	Ineffective	Others (including one to one communication (PtoPData-PDU) using a single LAN)	0 (recommend)

5.3.2.12.2 Management at a transmitting node

Assignment:

Each node has management information (S_PSEQ) of a transmitting packet sequence number.

a) Multicast communication with retransmission

   A S_PSEQ shall be assigned by Dfn (Data field number) and by Mgn (Multicast group number).

b) One to one communication using a duplex LAN

   A S_PSEQ shall be assigned by Dfn (Data field number) and by the TCP connection for each node.

Initialization:

a) Multicast communication with retransmission

   On opening a multicast communication port, the S_PSEQ corresponding to the port shall be set to 0.

b) One to one communication using a duplex LAN

   On establishing the TCP connection, the S_PSEQ corresponding to the connection shall be set to 0.

Transmission:

On transmission of a packet, the updated S_PSEQ is set to hd_pseq in the FALAR_N Header.

S_PSEQ will be updated as follows.

– If S_PSEQ is not 0xFFFFFFFF, S_PSEQ is incremented by one.
– If S_PSEQ is 0xFFFFFFFF, S_PSEQ is reset to 1 without increment.

NOTE In "Multicast communication with retransmission ", these three PDU (RetransEnq-PDU, RetransConfirm-PDU, RetransNak-PDU) don't use the hd_pseq field.

5.3.2.12.2.1 Management at a receiving node

Assignment:

Each node has management information (R_PSEQ) of a received packet sequence number.

a) Multicast communication with retransmission

A R_PSEQ shall be managed by Dfn (Data field number), by Mgn (Multicast group number) and by source node.

b) One to one communication using a duplex LAN

A R_PSEQ shall be assigned by Dfn (Data field number) and by the TCP connection for each node.

Initialization:

a) Multicast communication with retransmission

- On opening a multicast communication port, the S_PSEQ corresponding to the port shall be set to 0.
- On the failure of retransmission request, the R_PSEQ corresponding to the port shall be set to 0.

b) One to one communication using a duplex LAN

On establishing the TCP connection, the R_PSEQ corresponding to the connection shall be set to 0.

Reception:

On reception of a packet, the R_PSEQ is retrieved by hd_pseq in the FALAR_N Header.

5.3.2.12.2.2 Packet sequence number check

a) Multicast communication with retransmission

On reception of a packet, comparison R_PSEQ with hd_pseq in the FALAR_N Header is executed.

R_PSEQ will be updated as follows.

1) If R_PSEQ is equal to 0,

the packet is unconditionally received and hd_pseq in the FALAR_N Header set to R_PSEQ.

2) If R_PSEQ is not equal to 0,

R_PSEQ and hd_pseq are compared for detecting a duplicated packet or lost packets. Table 21 and Figure 6 show conditions for packet sequence number check.

Table 21 – Detailed conditions for sequence number check of reception packet (Multicast communication with retransmission)

No.	Conditions	Result
1	R_PSEQ != 0xffffffff && R_PSEQ + 1 == PSEQ	Normal packet reception (R_PSEQ = PSEQ)
2	R_PSEQ == 0xffffffff && PSEQ == 1	Normal packet reception (R_PSEQ = PSEQ)
3	R_PSEQ > N1 && R_PSEQ – N1 < PSEQ <= R_PSEQ	Duplicated packet reception
4	R_PSEQ <= N1 && (0 < PSEQ <= R_PSEQ || 0xffffffff – (N1 – R_PSEQ) < PSEQ <= 0xffffffff)	Duplicated packet reception
5	Other conditions	Lost packets
PSEQ: Value of hd_pseq in the FALAR_N Header. N1: Transmission packet sequence number assumed to be duplicated. This value is implemented matter.		

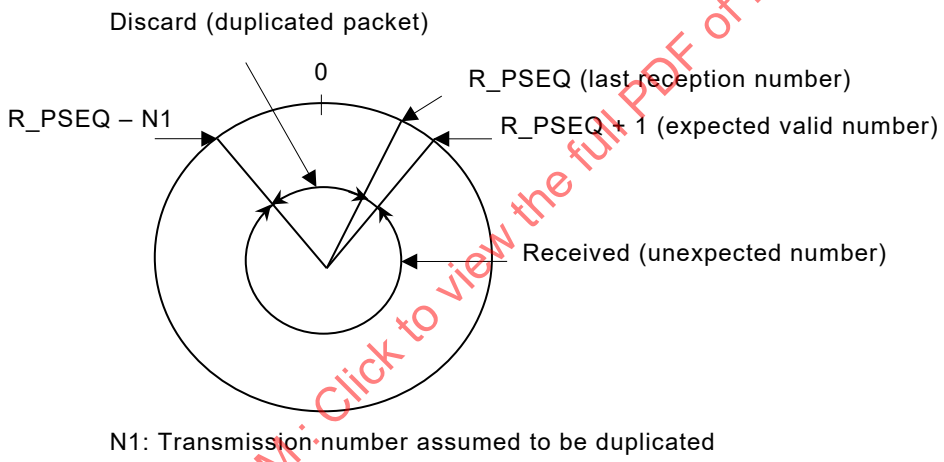


Figure 6 – Valid reception packet sequence number

b) One to one communication using a duplex LAN

1) If R_PSEQ is equal to 0 and hd_pseq in the FALAR_N Header is equal to 1, set the value of hd_pseq to R_PSEQ (first normal packet received).

2) If R_PSEQ is equal to 0 and hd_pseq in the FALAR_N Header is not equal to 1, the receiving node discards the received packet and disconnects the TCP connection.

3) If R_PSEQ is not equal to 0,

R_PSEQ and hd_pseq are compared for detecting a duplicated packet or lost packets. Table 22 shows conditions for packet sequence number check.

Table 22 – Detailed conditions for packet sequence number check

No.	Conditions	Result
1	R_PSEQ != 0xffffffff && R_PSEQ + 1== PSEQ	Normal packet reception (R_PSEQ = PSEQ)
2	R_PSEQ == 0xffffffff && PSEQ == 1	Normal packet reception (R_PSEQ = PSEQ)
3	Other conditions	Duplicated packet reception or lost packets
One to one communication using a duplex LAN		

On detecting a duplicated packet or lost packets, the receiving node discards the received packet.

5.3.2.13 hd_mode

5.3.2.13.1 Overview

This protocol allows adding a node mode to a message to indicate whether the node is online or under testing. Since a node can control transmitting/receiving messages selectively based on the discriminating mode (online / testing), the node can eliminate any interrupting test message when it is on the way of testing.

When a node is performing a simulation test, it can receive either online message or test message.

5.3.2.13.2 Message modes

A message mode is assigned by every message. There are two types of message modes:

1) Online mode,
2) Test mode.

5.3.2.13.3 Node modes

A node mode is assigned by every Dfn (Data field number). There are two types of node modes:

1) Online mode,
2) Test mode.

A node with online mode shall set online message mode to hd_mode field.

A node with test mode shall set test message mode to hd_mode field.

5.3.2.13.4 Transmitting/Receiving message control

Table 23 shows the relations between message modes and node modes. Message control using these relations is executed.

Table 23 – Relation between message transmission/reception

Message mode	Node mode			
	Online		Test	
	Transmission	Reception	Transmission	Reception
Online	Effective	Effective	Not effective	Effective
Test	Not effective	Discard	Effective	Effective
Effective: transmitting / receiving a message; Not effective: no transmitting a message				

5.3.2.13.5 hd_mode

This field shall specify a message mode as shown in Table 24.

Table 24 – hd_mode

Value	Description
0	Online mode
1	Test mode

5.3.2.14 hd_pver

This field shall specify a protocol version. A current value is 1.

5.3.2.15 hd_pri

This field shall specify a message priority level as shown in Table 25.

Table 25 – Message priority level

Priority level[a]	Description
0	Out of priority control (No priority control is implemented)
1	Message with the highest priority level 1
2	Message with the priority level 2
3	Message with the priority level 3
4	Message with the priority level 4
5	Message with the priority level 5
6	Message with the priority level 6
7	Message with the lowest priority level 7
[a] Higher numbers means lower priority.	

5.3.2.16 hd_cbn

5.3.2.16.1 Overview for fragmenting and assembling messages

The specifications limit the maximum user message size to 16 KB for the TCP/UDP based communication, while the maximum transfer size (MTU) for LANs is limited to 1 500 octets under the Ethernet standard.

For the TCP based communication, the IPv4 and TCP headers use 20 octets respectively and the FALAR-N header uses 64 octets, so available free data storage size can be calculated as (MTU – 104) octets.

On the other hand, the IPv6 header uses 40 octets and the FALAR-N header uses 64 octets, so available free data storage size can be calculated as (MTU – 124) octets.

For the UDP based communication, available free data storage size can be calculated as (MTU – 92) octets or as (MTU-112) octets, respectively.

NOTE The maximum free user's area in the TCP segment can be less than 1 460 octets or less than 1 440 octets through a negotiation on the TCP segment size when establishing a TCP connection.

If a message size is more than α octets, the message shall be divided into multiple PDUs with added FALAR-N headers respectively. Table 26 shows the value of α. The PDUs are sent to the data field in order. The receiver node receives the PDUs and assembles them into one original message.

Table 26 – Value of α

Communication type	Value of α
IPv4 + TCP	MTU – 104
IPv4 + UDP	MTU – 92
IPv6 + TCP	MTU – 124
IPv6 + UDP	MTU – 112

5.3.2.16.2 Header information for fragmentation and assembly

Table 27 shows one example of the header information for a UDP message fragmentation.

Table 28 shows one example of the header information for a TCP message fragmentation. hd_cbn, hd_tbn and hd_bsize refer to 5.3.2.16, 5.3.2.18 and 5.3.2.19 respectively.

Table 27 – Example of header information for a UDP message fragmentation

Example of fragmentation	Fragmented into four packets				No fragmentation
	1st packet	2nd packet	3rd packet	4th packet	Single packet
hd_seq	100	100	100	100	101
hd_cbn / hd_tbn	1 / 4	2 / 4	3 / 4	4 / 4	1 / 1
hd_bsize	1 472	1 472	1 472	340	1 024
hd_ml	4 564	4 564	4 564	4 564	1 024

Table 28 – Example of header information for a TCP message fragmentation

Example of fragmentation	Fragmented into four packets				No fragmentation
	1st packet	2nd packet	3rd packet	4th packet	Single packet
hd_seq	100	100	100	100	101
hd_cbn / hd_tbn	1 / 4	2 / 4	3 / 4	4 / 4	1 / 1
hd_bsize	1 460	1 460	1 460	376	1 024
hd_ml	4 564	4 564	4 564	4 564	1 024

5.3.2.17 hd_cbn

This field shall specify a current fragmented PDU block number (more than 1).

5.3.2.18 hd_tbn

This field shall specify a total block number of fragmented PDUs (more than 1).

5.3.2.19 hd_bsize

This field shall specify a PDU block size (including FALAR-N header size).

5.3.2.20 inqid_inq_sa

This field shall specify a source address. This hd_inqid.inq_sa value is specified in Table 29.

Table 29 – inqid_inq_sa value

Elements of hd_inqid field	Description		
	Inq-PDU, Ninq-PDU	Reply-PDU	Other PDU
Inqid_inq_sa	Source address(= hd_sa)	Same value that is set to inqid_inq_sa of the received Inq-PDU or Ninq-PDU	0

5.3.2.21 inqid_tr_adr

This field shall indicate the inquiry-transaction sequence number of the inquiry / N-inquiry message. The range of values is from 0x00000001 to 0x0FFFFFFF, and used cyclically. When an N-inquiry message is transmitted, this value shall be incremented by one.

In the case of retransmitting the N-inquiry message, the same value shall be used.

This inqid_tr_adr value is managed separately for each inqid_id_seq. Refer to 5.3.2.22.

This inqid_ tr_adr value is specified in Table 30.

Table 30 – inqid_tr_adr value

Elements of hd_inqid field	Description			
	Inq-PDU	Ninq-PDU	Reply-PDU	Other PDU
Inqid_tr_adr	any	1 .. 0x0FFFFFFF	Same value that is set to inqid_tr_adr of the received Inq-PDU or Ninq-PDU	0

5.3.2.22 inqid_id_seq

This field shall specify the inquiry-identification sequence number. Each inquiry / N-inquiry message requires a unique inqid_inq_seq value in the source node. This inqid_id_seq value is specified in Table 31.

Table 31 – inqid_inq_seq value

Elements of hd_inqid field	Description		
	Inq-PDU, Ninq-PDU	Reply-PDU	Other PDU
Inqid_id_seq	1 .. MaxIdSeq MaxIdSeq = 0x0400 (default value = 0x0080) Not to use the same value simultaneously.	Same value that is set to inqid_id_seq of the received Inq-PDU or Ninq-PDU	0

Table 32 shows relationship between inqid_id_seq and inqid_tr.

Table 32 – Relationship between inqid_id_seq and inqid_tr_adr

Elements of hd_inqid field	Description
Inqid_id_seq	**hd_inqid.tr_adr**
1	1 .. 0x0FFFFFFF
2	1 .. 0x0FFFFFFF
. .	1 .. 0x0FFFFFFF
128	1 .. 0x0FFFFFFF

5.3.3 CyclicData-PDU

5.3.3.1 General

CyclicData-PDU transmits cyclic transfer memory in a node to all nodes belonging to a specified multicast group, cyclically. This PDU uses a UDP port.

5.3.3.2 falArHeader

Refer to 5.3.2.

5.3.3.3 tmid

Cyclic transfer memory has unique transfer memory identifiers (TMIDs) defined for each data field. This field shall indicate a tmid in a target data field.

Tmid value: 1 to 8 per a data field.

5.3.3.4 blockNumber

This protocol transfers cyclic transfer memory I units of blocks, each having an assumed size of 64 octets. This blockNumber field shall indicate the first block number of cyclicData in PDU.

5.3.3.5 blockCount

This field shall indicate a number of blocks of cyclicData in PDU.

5.3.3.6 cyclicData

This field shall contain cyclic transfer memory data.

5.3.4 MulticastData-PDU

5.3.4.1 General

MulticastData-PDU transmits a user multicast data to all nodes belonging to a specified multicast group. This PDU uses a UDP port. A multicast communication with retransmission also uses this MulticastData-PDU to prevent loss of receiving messages.

5.3.4.2 falArHeader

Refer to 5.3.2.

5.3.4.3 multicastData

This field contains user multicast data.

5.3.5 PtoP Data-PDU

5.3.5.1 General

PtoP Data-PDU transmits a user data to the specified node. This PDU uses TCP port.

5.3.5.2 falArHeader

Refer to 5.3.2.

5.3.5.3 pointToPointData

This field contains user point- to- point data.

5.3.6 Aliveinfo-PDU

5.3.6.1 General

In IPv4 network environment, Aliveinfo-PDU transmits periodically the status of own node to all nodes belonging to a specified data field. This PDU uses UDP port.

5.3.6.2 falArHeader

Refer to 5.3.2.

5.3.6.3 al_nd_name

This field shall specify a node name (An ASCII string, up to ten characters).

5.3.6.4 al_os_name

This field shall specify a name of vender device (An ASCII string, up to ten characters).

5.3.6.5 al_tm_out

This field shall specify the monitoring interval time (in seconds) for an alive-message. This monitoring will be restart each time a new alive-message arrives from a monitored node. If the monitoring node did not receive an alive-message from the monitored node within the time specified by the al_tm_out, the monitoring node recognizes that a monitored node has been dead.

5.3.6.6 al_msgserno

This field shall specify the message sequence number of an alive-message. The range of values is from 0x0001 to 0x7FFF, and used cyclically. In sequence number check for a received alive-message, both hd_v_seq and al_msgserno are effective.

5.3.6.7 al_mode

This field shall specify the type of an alive-message as shown in Table 33.

Table 33 – Type of an alive-message

Value	Description	Remark
1	Normal report	Mandatory.
2	Notice (Shutdown)	Option. Cause of terminating the transmission of alive-message: Shutdown of the equipment
3	Notice (Maintenance)	Option. Cause of terminating the transmission of alive-message : Maintenance of the equipment

5.3.6.8 al_protocol

This field shall specify the type of an alive-message protocol as shown in Table 34.

Table 34 – Type of an alive-message protocol

Value	Description	Remark
1	Category N_f	Autonomous decentralized system protocol in this specifications
2 to 3	Reserved	for other existing protocols
4	Category N_l	Autonomous decentralized system protocol (light weight type)
5 to 255	Reserved	For future uses

5.3.6.9 al_tg_cmn_cnt

This field shall specify the number of times that have changed an extended information (al_ExtentionInfo).

5.3.6.10 al_tg_cflag

This field is for future use.

5.3.6.11 al_tg_max

This field shall specify the maximum number of tasks. The range of values is from 0x0000 to 0x0053.

5.3.6.12 al_tg_usecnt

This field shall specify the number of monitored tasks information.

5.3.6.13 al_chg_time

This field shall specify the time when the node status changed. It is recommended to set the time defined as the number of seconds that have elapsed since 00:00:00 Greenwich Mean Time (GMT), 1 January 1970.

If setting in GMT is unavailable, specify 0. Time of each al_mode is as shown in Table 35.

Table 35 – Time of each al_mode

Value of al_mode	Description
1	Time when the status of node changed from “dead” state to “alive“ state.
2	Time when the status of node changed from “alive“ state to “acceptance of the shutdown request” state.
3	Time when the status of node changed from “alive“ state to “acceptance of the maintenance request” state.

5.3.6.14 al_ipv4addr1

This field shall specify IPv4 address of LAN1.

5.3.6.15 al_ipv4addr2

This field shall specify IPv4 address of LAN2.

Specify 0 in the case of a single LAN (i.e. not duplex LANs).

5.3.6.16 al_ver

This field shall specify Alive Message version (= 1).

5.3.6.17 al_ta_chgalvstat

This field shall specify the changes in tasks status as shown in Table 36.

Table 36 – Status change of tasks

Value	Description
0	No change
1	Status change (from “dead “ state to “alive” state)
2	Status change (from “alive “ state to “dead” state)

5.3.6.18 al_ta_chginfostat

This field shall specify the changes in tasks content as shown in Table 37.

Table 37 – Change of tasks content

Value	Description
0	No changes in tasks content
1	Changes in tasks content

5.3.6.19 al_ta_tid

This field shall specify the identification of tasks.

5.3.6.20 al_ta_data

This field indicates the status information of tasks.

5.3.6.21 al_ExtentionInfo

This field is an extended information area that allows user to set data freely.

5.3.7 Aliveinfo6-PDU

5.3.7.1 General

In IPv6 network environment, Aliveinfo-PDU transmits periodically the status of own node to all nodes belonging to a specified data field. This PDU uses UDP port.

5.3.7.2 falArHeader

Refer to 5.3.2.

5.3.7.3 al_nd_name

Refer to 5.3.6.3.

5.3.7.4 al_os_name

Refer to 5.3.6.4.

5.3.7.5 al_tm_out

Refer to 5.3.6.5.

5.3.7.6 al_msgserno

Refer to 5.3.6.6.

5.3.7.7 al_mode

Refer to 5.3.6.7.

5.3.7.8 al_protocol

Refer to 5.3.6.8.

5.3.7.9 al_tg_cmn_cnt

Refer to 5.3.6.9.

5.3.7.10 al_tg_cflag

Refer to 5.3.6.10.

5.3.7.11 al_tg_max

Refer to 5.3.6.11.

5.3.7.12 al_tg_usecnt

Refer to 5.3.6.12.

5.3.7.13 al_chg_time

Refer to 5.3.6.13.

5.3.7.14 al_ipv4addr1

This dummy field shall specify 0.

5.3.7.15 al_ipv4addr2

This dummy field shall specify 0.

5.3.7.16 al_ver

This field shall specify Alive Message version (= 2).

5.3.7.17 al_ipv6addr1

This field shall specify IPv6 address of LAN1.

5.3.7.18 al_ipv6addr2

This field shall specify IPv6 address of LAN2.

Specify 0 in the case of a single LAN (i.e. not duplex LANs).

5.3.7.19 al_ta_chgalvstat

Refer to 5.3.6.17.

5.3.7.20 al_ta_chginfostat

Refer to 5.3.6.18.

5.3.7.21 al_ta_tid

Refer to 5.3.6.19.

5.3.7.22 al_ta_data

Refer to 5.3.6.20.

5.3.7.23 al_ExtentionInfo

Refer to 5.3.6.21.

5.3.8 Inq-PDU

5.3.8.1 General

Inq-PDU transmits an inquiry message to nodes belonging to a specified multicast group.

Each receiving node transmits Reply-PDU to the sending node. Two usages are as follows:

a) Inquiry to all nodes using UDP port,

b) Inquiry to the specific node using TCP port.

5.3.8.2 falArHeader

Refer to 5.3.2.

5.3.8.3 inquirydata

This field contains user inquiry data.

5.3.9 Ninq-PDU

5.3.9.1 General

Ninq-PDU transmits an inquiry message to all nodes belonging to a specified multicast group.

Each node specified in this PDU transmits a Reply-PDU to all nodes. This PDU uses a UDP port.

5.3.9.2 falArHeader

Refer to 5.3.2.

5.3.9.3 node-list

This field shall specify a node list, as shown in Figure 7. Each bit corresponds to node number.

(Example)
The first bit (0) of the first octet (0) indicates node number 1.
The second bit (1) of the first octet (0) indicates node number 2.
The eighth bit (7) of the 512th octet (511) indicates node number 4 096.

	0	1	2	3	4	5	6	7	bit
0	R	R	R	R	R	R	R	R	
1	R	R	R	R	R	R	R	R	
2	R	R	R	R	R	R	R	R	
.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	
511 Octet	R	R	R	R	R	R	R	R	

R = 1 : Request the target node to reply , R = 0: No request

Figure 7 – Node-list

5.3.9.4 ninquirydata

This field contains user ninquiry (one-to-N inquiry/ multi-response) data.

5.3.10 Reply-PDU

5.3.10.1 General

Reply-PDU transmits a reply to a node that sent an Inq-PDU or a Ninq-PDU. This PDU uses TCP port.

5.3.10.2 falArHeader

Refer to 5.3.2.

5.3.10.3 replydata

This field contains user reply data.

5.3.11 RetransEnq-PDU

5.3.11.1 General

In Multicast communication with retransmission, RetransEnq-PDU transmits a request to retransmission of specified messages to all nodes belonging to a specified multicast group.

This PDU uses UDP port of retransmission control.

A node requested for retransmission transmits all messages that have sent, including the specified message.

Timing that a RetransEnq-PDU is transmitted is as follows;

- Timing that a receiving node detects the lost messages,
- Timing that a receiving node detects the loss of messages that is informed by RetransConfirm-PDU.

5.3.11.2 falArHeader

Refer to 5.3.2.

5.3.11.3 retransRequest

This field shall specify the retransmission request code (= 1).

5.3.11.4 retransRequestNode

This field shall specify the node number requested to retransmit messages.

5.3.11.5 retransNumberOfRequests

This field shall specify the number of requests contained in this PDU. One request consists of both retransMcg and retransPseqNo.

5.3.11.6 retransMcg

This field shall specify multicast group number (MCG) to which the retransmitting node belongs.

5.3.11.7 retransPseqNo

This field shall specify the packet sequence number of the first retransmitting packet.

5.3.12 RetransConfirm-PDU

5.3.12.1 General

In Multicast communication with retransmission, RetransConfirm-PDU transmits a request to confirm that the specified message has arrived at the destination nodes belonging to a specified multicast group.

If a receiving node confirms that the specified message has not arrived, it transmits a RetransEnq-PDU to request retransmission of a message.

This PDU uses UDP port of retransmission control.

5.3.12.2 falArHeader

Refer to 5.3.2.

5.3.12.3 retransRequest

This field shall specify the retransmission confirmation code (= 2). When a transmitting packet did not occur within the predefined time, the source node transmits this PDU to confirm that the specified packet has reached to the target node.

5.3.12.4 retransNumberOfRequests

Refer to 5.3.11.5.

5.3.12.5 retransMcg

This field shall specify the multicast group number (MCG) to which the target node belongs.

5.3.12.6 retransPseqNo

This field shall specify the packet sequence number which the target node is requested to confirm.

5.3.13 RetransNak-PDU

5.3.13.1 General

In Multicast communication with retransmission, when a node cannot transmit a message requested by a RetransEnq-PDU, a node transmits a RetransNak-PDU.

This PDU uses UDP port of retransmission control.

5.3.13.2 falArHeader

Refer to 5.3.2.

5.3.13.3 retransRequest

This field shall specify the retransmission reject code (= 3).

5.3.13.4 retransNumberOfRequests

Refer to 5.3.11.5.

5.3.13.5 retransMcg

This field shall specify multicast group number (MCG) to which the retransmission request node belongs.

5.3.13.6 retransPseqNo

When the target node cannot retransmit the packet specified in RetransEnq-PDU, the target node transmits this RetransNak-PDU to inform this status. This field shall specify the packet sequence number specified in RetransEnq-PDU.

6 Structure of the FAL protocol state machine

The FAL protocol state machine consists of three protocol state machines as shown in Figure 8. These three protocol state machines, in the order from the data link layer side are as follows, data link layer mapping protocol machine (DMPM), application relationship protocol machine (ARPM), and FAL service protocol machine (FSPM).

The role of FSPM is to receive service primitives from FAL users and convert the primitives to internal primitives, select an ARPM state machine, and receive the internal primitives from ARPM and convert the primitives to FA service primitives ARPM and transmit them to FAL users.

The role of ARPM is to convert services primitives between ARPM and DMPM.

The role of DMPM is to map into the data link layer.

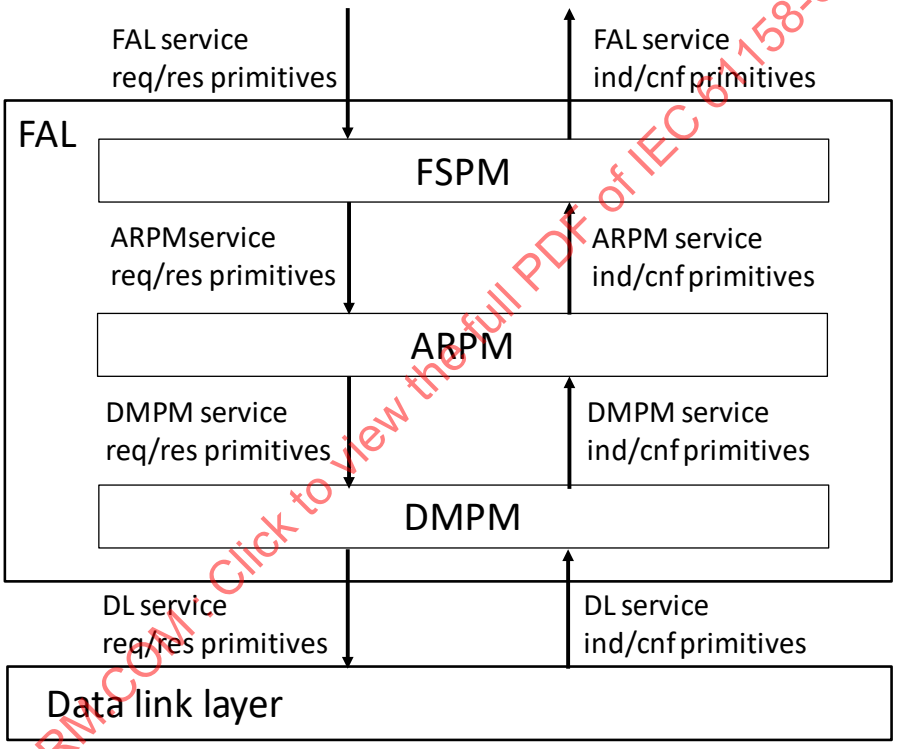


Figure 8 – Relationships between protocol machines

7 FAL service protocol machine (FSPM)

7.1 Overview

The FSPM provides an interface to FAL users. It performs the mapping between FAL user services and FAL internal services.

7.2 FSPM type S

7.2.1 Overview

The FSPM consists of three protocol machines: Cyclic data, acyclic data and management. The relationship between protocol machines is shown in Figure 9.

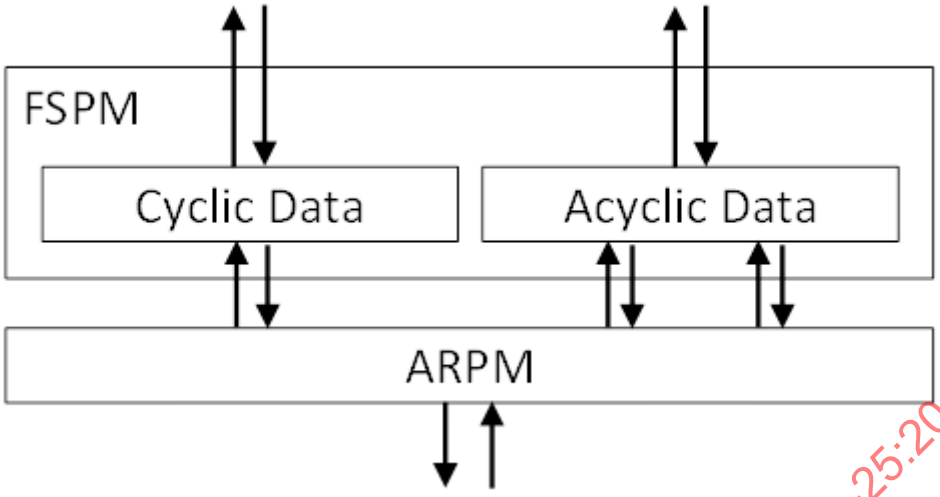


Figure 9 – Structure of FSPM type S

The following primitives are issued from the FAL user to the FSPM:

Put_cyclicdata.req
Get_cyclicdata.req
Ctl_cyclic.req
Send_ctldata.req
Send_infodata.req
Send_rmtctl.req

The following primitives are issued from the FSPM to the FAL user.

Put_cyclicdata.cnf
Get_cyclicdata.cnf
Ctl_cyclic.cnf
Send_ctldata.ind
Send_ctldata.cnf
Send_infodata.ind
Send_infodata.cnf
Send_rmtctl.cnf

7.2.2 Interface of cyclic communication to FAL users

In cyclic communication, each node independently schedules transmission of the cyclic communication frame using its own timer. To manage cyclic communication within transmission cycle, the communication traffic of each node is restricted.

In type S network, the amount of data traffic called “Transmission factor” is smaller than “Threshold.” Transmission factor is defined as the number of blocks (64 octet/block) sent per cycle. The relation among the cycle time, the number of blocks, and the transmission factor is shown in the equation below

(Transmission factor)
= (Number of blocks for 1 ms cycle)/1 + (Number of blocks for 2 ms cycle)/2
+ (Number of blocks for 5 ms cycle)/5 + (Number of blocks for 10 ms cycle)/10
+ (Number of blocks for 20 ms cycle)/20 + (Number of blocks for 50 ms cycle)/50
+ (Number of blocks for 100 ms cycle)/100
≤ (Threshold)

The minimum cycle time is determined by the number of nodes. When a network is composed of 16 nodes, the minimum cycle time is 1 ms. Composed of 16 to 32 nodes, the network can use 2 ms cycle at the minimum. From 33 to 80 nodes, the minimum cycle is 5 ms. From 81 to 127 nodes, it only use over 10 ms cycle time. Therefore, threshold is different according to the total of cable length and the minimum cycle time. Table 38 shows the threshold of transmission factor.

Table 38 – The threshold of transmission factor

No.	Total of cable length [km]	Minimum cycle time [ms]	Threshold
1	0 km to 10 km	1 ms to 5 ms	256.0
2	0 km to 10 km	Over 10 ms	409.6
3	Over 10 km	1 ms to 5 ms	128.0
4	Over 10 km	Over 10 ms	204.8

The blocks used in cyclic communication are specified by FAL users. Table 39 shows an example of the traffic control configuration menu. It gives the total available blocks according to the number of nodes and used cycle time in a type S network. For example, No. 2 shows the case of connecting 16 nodes and using 1 ms and 2 ms cycle time. In this case, the network can use 128 blocks in 1 ms cycle and 256 blocks in 2 ms cycle.

Table 39 – Example of the traffic control configuration menu

No.	Number of Node	Available block number in transfer cycle [ms] (64Byte/Block)						
		1	2	5	10	20	50	…
1	16	256						
2		128	256					
3		128		640				
4		128		320	640			
5		128		320		1280		
6	32		512					
7			256	640				
8			256		1280			
9			256		640	1280		
10			256		640		3200	
:								

Figure 10 shows an example of shared memory allocation in Type S network. Each node has the memory for cyclic communication and allocates the transmission area to store its own data for sending to other nodes. It schedules the cyclic communication by using its own timer. Number of the allocated blocks is less than the available block number.

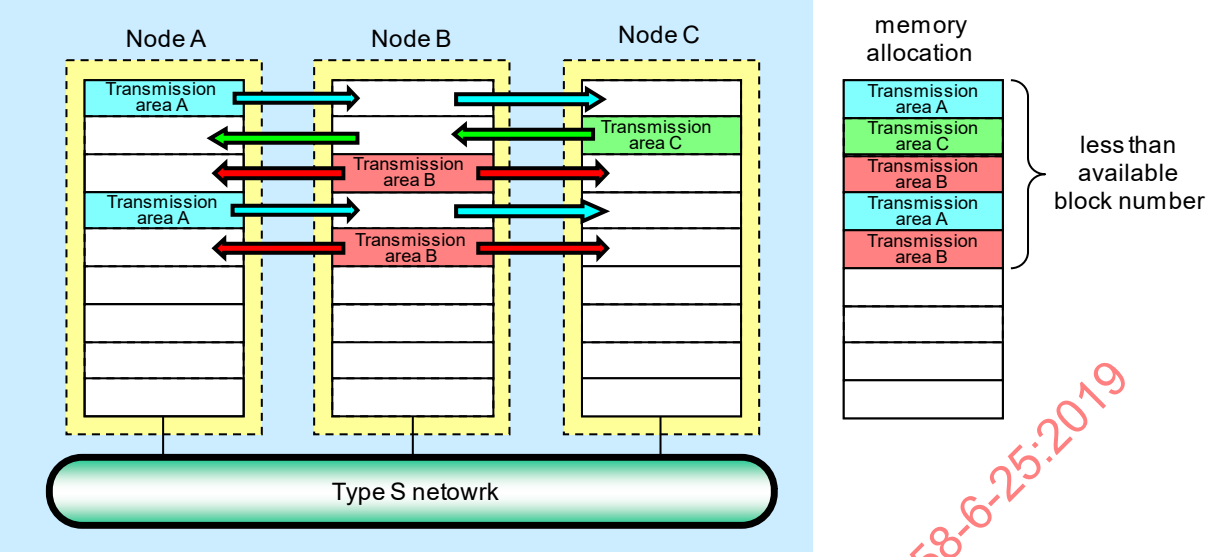


Figure 10 – Shared memory allocation in type S network

7.2.3 State machine of FSPM

7.2.3.1 Cyclic data

Details of Cyclic data state machine are shown in Table 40.

Table 40 – Cyclic data state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Put_cyclicdata.req{CYC_ID,Block_ID,CYCdata} => CYC_WRITE.req{CYC_ID,Block_ID,CYCdata}	ACTIVE
2	ACTIVE	CYC_WRITE.cnf{CYC_ID,Status} => Put_cyclicdata.cnf{CYC_ID,Status}	ACTIVE
3	ACTIVE	Get_cyclicdata.req{CYC_ID,Block_ID} => CYC_READ.req{CYC_ID,Block_ID}	ACTIVE
4	ACTIVE	CYC_READ.cnf{CYC_ID,CYCdata,Status} => Get_cyclicdata.cnf{CYC_ID,CYCdata,Status}	ACTIVE
5	ACTIVE	Ctl_cyclic.req{CYC_ID,CYCctl,CYC-CTLdata} => CTL_CYCLIC.req{CYC_ID,CYCctl,CYC-CTLdata}	ACTIVE
6	ACTIVE	CTL_CYCLIC.cnf{CYC_ID,Status} => Ctl_cyclic.cnf{CYC_ID,Status}	ACTIVE

7.2.3.2 Acyclic data

Details of Acyclic data state machine are shown in Table 41.

Table 41 – Acyclic data state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Send_ctldata.req{D_add,S_add,CTLdata} => SendCTL.req{D_add,S_add,CTLdata}	ACTIVE
2	ACTIVE	SendCTL.cnf{Status} => Send_CTLdata.cnf{Status}	ACTIVE
3	ACTIVE	SendCTL.ind{D_add,S_add,CTLdata} => Send_ctldata.ind{D_add,S_add,CTLdata}	ACTIVE
4	ACTIVE	Send_infodata.req{D_add,S_add,INFOdata} => SendINFO.req{D_add,S_add,INFOdata};	ACTIVE
5	ACTIVE	SendINFO.cnf{Status} => Send_infodata.cnf{Status}	ACTIVE
6	ACTIVE	SendINFO.ind{D_add,S_add,INFOdata} => Send_infodata.ind{D_add,S_add,INFOdata}	ACTIVE
7	ACTIVE	Send_rmtctl.req{D_add,S_add,CMD,CMDdata} => SendRMTCTL.req{D_add,S_add,CMD,CMDdata}	ACTIVE
8	ACTIVE	SendRMTCTL.cnf{D_add,S_add,CMD,CMDdata} => Send_rmtctl.cnf{D_add,S_add,CMD,CMDdata}	ACTIVE

7.3 FSPM type N

7.3.1 Overview

The FSPM consists of three protocol machines: Cyclic data, Acyclic data and Management. The relationship between protocol machines is shown in Figure 11.

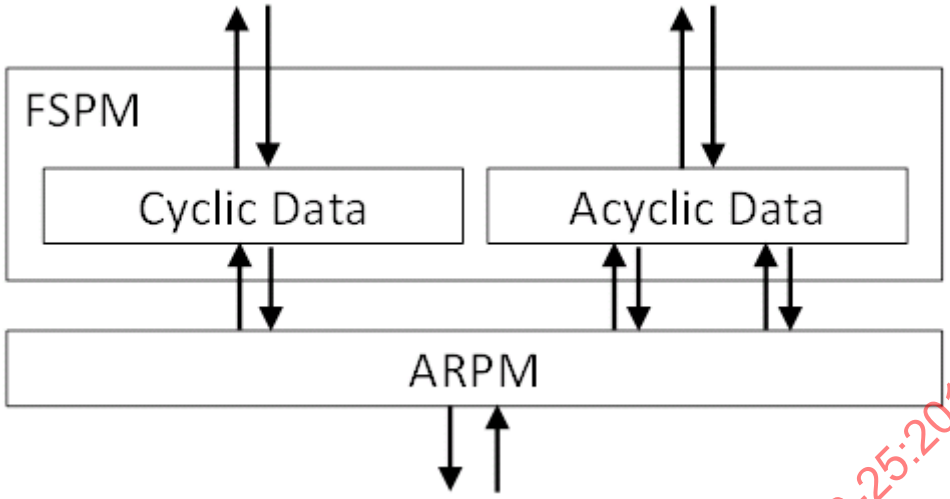


Figure 11 – Structure of FSPM type N

The following primitives are issued from the FAL user to the FSPM:

Put_cyclicdata.req
Get_cyclicdata.req
Put_message.req
Get_message.req
Put_inquiry message.req
Put_reply_message.req
Put_ninquiry_message.req
Send_aliveinfo.req
Receive_aliveinfo.req
Control_acyclic.req
Control_cyclic.req

The following primitives are issued from the FSPM to the FAL user.

Get_cyclicdata.cnf
Put_message.ind
Get_message.cnf
Put_inquiry_message.ind
Put_reply_message.ind
Put_ninquiry_message.ind
Receive_aliveinfo.cnf

7.3.2 FSPM

7.3.2.1 Cyclic data

Details of Cyclic data state machine are shown in Table 42.

Table 42 – Cyclic data state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Put_cyclicdata.req => Write_cyclicdata.req	ACTIVE
2	ACTIVE	Get_cyclicdata.req => Get_cyclicdata.cnf	ACTIVE
3	ACTIVE	Write_cyclicdata.ind => Update cyclicdata	ACTIVE
4	ACTIVE	Control_cyclic.req => Ctl_cyclic.req	ACTIVE

7.3.2.2 Acyclic data

Details of Acyclic data state machine are shown in Table 43.

Table 43 – Acyclic data state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Put_message.req => Transmit_acyclicdata1.req	ACTIVE
2	ACTIVE	Transmit_acyclicdata1.ind / (CtlType == MLT) || (CtlType == ONE) => Put_message.ind	ACTIVE
3	ACTIVE	Get_message.req => Get_message.cnf	ACTIVE
4	ACTIVE	Put_inquiry_message.req => Transmit_acyclicdata1.req	ACTIVE
5	ACTIVE	Transmit_acyclicdata1.ind / CtlType == INQ => Put_inquiry_message.ind	ACTIVE
6	ACTIVE	Put_ninquiry_message.req => Transmit_acyclicdata1.req	ACTIVE

#	Current state	Event /condition => actions	Next state
7	ACTIVE	Transmit_acyclicdata1.ind / CtlType == NIQ => Put_ninquiry_message.ind	ACTIVE
8	ACTIVE	Put_reply_message.req => Transmit_acyclicdata1.req	ACTIVE
9	ACTIVE	Transmit_acyclicdata1.ind / CtlType == RPL => Put_reply_message.ind	ACTIVE
10	ACTIVE	Send_aliveinfo.req => Transmit_acyclicdata2.req	ACTIVE
11	ACTIVE	Receive_aliveinfo.req => Receive_aliveinfo.cnf	ACTIVE
12	ACTIVE	Control_acyclic.req => Ctl_acyclic.req	ACTIVE
13	ACTIVE	Transmit_acyclicdata2.ind => Send_aliveinfo.req	ACTIVE

8 Application relationship protocol machine (ARPM)

8.1 ARPM type S

8.1.1 Overview

The ARPM consists of four sub-protocols: Cyclic control, Remote control, RCL communication control, and RT communication control. The structure of ARPM is shown in Figure 12.

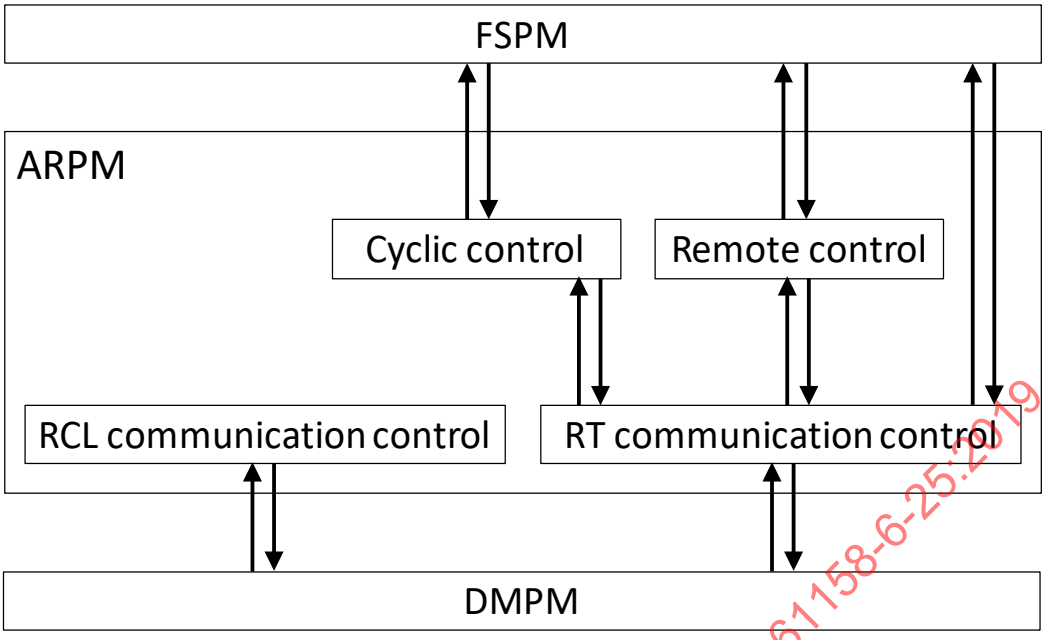


Figure 12 – Structure of ARPM type S

8.1.2 Cyclic control

8.1.2.1 Cyclic communication menu

The cyclic control generates the traffic control configuration menu based on configuration data from the FAL user. The menu is created by each node at each cycle time. The number of frames in each node is specified.

The relation among the cycle time n[ms],used blocks x[blocks] and transmission number per cycle N_nc [blocks] is shown as below.

Cycle time n [ms] (n={1,2,5,10,20,50,100,200,500,1 000})
The number of blocks x (0 ≤ x ≤ 21 × n)
The number of blocks per cycle N_nc= min(21, x-21×c) (c={0,1,⋯,[x/21]-1})

Figure 13 shows the sequence of cyclic communication based on above formulas. In the sequence, cycle 1 transmits frames with the cycle time of 10 ms and uses 105 blocks. Cycle 2 transmits frames with cycle time of 5 ms and uses 42 blocks. Cycle 3 transmits a frame with cycle time of 1 ms and uses 21 blocks. The boxes in Figure 13 mean sent frames, the number in each box means cyclic 1, 2, and 3 respectively.

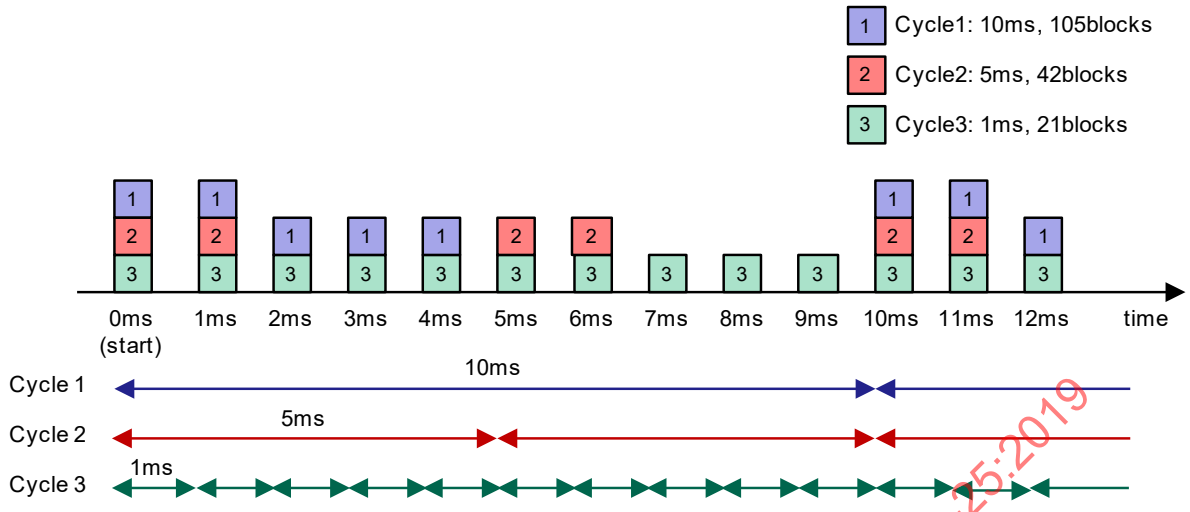


Figure 13 – Sequence of cyclic communication

8.1.2.2 Primitive definitions

FSPM issues a CYC_READ.req service, CYC_WRITE.req service, or CTL_CYCLIC.req service to Cyclic control. RT communication control issues a SendCYC.ind service to Cyclic control. Cyclic control issues a CYC_READ.cnf service, CYC_WRITE.cnf service, or CTL_CYCLIC.cnf service to FSPM. Cyclic control issues a SendCYC.req service to RT communication control.

The primitives of Cyclic control are shown in Figure 14.

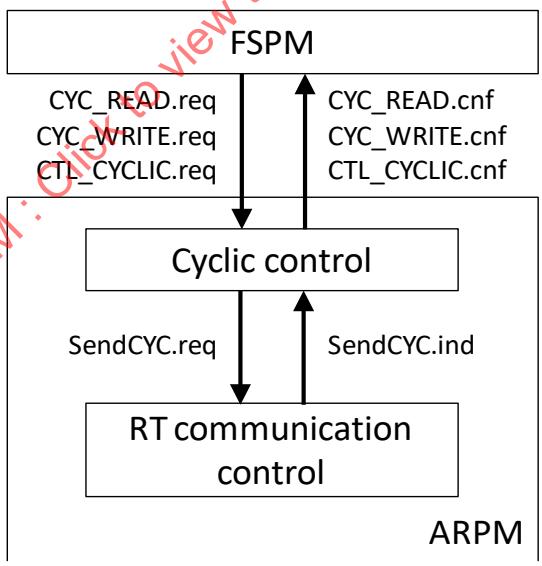


Figure 14 – The primitives for cyclic control

8.1.2.3 Cyclic control state table

Details of Cyclic control state machine are shown in Table 44.

Table 44 – Cyclic control state table

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset => CYCMenu_flag = 0 Current_cycle = 0	READY
2	Any states	CYC_READ.req{CYC_ID,Block_ID} /CYC_rcheck(Block_ID) == True => CYCdata = Read_memory(Block_ID) Status = “Success” CYC_READ.cnf{CYC_ID,Block_ID, CYCdata,Status}	Any states (no change)
3	READY	CYC_WRITE.req{CYC_ID,Block_ID,CYCdata} /CYC_wcheck(Block_ID,CYCdata) == True => WRITE_Memory(Block_ID,CYCdata) Status = “Success” CYC_WRITE.cnf{CYC_ID,Status}	READY
4	READY	CTL_CYCLIC.req{CYC_ID,CYCctl,CYC-CTLdata} /Config_check(CYCctl,CYC-CTLdata) == True && CYCctl == 1 /* Start */ => Make_CYCMenu(CYC-CTLdata) START_TIMER(CYCtask,CYC_Cycle) Status = “Success” CTL_CYCLIC.cnf{CYC_ID,Status}	ACTIVE
5	ACTIVE	CYC_WRITE.req{CYC_ID,Block_ID,CYCdata} /CYC_wcheck(Block_ID,CYCdata) == True => WRITE_Memory(Block_ID,CYCdata);	ACTIVE

6	ACTIVE	/EXPIRED_TIMER(CYCtask) == “True”
		=>
		Current_cycle++
		CYCdata= Send_CYCMenu(Current_cycle)
		D_add = Broadcast
		S_add = Myadd
		SendCYC.req{D_add,S_add,CYCdata}
		IF(Current_cycle == Max_CYCLE) THEN
		Current_cycle = 0
		ENDIF
			ACTIVE

#	Current state	Event /condition => actions	Next state
7	ACTIVE	SendCYC.ind{D_add,S_add,CYCdata} / D_add = Myadd => BlockID = Get_BlockID(CYCdat) WRITE_Memory(Block_ID,CYCdata)	ACTIVE
8	ACTIVE	CTL_CYCLIC.req{CYC_ID,CYCCTL,CYC-CTLdata} /CYCCTL == 0 /* Stop */ => STOP_TIMER(CYCtask,CYC_Cycle) Status = “Success” CTL_CYCLIC.cnf{CYC_ID,Status}	READY

8.1.2.4 Functions

All the functions used by the cyclic control are summarized in Table 45.

Table 45 – Cyclic control functions

Function Name	Input	Output	Description
START_TIMER	CYCtask, CYC_Cycle	(<none>)	Timer CYCtask is set by value of CYC_Cycle, and is activated.
EXPIRED_TIMER	CYCtask	True/False	When the requested timer CYCtask has expired, “True” is returned, otherwise “False” is returned.
CYC_rcheck	Block_ID	True/False	This function is used to check whether the Block_ID is valid for reading from the shared memory.
CYC_wcheck	Block_ID, CYCdata	True/False	This function is used to check whether the Block_ID and CYCdata are valid for writing to the shared memory.
Read_memory	Block_ID	CYCdata	This function loads the cyclic data from block in shared memory specified Block_ID.
WRITE_Memory	Block_ID, CYCdata	True/False	This function writes CYCdata to block in shared memory specified Block_ID.
Config_check	CYCctl, CYC-CTLdata	True/False	This function is used to check whether the Block_ID is valid for reading from the shared memory.
Make_CYCMenu	CYC-CTLdata	True/False	This function makes cyclic menu to send cyclic communication frames.
Send_CYCMenu	Current_Cycle	CYCdata	This function returns the CYCdata to send cyclic frames. This function load CYCdata[Current_Cycle] table and return the variable data.

8.1.2.5 Variables

All the variables used by the cyclic control are summarized in Table 46.

Table 46 – Cyclic control variables

Name	Description
Current_cycle	This variable indicates the current cycle to send cyclic communication frames. If this variable is 0, cyclic communication is not activated in the node. If cyclic communication is activated, this variable takes from 1 to Max_CYCLE.
Broadcast	This variable is broadcast MAC address: FF-FF-FF-FF-FF-FF.
Myadd	This variable indicates the address of own node.
Max_CYCLE	This variable indicates the maximum cycle that can be used in cyclic communication on Type S network.

8.1.3 Remote control

8.1.3.1 Primitive definitions

FSPM issues a SendRMTCTL.req service to Remote control. RT communication control issues a SendCTL_RMT.ind service and SendCTL_RMT.cnf service to Remote control. Remote control issues a SendRMTCTL.cnf service to FSPM. Remote control issues a SendCTL_RMT.req service and SendCTL_RMT.rsp to RT communication control.

The primitives of Remote control are shown in Figure 15.

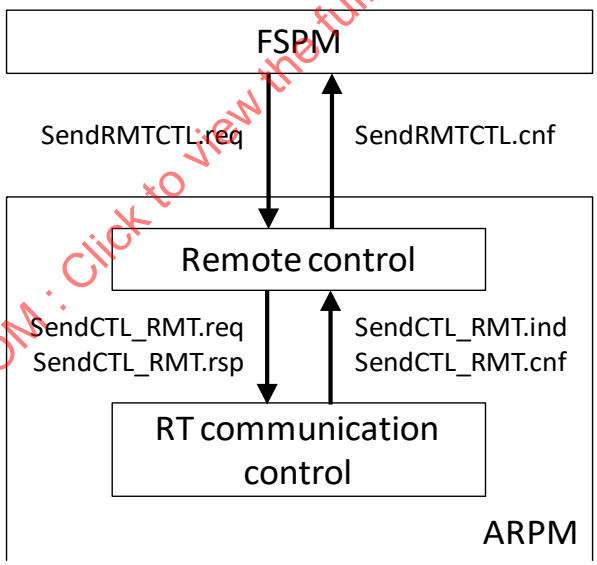


Figure 15 – The primitives for Remote control

8.1.3.2 Remote control state table

Details of Remote control state machine are shown in Table 47.

Table 47 – Remote control state table

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset =>	ACTIVE
2	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} /D_add != My_add => CTLdata = CreateData(CMD,CMDdata) SendCTL_RMT.req{D_add,S_add,CTLdata}	ACTIVE

3	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata}
		/D_add == My_add &&
		CMD_code(CMD) == “RMT_Read”
		=>
		D_add_rsp = S_add
		S_add_rsp = My_add
		R_add = CMD_address(CMD)
		R_size = CMD_size(CMD)
		CMDdata_rsp = Readmemory(R_add,R_size)
		if(CMDdata_rsp != NULL) THEN
		Status = “Success”
		ELSE
		Status = “Soft I/F Error”
		ENDIF
		CMD_rsp = MakeRSPCMD(CMD,Status)
		SendRMTCTL.cnf{D_add_rsp,S_add_rsp,
		CMD_rsp,CMDdata_rsp}
			ACTIVE

4	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} /D_add == My_add && CMD_code(CMD) == “RMT_Reset” => Reset()	-
5	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} / D_add == My_add && CMD != “RMT_Read” && CMD != “RMT_Reset” => D_add_rsp = S_add S_add_rsp = My_add Status = “Command parameter error” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL SendRMTCTL.cnf{D_add_rsp,S_add_rsp, CMD_rsp,CMDdata_rsp}	ACTIVE

#	Current state	Event /condition => actions	Next state
6	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} / D_add == My_add && CMD_check(CTLdata) == “RMT_Start” => Start_CPU() D_add_rsp = S_add S_add_rsp = My_add Status = “Success” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE
7	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /D_add == My_add && CMD_check(CTLdata) == “RMT_Stop” => Stop_CPU() D_add_rsp = S_add S_add_rsp = My_add Status = “Success” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE

#	Current state	Event /condition => actions	Next state
8	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /D_add == My_add && CMD_check(CTLdata) == “RMT_Write” => D_add_rsp = S_add S_add_rsp = My_add CMD = CTL2CMD(CTLdata) CMDdata = CTL2DATA(CTLdata) W_add = CMD_address(CMD) W_size = CMD_size(CMD) Status = Writememory(W_add,W_size) CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE

9	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata}
		/D_add == My_add &&
		CMD_check(CTLdata) == “RMT_Read”
		=>
		D_add_rsp = S_add
		S_add_rsp = My_add
		CMD = CTL2CMD(CTLdata)
		CMDdata = CTL2DATA(CTLdata)
		R_add = CMD_address(CMD)
		R_size = CMD_size(CMD)
		CMDdata_rsp = Readmemory(R_add,R_size)
		if(CMDdata_rsp != NULL) THEN
		Status = “Success”
		ELSE
		Status = “Command parameter error”
		ENDIF
		CMD_rsp = MakeRSPCMD(CMD,Status)
		CTLdata_rsp = CreateData(CMD_rsp,
		CMDdata_rsp)
		SendCTL.rsp{D_add_rsp,S_add_rsp,
		CTLdata_rsp}
			ACTIVE

10	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /S_add == My_add && CMD_check(CTLdata) == “RMT_Reset” => Reset()	-

#	Current state	Event /condition => actions	Next state
11	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /S_add == My_add && CMD_check(CTLdata) == “Not_Valid” => D_add_rsp = S_add S_add_rsp = My_add Status = “Command parameter error” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE
12	ACTIVE	SendCTL_RMT.cnf{D_add,S_add,CTLdata} /CMD_check(CTLdata) != “Not_Valid” => CMD = CTL2CMD(CTLdata) CMDdata = CTL2DATA(CTLdata) SendRMTCTL.cnf{D_add,S_add,CMD,CMDdata}	ACTIVE

8.1.3.3 Functions

All the functions used by the Remote control are summarized in Table 48.

Table 48 – Remote control functions

Function Name	Input	Output	Description
CreateData	CMD, CMDdata	CTLdata	This function creates data to send remote control PDU.
CMD_code	CMD	CMD_type	This function gets the type of the remote control command.
CMD_address	CMD	R_add or W_add	This function gets the address for memory access.
CMD_size	CMD	R_size or W_size	This function gets the data size for memory access.
Readmemory	R_add, R_size	CMDdata_rsp	This function reads the data specified R_add and R_size from the node memory.
MakeRSPCMD	CMD, Status	CMD_rsp	This function makes response command to receive remote control command.
Reset	(<none>)	Status	This function is used to reset the node.
CMD_check	CTLdata	CMD_type	This function checks the command from CTLdata.
Start_CPU	(<none>)	Status	This function starts the CPU connected Type S network.
Stop_CPU	(<none>)	Status	This function stops the CPU connected Type S network.

Function Name	Input	Output	Description
CTL2CMD	CTLdata	CMD	This function makes remote control command from CTLdata.
CTL2DATA	CTLdata	CMDdata	This function makes data for remote control from CTLdata.
Writememory	W_add, W_size	Status	This function writes the data to the node memory.

8.1.3.4 Variables

All the variables used by the Remote control are summarized in Table 49.

Table 49 – Remote control variables

Name	Description
CMD_type	This variable indicates the command type for remote control including “RMT_Read”, “RMT_Reset”, “RMT_Start”, “RMT_Stop”, or “RMT_Write”.
My_add	This variable indicates the address of own node.
R_add	This variable is used to specify the address for reading from the node memory.
R_size	This variable is used to specify size of the data for reading from the node memory.
NULL	This variable indicates the no data.
W_add	This variable is used to specify the address for writing to the node memory.
W_size	This variable is used to specify size of the data for writing to the node memory.

8.1.4 RCL communication control

8.1.4.1 Primitive definitions

DMPM issues an RCL_START.ind service, RCL_STOP.ind service, or Node_ST.ind service to RCL communication control. RCL communication control issues a FAL-RCL.req service to DMPM.

The primitives of RCL communication control are shown in Figure 16.

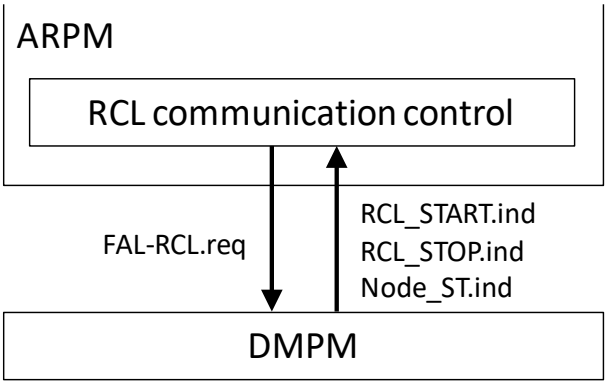


Figure 16 – The primitives for RCL communication control

8.1.4.2 RCL communication control state table

Table 50 shows the state table of the RCL communication control.

Table 50 – RCL communication control state table

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset => LCC_flag = “False” LCN_flag = “False” SCR_count = 0 START_TIMER(Fixedtask,Fixed_Cycle)	ACTIVE
2	ACTIVE	EXPIRED_TIMER(Fixedtask) == “True” => RCL_TIM_RHE() RCL_TIM_LCC() RCL_TIM_LCN() RCL_TIM_SCR()	ACTIVE
3	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCC => LCC_flag = “True” LCC_PDU = CreateLCCframe()	ACTIVE
4	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCA => LCA_PDU = CreateLCAframe(D_add) PortNum = RCLPort FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,LCA_PDU}	ACTIVE
5	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCN => LCN_flag = “True” LCN_PDU = CreateLCNframe()	ACTIVE

#	Current state	Event /condition => actions	Next state
6	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LNA => LNA_PDU = CreateLNAframe(D_add) PortNum = RCLPort FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,LNA_PDU}	ACTIVE
7	ACTIVE	RCL_STOP.ind{RCLType} /RCLType == LCC => LCC_flag = “False”	ACTIVE
8	ACTIVE	RCL_STOP.ind{RCLType} /RCLType == LCN => LCN_flag = “False”	ACTIVE
9	ACTIVE	Node_ST.ind{NodeST,PortAST,PortBST} => SCR_count = SCR_Sndnum SCR_PDU = CreateSCRframe(NodeST,PortAST, PortBST)	ACTIVE

8.1.4.3 Functions

All the functions used by the RCL communication control are summarized in Table 51.

Table 51 – RCL communication control functions

Function Name	Input	Output	Description
RCL_TIM_RHE	(<none>)	(<none>)	Fixed cycle function for sending RHE-PDU. RCL_TIM_RHE() is assembled as follows: RHE_PDU-A = CreateRHEframe(PortA) RHE_PDU-B = CreateRHEframe(PortB) S_add = Myadd RCLPri = VLAN_RCL RCLType = RHE PortNum = PortA FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,RHE_PDU-A} PortNum = PortB FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,RHE_PDU-B}

Function Name	Input	Output	Description

RCL_TIM_LCC	(<none>)	(<none>)	Fixed cycle function for sending LCC-PDU.
			RCL_TIM_LCC() is assembled as follows:
			IF(LCC_flag == “TRUE”) THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = LCC
			PortNum = Both
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			LCC_PDU}
			ENDIF

RCL_TIM_LCN	(<none>)	(<none>)	Fixed cycle function for sending LCN-PDU.
			RCL_TIM_LCN() is assembled as follows:
			IF(LCN_flag == “TRUE”) THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = LCN
			PortNum = PortB
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			LCN_PDU}
			ENDIF

RCL_TIM_SCR	(<none>)	(<none>)	Fixed cycle function for sending SCR-PDU.
			RCL_TIM_SCR() is assembled as follows:
			IF(SCR_count > 0) THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = SCR
			PortNum = Both
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			SCR_PDU}
			SCR_count--
			ENDIF

EXPIRED_TIMER	Fixedtask	True/False	When requested timer Fixedtask has expired, “True” is returned, otherwise “False” is returned.
START_TIMER	Fixedtask, Fixed_Cycle	(<none>)	Timer Fixedtask is set by value of Fixed_Cycle, and is activated.
CreateLCCframe	(<none>)	LCC_PDU	Generate the LCC-PDU
CreateLCAframe	D_add	LCA_PDU	Generate the LCA-PDU
CreateLCNframe	(<none>)	LCN_PDU	Generate the LCN-PDU
CreateLNAframe	D_add	LNA_PDU	Generate the LNA-PDU

Function Name	Input	Output	Description
CreateSCRframe	NodeST, PortAST, PortBST	SCR_PDU	Generate the SCR-PDU

8.1.4.4 Variables

All the variables used by the RCL communication control are summarized in Table 52.

Table 52 – RCL communication control variables

Name	Description
LCC_flag	This flag indicates whether sending LCC-PDU or not.
LCN_flag	This flag indicates whether sending LCN-PDU or not.
SCR_count	This counter contains the remaining number of transmissions of SCR-PDU.
SCR_Sndnum	This variable indicates the number of transmissions of SCR-PDU per an event.
Myadd	This variable indicates the address of own node.
VLAN_RCL	This variable indicates the VLAN priority to RCL communication.

8.1.5 RT communication control

8.1.5.1 Primitive definitions

FSPM issues a SendCTL.req service or SendINFO.req service to RT communication control. Cyclic control issues a SendCYC.req service to RT communication control. Remote control issues a SendCTL_RMT.req service or SendCTL_RMT.rsp service to RT communication control. DMPM issues a FAL-RTC.ind service or FAL-RTC.cnf service to RT communication control. RT communication control issues a SendCTL.ind service, SendCTL.cnf service, SendINFO.ind service, or SendINFO.cnf service to FSPM. RT communication control issues a SendCYC.ind service or SendCYC.cnf service to cyclic control. RT communication control issues a SendCTL_RMT.ind service or SendCTL_RMT.cnf service to Remote control. RT communication control issues a FAL-RTC.req service or FAL-RTC.rsp service to DMPM.

The primitives of RT communication control are shown in Figure 17.

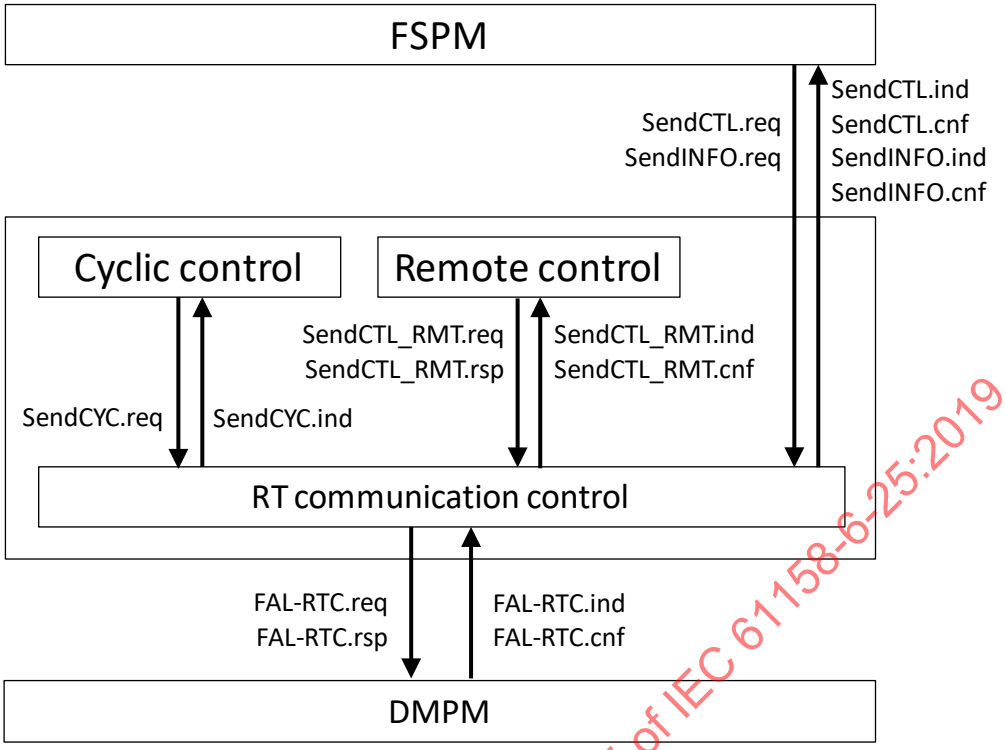


Figure 17 – The primitives for RT communication control

8.1.5.2 RT communication control state table

Table 53 shows the state table of the RT communication control.

Table 53 – RT communication control state table

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset =>	ACTIVE
2	ACTIVE	SendCTL.req{D_add,S_add,CTLdata} => RTC_PDU = CreateRTCframe(CTLdata) Frame_pri = VLAN_CTL FAL-RTC.req{D_add,S_add,Frame_pri, RTC_PDU} Status = “Success” SendCTL.cnf{Status}	ACTIVE
3	ACTIVE	FAL-RTC.ind{D_add,S_add,Frame_pri, RTC_PDU} /Frame_pri == VLAN_CTL && PDU_check(RTC_PDU) != “RMTCTL” => CTLdata = CreateRTCdata(RTC_PDU) SendCTL.ind{D_add,S_add,CTLdata}	ACTIVE

#	Current state	Event /condition => actions	Next state
4	ACTIVE	SendINFO.req{D_add,S_add,INFOdata} => RTC_PDU = CreateRTCframe(INFOdata) Frame_pri = VLAN_INFO FAL-RTC.req{D_add,S_add,Frame_pri, RTC_PDU} Status = “Success” SendINFO.cnf{Status}	ACTIVE
5	ACTIVE	FAL-RTC.ind{D_add,S_add,Frame_pri, RTC_PDU} /Frame_pri == VLAN_INFO => INFOdata = CreateRTCdata(RTC_PDU) SendINFO.ind{D_add,S_add,INFOdata}	ACTIVE
6	ACTIVE	SendCTL_RMT.req{D_add,S_add,CTLdata} => RTC_PDU = CreateRTCframe(CTLdata) Frame_pri = VLAN_CTL FAL-RTC.req{D_add,S_add,Frame_pri, RTC_PDU}	ACTIVE
7	ACTIVE	FAL-RTC.ind{D_add,S_add,Frame_pri, RTC_PDU} /Frame_pri == VLAN_CTL && PDU_check(RTC_PDU) == “RMTCTL” => CTLdata = CreateRTCdata(RTC_PDU) SendCTL_RMT.ind{D_add,S_add,CTLdata}	ACTIVE
8	ACTIVE	SendCTL_RMT.rsp{D_add,S_add,CTLdata} => RTC_PDU = CreateRTCframe(CTLdata) Frame_pri = VLAN_CTL FAL-RTC.rsp{D_add,S_add,Frame_pri, RTC_PDU}	ACTIVE
9	ACTIVE	FAL-RTC.cnf{D_add,S_add,Frame_pri, RTC_PDU} /Frame_pri == VLAN_CTL PDU_check(RTC_PDU) == “RMTCTL” => CTLdata = CreateRTCdata(RTC_PDU) SendCTL_RMT.cnf{D_add,S_add,CTLdata}	ACTIVE

#	Current state	Event /condition => actions	Next state
10	ACTIVE	SendCYC.req{D_add,S_add,CYCdata} => RTC_PDU = CreateRTCframe(CYCdata) Frame_pri = VLAN_CYC FAL-RTC.req{D_add,S_add,Frame_pri, RTC_PDU} Status = “Success” SendCYC.cnf{Status}	ACTIVE
11	ACTIVE	FAL-RTC.ind{D_add,S_add,Frame_pri, RTC_PDU} /Frame_pri == VLAN_CYC => CYCLdata = CreateRTCdata(RTC_PDU) SendCYC.ind{D_add,S_add,CYCdata}	ACTIVE

8.1.5.3 Functions

All the functions used by the RT communication control are summarized in Table 54.

Table 54 – RT communication control functions

Function Name	Input	Output	Description
CreateRTCframe	CTLdata or INFOdata	RTC_PDU	This function generates the RTC-PDU for sending the cyclic, control, and information communication.
CreateRTCdata	RTC_PDU	CTLdata or INFOdata	This function converts the RTC-PDU to the data unit for FAL service.
PDU_check	RTC_PDU	PDU_Type	This function checks the RTC_PDU for the remote control data or the general control communication data.

8.1.5.4 Variables

All the variables used by the RT communication control are summarized in Table 55.

Table 55 – RT communication control variables

Name	Description
PDU_Type	This variable indicates the PDU type and takes character “CTL”, “INFO”, “RMTCTL”, or “CYCLIC”.
VLAN_CTL	This variable indicates the VLAN priority to Control communication.
VLAN_INFO	This variable indicates the VLAN priority to Information communication.
VLAN_CYC	This variable indicates the VLAN priority to Cyclic communication.

8.2 ARPM type N

8.2.1 Overview

The ARPM consists of four sub-protocols: cyclic transmission control, acyclic transmission control, and RT communication control. The structure of ARPM is shown in Figure 18.

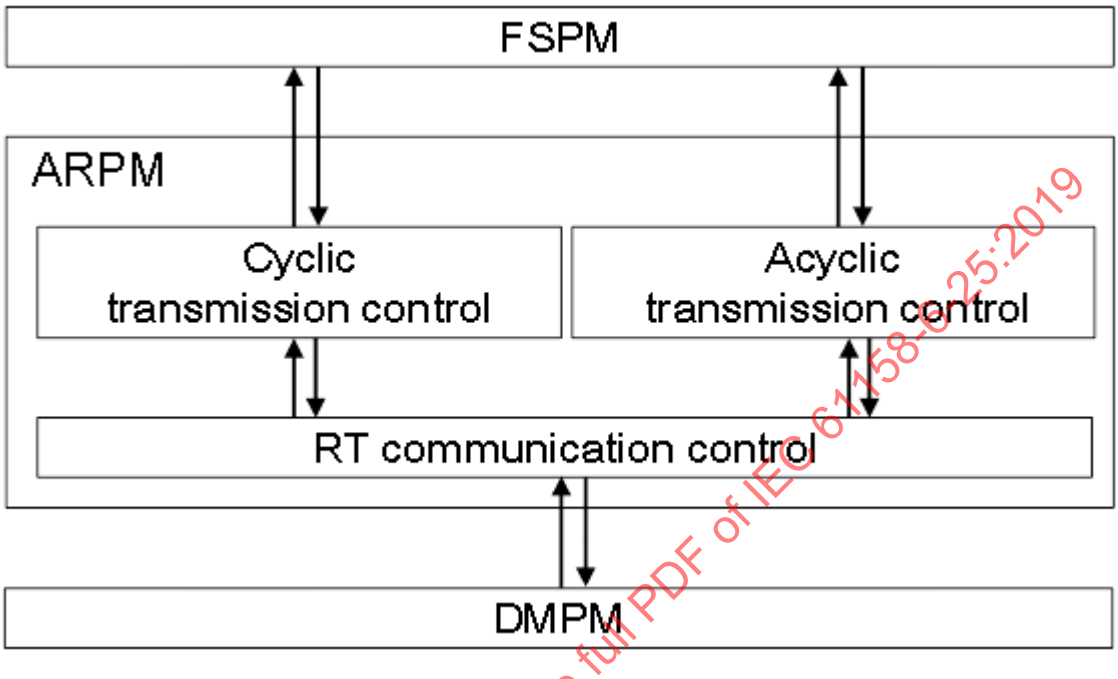


Figure 18 – Structure of ARPM type N

8.2.2 General control

8.2.2.1 Duplex LANs control

Type N protocol provides a function to transmit messages using duplex LANs in order to acquire the high reliability of transmission. In this case, this function does not affect the user.

1) Multicast communication

   Multicast communication method uses both LANs of duplex LANs.

   a) Multicast communication (Transmission)

      This method transmits the same message into both LANs. In this case, the same header, same UDP port number and different destination IP address are used.

   b) Multicast communication (Reception)

   This communication method receives messages from both LANs of duplex LANs. On receiving a message, the checking procedure for a receiving message is executed by means of using a message sequence number (hd_seq) within a NX header. Duplicated messages are detected and discarded.

2) Peer to Peer communication

   Peer to Peer communication method treats duplex LANs as an active LAN and a standby LAN. Each node establishes a TCP connection in both an active LAN and a standby LAN, respectively. This communication method uses a TCP connection of only an active LAN. On detecting a fault of an active LAN, a detecting node disconnects the TCP connection of the active LAN, and changes from a standby LAN into a new active LAN.

   a) Peer to Peer communication (Transmission)

      Each node transmits packets (messages) into a TCP connection of an active LAN.

   b) Peer to Peer communication (Reception)

Each node receives packets (messages) from a TCP connection of an active LAN. On receiving a packet, the checking procedure for a receiving packet is executed by means of using a packet sequence number (hd_pseq) within a NX header.

Duplicated packets or lost packets are detected and discarded. On receiving packets with an inconsistent packet sequence number, a detecting node disconnects the TCP connection and establishes a new TCP connection. The checking procedure for a receiving message is also executed by means of using a message sequence number (hd_seq) within a NX header.

8.2.2.2 Node monitoring control

Each node transmits periodically alive-messages into data fields in order to monitor the state (“alive” or “dead”) of nodes.

1) Range of transmitting data fields

   A node transmits an alive-message into all data fields corresponding to connected networks. In the duplex LAN environment, a node transmits the same alive-message into each LAN, respectively.

2) Interval of transmitting an alive-message

   A node transmits immediately an alive-message after becoming online status. Then, a node transmits an alive-message according to a predefined transmitting interval time in each data field.

3) Monitoring node status

   On starting of monitoring a data field, nodes status of all nodes that belong to the data field are recognized as "dead" state. When a monitoring node receives an alive-message from a monitored node, a node state is changed from “dead” state to “alive” state. If a monitoring node did not receive an alive-message from a monitored node within the time specified by the al_tm_out, the monitoring node recognizes that a monitored node is dead. When a type of a receiving alive-message (al_mode) is a “shutdown” or “maintenance”, a monitoring node recognizes a monitored node with “dead” state by means of “shutdown” operation or “maintenance” operation.

NOTE Even if a monitored node’s state is alive, faults of network-paths may intercept an alive-message from a monitored node. Therefore, a monitoring node can improve the recognition accuracy of the monitored node’s state by means of detecting a reception state of the alive-message from the plural network-paths or detecting a reception state of the alive-message from plural nodes.

8.2.2.3 Multi layers traffic control

Type N protocol provides the multi layers traffic control function.

Each node shall control the traffic of message (data) in multi layers as follows:

1) Traffic Bandwidth Allocation

   Allocating the network’s bandwidth, and controlling network traffic with transmission-rate control,

2) Message Priority Control

   Controlling a priority of the send/receive-message according to the priority with max 7-level. Here, cyclic transmission’s priority is higher than acyclic transmission’s priority,

3) Traffic Filtering Control

   Excluding the worse influence for send-traffic by filtering much receive-traffic,

4) Network traffic control

   This control excludes the worse influence from other network traffic.

   a) Dividing a broadcast domain (VLAN);

   b) Allocating the network communications to physically distinct LANs;

   c) Priority control by network switches（ToS、VLAN-tag).

System design is important regarding the total network traffic.

8.2.3 Cyclic transmission control

8.2.3.1 Primitive definitions

Figure 19 shows Primitives of Cyclic transmission control. The FSPM issues Write_cyclicdata.req and Ctl_cyclic.req to the Cyclic transmission control. The Cyclic transmission control issues Write_cyclicdata.ind to the FSPM.

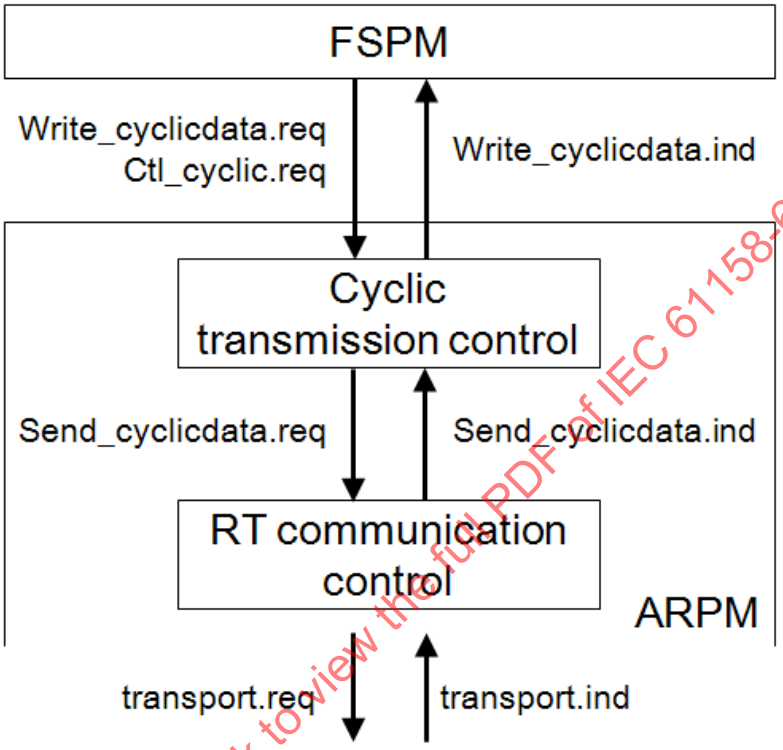


Figure 19 – Primitives of Cyclic transmission control

8.2.3.2 Cyclic transmission control state table

Details of Cyclic transmission control state machine are shown in Table 56.

Table 56 – Cyclic transmission control state table

#	Current state	Event /condition => actions	Next state
1	Any state	Initial => Create_Cyclic_Control_Information (Dfn , Tmid , Info)	READY
2	READY	Ctl_cyclic.req {TargetDfn ,TargetMgn, Tmid , CtlCmd } / (CtlCmd == START) && (Check_CyclicEnvironment == True) => Open_MCG_port (TargetDfn) Start_Timer(Cyclic_interval_timer)	ACTIVE
3	ACTIVE	Ctl_cyclic.req {TargetDfn , TargetMgn, Tmid , CtlCmd } / (CtlCmd == STOP) && (Check_CyclicEnvironment == True) => Stop_Timer(Cyclic_interval_timer)	READY
4	ACTIVE	Write_cyclicdata.req{ DstDfn , DstMgn , Tmid , Pri, Offset-address , Size , CyclicData } / => Update_a_CyclicDataMemory (DstDfn , DstMgn ,Tmid , Offset-address , Size , CyclicData)	ACTIVE

5	ACTIVE	/ Expire_Timer (Cyclic_interval_timer) == True
		=>
		Adjust_CyclicIntervalTimerInfo
		Select_CyclicData(Data, DataLength)
		Create_CyclicData-PDU (DstDfn, DstMgn, Tcd, Pri, CtlType, Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_cyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

6	ACTIVE	/ Expire_Timer (Cyclic_interval_timer , DstDfn , DstMgn) == True && NodeMode == Test-mode => (Nop)	ACTIVE

#	Current state	Event /condition => actions	Next state

7	ACTIVE	Send_cyclicdata.ind { CyclicData-PDU }
		=>
		/* Assembly processing */
		Assembly_c_Processing（CyclicData-PDU, Cdata, CdataLength, Diagnosis）
		IF (Diagnosis == OK_1)
		THEN {
		BlockInfo = Get_BlockInfo (Cdata, CdataLength)
		Update_b_CyclicDataMemory (DstDfn , DstMgn ,Tmid , BlockInfo)
		Write_cyclicdata.ind
		}
		ELSE IF ((Diagnosis == OK_2) || (Diagnosis == NG)
		(Nop)
			ACTIVE

8	ACTIVE	Read_cyclicdata.req { DstDfn , DstMgn ,Tmid , Offset-address , Size , CyclicData } => Read_cyclicdata.cnf { DstDfn , DstMgn ,Tmid , Offset-address , Size , CyclicData }	ACTIVE

8.2.3.3 Functions

All the functions used by the Cyclic transmission control are summarized in Table 57.

Table 57 – Cyclic transmission control functions

Name	Input	Output	Description
Adjust_CyclicIntervalTimer Info	CyclicIntervalTimerInformation	CyclicIntervalTimerInformation	This function updates cyclic-transmitting information (transmitting timing, transmitting counter etc)
Assembly_c_Processing	PDUdata	Cdata, CdataLength, Diagnosis	This function reassembles the received fragmentation packets (PDUdata). This function reassembles the received fragmentation packets (PDUdata). After completing a reassembling processing, a reassembled data is set to a Cdata. Here, a falAr-Header is removed from a Cdata. (s1) If a received packet (PDUdata) is not a fragmented packet, finish a processing. →(s 6) (s2) If a received packet is a top packet of a fragmented packet, register the packet. Start a monitoring timer of reassembling. →(s 7) (s3) If a received packet is not a top packet of a fragmented packet, register the packet and execute a reassembling processing. →(s 4) (s4) If a reassembling processing completed, stop a monitoring timer of reassembling. →(s 6)If a reassembling processing did not complete yet, continue an ongoing processing state.→(s 7) (s5) If a reassembling processing did not complete within a monitoring time of reassembling, discard all packets registered in a reassembling queue and return an uncompleted error. →(s8) (s6) Return the information below. Cdata = Reassembling completed data Cdata Length = Length of Cdata Diagnosis = OK_1 (Completed) (s7) Return the information below. Diagnosis = OK_2 (Continuing reassembling) (s8) Return the information below. Cdata =Current reassembling data Cdata Length = Length of Cdata Diagnosis = NG ErrorTransactionCode = Not completed
Check_CyclicEnvironment	-	True / False	This function checks whether or not the environment of cyclic transmission is valid. Returns True if so, and False if not.

Name	Input	Output	Description
Create_CyclicData-PDU	DstDfn, DstMgn, Tcd, Pri, CtlType, Data, DataLength	FragmentedPduList	This function creates a Create_CyclicData-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_Cyclic_Control_Information	Dfn , Tmid , Info	CyclicEnvironment	This function Initializes information of cyclic transfer memory. Obviously, it defines cyclic transfer intervals, block numbers, number of blocks, multicast group numbers of which consisting transfer memory.
Expire_Timer	cyclic_interval_timer		This function returns True when a cyclic_interval_timer has expired.
Get_BlockInfo	Cdata, CdataLength	Blockinfo	This function gets information of transfer memory consisting of DstDfn, DstMgn, Tmid, transfer block number, number of transfer blocks and cyclic transfer data. It sets above information to Blockinfo variable.
Open_MCG_port	Dfn	-	This function enables a multicast communication port belonging to Dfn to communicate. In the case, to initialize a sequence number of each multicast group. (S_V_SEQ = current time, S_SEQ =0, R_V_SEQ = 0)
RetrieveList	FragmentedPduList	targetPDU	This function gets the fragmented PDU from FragmentedPduList and sets it to targetPDU variable. When FragmentedPduList becomes empty, sets Null to FragmentedPduList.
Select_CyclicData	-	Data, DataLength	This function extracts contents of transmitting memory from cyclic transfer memory. It sets the contents and size of the contents to Data and DataLength, respectively. Here, structure of Data shall have a structure of CyclicData-PDU which includes a tmid, a blockNumber and a blockCount field.
Start_Timer	Cyclic_interval_timer	-	This function starts a cyclic_interval_timer.
Stop_Timer	Cyclic_interval_timer	-	This function stops a cyclic_interval_timer.
Update_a_CyclicDataMemory	DstDfn , DstMgn , Tmid , Offset-address , Size , CyclicData	CyclicMemoryArea	This function updates the contents of cyclic transfer memory which is specified by DstDfn, DstMgn and Tmid. Obviously, it updates memory area with the contents which is specified by both the offset-address (Offset-address) from a top-address of cyclic transfer memory (CyclicData) and the Size.

Name	Input	Output	Description
Update_b_CyclicDataMemory	Blockinfo	CyclicMemoryArea	This function updates the contents of cyclic transfer memory by using information of cyclic transfer memory (BlockInfo), which is get from CyclicData-PDU.

8.2.3.4 Variables

All the variables used by the cyclic transmission control are summarized in Table 58.

Table 58 – Cyclic transmission control variables

Name	Description
BlockInfo	Information of cyclic transfer memory creating from a reception PDU
CtlCmd	Control command for cyclic transmission
CtlType	Control information within PDU
CyclicData , Data	Data of cyclic transfer memory
DataLength	Length of Data
Dfn	Data field number
DstDfn	Destination data field number
DstMgn	Destination multicast group number
FragmentedPduList	List of PDU fragmenting data
Info	Information of configuring cyclic transfer memory
NodeMode	Operation mode of a node (Online-mode/ Test-mode)
Offset-address	Offset-address from a top address of area
Size	Length of CyclicData
Pri	transmission priority
TargetDfn	Target data field number
TargetMgn	Target multicast group number
Tcd	Transaction code
Tmid	Identifier of cyclic transfer memory
targetPDU	Transmitting PDU to network

8.2.4 Acyclic transmission control

8.2.4.1 Primitive definitions

Figure 20 shows the primitives of acyclic transmission control. The FSPM issues Transmit_acyclicdata1.req, Transmit_acyclicdata2.req and Ctl_acyclic.req to the acyclic transmission control. The acyclic transmission control issues Transmit_acyclicdata1.ind, transmit_acyclicdata2.ind to the FSPM.

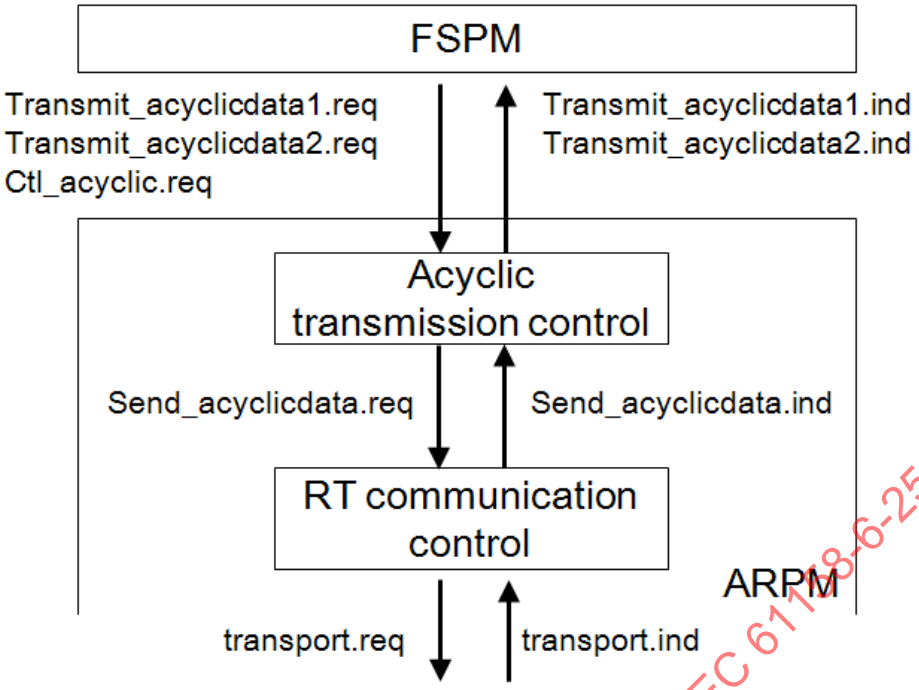


Figure 20 – Primitives of acyclic transmission control

8.2.4.2 Acyclic transmission control state table

Details of Acyclic transmission control state machine are shown in Table 59.

Table 59 – Acyclic transmission control state table

#	Current state	Event /condition => actions	Next state
1	Any state	Initial => Create_Acyclic_Control_Information (Dfn , Info)	READY
2	READY	Ctl_acyclic.req {TargetDfn , CtlCmd } / (CtlCmd == ACTIVATE) && (Check_AyclicEnvironment == True) => Open_MCG_port (TargetDfn) Open_TCP_connection (TargetDfn) Set_All_ TargetNodeAliveMode(TargetDfn, “Dead”)	ACTIVE
3	ACTIVE	Ctl_acyclic.req {TargetDfn , CtlCmd } / (CtlCmd == INACTIVATE) && (Check_AcyclicEnvironment == True) => Set_All_ TargetNodeAliveMode(TargetDfn, “Dead”)	READY

#	Current state	Event /condition => actions	Next state

4	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstMgn , Tcd, Pri, CtlType , Inq-id, Node-list , Data , DataLength}
		/ CtlType == MLT
		=>
		Create_ MulticastData-PDU (DstDfn, DstMgn, Tcd, Pri, CtlType, Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

5	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstLnn
		, Tcd, Pri, CtlType , Inq-id, Node-list , Data, DataLength }
		/ CtlType == ONE
		=>
		Create_ PtoPData-PDU (DstDfn, DstLnn, Tcd, Pri, CtlType, Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstLnn
		, targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

6	ACTIVE	Transmit_acyclicdata1.req {DstDfn , DstLnn , Tcd, Pri, CtlType , Inq-id, Node-list , Data, DataLength }
		/ CtlType == INQ & ONE
		=>
		Create_ Inq-PDU (DstDfn, DstLnn, Tcd, Pri, , CtlType, Inq-id, Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstLnn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

7	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstMgn , Tcd, Pri, CtlType , Inq-id, Node-list , Data, DataLength }
		/ CtlType == INQ & MLT
		=>
		Create_ Inq-PDU (DstDfn, DstMgn, Tcd, Pri, CtlType, Inq-id, Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

8	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstMgn , Tcd, Pri, CtlType , Inq-id, Node-list , Data, DataLength }
		/ CtlType == NIQ
		=>
		Create_ Ninq-PDU (DstDfn, DstMgn, Tcd, Pri, CtlType, Inq-id, Node-list , Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			WAIT_Reply

9	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstLnn , Tcd, Pri, CtlType , Inq-id, Node-list , Data ,DataLength }
		/ CtlType == RPL
		=>
		Create_ Reply-PDU (DstDfn, DstLnn, Tcd, Pri, CtlType, Inq-id, Node-list , Data, DataLength, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstLnn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

10	ACTIVE	Transmit_acyclicdata2.req { DstDfn , DstMgn , AliveMode, Task-info1, Task-info2 }
		/ NodeIPversion == IPv4
		=>
		NodeStatus = AliveMode
		Create_ Aliveinfo -PDU (DstDfn , DstMgn , AliveMode, Task-info1, Task-info2, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
		Start_Timer(Alive_send_interval_timer, DstDfn , DstMgn)
			WAIT_Alive

11	ACTIVE	Transmit_acyclicdata2.req { DstDfn , DstMgn , AliveMode, Task-info1, Task-info2 }
		/ NodeIPversion == IPv6
		=>
		NodeStatus = AliveMode
		Create_ Aliveinfo6 -PDU (DstDfn , DstMgn , AliveMode, Task-info1, Task-info2, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
		Start_Timer(Alive_send_interval_timer, DstDfn , DstMgn)
			WAIT_Alive

#	Current state	Event /condition => actions	Next state

12	WAIT_Alive	/ Expire_Timer(Alive_send_interval_timer, DstDfn , DstMgn , AliveMode, Task-info1, Task-info2,) &&
		NodeIPversion == IPv4
		=>
		Create_ Aliveinfo -PDU (DstDfn , DstMgn , AliveMode, Task-info1, Task-info2, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
		Start_Timer(Alive_send_interval_timer, DstDfn , DstMgn)
			WAIT_Alive

13	WAIT_Alive	/ Expire_Timer(Alive_send_interval_timer, DstDfn , DstMgn , AliveMode, Task-info1, Task-info2,) &&
		NodeIPversion == IPv6
		=>
		Create_ Aliveinfo6 -PDU (DstDfn , DstMgn , AliveMode, Task-info1, Task-info2, FragmentedPduList)
		targetPDU = RetrieveList(FragmentedPduList)
		While (targetPDU != Null) {
		Send_acyclicdata.req(DstDfn , DstMgn , targetPDU)
		targetPDU = RetrieveList(FragmentedPduList)
		}
		Start_Timer(Alive_send_interval_timer, DstDfn , DstMgn
			WAIT_Alive

14	ACTIVE	/ Expire_Timer(Alive_monitor_interval_timer , TargetDfn, TargetNode) => MonitorCounter = MonitorCounter – 1 IF (MonitorCounter = 0) THEN /* dead */ TargetNodeAliveMode = “Dead” Change_LAN(primaryLAN) Transmit_acyclicdata2.ind(TargetDfn, TargetNode, , TargetNodeAliveMode)	ACTIVE

#	Current state	Event /condition => actions	Next state

15	ACTIVE	Send_acyclicdata.ind{ DstDfn, DstMgn, PDUdata }
		/
		(hd_tcd != Alive_TCD) && (hd_m_ctl != ONE)
		=>
		/* Assembly processing */
		Assembly_a_Processing（PDUdata, Cdata, CdataLength, Diagnosis, ErrorTransactionCode ）
		IF (Diagnosis == OK_1) || (Diagnosis == NG)
		THEN {
		Message_SEQ_Control (PDUdata, result-m)
		IF (result-m == 1) /* normal message reception */
		THEN {
		Set_Transaction(ErrorTransactionCode)
		IF(hd_m_ctl == NIQ)
		THEN
		Transmit_acyclicdata1.ind{ hd_da.Dfn, hd_da.Mgn, hd_tcd, hd_pri, hd_m_ctl, hd_ inqid, node-list, Cdata + NodeListSize, CdataLength – NodeListSize }
		ELSE
		Transmit_acyclicdata1.ind{ hd_da.Dfn, hd_da.Mgn, hd_tcd, hd_pri, hd_m_ctl, hd_ inqid, , Cdata, CdataLength }
		}
		}
		ELSE IF (Diagnosis == OK_2)
		(Nop)
			ACTIVE

#	Current state	Event /condition => actions	Next state

16	WAIT_Reply	Send_acyclicdata.ind{ DstDfn, DstLnn, PDUdata }
		/
		(hd_tcd != Alive_TCD) && (hd_m_ctl == ONE)
		=>
		
		/* Assembly processing */
		Assembly_a_Processing（ PDUdata, Cdata, CdataLength, Diagnosis, ErrorTransactionCode ）
		IF (Diagnosis == OK_1) || (Diagnosis == NG)
		THEN {
		Message_SEQ_Control (PDUdata, result-m)
		IF (result-m == 1) /* normal message reception */
		THEN {
		
		Set_Transaction(ErrorTransactionCode)
		
		Transmit_acyclicdata1.ind{ hd_da.Dfn, hd_da.Lnn, hd_tcd, hd_pri, hd_m_ctl, hd_ inqid, , Cdata, CdataLength }
		}
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

17	ACTIVE	Send_acyclicdata.ind { DstDfn, DstMgn, PDUdata }
		/
		(hd_tcd == Alive_TCD) && (al-mode == Alive)
		=>
		
		/* Assembly processing */
		Assembly_a_Processing（ PDUdata, Cdata, CdataLength, Diagnosis, ErrorTransactionCode ）
		IF (Diagnosis == OK_1) || (Diagnosis == NG)
		THEN {
		Message_SEQ_Control (PDUdata, result-m)
		IF (result-m == 1) /* normal message reception */
		THEN {
		
		Update_AliveInfo (Cdata, NodeAliveInfo)
		Update_ExtensionInfo(Cdata, NodeExtensionInfo)
		node = hd_sa.Lnn
		Dfn = hd_sa.Dfn
		MonitorCounter = MaxMonitorCounter
		IF (TargetNodeAliveMode != “Alive”) {
		TargetNodeAliveMode = “Alive”
		Start_Timer(Alive_monitor_interval_timer, Dfn, node)
		
		Transmit_acyclicdata2.ind (hd_da.Dfn, hd_da.Mgn , al-mode)
		
		}
		}
		
		ELSE IF (Diagnosis == OK_2)
		(Nop)
			ACTIVE

#	Current state	Event /condition => actions	Next state

18	ACTIVE	Send_acyclicdata.ind { DstDfn, DstMgn, PDUdata }
		/
		(hd_tcd == Alive_TCD) && (al-mode == Shutdown)
		=>
		
		/* Assembly processing */
		Assembly_a_Processing（ PDUdata, Cdata, CdataLength, Diagnosis, ErrorTransactionCode ）
		IF (Diagnosis == OK_1) || (Diagnosis == NG)
		THEN {
		Message_SEQ_Control (PDUdata, result-m)
		IF (result-m == 1) /* normal message reception */
		THEN {
		
		Update_AliveInfo (Cdata, NodeAliveInfo)
		Update_ExtensionInfo(Cdata, NodeExtensionInfo)
		node = hd_sa.Lnn
		Dfn = hd_sa.Dfn
		TargetNodeAliveMode = “Shutdown”
		Stop_Timer(Alive_monitor_interval_timer, Dfn, node)
		
		Transmit_acyclicdata2.ind (hd_da.Dfn, hd_da.Mgn , al-mode)
		}
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

19	ACTIVE	Send_acyclicdata.ind { DstDfn, DstMgn, PDUdata }
		/
		(hd_tcd == Alive_TCD) &&
		(al-mode == Maintenance)
		=>
		
		/* Assembly processing */
		Assembly_a_Processing （ PDUdata, Cdata, CdataLength, Diagnosis, ErrorTransactionCode ）
		IF (Diagnosis == OK_1) || (Diagnosis == NG)
		THEN{
		Message_SEQ_Control (PDUdata, result-m)
		IF (result-m == 1) /* normal message reception */
		THEN {
		
		Update_AliveInfo (Cdata, NodeAliveInfo)
		Update_ExtensionInfo(Cdata, NodeExtensionInfo)
		node = hd_sa.Lnn
		Dfn = hd_sa.Dfn
		TargetNodeAliveMode = “Maintenance”
		Stop_Timer(Alive_monitor_interval_timer, Dfn, node)
		
		Transmit_acyclicdata2.ind (hd_da.Dfn, hd_da.Mgn , al-mode)
		}
		}
			ACTIVE

20	ACTIVE	Transmit_acyclicdata1.req { DstDfn , DstMgn , Tcd, Pri, CtlType , Inq-id, Node-list , Data , DataLength } / NodeMode == Test-mode => (Nop)	ACTIVE

8.2.4.3 Functions

All the functions used by the Acyclic transmission control are summarized in Table 60.

Table 60 – Acyclic transmission control functions

Function name	Input	Output	Description
Assembly_a_Processing	PDUdata	Cdata, CdataLength, Diagnosis, ErrorTransactionCode	This function reassembles the received fragmentation packets (PDUdata). After completing a reassembling processing, a reassembled data is set to a Cdata. Here, a falAr-Header is removed from a Cdata. (s1) If a received packet (PDUdata) is not a fragmented packet, finish a processing. →(s 6) (s2) If a received packet is a top packet of a fragmented packet, register the packet. Start a monitoring timer of reassembling. →(s 7) (s3) If a received packet is not a top packet of a fragmented packet, register the packet and execute a reassembling processing. →(s 4) (s4) If a reassembling processing completed, stop a monitoring timer of reassembling. →(s 6) If a reassembling processing did not complete yet, continue an ongoing processing state.→(s 7) (s5) If a reassembling processing did not complete within a monitoring time of reassembling, discard all packets registered in a reassembling queue and return an uncompleted error. →(s8) (s6) Return the information below. Cdata = Reassembling completed data Cdata Length = Length of Cdata Diagnosis = OK_1 (Completed) ErrorTransactionCode = Completed (s7) Return the information below. Diagnosis = OK_2 (Continuing reassembling) (s8) Return the information below. Cdata =Current reassembling data Cdata Length = Length of Cdata Diagnosis = NG ErrorTransactionCode = Not completed
Change_LAN(primaryLAN)			This function changes a primary LAN connecting to a target node in duplex LANs environment. (secondary LAN → primary LAN, primary LAN → secondary LAN)
Check_AyclicEnvironment	-	True / False	This function checks the effectiveness of acyclic communication environment. Enabled: True , Disabled: False
Create_ Aliveinfo -PDU	DstDfn , DstMgn , AliveMode, Task-info1, Task-info2	FragmentedPduList	This function creates an Aliveinfo-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.

Function name	Input	Output	Description
Create_ Aliveinfo6 -PDU	DstDfn , DstMgn , AliveMode, Task-info1, Task-info2	FragmentedPduList	This function creates an Aliveinfo6-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_ Inq-PDU	DstDfn, DstMgn(DstLnn), Tcd, Pri, CtlType, Inq-id, Data, DataLength	FragmentedPduList	This function creates an Inq-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16 it registers a fragmented PDU into a list of FragmentedPduList.
Create_ MulticastData-PDU	DstDfn, DstMgn, Tcd, Pri, CtlType, Data, DataLength	FragmentedPduList	This function creates a MulticastData-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_ Ninq-PDU	DstDfn, DstMgn, Tcd, Pri, CtlType, Inq-id, Node-list , Data, DataLength	FragmentedPduList	This function creates a Ninq-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_ PtoPData-PDU	DstDfn, DstLnn, Tcd, Pri, CtlType, Data, DataLength	FragmentedPduList	This function creates a PtoPData-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_ Reply-PDU	DstDfn, DstLnn, Tcd, Pri, CtlType, Inq-id, Node-list , Data, DataLength	FragmentedPduList	This function creates a Reply-PDU from a specified message (Data). Here, it sets a state of message mode corresponding to operation mode of the node (NodeMode) into hd_mode field. After processing a fragmentation according to 5.3.2.16, it registers a fragmented PDU into a list of FragmentedPduList.
Create_Acyclic_Control_Information	Dfn , Info	AcyclicEnvironment	This function initializes management information of a specified data field (Dfn).
Expire_Timer	-	Alive_send_interval_timer, DstDfn , DstMgn , AliveMode, Task-info1, Task-info2,	This function returns True when Alive_send_interval_timer corresponding to both a TargetDfn and a DstDfn has expired.
Expire_Timer	-	Alive_monitor_interval_timer , TargetDfn, TargetNode	This function returns True when Alive_monitor_interval_timer corresponding to both a TargetDfn and a TargetNode has expired.

Function name	Input	Output	Description
Message_SEQ_Control	PDU	result-m	This function compares a R_V_SEQ with R_SEQ by using the information of hd_sa, hd_m_ctl, hd_v_seq, and hd-seq. A R_V_SEQ and a R_SEQ are updated according to the result of comparison, (Ref. 5.3.2.6) In the case of detecting duplicated message or lost messages, a received message is discarded. ·result-m = 1 （normal message reception） ·result-m = 2 （duplicate message reception） ·result-m = 3 （lost messages detection）
Open_MCG_port	Dfn	-	This function enables multicast communication ports belonging to a specified Dfn to communicate. Here, it initializes a message sequence number corresponding to each multicast group. (S_V_SEQ = the current date and time, S_SEQ =0, R_V_SEQ = 0, R_SEQ = 0)
Open_TCP_connection	Dfn	-	This function establishes a TCP connection between nodes belonging to a specified Dfn, enabling nodes to communicate. Here, it initializes a message sequence number corresponding to each TCP connection. (S_V_SEQ = the current date and time S_PSEQ =0, R_V_SEQ = 0, R_PSEQ = 0)
RetrieveList	FragmentedPduList	targetPDU	This function extracts a fragmented PDU in registration order from a list of FragmentedPduList, and sets it to a targetPDU. In the case of detecting no PDU to extract, it sets Null into a TargetPDU
Set_All_ TargetNodeAliveMode	TargetDfn, “Dead”	TargetNodeAliveMode	This function sets the dead-state into state of all nodes (TargetNodeAliveMode) belonging a specified data field.
Set_Transaction	ErrorTransactionCode	-	This function sets transaction information.
Start_Timer	Alive_send_interval_timer, DstDfn , DstMgn	-	This function starts an Alive_send_interval_timer which is used to send an alive-message corresponding to both a DstDfn and a DstMgn.
Start_Timer	Alive_monitor_interval_timer, Dfn, node	-	This function starts an Alive_monitor_interval_timer which is used to monitor a reception of an alive-message corresponding to a monitored node (Dfn, node)
Stop_Timer	Alive_monitor_interval_timer, Dfn, node	-	This function stops an Alive_monitor_interval_timer which is used to monitor a reception of an alive-message corresponding to a monitored node (Dfn, node)
Update_AliveInfo	Cdata	NodeAliveInfo	This function updates the alive-information which corresponds to the monitored node (NodeAliveInfo) by using an alive-information (AliveInfo) within a Cdata.
Update_ExtensionInfo	Cdata	NodeExtensionInfo	This function updates the extension-information which corresponds to the monitored node (NodeExtensionInfo) by using an extension-information (NodeExtensionInfo) within a Cdata.

8.2.4.4 Variables

All the variables used by the Acyclic transmission control are summarized in Table 61.

Table 61 – Acyclic transmission control variables

Name	Description
AliveMode	Mode of an alive-message (Alive、Shutdown、Maintenance)
Alive_TCD	TCD of an alive-message
Alive_monitor_interval_timer	Monitoring timer for a alive-message
Alive_send_interval_timer	Interval timer of sending an alive-message
Cdata	Data which is reassembled from a received PDU
CdataLength	Length of Cdata
CtlCmd	Control command of an acyclic communication. (ACTIVATE / INACTIVATE)
CtlType	Type of a packet
Data	Message data
DataLength	Length of Data
Dfn	Data field number
Diagnosis,	Diagnosis of the function. (OK、NG、OK_1, OK_2)
DstDfn	Destination data field number
DstMgn	Destination multicast group number
ErrorTransactionCode	Error transaction code
FragmentedPduList	List of PDUs created by fragmenting a message
Inq-id	Identifier of an inquiry message
MaxMonitorCounter	Monitoring counter of receiving for an alive-message (initial value)
MonitorCounter	Monitoring counter of receiving for an alive-message
Node-list	List of the nodes which are requested to reply in an inquiry/ response communication
NodeAliveInfo	Alive-information of the target node
NodeExtensionInfo	Alive-extended information of the target node
NodeIPversion	IP address version of the target node
NodeListSize	Size of a NodeList
NodeMode	Operation mode of a node. (Online-mode, Test-mode)
NodeStatus	State of own node (State: Alive or Shutdown or Maintenance)
Pri	Message priority
TCD	Transaction code
TargetDfn	Data field number of a target node
TargetNode	Node number of a target node
TargetNodeAliveMode	State of a target node (State: Alive or Shutdown or Maintenance)
Task-info1	Information of a task (type1)
Task-info2	Information of a task (type2)
hd_da.Dfn	Source data field number which is set in PDU
hd_da.Mgn	Source multicast group number which is set in PDU
hd_sa.Dfn	Source data field number which is set in PDU
hd_sa.Lnn	Source node number which is set in PDU
node	Node number
primaryLAN	Primary LAN in duplexed communication paths system
result-m	Result of a function
targetPDU	Sending node

8.2.5 RT communication control

8.2.5.1 RT communication control state table

Table 62 shows the state table of the RT communication control.

Table 62 – RT communication control state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Send_cyclicdata.req{ DstDfn, DstMgn, PDUdata } / DuplicateLAN_Topology (DstDfn,DstMgn) == True /* Duplicate LAN , Cyclic Communication */ => Priority_Queing_control(DstDfn, DstMgn, PDUdata, trans-mode) Get_transport_identifier(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2) Transport_write_data (transport_IDs-1 , PDUdata) Transport_write_data (transport_IDs-2 , PDUdaat)	ACTIVE
2	ACTIVE	Send_cyclicdata.req{ DstDfn, DstMgn, PDUdata } / DuplicateLAN_Topology (DstDfn,DstMgn) == False /* Single LAN , Cyclic Communication */ => Priority_Queing_control(DstDfn, DstMgn, PDUdata, trans-mode) Get_transport_identifier(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2) Transport_write_data (transport_IDs-1 , PDUdata)	ACTIVE

3	ACTIVE	Send_acyclicdata.req{ DstDfn, DstMgn, PDUdata }
		/ DuplicateLAN_Topology (DstDfn,DstMgn) == True
		&&
		Re-transmission_control (DstDfn,DstMgn) == True
		/* Duplicate LAN , Acyclic & Re-trans-mode Communication */
		=>
		
		PDUdata.hd_pseq = Update_S_PSEQ(DstDfn, DstMgn)
		Enque_Multicast_Retransmit_Que(DstDfn,DstMgn, PDUdata)
		Priority_Queing_control(DstDfn,DstMgn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier(DstDfn,DstMgn, transport_IDs-1, transport_IDs-2)
		Transport_write_data (transport_IDs-1 , PDUdata)
		Transport_write_data (transport_IDs-2 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn,DstMgn)
			ACTIVE

#	Current state	Event /condition => actions	Next state

4	ACTIVE	Send_acyclicdata.req{ DstDfn, DstMgn, PDUdata }
		/ DuplicateLAN_Topology (DstDfn, DstMgn) == False
		&&
		Re-transmission_control(DstDfn, DstMgn) == True
		/* Single LAN , Acyclic & Re-trans-mode Communication *
		=>
		
		PDUdata.hd_pseq = Update_S_PSEQ(DstDfn, DstMgn)
		Enque_Multicast_Retransmit_Que(DstDfn, DstMgn, PDUdata)
		Priority_Queing_control(DstDfn,DstMgn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier (DstDfn, DstMgn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn, DstMgn)
			ACTIVE

5	ACTIVE	Send_acyclicdata.req{ DstDfn, DstMgn, PDUdata }
		/ DuplicateLAN_Topology(DstDfn, DstMgn) == True
		&&
		Re-transmission_control(DstDfn, DstMgn) == False
		/* Duplicate LAN , Acyclic & No-Re-trans-mode & Multicast Communication */
		
		=>
		
		Priority_Queing_control(DstDfn,DstMgn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1 , PDUdata)
		Transport_write_data(transport_IDs-2 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn,DstMgn)
			ACTIVE

#	Current state	Event /condition => actions	Next state

6	ACTIVE	Send_acyclicdata.req{ DstDfn, DstLnn, PDUdata }
		/ DuplicateLAN_Topology(DstDfn, DstLnn) == True
		
		/* Duplicate LAN , Acyclic & No-Re-trans-mode & PtoP Communication */
		
		=>
		
		PDUdata.hd_pseq = Update_S_PSEQ(DstDfn, DstLnn)
		Enque_PtoP_Retransmit_Que(DstDfn,DstLnn, PDUdata)
		Priority_Queing_control(DstDfn, DstLnn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn, DstMgn)
			ACTIVE

7	ACTIVE	Send_acyclicdata.req{ DstDfn, DstMgn, PDUdata }
		/ DuplicateLAN_Topology(DstDfn, DstMgn) == False
		&&
		Re-transmission_control (DstDfn, DstMgn) == False
		/* Single LAN , Acyclic & No-Re-trans-mode Communication */
		=>
		
		Priority_Queing_control(DstDfn,DstMgn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn, DstMgn)
			ACTIVE

8	ACTIVE	Send_acyclicdata.req{ DstDfn, DstLnn, PDUdata }
		/ DuplicateLAN_Topology(DstDfn, DstLnn) == False
		
		/* Single LAN , Acyclic & No-Re-trans-mode & PtoP Communication */
		=>
		
		Priority_Queing_control(DstDfn,DstLnn, PDUdata, trans-mode)
		IF(trans-mode == immediate)
		THEN
		Get_transport_identifier(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1 , PDUdata)
		ELSE
		Start_Timer(Rate_control_interval, DstDfn,DstLnn)
			ACTIVE

#	Current state	Event /condition => actions	Next state

9	ACTIVE	/ Expire_Timer(Rate_control_interval, DstDfn,DstMgn)
		&&
		DuplicateLAN_Topology(DstDfn, DstMgn) == True
		/* Duplicate LAN , Multicast Communication */
		
		=>
		trans-PDU = Select_transaction(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		
		While (trans-PDU != Null) {
		Transport_write_data(transport_IDs-1, trans-PDU)
		Transport_write_data(transport_IDs-2, trans-PDU)
		trans-PDU = Select_transaction(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		}
			ACTIVE

10	ACTIVE	/ Expire_Timer(Rate_control_interval, DstDfn, DstMgn)
		&&
		DuplicateLAN_Topology(DstDfn, DstMgn) == False
		
		/* Single LAN , Multicast Communication */
		=>
		
		trans-PDU = Select_transaction(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		
		While (trans-PDU != Null) {
		Transport_write_data (transport_IDs-1 , trans-PDU)
		trans-PDU = Select_transaction(DstDfn, DstMgn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

11	ACTIVE	/ Expire_Timer(Rate_control_interval, DstDfn, DstLnn)
		&&
		DuplicateLAN_Topology(DstDfn, DstLnn) == True
		/* Duplicate LAN , PtoP Communication */
		=>
		
		trans-PDU = Select_transaction(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		
		While (trans-PDU != Null) {
		IF (ACK_req_identifier == True)
		trans-PDU.hd_pkind = 1 /* ACK.req */
		Transport_write_data(transport_IDs-1 , trans-PDU)
		
		trans-PDU = Select_transaction(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		}
			ACTIVE

12	ACTIVE	/ Expire_Timer(Rate_control_interval, DstDfn, DstLnn)
		DuplicateLAN_Topology(DstDfn, DstMgn) == False
		/* Single LAN , PtoP Communication */
		=>
		
		trans-PDU = Select_transaction(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		
		While (trans-PDU != Null) {
		IF (ACK_req_identifier == True)
		trans-PDU.hd_pkind = 1 /* ACK.req */
		Transport_write_data (transport_IDs-1, trans-PDU)
		
		trans-PDU = Select_transaction(DstDfn, DstLnn, transport_IDs-1, transport_IDs-2, ACK_req_identifier)
		}
			ACTIVE

13	ACTIVE	/ Transport_read_data(receive_cyclic_port , PDU) == True
		=>
		
		Check_TestMessage(PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		THEN {
		Priority_QueingR_control(hd_da.Dfn, hd_da.Mgn, PDU)
		Send_cyclicdata.ind(hd_da. Dfn, hd_da.Mgn, PDU)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

14	ACTIVE	/ Transport_read_data(receive_acyclic_port_for_MulticastWithRetrans , PDU) == True
		&&
		(hd_m_ctl == MCR && hd_tcd == Retrans-TCD)
		/* Multicastdata with retransmission */
		=>
		
		Check_TestMessage (PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		THEN {
		Packet_SEQ_Control(PDU, result-p)
		
		IF (result-p == 1) /* normal packet reception */
		THEN {
		Priority_QueingR_control(hd_da.Dfn, hd_da.Mgn, PDU)
		
		Send_acyclicdata.ind(hd_da.Dfn, hd_da.Mgn, PDU)
		}
		ELSE IF (result-p == 3) /* request to retransmission */
		THEN {
		RequestPDU =Create_b_RetransEnq –PDU(PDU)
		Get_transport_identifier(RequestPDU.hd_da.Dfn, RequestPDU.hd_da.Mgn, transport_IDs-1, transport_IDs-2)
		
		IF (transport_IDs-2 != False)
		THEN
		Transport_write_data(transport_IDs-1 , RequestPDU)
		Transport_write_data(transport_IDs-2 , RequestPDU)
		ELSE
		Transport_write_data(transport_IDs-1 , RequestPDU)
		
		}
		ELSE IF (result-p == 4) /* request to disiconnect TCP connection */
		Close_TCP_connection (PDU.hd_sa)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

15	ACTIVE	/ Transport_read_data(receive_acyclic_port_for_MulticastWithRetransControl , PDU) == True
		&&
		(hd_m_ctl == MCR && hd_tcd == Retrans-TCD)
		&&
		(PDU == RetransEnq-PDU)
		/* Re-transmission –request PDU */
		
		Check_TestMessage (PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		
		THEN {
		
		Produce_ RetransPduList(PDU, RetransPduList)
		trans = Get_RetransPDU(RetransPduList, RetransPDU)
		
		While(trans) {
		Get_transport_identifier(RetransPDU.hd_da. Dfn, RetransPDU.hd_da. Mgn, transport_IDs-1, transport_IDs-2)
		
		IF (transport_IDs-2 != False)
		THEN
		Transport_write_data(transport_IDs-1 , RetransPDU)
		Transport_write_data(transport_IDs-2 , RetransPDU)
		ELSE
		Transport_write_data (transport_IDs-1 , RetransPDU)
		
		trans = Get_RetransPDU(RetransPduList, RetransPDU)
		}
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

16	ACTIVE	/ Transport_read_data(receive_acyclic_port_for_MulticastWithRetransControl , PDU) == True
		&&
		(hd_m_ctl == MCR && hd_tcd == Retrans-TCD)
		&&
		(PDU == RetransConfirm-PDU)
		/* Re-transmission-CofirmationRequest PDU */
		RequestPDU = Create_a_RetransEnq–PDU(PDU)
		IF(RequestPDU != Null)
		{Get_transport_identifier(RequestPDU.hd_da.Dfn, RequestPD.hd_da.Mgn, transport_IDs-1, transport_IDs-2)
		IF (transport_IDs-2 != False)
		THEN
		Transport_write_data(transport_IDs-1 , RequestPDU)
		Transport_write_data(transport_IDs-2 , RequestPDU)
		ELSE
		Transport_write_data(transport_IDs-1 , RequestPDU)
		}
			ACTIVE

17	ACTIVE	/ Transport_read_data(receive_acyclic_port_for_MulticastWithRetransControl , PDU) == True
		&&
		(hd_m_ctl == MCR && hd_tcd == Retrans-TCD)
		&&
		(PDU == RetransNak-PDU)
		/* Re-transmission–NAK PDU */
		=>
		Check_TestMessage(PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		THEN {
		(Nop)
		ErrorTransaction(LocalMatter)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

18	ACTIVE	/ Transport_read_data(receive_acyclic_port_for_multicast , PDU) == True
		&&
		(hd_m_ctl != MCR && hd_tcd != Retrans-TCD)
		/* No Re-transmision PDU */
		=>
		
		Check_TestMessage(PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		
		THEN {
		
		Priority_QueingR_control(hd_da.Dfn, hd_da.Mgn, PDU)
		
		Send_acyclicdata.ind(hd_da.Dfn, hd_da.Mgn, PDU)
		}
			ACTIVE

#	Current state	Event /condition => actions	Next state

19	ACTIVE	/ Transport_read_data (receive_acyclic_port_for_PtoP , PDU) == True
		&&
		DuplicateLAN_Topology(hd_sa.Dfn, hd_sa.Lnn) == True
		=>
		Check_TestMessage(PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		THEN {
		Packet_SEQ_Control (PDU, result-p)
		
		IF (result-p == 4) /* request to disconnect TCP connection */
		THEN
		Close_TCP_connection (PDU.hd_sa)
		
		ELSE IF(result-p == 1) /* normal packet reception */
		THEN {
		IF(hd_pkind == ACK.req)
		/* Receive ACK.req(=1) packet from a primary LAN */
		THEN
		{
		/* Create ACK_Inform_packet（PDU） */
		AckInformPDU = GetHeader(PDU)
		AckInformPDU.hd-sa = hd_da
		AckInformPDU.hd_da = hd_sa
		AckInformPDU.hd_ml = 64
		AckInformPDU.hd_bsize =64
		AckInformPDU.hd_pkind = 2 /* AckInform */
		AckInformPDU.hd_cbn =1
		AckInformPDU.hd_tbn = 1
		
		/* Send ACK_Inform_packet */
		
		Primary_transport_IDs = Get_ transport_identifier(AckInformPDU.hd_da.Dfn, AckInformPDU.hd_da.Lnn, transport_IDs-1, transport_IDs-2)
		Transport_write_data(transport_IDs-1, AckInformPDU)
		
		/* Processing receive-packet */
		Send_acyclicdata.ind(hd_da.Dfn, hd_da.Lnn, PDU)
		}
		ELSE IF(hd_pkind == ACK.inform)
		/* Receive ACK.inform (=2) from primary LAN) */
			ACTIVE

#	Current state	Event /condition => actions	Next state

		THEN
		Dequeue_PtoP_Retransmit_Que(hd_sa.Dfn, hd_sa.Lnn, hd_pseq)
		/* Delete all acknowleged packets from the specified PyoP_Retransmit_Que */
		ELSE IF(hd_pkind == NoACK)
		/* Receive NoACK (=0) from primary LAN) */
		THEN
		{
		/* Processing receive-packet */
		Send_acyclicdata.ind(hd_da.Dfn, hd_da.Lnn, PDU)
		}
		}	

20	ACTIVE	/ Transport_read_data (receive_acyclic_port_for_PtoP , PDU) == True && DuplicateLAN_Topology(hd_sa.Dfn, hd_sa.Lnn) == False /* Single LAN PtoP(TCP) communication */ => Check_TestMessage (PDU, result-mode) IF (result-mode == 1) /* effective message */ /* Processing receive-packet */ Send_acyclicdata.ind(hd_da.Dfn, hd_da.Lnn, PDU)	ACTIVE

21	ACTIVE	/ Transport_error_detection(transport_identifier) == True
		&&
		DuplicateLAN_Topology(transport_identifier) == True
		&&
		PtoP_Communication(transport_identifier) == True
		/* Error detection, DuplicateLAN and PtoP(TCP) communication */
		=>
		
		Change_PrimaryLAN(transport_identifier)
		Check_TestMessage(PDU, result-mode)
		IF (result-mode == 1) /* effective message */
		THEN {
		NewPrimary_transport_IDs = Get_Primary_transport_identifier(transport_identifier)
		
		/* Retransmit all PDUs in PtoP_Retransmit_Que */
		Retrans =Get_PDU(transport_identifier, PtoP_Retransmit_Que, RetransPDU)
		while(Retrans== True) {
		Transport_write_data(NewPrimary_transport_IDs , RetransPDU)
		}
		}
			ACTIVE

8.2.5.2 Functions

All functions used by the RT communication control are summarized in Table 63.

Table 63 – RT communication control functions

Name	Input	Output	Description
Change_PrimaryLAN	transport_identifier	-	This function changes a primary LAN which is used to transmit packets to destination node specified by transport_identifier. Specifically, it changes the primary/ secondary state of duplicated LANs. (secondary LAN → primary LAN, primary LAN → secondary LAN)
Close_TCP_connection	PDU.hd_sa	-	This function closes TCP connection to the node specified by PDU.hd_sa.
Create_a_RetransEnq-PDU	PDU	RequestPDU	This function creates a RetransEnq-PDU from information of specified PDU (RetransConfirm-PDU). In this case, it sets a message-mode state corresponding to an operation-mode state of node (NodeMode) into an hd_mode field. For each case of a combination of a retransMcg and a retransPseqNo within a RetransConfirm-PDU, execute the following steps. (Step1) (a) If the node has already received the PDU having the same sequence number as retransPseqNo from a multicast group (retransMcg), execute step2. (b) If the node has not received the PDU having the same sequence number as retransPseqNo from a multicast group (retransMcg) yet, create RetransEnq-PDU having the same retransmission sequence number with retransPseqNo. If a RetransEnq-PDU has been already created, set the additional information of combination of a retransMcg and a retransPseqNo into a RetransEnq-PDU. Go to step2. (Step2) Repeat the procedure of step 1 for all cases. When these all procedures completed, set the return value to a RequestPDU. (a) in the case of that a RetransEnq-PDU has been created. ・ RequestPDU ＝ RetransEnq-PDU (b) in the case of that a RetransEnq-PDU has not been created. ・ RequestPDU = Null
Create_b_RetransEnq-PDU	PDU	RequestPDU	This function creates a RetransEnq-PDU from information of a specified PDU (MulticastData-PDU). In this case, set a value of (R_PSEQ + 1) to hd_pseq field. Here, if a value of R_PSEQ equals 0xffffffff, set 1 to to hd_pseq field. Set RetransEnq-PDU to RequestPDU.

Name	Input	Output	Description
Check_TestMessage	PDU	result-mode	This function checks the effectiveness of a message. Specifically, if a message mode () within a receiving PDU is a Test mode, and operation mode of own node (Node mode) is an Online mode, it discards the PDU and returns the result-mode = 2. Otherwise, it returns the result-mode = 1. ・result-mode =1 (Valid message) ・result-mode = 2 (Invalid message)
Dequeue_PtoP_Retransmit_Que	Dfn, Lnn, hd_pseq	-	This function deletes the packets from a retransmission queue for a peer-to-peer communication specified by a Data field number (Dfn) and a node number (Lnn). Here, these deleted packets indicate the packet with the same sequence number as the hd_seq, and the packets that are registered faster than this packet.
DuplicateLAN_Topology	DstDfn, DstMgn(DstLnn) or transport_identifier	-	This function returns a True, if the communication with a node specified by either (a DstDfn, a DstMgn (DstLnn)) or a transport_identifier uses the duplexed LANs.
Enque_Multicast_Retransmit_Que	DstDfn, DstMgn, PDU	-	This function registers a PDU to a multicast communication retransmission queue specified by DstDfn and DstMgn.
Enque_PtoP_Retransmit_Que	DstDfn, DstLnn, PDU	-	This function registers a PDU to a peer-to-peer communication retransmission queue specified by DstDfn and DstLnn.
Expire_Timer	Rate_control_interval, DstDfn, DstMgn(DstLnn)	True / False	This function returns True when a transmission control timer of a transmission queue specified by DstDfn and DstMgn (DstLnn) expired.
GetHeader	PDU	newPDU	This function extracts a falArHeader from a PDU, sets it to a newPDU.
Get_Primary_transport_identifier	transport_identifier	NewPrimary_transport_IDs	This function gets the transport identifier of a primaryLAN (NewPrimary_transport_IDs), which is used at the time of transmitting packets to a destination node specified by a transport_identifier.
Get_PDU	transport_identifier , Retransmit_Que ,	RetransPDU ,Retrans	This function extracts the PDU in registered order from a retransmission queue for a destination node specified by a transport_identifier. If an extracted PDU exists, it sets the extracted PDU to a RetransPDU, and sets True to a Retrans, Otherwise, it sets False to a Retrans.
Get_RetransPDU	RetransPduList	RetransPDU, trans	This function extracts a PDU in registered order from a RetransPduList, and sets the PDU and the True to a RetransPDU and a trans, respectively. In the case that a RetransPduList is NULL or all PDUs have been extracted, sets False to a trans.
Get_transport_identifier	DstDfn, DstMg（DstLnn）	transport_IDs-1, transport_IDs-2	This function gets transport identifiers (port information, IP address etc) which are necessary to transmit a PDU, and sets these information to transport_IDs-1 and transport_IDs-2.

Name	Input	Output	Description
			(In the case of Duplexed LAN) ・ transport_IDs-1 = transport information of a primary LAN ・ transport_IDs-2 = transport information of a secondary LAN (In the case of a Single LAN) ・ transport_IDs-1 = transport information of a primary LAN ・ transport_IDs-2 = Null
Packet_SEQ_Control	PDU	result-p	This function compares a R_V_PSEQ with R_PSEQ by using the information of hd_sa, hd_m_ctl and hd_pseq. A R_V_PSEQ and a R_PSEQ are updated according to the result of comparison, (Ref. 5.3.2.12) (a) In the case of that R_V_SEQ ≠ 0 and R_V_SEQ ≠ hd_w_seq, the received PDU is discarded, set result-p = 4 (b) In the case of detecting duplicated packet or lost packets, the received packet is discarded. ・ result-p = 1 （normal packet reception） ・ result- p = 2 （duplicate packet reception） ・ result- p = 3 （lost packet detection） ・ result-p = 4 (Request to disconnect TCP connection)
Priority_QueingR_control	hd_da.Dfn, hd_da.Mgn (hd_da.Lnn), PDU	–	This function registers a PDU to a receiving queue specified by the hd_da.Dfn and hd_da.Mgn (hd_da.Lnn) corresponding priority level, and controls a priority receiving processing.
Priority_Queing_control	DstDfn, DstMgn (DstLnn), PDUdata	trans-mode	This function registers a PDUdata to a transmitting queue specified by the DstDfn and DstMgn (DstLnn) corresponding priority level, and controls a priority transmitting. Returns the transmitting mode to trans-mode. ・ trans-mode = Immediate (Immediate transmission) ・ trans-mode = Periodic (Interval transmission)
Produce_ RetransPduList (PDU, RetransPduList)	PDU	RetransPduList,	In accordance with the retransmitting request information within a RetransEnq-PDU, this function produces the list of all PDU to be retransmitted. Specially, execute a Step1 against each case which consists of both retransMcg and retransPseqNo within a ReturansEnq-PDU. (Step1) After extracting the specific PDUs from a retransmitting queue which corresponds to a retransMcg, set the PDUs to a RetransPduList in order of registering. Here, the specific PDUs mean the PDU having the same sequence number with a retransPseqNo, and mean all PDU registered subsequently. (Step2) If there is not a PDU having the same

Name	Input	Output	Description
			sequence number with a retransPseqNo in a retransmitting queue which corresponds to a retransMcg, create a RetransNAK-PDU. In this case, if a RetransNAK-PDU has already been created, adds both a retransMcg and a retransPseqNo to the fields of a RetransNAK-PDU. (Step3) With respect to a retransMcg of the case of the rest, repeat procedures of both step1 and step2. If a RetransNAK-PDU has already been created at the timing of completing all cases, set a RetransNAK-PDU to a RetransPduList.
Re-transmission_control	DstDfn, DstMgn,	True / False	This function checks whether the communication specified by the DstDfn and the DstMgn is a multicast communication with a retransmission. In the case of a multicast communication with retransmission, it returns True.
Select_transaction	DstDfn, DstMgn (DstLnn), transport_IDs-1, transport_IDs-2,	trans-PDU , ACK_req_identifier ,	After this function selects the queue which has exceeded a transmitting interval time from the transmitting queues specified by the DstDfn and the DstMgn (DstLnn), it extracts the sending PDU from the selected queue. If the corresponding multiple queues exist, process in order from the highest priority queue to the lowest priority queue. Here, a total number of transmissions don't exceed a pre-configured maximum number. (Result) ・ trans-PDU = transmitting PDU if no transmitting PDU exists, trans-PDU = Null ・ transport_IDs-1 = transport information of a primary LAN ・ transport_IDs-2 = transport information of a secondary LAN if LANs are not duplexed, transport_IDs-2 = Null ・ ACK_req_identifier = True/ False Here, an ACK_req_identifier variable is valid only when the target communication is a peer-to-peer (TCP) communication using a duplexed LAN. In the case of the following, set True to an ACK_req_identifier. Otherwise, set False to an ACK_req_identifier. (a) When counts of a transmission of PDU exceed a predefined maximum service counts which is corresponding to each transmitting queue. (b) Within each transmitting queue corresponding to priority levels, there exists no other PDU except for these PDU described above.
Start_Timer	Rate_control_interval, DstDfn, DstMgn	-	Invokes a transmitting control timer for the transmitting queue specified by DstDfn and DstMgn

Name	Input	Output	Description
Transport_error_detection	transport_identifier	True / False	To return True when a lower layer than transport layer detects an error that it cannot maintain communication. To return False in the case of no detection.
Transport_read_data	receive_port	PDU, True / False	To set the data into a PDU after reading data from a receiving port of transport layer. Here, at the timing of reading data, if data has already arrived at this port, returns “true”.
Transport_write_data	transport_IDs-i, PDU	-	To transmit a PDU into the transport layer, according to information of transport_ID_i. (i = 1, 2)
Update_S_PSEQ	DstDfn, DstMgn (DstLnn)	UpdatedNumber	(1) To return an updated number after updating a sending sequence number (S-PEQ) of packet for multicast communication specified DstDfn and DstMgn. (2) To return an updated number after updating a sending sequence number (S-PEQ) of packet for peer to peer communication specified DstDfn and DstLnn. Ref. 5.3.2.12

8.2.5.3 Variables

All the variables used by the RT communication control are summarized in Table 64.

Table 64 – RT communication control variables

Name	Description
ACK.inform	ACK-reply for transmission request of ACK packet
ACK.req	Transmission request of ACK packet
ACK_req_identifier	Identifier indicating the presence or absence of ACK-request. (Presence: True, Absence: False)
AckReplyPDU	Reply PDU for transmission request of ACK packet
Dfn	Data field number
DstDfn	Destination data field number
DstLnn	Destination node number
DstMgn	Destination multicast group number
Lnn	Logical node number
NewPrimary_transport_IDs	Transport identifier which was newly changed to a primary LAN
NodeMode	Operation mode of node（online mode or test mode）
Rate_control_interval	Controlling period of transmitting data
RequestPDU.hd_da.Dfn	Destination data field number set in a hd_da field of RequestPDU
RequestPDU.hd_da.Mgn	Destination multicast group number set in a hd_da field of RequestPDU
Retrans	Variable indicating the presence or absence of RetransPDU
Retrans-TCD	Transaction code of multicast communication with function of retransmission

Name	Description
Retransmit_Que	Queue of retransmission
hd_da	hd_da field in PDU
hd_pkind	hd_pkind field in PDU
hd_pri	hd_pri field in PDU
hd_pseq	hd_pseq field in PDU
hd_sa	hd_sa field in PDU
primaryLAN	Primary LAN in duplexed communication paths system
receive_acyclic_port_for_Alive	Receiving port number of an alive-communication
receive_acyclic_port_for_Multicast	Receiving port number of a multicast communication
receive_acyclic_port_for_MulticastWithRetrans	Receiving port number with function of retransmission
receive_acyclic_port_for_MulticastWithRetransControl	Receiving port number with function of retransmission control
receive_acyclic_port_for_PtoP	TCP's receiving port number of acyclic communication
receive_cyclic_port	Receiving port number of cyclic communication
receive_port	Receiving port number
result-m, result -p , result -mode	Results of function
trans	Variable indicating the presence or absence of retransmission PDU
trans-mode	Transmitting timing of PDU （Immediate / Periodic）
transport_identifier, transport_IDs-1, transport_IDs-2	Transport information which is needed to transmit a PDU from a transport layer, which consists of port number or IP address etc. （transport_IDs-1: transport information for LAN-1、 transport_IDs-2: transport information for LAN-2）

9 DLL mapping protocol machine (DMPM)

9.1 DMPM type S

The DMPM maps the ARPM service requests/responses to DL service requests/responses (converting APDUs to DLSDUs) and DL service indications/confirms to ARPM service indications/confirms (converting DLSDUs to APDUs).

Table 65 shows the relationships between ARPM and DL service.

Table 65 – ARPM to DL mapping

ARPM primitive	DL-Service	
	Service	Primitives
FAL-RCL.req FAL-RCL.cnf FAL-RCL.ind	Transmit/Receive DLSDU (RCL communication)	DL-RCL.req DL-RCL.cnf DL-RCL.ind
FAL-RTC.req FAL-RTC.cnf FAL-RTC.ind	Transmit/Receive DLSDU (RT communication)	DL-RTC.req DL-RTC.cnf DL-RTC.ind
RCL_STOP.ind	Stop RCL frames	DLM_RCL_STOP.ind
RCL_START.ind	Start RCL frames	DLM_RCL_START.ind
Node_ST.ind	Indicate the node of status	DLM_Node_ST.ind

9.2 DMPM type N

9.2.1 General

Subclause 9.2 defines the requirement for mapping the transmission of TPDUs over an Ethernet-TCP/UDP/IP-based network using the TCP/UDP/IP protocol suite.

9.2.2 Communication port in transport layer

The transmission of TPDUs (transport.req, transport.ind) uses TCP/UDP communication ports predefined configuration.

Table 66 shows an assignment policy of communication ports.

Table 66 – Assignment policy of communication ports

Communication type	Protocol	Port	Description
Multicast communication	UDP	transmitting port	One transmitting UDP port shall be assigned for each data field.
		receiving port	One receiving UDP port shall be assigned for each multicast group.
Peer -to- peer communication	TCP	transmitting port	Define one transmitting TCP port and one receiving TCP port between each node in the data field. Establish a TCP connection between these ports in each node.
		receiving port	
Alive-message communication	UDP	transmitting port	One transmitting alive-message UDP port shall be assigned for each data field. Each node shall use the same port number.
		receiving port	One receiving alive-message UDP port shall be assigned for each data field. Each node shall use the same port number.

9.2.3 Quality of Service

9.2.3.1 Overview

Quality of Service (QoS) is important for time sensitive applications. The type N defines requirements and recommendations for how type N equipment uses three QoS mechanisms: IEEE 802.1D, IEEE 802.1Q and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

Equipment may optionally support sending and receiving IEEE 802.1Q tagged frames and DiffServ Code Points (DSCP) processing, depending on their implementation capabilities.

9.2.3.2 Mapping type N traffic to DSCP, IEEE 802.1D and IEEE 802.1Q

DSCP is defined for specifying the relative priority of traffic based on the IPv4 Type of Service (ToS) field or on the IPv6 Traffic Class field. Figure 21 shows the DSCP field in the IPv4/IPv6 header.

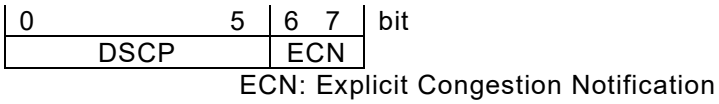


Figure 21 – DSCP format

IEEE 802.1Q tagged frame defines a PRIORITY (3-bit) field to specify 8 priority levels as shown in Figure 22. Priority 7 is the highest. Priority 0 is the lowest.

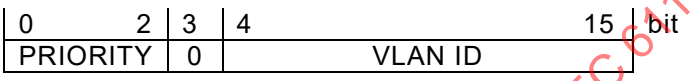


Figure 22 – IEEE 802.1Q tag frame format

Table 67 defines the default DSCP, IEEE 802.1D and IEEE 802.1Q priority mapping.

Table 67 – Default DSCP, IEEE 802.1D and IEEE 802.1Q priority mapping

Traffic usage (recommended)	DSCP (default)	IEEE 802.1D priority
-	-	7
Urgent Control	52	6
Cyclic transmission（short interval time）	44	5
High priority message / Control	36	4
Middle priority message / Control	28	3
Low priority message	20	2
-	-	1
-	-	0

These priority controls are valid between end equipments implementing a QoS function.

All type N equipment shall support receiving packets with non-zero DSCP values. Network Switches or Routers etc. can potentially alter DSCP values. Receiving equipment shall not assume or otherwise check whether incoming DSCP values are the same as in Table 67, or the same as the values sent from the sending equipment.

When sending traffic with IEEE 802.1Q tagged frames, type N equipment recommend to use the priority values specified in Table 67.

System design regarding QoS requires harmonizing the capability of the equipment and that of the network equipment, etc.

Bibliography

IEC 61158-1, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

SOMMAIRE

AVANT-PROPOS 131
INTRODUCTION 133
1 Domaine d'application 134
1.1 Généralités 134
1.2 Spécification 135
1.3 Conformité 135
2 Références normatives 135
3 Termes, définitions, symboles, abréviations et conventions 136
3.1 Termes et définitions du modèle de référence 136
3.1.1 Termes de l'ISO/IEC 7498-1 136
3.1.2 Termes de l'ISO/IEC 8822 136
3.1.3 Termes de l'ISO/IEC 9545 137
3.1.4 Termes de l'ISO/IEC 8824-1 137
3.2 Termes et définitions supplémentaires de type 25 137
3.3 Symboles et abréviations 139
3.4 Conventions 140
3.4.1 Conventions générales 140
3.4.2 Conventions pour les définitions de classe 140
3.4.3 Conventions pour la description de bit en octets 141
3.4.4 Conventions relatives aux descriptions des diagrammes d'états 141
4 Description de la syntaxe de FAL 142
4.1 Syntaxe abstraite des PDU de la FAL de type S 142
4.1.1 Syntaxe abstraite de base 142
4.2 Syntaxe abstraite des PDU de la FAL de type N 146
4.2.1 Syntaxe abstraite de base 146
4.2.2 CyclicData-PDU 147
4.2.3 MulticastData-PDU 147
4.2.4 PtoPData-PDU 147
4.2.5 Aliveinfo-PDU 148
4.2.6 Aliveinfo6-PDU 148
4.2.7 Inq-PDU 149
4.2.8 Ninq-PDU 149
4.2.9 Reply-PDU 149
4.2.10 RetransEnq-PDU 149
4.2.11 RetransConfirm-PDU 149
4.2.12 RetransNak-PDU 149
4.3 Affectation des types de données pour le type S 150
4.4 Affectation des types de données pour le type N 150
5 Syntaxe de transfert FAL 151
5.1 Règles de codage 151
5.1.1 Codage de valeur Unsigned 151
5.1.2 Codage de chaîne d'octets 151
5.1.3 Codage de SEQUENCE 152
5.2 Codage des éléments FAL-PDU de type S 152
5.2.1 RCL_header 152
5.2.2 RHE-PDU 153

5.2.3 LCC-PDU 155
5.2.4 LCA-PDU 155
5.2.5 LCN-PDU 156
5.2.6 LNA-PDU 157
5.2.7 SCR-PDU 157
5.2.8 Cyclic_S-PDU 157
5.2.9 Cyclic_header 158
5.2.10 Control-PDU 158
5.2.11 RMTCTL-PDU 158
5.2.12 INFO-PDU 159
5.3 Codage des éléments FAL-PDU de type N 160
5.3.1 Généralités 160
5.3.2 En-tête FALAR-N 160
5.3.3 CyclicData-PDU 175
5.3.4 MulticastData-PDU 176
5.3.5 PtoP Data-PDU 176
5.3.6 Aliveinfo-PDU 177
5.3.7 Aliveinfo6-PDU 179
5.3.8 Inq-PDU 181
5.3.9 Ninq-PDU 181
5.3.10 Reply-PDU 182
5.3.11 RetransEnq-PDU 182
5.3.12 RetransConfirm-PDU 183
5.3.13 RetransNak-PDU 184
6 Structure du diagramme d'états de protocole FAL 184
7 Machine de protocole FSPM 185
7.1 Vue d'ensemble 185
7.2 Type S de FSPM 186
7.2.1 Vue d'ensemble 186
7.2.2 Interface de communication cyclique avec les utilisateurs de la FAL 187
7.2.3 Diagramme d'états de FSPM 189
7.3 FSPM de type F 190
7.3.1 Vue d'ensemble 190
7.3.2 FSPM 191
8 Machine protocolaire de relations entre applications (ARPM) 193
8.1 Type S d'ARPM 193
8.1.1 Vue d'ensemble 193
8.1.2 Commande cyclique 194
8.1.3 Commande à distance 199
8.1.4 Commande de communication RCL 205
8.1.5 Commande de communication RT 210
8.2 Type N d'ARPM 214
8.2.1 Vue d'ensemble 214
8.2.2 Commande générale 215
8.2.3 Commande de transmission cyclique 217
8.2.4 Commande de transmission acyclique 223
8.2.5 Commande de communication RT 238
9 Machine de protocole DMPM 256

9.1 Type S de DMPM 256
9.2 Type N de DMPM 256
9.2.1 Généralités 256
9.2.2 Port de communication dans la couche transport 256
9.2.3 Quality of Service (Qualité de service) 257
Bibliographie 259

Figure 1 – Description de bit en octets 141
Figure 2 – hd_sa 160
Figure 3 – hd_da 161
Figure 4 – Numéro de séquence valide pour un message de réception 165
Figure 5 – hd_m_ctl 166
Figure 6 – Numéro de séquence de paquet de réception valide 170
Figure 7 – Liste des nœuds 182
Figure 8 – Relations entre les diagrammes de protocole 185
Figure 9 – Structure de type S de FSPM 186
Figure 10 – Attribution de mémoire partagée dans un réseau de type S 188
Figure 11 – Structure de type N de FSPM 191
Figure 12 – Structure de type F d'ARPM 194
Figure 13 – Séquence de communication cyclique 195
Figure 14 – Primitives de commande cyclique 196
Figure 15 – Primitives de commande à distance 200
Figure 16 – Primitives pour la commande de communication RCL 206
Figure 17 – Primitives pour la commande de communication RT 211
Figure 18 – Structure de type N d'ARPM 215
Figure 19 – Primitives de la commande de transmission cyclique 218
Figure 20 – Primitives de commande de transmission acyclique 224
Figure 21 – Format du DSCP 257
Figure 22 – Format de trame étiquetée IEEE 802.1Q 257

Tableau 1 – Descriptions des transitions d'état 142
Tableau 2 – Description des éléments d'un diagramme d'états 142
Tableau 3 – Conventions utilisées dans les diagrammes d'états 142
Tableau 4 – Classe de trame 152
Tableau 5 – DA_STaddress – DA_STaddress 152
Tableau 6 – DA_MACaddress 152
Tableau 7 – Format du champ CMD 153
Tableau 8 – Send Direction (direction d'envoi) 153
Tableau 9 – RHE ReceiveStatus 154
Tableau 10 – Physical Linkdown 154
Tableau 11 – RHE_pattern 1~4 155
Tableau 12 – LCC-Kind 155
Tableau 13 – RCL Status 156
Tableau 14 – hd_h_type 160

Tableau 15 – Utilisation de Mgn ou Lnn 162
Tableau 16 – Conditions détaillées pour la vérification des numéros de séquence du message de réception 165
Tableau 17 – Bits valides de hd_m_ctl 166
Tableau 18 – TCD spécifié 167
Tableau 19 – hd_pkind 167
Tableau 20 – PDU avec un hd_pseq efficace 168
Tableau 21 – Conditions détaillées de vérification des numéros de séquence de paquets de réception (communication multidiffusion avec retransmission) 170
Tableau 22 – Conditions détaillées de vérification des numéros de séquence de paquets 171
Tableau 23 – Relation entre transmission/réception de messages 172
Tableau 24 – hd_mode 172
Tableau 25 – Niveau de priorité des messages 172
Tableau 26 – Valeur de α 173
Tableau 27 – Exemple d'informations d'en-tête pour une fragmentation de message UDP 173
Tableau 28 – Exemple d'informations d'en-tête pour une fragmentation de message TCP 174
Tableau 29 – Valeur inqid_inq_sa 174
Tableau 30 – Valeur inqid_tr_adr 175
Tableau 31 – Valeur inqid_inq_seq 175
Tableau 32 – Relation entre inqid_id_seq et inqid_tr_adr 175
Tableau 33 – Type d'un message "alive" 177
Tableau 34 – Type d'un protocole de message "alive" 178
Tableau 35 – Temps de chaque al_mode 178
Tableau 36 – Changement d'état des tâches 179
Tableau 37 – Changement de contenu des tâches 179
Tableau 38 – Seuil du facteur de transmission 187
Tableau 39 – Exemple de menu de configuration de commande de trafic 188
Tableau 40 – Table d'états des données cycliques 189
Tableau 41 – Table d'états des données acycliques (Acyclic data) 190
Tableau 42 – Table d'états des données cycliques 192
Tableau 43 – Table d'états des données acycliques (Acyclic data) 192
Tableau 44 – Tableau d'états de la commande cyclique 197
Tableau 45 – Fonctions de commande cyclique 198
Tableau 46 – Variables de commande cyclique 199
Tableau 47 – Table d'états de commande à distance 201
Tableau 48 – Fonctions de commande à distance 204
Tableau 49 – Variables de commande à distance 205
Tableau50 – Tableau d'états de la commande de communication RCL 206
Tableau 51 – Fonctions de commande de communication RCL 208
Tableau 52 – Variables de la commande de communication RCL 209
Tableau 53 – Tableau d'états de la commande de communication RT 212
Tableau 54 – Fonctions de commande de communication RT 214
Tableau 55 – Variables de la commande de communication RT 214

Tableau 56 – Tableau d'états de la commande de transmission cyclique 219
Tableau 57 – Fonctions de commande de transmission cyclique 221
Tableau 58 – Variables de commande de transmission cyclique 223
Tableau 59 – Tableau d'états de la commande de transmission acyclique 225
Tableau 60 – Fonctions de commande de transmission acyclique 234
Tableau 61 – Variables de commande de transmission acyclique 237
Tableau 62 – Tableau d'états de la commande de communication RT 238
Tableau 63 – Fonctions de commande de communication RT 250
Tableau 64 – Variables de la commande de communication RT 255
Tableau 65 – ARPM to DL mapping 256
Tableau 66 – Politique d'attribution des ports de communication 257
Tableau 67 – Mapping de priorité DSCP par défaut, IEEE 802.1D et IEEE 802.1Q 258

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-25: Spécification du protocole de la couche application – Éléments de type 25

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 61158-6-25 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

La présente version bilingue (2020-12) correspond à la version anglaise monolingue publiée en 2019-06.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain* peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

Le présent document fait partie d'une série élaborée pour faciliter l'interconnexion des composants d'un système d'automation. Il est lié à d'autres normes de la série tel que défini par le modèle de référence des bus de terrain "à trois couches" décrit dans l'IEC 61158-1.

Le protocole application fournit le service application en utilisant les services disponibles de la liaison de données ou autre couche immédiatement inférieure. Le principal objectif du présent document est de définir un ensemble de règles de communication, exprimées en termes de procédures que doivent suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles pour la communication visent à fournir une base solide pour le développement afin de servir une diversité de besoins:

- guider les implémenteurs et les concepteurs;
- pour une utilisation dans les essais et achats d'équipements;
- comme partie intégrante d'un accord pour l'admission de systèmes dans l'environnement de systèmes ouverts;
- comme affinement pour la compréhension de communications prioritaires au sein de l'OSI (Open Systems Interconnexion, c'est-à-dire Interconnexion des systèmes ouverts).

Le présent document traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. L'utilisation conjointe du présent document avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain permet à des systèmes qui ne pourraient pas, sans cela, fonctionner ensemble dans toute combinaison.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-25: Spécification du protocole de la couche application – Éléments de type 25

1 Domaine d'application

1.1 Généralités

La couche application de bus de terrain (FAL, Fieldbus Application Layer) fournit aux programmes d'utilisateur un moyen d'accéder à l'environnement de communication du bus de terrain. A cet égard, la FAL peut être vue comme une "fenêtre entre des programmes d'application correspondants".

La présente Norme internationale fournit les éléments communs pour les communications de messagerie de base prioritaires et non prioritaires entre des programmes d'application dans un environnement d'automation et le matériau spécifique au bus de terrain de type 25. On utilise le terme "prioritaire" pour traduire la présence d'une fenêtre temporelle, à l'intérieur de laquelle une ou plusieurs actions spécifiées doivent être terminées avec un niveau de certitude défini. Si les actions spécifiées ne sont pas réalisées dans la fenêtre temporelle, les applications demandant les actions risquent de connaître une défaillance, avec les risques que cela comporte pour les équipements, les installations et éventuellement la vie humaine.

Le présent document définit de manière abstraite le comportement, visible par un observateur externe, assuré par les différents types de la couche application de bus de terrain, en termes

a) de syntaxe abstraite définissant les unités de données de protocole de la couche application, transmises entre les entités d'application en communication;
b) de syntaxe de transfert définissant les unités de données de protocole de la couche application, transmises entre les entités d'application en communication;
c) de diagramme d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication; et
d) de diagrammes d'états de relations d'applications définissant le comportement de communication visible entre les entités d'application en communication.

Le présent document vise à définir le protocole mis en place pour:

a) définir la représentation filaire des primitives de service définies dans l'IEC 61158-5-25; et
b) définir le comportement visible de l'extérieur qui est associé à leur transfert.

Le présent document spécifie le protocole de la couche application de bus de terrain de l'IEC, en conformité avec le modèle de référence de base OSI (ISO/IEC 7498-1) et avec la structure de la couche application OSI (ISO/IEC 9545).

Les services et protocoles de la FAL sont fournis par des entités d'application (AE, "Application Entity") de la FAL contenues dans les processus application. L'AE de la FAL se compose d'un jeu d'éléments de service application (ASE, "Application Service Element") orientés objet et d'une entité de gestion de couche (LME, "Layer Management Entity") qui gère l'AE. Les ASE fournissent des services de communication qui fonctionnent sur un jeu de classes d'objets de processus application (APO, "Application process object") connexes. L'un des ASE de la FAL est un ASE de gestion qui fournit un ensemble commun de services pour la gestion des instances des classes FAL.

Bien que ces services spécifient, du point de vue des applications, la manière dont la demande et les réponses sont émises et délivrées, ils n'incluent pas une spécification de ce que les applications qui demandent et qui répondent sont supposées en faire. Autrement dit, les aspects comportementaux des applications ne sont pas spécifiés; seules sont définies les demandes et les réponses que ces applications peuvent envoyer/recevoir. Cela offre aux utilisateurs de la FAL une plus grande flexibilité pour normaliser le comportement de ces objets. En plus de ces services, certains services d'appui sont également définis dans le présent document pour fournir l'accès à la FAL afin de maîtriser certains aspects de son fonctionnement.

1.2 Spécification

Le présent document a pour objectif principal de spécifier la syntaxe et le comportement du protocole de la couche application qui véhicule les services de la couche application définis dans l'IEC 61158-5-25. Un objectif secondaire est de fournir des trajets de migration à partir de protocoles de communication industriels préexistants. Ce dernier objectif explique la diversité des protocoles normalisés dans les sous-parties de l'IEC 61158-6.

1.3 Conformité

Le présent document ne spécifie pas de mises en œuvre individuelles ou de produits individuels et ne contraint pas les mises en œuvre d'entités de la couche application au sein des systèmes d'automatisation industriels.

Il n'est pas défini de conformité d'équipement à la présente norme de définition des services de couche application. A la place, la conformité est obtenue par la mise en œuvre de cette spécification de protocole de couche application.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

NOTE Toutes les parties de la série IEC 61158, ainsi que l'IEC 61784-1 et l'IEC 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

IEC 61158-3-25:2019, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 3-25: Définition des services de la couche liaison de données – Eléments de type 25*

IEC 61158-5-25:2019, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-25: Définition des services de la couche application – Eléments de type 25*

ISO/IEC 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/IEC 8822, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Définition du service de présentation*

ISO/IEC 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

ISO/IEC 8824-1, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base*

ISO/IEC/IEEE 8802-3, *Technologies de l'information – Télécommunications et échange d'informations entre systèmes – Réseaux locaux et métropolitains – Prescriptions spécifiques – Partie 3: Norme pour Ethernet*

IEEE Std 802.1D, *IEEE Standard for Local and metropolitan area networks – Media access Control (MAC) Bridges*; disponible à l'adresse http://www.ieee.org [affiché le 17-09-2018]

IEEE Std 802.1Q, *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks*; disponible à l'adresse http://www.ieee.org [affiché le 17-09-2018]

IETF RFC 768, *User Datagram Protocol*; disponible à l'adresse http://www.ietf.org> [affiché le 17-09-2018]

IETF RFC 791, *Internet Protocol*; disponible à l'adresse http://www.ietf.org [affiché le 17-09-2018]

3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/;
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp.

3.1 Termes et définitions du modèle de référence

3.1.1 Termes de l'ISO/IEC 7498-1

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/IEC 7498-1 s'appliquent:

a) entité d'application
b) processus application
c) unité de données de protocole d'application
d) élément de service d'application
e) invocation d'entité d'application
f) invocation de processus application
g) transaction d'application
h) système ouvert réel
i) syntaxe de transfert

3.1.2 Termes de l'ISO/IEC 8822

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/IEC 8822 s'appliquent:

a) syntaxe abstraite
b) contexte de présentation

3.1.3 Termes de l'ISO/IEC 9545

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/IEC 9545 s'appliquent:

a) association d'applications
b) contexte d'application
c) nom de contexte d'application
d) invocation d'entité d'application
e) type d'entité d'application
f) invocation de processus application
g) type de processus application
h) élément de service d'application
i) élément de service de contrôle d'application

3.1.4 Termes de l'ISO/IEC 8824-1

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/IEC 8824-1 s'appliquent:

a) identificateur d'objet
b) Type

3.2 Termes et définitions supplémentaires de type 25

Pour les besoins du présent document, les définitions suivantes s'appliquent également:

NOTE De nombreuses définitions sont typiques de plusieurs types de protocoles; elles ne sont pas forcément utilisées par tous les types de protocoles.

3.2.1
message ADP
message transmis par un protocole système décentralisé autonome

3.2.2
message d'état "alive"
message signalant l'état de son propre nœud, transmis de manière périodique

3.2.3
bloc
unité de donnée de base transférée dans le cadre d'une communication cyclique, ayant chacune une taille de 64 octets

3.2.4
catégorie N_f
catégorie d'un protocole système décentralisé autonome (spécifications complètes) de type N

3.2.5
catégorie N_l
catégorie d'un protocole système décentralisé autonome (spécifications simplifiées) de type N

3.2.6
communication de commande
communication acyclique de données pour des applications plus prioritaires dans un réseau de type S

3.2.7
transmission cyclique
communication périodique de données à des fins de communication en temps réel

3.2.8
mémoire de transfert cyclique
mémoire qui est attribuée à chaque nœud du champ de données, que chaque nœud transmet périodiquement dans le but de partager logiquement cette zone de mémoire

3.2.9
champ de données
lieu logique par lequel passent des données spécifiques, correspondant à des réseaux réels

3.2.10
domaine
ensemble administratif constitué de champs de données multiples

3.2.11
réseau local duplex
deux chemins de réseau différents entre des nœuds d'extrémité

3.2.12
équipement
matériel physique connecté au réseau

EXEMPLE Station, appareil et serveur.

3.2.13
communication d'information
communication acyclique de données pour une application moins prioritaire dans un réseau de type S

3.2.14
nœud logique, nœud
équipement dans les champs de données, dans lequel est installée une fonction de système décentralisé autonome

3.2.15
mode de message
identifiant qui indique les utilisations d'un message (qu'il s'agisse d'un message en ligne ou d'un message d'essai)

EXEMPLE Message en ligne ou message d'essai.

3.2.16
groupe de multidiffusion
groupe de nœuds appartenant à un champ de données pour déterminer s'il faut recevoir les mêmes données provenant d'un autre nœud

3.2.17
liste de nœuds
structure de matrice binaire permettant de spécifier un nœud pour recevoir des données de demande

3.2.18
mode de nœud
identifiant qui indique l'utilisation d'un nœud

EXEMPLE Nœud en ligne ou nœud d'essai.

3.2.19
réseau local primaire
réseau local (LAN) utilisé à l'état normal dans un système LAN duplex

3.2.20
commande de priorité
commande de l'ordre de transmission pour les paquets provenant de réseaux, selon le réseau local virtuel (VLAN)

3.2.21
communication de commande d'anneau
commande du réseau en anneau de type S utilisant des trames de commande d'anneau (RCL, ring control)

3.2.22
commande à distance
commande destinée aux autres nœuds connectés à un réseau

3.2.23
réseau local secondaire
réseau local (LAN) permettant la sauvegarde d'un LAN primaire dans un système LAN duplex

3.2.24
code de transaction
informations permettant d'identifier les caractéristiques du message au sein d'une PDU

3.3 Symboles et abréviations

ADP	Autonomous Decentralized Protocol (Protocole décentralisé autonome)
ADS-net	Autonomous Decentralized System – network (Système décentralisé autonome – réseau)
AE	Application Entity (Entité d'application)
AL	Application Layer (Couche application)
AP	Application Process (Processus d'application)
APDU	Application Protocol Data Unit (Unité de données de protocole d'application)
APO	Application Project Object (Objet de processus d'application)
AR	Application Relationship (Relation d'application)
AREP	Application Relationship Endpoint (Point d'extrémité de relation d'applications)
ARPM	Application Relationship Protocol Machine (Machine de protocole de relation entre applications)
ASCII	American Standard Code for Information Interchange (Code normalisé américain pour l'échange d'informations)
ASE	Application Service Element (Elément de service d'application)
ASN.1	Abstract Syntax Notation 1 (Notation de syntaxe abstraite n° 1)
CRC	Cyclic Redundancy Check (Contrôle de Redondance Cyclique)
DF	Data Field (Champ de données)
Dfn	Data Field Number (Numéro du champ de données)
DLL	Data Link Layer (Couche liaison de données)
Dmn	DoMain Number (Numéro de domaine)
DMPM	DLL Mapping Protocol Machine (Machine de protocole de mapping DLL)
DSCP	DiffServ Code Point (Point de code de service différencié)
FAL	Fieldbus Application Layer (Couche application de bus de terrain)

FSPM	FAL Service Protocol Machine (Machine de protocole de service FAL)
GMT	Greenwich Mean Time (Temps moyen de Greenwich)
IP	Internet Protocol (Protocole Internet)
IPv4	Internet Protocol version 4 (Protocole Internet version 4)
IPv6	Internet Protocol version 6 (Protocole Internet version 6)
LAN	Local area network (Réseau local)
LCA	Loop condition alert (Type S frame type) (Alarme d'état de boucle (trame de type S))
LCC	Loop condition check (Type S frame type) (Vérification de l'état d'une boucle (trame de type S))
LCN	Loop Condition Notify (Type S frame type) (Notification d'état de boucle (trame de type S))
LLD	Logical Link Down (Liaison logique descendante)
LLU	Logical Link Up (Liaison logique montante)
LNA	Loop Notify Answer (Type S frame type) (Réponse à une notification de boucle (trame de type S))
Lnn	Logical Node Number (Numéro de nœud logique)
LSB	Least Significant Bit (Bit de poids faible)
MCG	MultiCast Group (Groupe de multidiffusion)
Mgn	Multicast Group Number (Numéro de groupe de multidiffusion)
MSB	Most Significant Bit (Bit de poids fort)
MTU	Maximum Transmission Unit (Unité de transmission maximale)
OSI	Open Systems Interconnection (Interconnexion de systèmes ouverts)
PDU	Application Data Unit (Unité de données de protocole)
PLD	Physical Link Down (Liaison physique descendante)
QoS	Quality-of-Service (Qualité de service)
RCL	Ring control (Commande de l'anneau)
RHE	Rapid hello (Type S frame type) (Hello rapide (trame de type S))
RMTCTL	Remote control (Commande à distance)
RT	Real Time (Temps réel)
SCR	Station condition report (Type S frame type) (Rapport d'état de station (trame de type S))
TCD	Transaction CoDe (Code de transaction)
TCP	Transmission Control Protocol (Protocole de contrôle de transmission)
TMID	Transfer Memory Identifier (Identifiant de mémoire de transfert)
UDP	User Datagram Protocol (Protocole de datagramme utilisateur)

3.4 Conventions

3.4.1 Conventions générales

Les conventions données dans l'IEC 61158-3-25 s'appliquent également dans le présent document.

3.4.2 Conventions pour les définitions de classe

Cette description de la FAL de type 25 utilise un sous-ensemble de l'ASN.1. Les structures suivantes sont utilisées. Type sélectif (CHOICE) – Représente une sélection parmi les types de candidats

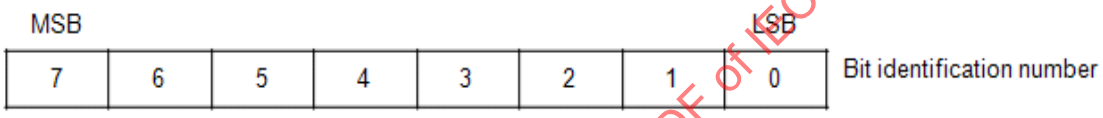
Type de séquence (SEQUENCE) – Représente une liste à ordre fixe comme dans l'exemple suivant:

```
DLPDU::= SEQUENCE {
preamble      Preamble,
sfd           SFD,
destaddr      DestAddr,
srcaddr       SrcAddr,
vlantag       VLANtag,
lt            LT,
dlsdu         FAL-PDU,
fcs           FCS
}
```

NOTE Cet exemple indique que la DLPDU, qui représente la trame Ethernet, est définie comme une SEQUENCE. La DLPDU comprend ce qui suit: Preamble, SFD, DestAddr, SrcAddr, VLANtag, LT, FALL-PDU et FCS.

3.4.3 Conventions pour la description de bit en octets

Lors de l'identification de chaque bit d'un octet, chaque bit est identifié par un numéro, tel qu'indiqué à la Figure 1 et décrit comme le Bit n.



Anglais	Français
Bit identification number	Numéro d'identification de bit
MSB	MSB (Bit de poids fort)
LSB	LSB (Bit de poids faible)

Figure 1 – Description de bit en octets

Lors de la spécification de plusieurs bits situés de façon séquentielle, le symbole de plage (..) est utilisé (par ex.: 7..0 spécifie les bits 7 à 0 inclusifs).

Lors de la spécification de plusieurs octets, le bit de poids faible (LSB) de l'octet le plus bas est considéré comme étant le Bit 0, puis les numéros d'identification de bit sont attribués par ordre croissant.

NOTE Par exemple, lors de la spécification de 4 octets, le bit de poids fort (MSB) de l'octet le plus haut est le Bit 31, le MSB du deuxième octet est le Bit 23, le MSB du troisième octet est le Bit 15 et le MSB de l'octet le plus bas est le Bit 7.

3.4.4 Conventions relatives aux descriptions des diagrammes d'états

La description du diagramme d'états est définie sous forme de tableau, tel qu'indiqué dans le Tableau 1. La signification des éléments est représentée dans le Tableau 2. Les conventions utilisées dans les diagrammes d'états sont présentées dans le Tableau 3.

Chaque ligne du tableau d'états représente une transition d'état. La première colonne indique le nom ou le numéro de la transition d'état. La deuxième colonne indique l'état actuel. La troisième colonne indique les événements, les conditions et les actions. La quatrième colonne indique l'état suivant. Lorsqu'un événement ou une condition se produit, l'action est exécutée et le diagramme d'états passe à l'état suivant.

Tableau 1 – Descriptions des transitions d'état

#	État courant	Evénement /condition => action	État suivant

Tableau 2 – Description des éléments d'un diagramme d'états

Elément de description	Signification
#	Numéro de la transition
Etat courant, État suivant	Noms de l'état de départ et de l'état de transition cible.
Evénement	Nom ou description de l'événement déclencheur qui provoque la transition.
/ conditions	Expression booléenne, qui doit être vraie pour que la transition ait lieu
=>actions	Liste des affectations et des appels de service ou de fonction. Il convient que l'action soit atomique. Le symbole "=>" qui précède ne fait pas partie de l'action.
Les "*/ conditions" peuvent être omises.	

Tableau 3 – Conventions utilisées dans les diagrammes d'états

Convention	Signification
=	Substitution du côté droit par le côté gauche
==	Condition logique indiquant qu'une entité de gauche est égale à une entité de droite.
!=	Condition logique indiquant qu'une entité de gauche n'est pas égale à une entité de droite.
<	Condition logique indiquant qu'une entité de gauche est inférieure à une entité de droite.
>	Condition logique indiquant qu'une entité de gauche est supérieure à une entité de droite.
&&	Logique AND
||	"OR" (c'est-à-dire: OU) logique
!	Opérateur de négation
+ – * /	Opérateur arithmétique
;	Point d'interruption
++	Opérateur d'incrémentation
--	Opérateur de décrémentation

4 Description de la syntaxe de FAL

4.1 Syntaxe abstraite des PDU de la FAL de type S

4.1.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```
FAL-PDU::= CHOICE {
        RCL-PDU,
        RT-PDU
}
```

```
RCL-PDU::= CHOICE {
        RHE-PDU,
        LCC-PDU,
        LCA-PDU,
        LCN-PDU,
        LNA-PDU,
        SCR-PDU
}

RT-PDU::= CHOICE {
        Cyclic_S-PDU,
        CTL-PDU,
        INFO-PDU
}

CTL-PDU::= CHOICE {
        Control-PDU,
        RMTCTL-PDU
}
```

4.1.1.1 RCL-PDU

Les éléments RCL_header à utiliser dans chaque RCL-PDU sont indiqués ci-dessous.

```
RCL_header::= SEQUENCE {
        Frame Class             CLASS,
        Dst Address             SEQUENCE {
                Reserved1                   Unsigned8,
                DA_STaddress                ST_Address,
                DA_MACaddress               MAC_Address
        },
        Src Address             SEQUENCE {
                Reserved2                   Unsigned8,
                SA_STaddress                ST_Address,
                SA_MACaddress               MAC_Address
        },
        RCL Command             RCLCMD,
        Frame Sequence          SequenceNum32,
        Reserved3               OCTET STRING(SIZE 20)
}
```

4.1.1.1.1 RHE-PDU

```
RHE-PDU::= SEQUENCE {
        RCL_header              RCL_HEADER,
        RHE_header              SEQUENCE {
                RHE_Common              SEQUENCE {
                        Send Direction              PortNum,
                        RHE Sequence                SequenceNum32
                        },
                Neighborhood address    SEQUENCE {
                        Reserved1                   Unsigned32,
                        NB_STaddress                ST_Address,
                        NB_MACaddress               MAC_Address
                        },
                RHE ReceiveStatus       RHE_RxStatus,
                True Status             SEQUENCE {
                        Reserved2                   Unsigned32,
                        Physical Linkdown           PLD,
                        Reserved3 ...Unsigned32
                        },
                Blocking Status                     BLK_Status,
```

```
                Node Status                     Node_Status,
                Reserved4                       OCTET STRING(SIZE 20)
},
        Reserved5           OCTET STRING(SIZE 64),
        RHE_pattern         SEQUENCE {
            RHE_pattern1            Pattern32,
            RHE_pattern2            Pattern32,
            RHE_pattern3            Pattern32,
            RHE_pattern4            Pattern32,
                .
                .
                .
}
```

4.1.1.1.2 LCC-PDU

```
LCC-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCC_header          SEQUENCE {
                LCC Kind                RCLKIND,
                LCC Priority    SEQUENCE {
                        LCC_Priority            Priority,
                        LCC_STadd               ST_Address,
                        LCC_MACadd              MAC_Address
                },
                Reserved1               OCTET STRING(SIZE 52)
        }
}
```

4.1.1.1.3 LCA-PDU

```
LCA-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCA_header          SEQUENCE {
                Reserved1                       Unsigned32,
                RCL Status                      RCL_Status,
                LCA Priority            SEQUENCE {
                        LCA_Priority            Priority,
                        LCA_STadd               ST_Address,
                        LCA_MACadd              MAC_Address
                },
        Reserved2                       OCTET STRING(SIZE 48)
        }
}
```

4.1.1.1.4 LCN-PDU

```
LCN-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LCN_header          SEQUENCE {
                Reserved1                       Unsigned32,
                RCL Status                      RCL_Status,
                Reserved2                       OCTET STRING(SIZE 56)
        }
}
```

4.1.1.1.5 LNA-PDU

```
LNA-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        LNA_header          SEQUENCE {
                Reserved1           Unsigned32,
                RCL Status          RCL_Status,
                Reserved2           OCTET STRING(SIZE 56)
        }
}
```

4.1.1.1.6 SCR-PDU

```
SCR-PDU::= SEQUENCE {
        RCL_header          RCL_HEADER,
        SCR_header          SEQUENCE {
                Entry Number        EntryNum,
                SendStat Time       Send_Time,
                Reserved1           OCTET STRING(SIZE 52)
},
        Event_info          SEQUENCE {
                Event0              EventData,
                Event1              EventData,
                    .
                    .
                    .
                Event16             EventData
        }
}
```

4.1.1.2 RT-PDU

4.1.1.2.1 Cyclic_S-PDU

```
Cyclic_S-PDU::= SEQUENCE {
        IP_header           IP_HEADER,
        UDP_header          UDP_HEADER,
        Reserved1           Unsigned16,
        Cyclic_header       CYCLIC_HEADER,
        Block_num           SEQUENCE {
                Block number1               BlockNum,
                Block number2               BlockNum,
                        .
                        .
                        .
                Block number21              BlockNum,
                BLock_num pattern           Pattern16
        },
        Cyclic_data         SEQUENCE {
                Block data1         BlockData,
                Block data2         BlockData,
                        .
                        .
                        .
                Block data21        BlockData
        },
        Cyclic_FCS          CFCS
}

Cyclic_header::= SEQUENCE {
        Cyclic_pattern      CYCLIC_PATTERN,
```

```
        Total_Block             TOTAL_BLOCK,
        Src STaddress           ST_Address,
        Reserved1 ...Unsigned8
}
```

4.1.1.2.2 CTL-PDU

4.1.1.2.2.1 Control-PDU,

```
Control-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        CTL_data                CTLdata
}
```

4.1.1.2.2.2 RMTCTL-PDU

```
CTL_cyclic-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        RMTCTL_CMD              SEQUENCE {
                Reserved1               Unsigned16,
                Header Type             HEADER_TYPE,
                Module ID               MODULE_ID,
                Station number          ST_NUM,
                Command code            RMTCTL_CMD,
                Length                  LENGTH,
                RMTCTL Status           STATUS,
                RMTCTL Kind             RMTKIND,
                RMTCTL Length           RMT_LENGTH,
                RMTCTL Address          RMT_ADD,
                Send Number             SND_NUM,
                Reserved1               Unsigned16,
                Reserved2 ...Unsigned32
                },
        RMTCTL data             RMTCTL_DATA
}
```

4.1.1.2.3 INFO-PDU

```
INFO-PDU::= SEQUENCE {
        IP_header               IP_HEADER,
        UDP_header              UDP_HEADER,
        INFO_data               INFOData
}
```


4.2 Syntaxe abstraite des PDU de la FAL de type N

4.2.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```
FAL-PDU::= CHOICE {
        CyclicData-PDU[0] CyclicData-PDU ,
        MulticastData-PDU       [1] MulticastData-PDU ,
        PtoP Data-PDU           [2] PtoP Data-PDU ,
        Aliveinfo-PDU           [3] Aliveinfo-PDU ,
        Aliveinfo6-PDU          [4] Aliveinfo6-PDU ,
        Inq-PDU                 [5] Inq-PDU ,
        Ninq-PDU                [6] Ninq-PDU ,
        Reply-PDU               [7] Reply-PDU ,
        RetransEnq-PDU          [8] RetransEnq-PDU,
```

```
        RetransConfirm-PDU    [9] RetransConfirm-PDU,
        RetransNak-PDU        [10] RetransNak-PDU
        }
```

Les éléments FALAR-N à utiliser dans chaque PDU sont indiqués ci-dessous.

```
FALAR-N Header::= SEQUENCE   {
        hd_h_type             Hd_h_type ,
        hd_ml                 Hd_ml ,
        hd_sa                 Hd_sa ,
        hd_da                 Hd_da ,
        hd_v_seq              Hd_v_seq ,
        hd_seq                Hd_seq ,
        hd_m_ctl              Hd_m_ctl ,
        hd_inqid              Hd_inqid ,
        hd_tcd                Hd_tcd ,
        hd_ver                Hd_ver ,
        reserved1             OctString SIZE(3) ,
        hd_pkind              Hd_pkind ,
        hd_pseq               Hd_pseq ,
        hd_mode               Hd_mode ,
        hd_pver               Hd_pver ,
        hd_pri                Hd_pri ,
        hd_cbn                Hd_cbn ,
        hd_tbn                Hd_tbn ,
        hd_bsize              Hd_bsize ,
        reserved2             OctString SIZE(4)
        }

Hd_inqid::= SEQUENCE {
        inqid_inq_sa          Inqid_inq_sa ,
        inqid_tr_adr          Inqid_tr_adr ,
        inqid_id_seq          Inqid_id_seq
        }
```

4.2.2 CyclicData-PDU

```
CyclicData-PDU::= SEQUENCE {
        falArHeader           FALAR-N Header ,
        tmid                  Tmid ,
        blockNumber           BlockNumber ,
        blockCount            BlockCount ,
        cyclicData            CyclicData
        }
```

4.2.3 MulticastData-PDU

```
 MulticastData-PDU::= SEQUENCE {
        falArHeader           FALAR-N Header ,
        multicastData         MulticastData
        }
```

4.2.4 PtoPData-PDU

```
PtoP Data-PDU::= SEQUENCE {
        falArHeader           FALAR-N Header ,
        pointToPointData      PointToPointData
        }
```

4.2.5 Aliveinfo-PDU

```
Aliveinfo-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        alive4Header            ALIVE4 Header ,
        al_ExtentionList        SEQUENCE of Al_ExtentionList ,
        al_ExtentionInfo        Al_ExtenyionInfo
        }

ALIVE4 Header::= SEQUENCE {
        al_nd_name              Al_nd_name ,
        al_os_name              Al_os_name ,
        al_tm_out               Al_tm_out ,
        al_msgserno             Al_msgserno ,
        al_mode                 Al_mode ,
        al_protocol             Al_protocol ,
        al_tg_cmn_cnt           Al_tg_cmn_cnt ,
        al_tg_cflag             Al_tg_cflag ,
        reserved1               OctString SIZE(1) ,
        al_tg_max               Al_tg_max ,
        al_tg_usecnt            Al_tg_usecnt ,
        al_chg_time             Al_chg_time ,
        al_ipv4addr1            Al_ipv4addr1 ,
        al_ipv4addr2            Al_ipv4addr2 ,
        al_ver                  AL_ver ,
        reserved2               OctString SIZE(15)
        }

Al_ExtentionList::= SEQUENCE {
        al_ta_chgalvstat        Al_ta_chgalvstat ,
        al_ta_chginfostat       Al_ta_chginfostat ,
        al_ta_tid               Al_ta_tid ,
        al_ta_data              Al_ta_data
        }
```

4.2.6 Aliveinfo6-PDU

```
Aliveinfo6-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header,
        alive6Header            ALIVE6 Header ,
        al_ExtentionList        SEQUENCE of Al_ExtentionList ,
        al_ExtentionInfo        Al_ExtentionInfo
        }

ALIVE6 Header::=  SEQUENCE {
        al_nd_name              Al_nd_name ,
        al_os_name              Al_os_name ,
        al_tm_out               Al_tm_out ,
        al_msgserno             Al_msgserno ,
        al_mode                 Al_mode ,
        al_protocol             Al_protocol ,
        al_tg_cmn_cnt           Al_tg_cmn_cnt ,
        al_tg_cflag             Al_tg_cflag ,
        reserved1               OctString SIZE(1) ,
        al_tg_max               Al_tg_max ,
        al_tg_usecnt            Al_tg_usecnt ,
        al_chg_time             Al_chg_time ,
        reserved2               Al_ipv4addr1 ,
        reserved3               Al_ipv4addr2 ,
        al_ver                  AL_ver ,
        reserved4               OctString SIZE(15) ,
        al_ipv6addr1            Al_ipv6addr1 ,
```

```
        al_ipv6addr2 …Al_ipv6addr2
        }
```

4.2.7 Inq-PDU

```
Inq-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        inquirydata             InquiryData
        }
```

4.2.8 Ninq-PDU

```
Ninq-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header ,
        node-list               Node-List ,
        ninquirydata            NinquiryData
        }
```

4.2.9 Reply-PDU

```
Reply-PDU::= SEQUENCE {
        falArHeader             FALAR-N Header
        }
```

4.2.10 RetransEnq-PDU

```
RetransEnq-PDU::= SEQUENCE {
        falArHeader                 FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransRequestNode          RetransRequestNode ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        reserved1                   OctString SIZE(4) ,
        retransInfo-List            SEQUENCE of RetransInfoList
        }

RetransInfoList::= SEQUENCE {
        retransMcg                  RetransMcg ,
        retransPseqNo               RetransPseqNo
        }
```

4.2.11 RetransConfirm-PDU

```
RetransConfirm::= SEQUENCE {
        falArHeader                 FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        retransInfo-List            SEQUENCE of RetransInfoList
        }
```

4.2.12 RetransNak-PDU

```
RetransNAK-PDU::= SEQUENCE {
        f alArHeader                FALAR-N Header ,
        retransRequest              RetransRequest ,
        retransNumberOfRequests     RetransNumberOfRequests ,
        retransInfo-List            SEQUENCE of RetransInfoList
        }
```

4.3 Affectation des types de données pour le type S

Les types de données utilisés dans la syntaxe abstraite des FAL-PDU de type F sont indiqués ci-dessous.

BLK_Status::= Unsigned32
BlockData::= OCTET STRING(SIZE=64)
BlockNum::= Unsigned16
CLASS::= Unsigned16
CLTData::= OCTET STRING(SIZE = 18..1472)
CYCLIC_FCS::= Unsighed32
EntryNum::= Unsigned32
ERR_ADD::= Unsigned32
HEADER_TYPE::= Unsigned16
INFOData::= OCTET STRING(SIZE = 18..1472)
IP_Header::= OCTET STRING(SIZE=20)
MODULE_ID::= Unsigned16
Node_Status::= Unsigned32
Pattern16::= Unsigned16
Pattern32::= Unsigned32
Pattern8::= Unsigned8
PLD::= Unsigned32
PortNum::= Unsigned32
Priority::= Unsigned8
RCL_Status::= Unsigned32
RCLCMD::= Unsigned32
RCLKIND::= Unsigned32
RMT_ADD::= Unsigned32
RMT_LENGTH::= Unsigned16
RMTCTL_CMD::= Unsigned16
RMTCTL_DATA::= OCTET STRING(SIZE = 0..1440)
RMTKIND::= Unsigned16
SendTime::= OCTET STRING(SIZE=8)
SequenceNum32::= Unsigned32
SND_NUM::= Unsigned16
ST_Address::= Unsigned8
ST_NUM::= Unsigned16
STATUS::= Unsigned32
TOTAL_BLOCK::= Unsigned8
UDP_Header::= OCTET STRING(SIZE=8)

4.4 Affectation des types de données pour le type N

Les types de données utilisés dans la syntaxe abstraite des FAL-PDU de type F sont indiqués ci-dessous.

AI_ExtentionInfo::= OctString SIZE(0..664)
AI_chg_time::= Unsigned32
AI_ipv4addr1::= OctString SIZE(4)
AI_ipv4addr2::= OctString SIZE(4)
AI_ipv6addr1::= OctString SIZE(16)
AI_ipv6addr2::= OctString SIZE(16)
AI_mode::= Unsigned8
AI_msgsernot::= Unsigned16
AI_nd_name::= OctString SIZE(10)
AI_os_name::= OctString SIZE(10)
AI_protocol::= Unsigned8
AI_ta_chgalvstat::= Unsigned8
AI_ta_chginfostat::= Unsigned8
AI_ta_data::= interger16
AI_ta_tid::= Unsigned16
AI_tg_cflag::= Unsigned8

Al_tg_cmn_cnt::= Unsigned16
Al_tg_max::= Unsigned16
Al_tg_usecnt::= Unsigned16
Al_tm_out::= Unsigned32
AL_ver::= Unsigned8
BlockCount::= Unsigned16
BlockNumber::= Unsigned16
CyclicData::= Octstring SIZE(64..16640)
Affectation des types de données
Hd_bsize::= unsigned16
Hd_cbn::= unsigned8
Hd_da::= unsigned32
Hd_h_type::= unsigned32
Hd_m_ctl::= unsigned32
Hd_ml::= unsigned32
Hd_mode::= integer16
Hd_pkind::= unsigned8
Hd_pri::= unsigned8
Hd_pseq::= unsigned32
Hd_pver::= unsigned8
Hd_sa::= unsigned32
Hd_seq::= unsigned32
Hd_tbn::= unsigned8
Hd_tcd::= unsigned16
Hd_v_seq::= unsigned32
Hd_ver::= unsigned16
Inqid_id_seq::= unsigned32
Inqid_inq_sa::= unsigned32
Inqid_tr_adr::= unsigned32
InquiryData::= OctString SIZE(0..262144)
MulticastData::= OctString SIZE(0..262144)
Node-List::= OctString SIZE(512)
PointToPointData::= OctString SIZE(0..262144)
RetransMcg::= unsigned32
RetransNumberOfRequests::= unsigned32
RetransPseqNo::= unsigned32
RetransRequest::= unsigned32
RetransRequestNode::= unsigned32
Tmid::= Unsigned32

5 Syntaxe de transfert FAL

5.1 Règles de codage

5.1.1 Codage de valeur Unsigned

Les valeurs non signées de longueur fixe Unsigned8, Unsigned16, Unsigned24 et Unsigned32 sont respectivement codées comme valeurs entières non signées d'un octet, deux octets, trois octets et quatre octets de longueur. Les valeurs Unsigned16, Unsigned24 et Unsigned32 sont codées comme gros-boutistes. L'octet de poids fort est considéré comme le premier octet, l'octet suivant comme le deuxième octet et l'octet de poids faible comme le dernier octet.

5.1.2 Codage de chaîne d'octets

La valeur OctetString, dont la chaîne d'octets est de longueur variable, est codée octet par octet, dans l'ordre séquentiel.

5.1.3 Codage de SEQUENCE

Le codage de SEQUENCE est exécuté selon une séquence, en commençant par l'élément initial. Les identificateurs et la longueur utilisés dans l'ASN.1 ne sont pas utilisés.

5.2 Codage des éléments FAL-PDU de type S

5.2.1 RCL_header

5.2.1.1 Classe de trame

Ce champ indique les classes de PDU décrites dans le Tableau 4.

Tableau 4 – Classe de trame

Valeur	Description
0x0001.	Trame de classe 1 (RHE-PDU)
0x0002.	Trame de classe 2 (LCC-PDU, LCA-PDU, LCN-PDU, LNA-PDU, SCR-PDU)

5.2.1.2 Adresse Dst

5.2.1.2.1 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.1.2.2 DA_STaddress

Ce champ indique l'adresse de la station de destination décrite dans le Tableau 5.

Tableau 5 – DA_STaddress – DA_STaddress

Valeur	Description
0xFF.	RHE-PDU, LCC-PDU, LCN-PDU, SCR-PDU
SA_STaddress dans la LCC-PDU reçue	LCA-PDU
SA_STaddress dans la LCN-PDU reçue	LNA-PDU

5.2.1.2.3 DA_MACaddress

Ce champ indique l'adresse MAC de destination décrite dans le Tableau 6.

Tableau 6 – DA_MACaddress

Valeur	Description
FF-FF-FF-FF-FF-FF	RHE-PDU, LCC-PDU, LCN-PDU, SCR-PDU
SA_MACaddress dans la LCC-PDU reçue	LCA-PDU
SA_MACaddress dans la LCN-PDU reçue	LNA-PDU

5.2.1.3 Adresse Src

5.2.1.3.1 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.1.3.2 SA_STaddress

Ce champ indique l'adresse de station de son propre nœud.

5.2.1.3.3 SA_MACaddress

Ce champ indique l'adresse MAC de son propre nœud.

5.2.1.4 RCL Command

Le Tableau 7 indique le format du champ RCL Command (CMD). Le champ sert à identifier la trame RCL. La signification de chaque bit de ce champ est la suivante:

Tableau 7 – Format du champ CMD

Bit	Valeur						Description
	RHE	LCC	LCA	LCN	LNA	SCR	
[3:0]	1						champ 1 réservé
[7:4]	0	0	1	2	3	0	numéro unique 1
[11:8]	0	1	1	1	1	3	numéro unique 2
[15:12]	0						champ 2 réservé
[23:16]	1	2	2	2	2	2	indique la classe de trame
[31:24]	0						champ 3 réservé

5.2.1.5 Séquence de trames

Le numéro de séquence de trames permet de détecter la perte et la duplication des trames reçues. La plage de cette valeur va de 0x0000000000 à 0xFFFFFFFFFF avec la taille de pas 1.

5.2.1.6 Reserved3.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.2 RHE-PDU

5.2.2.1 RHE_Common

5.2.2.1.1 Send Direction

Ce champ contient le port de transmission de RHE-PDU tel que décrit dans le Tableau 8.

Tableau 8 – Send Direction (direction d'envoi)

Valeur	Description
0x01.	Port A
0x02.	Port B

5.2.2.1.2 RHE Sequence

Ce champ contient le numéro d'identification de RHE-PDU. La plage pour cette valeur est 0x0000 à 0xFFFF. La même valeur est utilisée pour les ports A et B.

5.2.2.2 Neighborhood address

5.2.2.2.1 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.2.2.2 NB_STaddress

Ce champ indique l'adresse de station du nœud voisin.

5.2.2.2.3 NB_MACaddress

Ce champ indique l'adresse MAC du nœud voisin.

5.2.2.3 RHE RecieveStatus

Ce champ indique l'état de réception de RHE-PDU tel que décrit dans le Tableau 9.

Tableau 9 – RHE ReceiveStatus

Valeur	Description
1	Réception de RHE-PDU
0	Non-réception de RHE-PDU

5.2.2.4 True Status

5.2.2.4.1 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0x00000000.

5.2.2.4.2 Physical Linkdown

Ce champ indique l'état de liaison au niveau du port tel que décrit dans le Tableau 10.

Tableau 10 – Physical Linkdown

Valeur	Description
1	Liaison physique montante
0	Liaison physique descendante

5.2.2.4.3 Reserved3.

Il est réservé à un usage ultérieur. La valeur est 0x00000000.

5.2.2.5 Blocking Status

Ce champ indique le code d'état du port.

5.2.2.6 Node Status

Ce champ indique le code d'état du nœud.

5.2.2.7 Reserved4.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.2.8 Reserved5.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.2.9 RHE_pattern14

Ce champ est utilisé pour vérifier le schéma des trames RHE, comme décrit dans le Tableau 11. Les schémas sont répétés jusqu'à la longueur maximale de la trame selon l'ISO/IEC 8802-3.

Tableau 11 – RHE_pattern 1~4

Motif	Octets	Valeur
RHE_pattern1.	4	0xFFFFFFFF.
RHE_pattern2.	4	0xAAAAAAAA.
RHE_pattern3.	4	0x55555555.
RHE_pattern4.	4	0x00000000.

5.2.3 LCC-PDU

5.2.3.1 LCC Kind

Ce champ indique la raison d'envoi de la LCC-PDU comme décrit dans le Tableau 12.

Tableau 12 – LCC-Kind

Valeur	Description
0x00000101.	Port A de liaison logique montante du nœud
0x00000201.	Possibilité de recevoir d'autres LCC-PDU

5.2.3.2 Priorité LCC

5.2.3.2.1 LCC_Priority

Ce champ indique la priorité de la LCC-PDU. La valeur est 0x0C.

5.2.3.2.2 LCC_STadd

Ce champ indique l'adresse de station de son propre nœud.

5.2.3.2.3 LCC_MACadd

Ce champ indique l'adresse MAC de son propre nœud.

5.2.3.3 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.4 LCA-PDU

5.2.4.1 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x01.

5.2.4.2 RCL Status

Ce champ indique l'état du nœud qui envoie la LCA-PDU comme décrit dans le Tableau 13.

Tableau 13 – RCL Status

Nom du champ	Bit	Valeur	Description
-	31 … 24	0x00.	reserved (réservées)
Node status	23	0 ou 1	1: le nœud est à l'état "Edge-A". Lorsque les bits 23 et 22 sont de "1", le nœud est à l'état "Isolated". 0: le nœud est dans un autre état ("Edge-B" ou "Intermediate").
	22	0 ou 1	1: le nœud est à l'état "Edge-B". Lorsque les bits 23 et 22 sont de "1", le nœud est à l'état "Isolated". 0: le nœud est dans un autre état ("Edge-A" ou "Intermediate").
-	21	0	reserved (réservées)
Blocking status	20	0 ou 1	1: le port A ou B est le port de liaison logique descendante.
-	19 … 8	0x000.	reserved (réservées)
Port status	7	0 ou 1	1: le port A est à l'état "Port_Linkup".
	6	0 ou 1	1: le port B est à l'état "Port_Linkup".
	5	0 ou 1	1: le port A est la liaison physique montante.
	4	0 ou 1	1: le port B est la liaison physique montante.
-	3 … 0	0x0.	reserved (réservées)

5.2.4.3 LCA Priority

5.2.4.3.1 LCA_Priority

Ce champ indique la priorité dans la LCC-PDU reçue.

5.2.4.3.2 LCA_STadd

Ce champ indique l'adresse de station dans la LCC-PDU reçue.

5.2.4.3.3 LCA_MACadd

Ce champ indique l'adresse MAC dans la LCC-PDU reçue.

5.2.4.4 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.5 LCN-PDU

5.2.5.1 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x01.

5.2.5.2 RCL Status

Ce champ indique l'état du nœud qui envoie la LCN-PDU. Il a le même format qu'en 5.2.4.2 RCL Status, décrit dans le Tableau 13.

5.2.5.3 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.6 LNA-PDU

5.2.6.1 Reserved1.

Ce champ concerne la zone de bourrage. La valeur est 0x00.

5.2.6.2 RCL Status

Ce champ indique l'état du nœud qui envoie la LNA-PDU. Il a le même format qu'en 5.2.4.2 RCL Status, décrit dans le Tableau 13.

5.2.6.3 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.7 SCR-PDU

5.2.7.1 Entry Number

Ce champ indique le numéro que doit contenir l'entrée SCR_Event.

5.2.7.2 SendStat Time

Ce champ contient le temps d'envoi en secondes.

5.2.7.3 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0.

5.2.7.4 Event016

Ce champ contient les informations d'événement et spécifie chaque fabrication.

5.2.8 Cyclic_S-PDU

5.2.8.1 IP_header

Le codage des champs doit être conforme à RFC 791.

5.2.8.2 UDP_header

Le codage des champs doit être conforme à RFC 768.

5.2.8.3 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.8.4 Block_number121

Ce champ indique le numéro d'identification du bloc de données cycliques et spécifie chaque fabrication.

5.2.8.5 Block_num pattern

Ce champ est utilisé pour vérifier la Cyclic-PDU et spécifie chaque fabrication.

5.2.8.6 Block data1~21

Ce champ contient les données cycliques.

5.2.8.7 Cyclic_FCS

Ce champ est un code de vérification d'erreur à partir de Cyclic-PDU.

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.9 Cyclic_header

5.2.9.1 Cyclic_pattern

Ce champ est utilisé pour vérifier le Cyclic_header et spécifie chaque fabrication.

5.2.9.2 Total_Block

Ce champ indique le nombre de blocs dans Cyclic-PDU.

5.2.9.3 Src STaddress

Ce champ indique l'adresse MAC de son propre nœud.

5.2.9.4 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.10 Control-PDU

5.2.10.1 IP_header

Voir 5.2.8.1.

5.2.10.2 UDP_header

Voir 5.2.8.2.

5.2.10.3 CTL_data

Ce champ contient les données de la CTL-PDU.

5.2.11 RMTCTL-PDU

5.2.11.1 IP_header

Voir 5.2.8.1.

5.2.11.2 UDP_header

Voir 5.2.8.2.

5.2.11.3 RMTCTL_CMD

5.2.11.3.1 Reserved1.

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.11.3.2 Header Type

Ce champ est utilisé pour reconnaître la trame de commande à distance. Cette valeur est "RC" dans le code ASCII ("RC" = 0x5243).

5.2.11.3.3 Module ID

Ce champ indique le type de module.

5.2.11.3.4 Station number

Ce champ indique l'adresse de station du nœud de destination.

5.2.11.3.5 Code de Command

Ce champ spécifie le type de commande.

5.2.11.3.6 Length

Ce champ indique la longueur de RMTCTL-PDU.

5.2.11.3.7 RMTCTL Status

Ce champ indique le code d'état concernant les opérations de commande à distance.

5.2.11.3.8 RMTCTL Kind

Ce champ indique le type de module pour la lecture de mémoire distante.

5.2.11.3.9 RMTCTL Length

Ce champ indique la taille des données d'écriture ou de lecture de mémoire distante.

5.2.11.3.10 RMTCTL Address

Ce champ indique l'adresse d'écriture ou de lecture de mémoire distante.

5.2.11.3.11 Send Number

Ce champ est utilisé pour gérer la commande à distance par l'utilisateur de la FAL.

5.2.11.3.12 Reserved2.

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.11.3.13 Reserved3.

Il est réservé à un usage ultérieur. La valeur est 0x00000000.

5.2.11.3.14 RMTCTL data

Ce champ contient les données de la commande à distance.

5.2.12 INFO-PDU

5.2.12.1 IP_header

Voir 5.2.8.1.

5.2.12.2 UDP_header

Voir 5.2.8.2.

5.2.12.3 INFO_data

Ce champ contient les données de INFO-PDU.

5.3 Codage des éléments FAL-PDU de type N

5.3.1 Généralités

Les en-têtes de FAL-PDU de type N, à l'exclusion des données utilisateur, sont conformes à la convention d'ordre des octets de réseau TCP/IP dite "gros-boutiste".

5.3.2 En-tête FALAR-N

5.3.2.1 hd_h_type

Ce champ doit indiquer la spécification de type d'en-tête décrite dans le

Tableau 14. Spécifier "NUXM" ou "SNUV6" dans l'ASCII. Les PDU transmises dans le champ de données IPv4 spécifient "NUXM". Les PDU transmises dans le champ de données IPv6 spécifient "NUV6".

Tableau 14 – hd_h_type

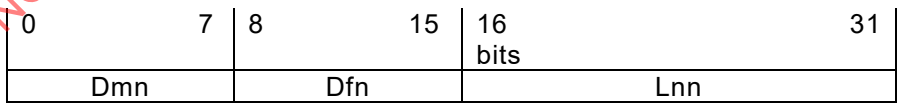
Valeur	Type de PDU
"NUXM"	PDU transmises dans le champ de données IPv4
"NUV6"	PDU transmises dans le champ de données IPv6

5.3.2.2 hd_ml

Ce champ doit indiquer la longueur du message total, y compris la longueur de l'en-tête FALAR-N (= 64 octets) et la longueur du message. (hd_ml = 64 + longueur du message).

5.3.2.3 hd_sa

Ce champ doit contenir l'adresse du nœud source (numéro de nœud logique) décrite à la Figure 2.



Dmn: numéro de domaine (=0)
Dfn: numéro du champ de données
Lnn: numéro de nœud logique

Figure 2 – hd_sa

Dmn:

Un domaine est défini comme étant le concept au-dessus du champ de données. Un domaine est constitué de plusieurs champs de données. Habituellement, un champ de données est défini sous forme d'emplacement local et connecté à d'autres domaines par l'intermédiaire de réseaux de communication étendus. Au sein d'un système, un nombre qui identifie de manière unique un domaine est appelé Domain Number (Dmn, numéro de domaine). Un Dmn unique compris entre 1 et 64 est attribué à chacun des domaines du système. Un numéro de domaine 0 (Dmn 0) ou un numéro de domaine non spécifié indique un domaine local (par défaut).

Dfn:

Data Field (DF, champ de données) est un emplacement où les messages ADP constitués d'informations de caractéristiques spécifiques peuvent circuler. La communication ADP est disponible entre les nœuds appartenant à un champ de données. Cela signifie que tout équipement d'un système doit appartenir à un ou plusieurs champs de données. Un champ de données est configuré par adresse réseau ou adresse de sous-réseau d'une adresse IP. Au sein d'un système décentralisé, un nombre qui identifie de manière unique un champ de données est appelé Data Field Number (Dfn, numéro de champ de données). Un Dfn unique est attribué à chacun des champs de données au sein du système, dans la plage comprise entre 1 et 255. Dfn 0 est réservé aux communications au sein du nœud en cours. Un réseau IPv4 doit être attribué avec IPv4 Dfn. Un réseau IPv6 doit être attribué avec IPv6 Dfn.

NOTE 1 Aucun champ de données ne peut être configuré sur plusieurs segments LAN. Chaque segment LAN est attribué avec Dfn.

NOTE 2 Un segment LAN peut contenir plusieurs champs de données.

Lnn:

Un nœud logique est un équipement dans un champ de données. Dans un champ de données, un nombre qui identifie de manière unique un nœud logique est appelé Logical Node Number (Lnn, numéro de nœud logique). Un Lnn unique compris entre 1 et 4 095 est attribué à chacun des appareils. Lnn 0 est réservé aux communications au sein du nœud en cours.

5.3.2.4 hd_da

Ce champ doit contenir l'adresse du groupe de multidiffusion de destination (Multicast Group Number) ou l'adresse du nœud de destination (Logical Node Number) décrites à la Figure 3.

Communication à multidestination

0 7	H	16 31 bits
Dmn	Dfn	Mgn

(Communication entre pairs)

0 7	8 15	16 31 bits
Dmn	Dfn	Lnn

Dmn: numéro de domaine (=0)
Dfn: numéro du champ de données
Mgn: numéro de groupe de multidiffusion
Lnn: numéro de nœud logique

Figure 3 – hd_da

Dmn,

Voir 5.3.2.3.

Dfn,

Voir 5.3.2.3.

Mgn,

Un groupe de multidiffusion (MCG) est un groupe de nœuds appartenant à un champ de données pour déterminer s'il faut recevoir des données de multidiffusion/diffusion provenant d'un nœud. Un groupe de multidiffusion doit être spécifié pour réaliser une transmission en multidiffusion. Tout nœud doit appartenir à un groupe de multidiffusion à utiliser pour une réception en multidiffusion. Un champ de données peut contenir plusieurs groupes de multidiffusion. Chaque nœud peut être ajouté à plusieurs groupes de multidiffusion dans la mesure où les groupes se trouvent dans un champ de données auquel le nœud appartient. Le Multicast Group Number (Mgn, numéro de groupe de multidiffusion) est un nombre qui identifie de manière unique un groupe de multidiffusion au sein d'un champ de données. Un Mgn unique compris entre 1 et 255 est attribué à chacun des groupes de multidiffusion du champ de données. Mgn 0 est réservé au système pour envoyer/recevoir des messages "alive".

Un Mgn de MulticastData-PDU doit être différent d'un Mgn de RetransX PDU.

Ici, RetransX PDU indique soit RetransEnq-PDU, soit RetransCinfirm-PDU, soit RetransNak-PDU.

Un Mgn de communication multidiffusion avec retransmission est également différent d'un Mgn de communication multidiffusion (sans retransmission) et d'un RetransX-PDU.

Lnn,

Voir 5.3.2.3.

Le Tableau 15 indique le Mgn ou le Lnn utilisés par chaque PDU.

Tableau 15 – Utilisation de Mgn ou Lnn

Champ (Mgn, Lnn)		Description (PDU)
Mgn	**Lnn**	
Applicable	N/A	CyclicData-PDU
Applicable	N/A	MulticastData-PDU
N/A	Applicable	PtoPData-PDU
Applicable (=0)	N/A	Aliveinfo-PDU
Applicable (=0)	N/A	Aliveinfo6-PDU
Applicable	Applicable	Inq-PDU
Applicable	N/A	Ninq-PDU
N/A	Applicable	Reply-PDU
Applicable	N/A	RetransEnq-PDU
Applicable	N/A	RetransConfirm-PDU
Applicable	N/A	RetransNak-PDU

5.3.2.5 hd_v_seq

Ce champ doit indiquer l'identification de version du numéro de séquence du message de transmission (hd_seq). Ce champ spécifie normalement l'heure à laquelle le numéro de séquence du message de transmission (hd_seq) est initialisé. L'objectif de ce champ est de permettre à un nœud destinataire de détecter lorsqu'un nœud émetteur vient d'initialiser le numéro de séquence. La nouvelle valeur de réglage de hd_v_seq doit être différente de son ancienne valeur.

5.3.2.6 hd_seq

5.3.2.6.1 hd_seq

Ce champ doit indiquer le numéro de séquence du message de transmission. La plage de valeurs va de 0x00000001 à 0x7FFFFFFF, et elle est utilisée de manière cyclique.

5.3.2.6.2 Gestion des numéros de version et des numéros de séquence de transmission

5.3.2.6.2.1 Gestion au niveau d'un nœud de transmission

Attribution:

Les numéros de version (V_SEQ) doivent être attribués à chaque nœud. Les numéros de version sont attribués avec des horodatages. La valeur non nulle de l'horodatage doit être différente de la valeur V_SEQ précédente.

a) Communication à multidestination

   Chaque nœud possède des numéros de séquence de transmission (S_SEQ). Les numéros de séquence de transmission (S_SEQ) doivent être gérés par chaque Dfn (numéro de champ de données), par chaque Mgn (numéro de groupe de multidiffusion) et par chaque niveau de priorité de message.

b) Communication entre pairs

   Chaque nœud possède des numéros de séquence de transmission (S_SEQ). Les numéros de séquence de transmission (S_SEQ) doivent être gérés par chaque Dfn, par la connexion pour chaque nœud et par chaque niveau de priorité de message.

Initialisation:

a) Communication à multidestination

   Lors de l'ouverture d'un port de communication multidiffusion, V_SEQ correspondant au port doit être défini sur l'heure d'ouverture et S_SEQ correspondant au port doit être défini sur 0.

b) Communication entre pairs

   Lors de l'établissement de la connexion TCP, V_SEQ correspondant à la connexion doit être défini sur l'heure d'établissement et S_SEQ correspondant à la connexion doit être défini sur 0.

Transmission:

Lors de la transmission d'un message, V_SEQ et S_SEQ mis à jour sont respectivement définis sur hd_v_seq et hd_seq dans l'en-tête FALAR_N. S_SEQ est mis à jour comme suit:

- Si S_SEQ est différent de 0x7FFFFFFF, S_SEQ est incrémenté de 1.
- Si S_SEQ est égal à 0x7FFFFFFF, S_SEQ est remis à 1 sans incrément.

5.3.2.6.2.2 Gestion au niveau d'un nœud destinataire

Attribution:

a) Communication à multidestination

Chaque nœud possède les numéros de séquence de réception (R_SEQ) reçus en dernier et un numéro de version de réception (R_V_SEQ) reçu en dernier. Le R_SEQ doit être géré par chaque Dfn (numéro de champ de données), par chaque Mgn (numéro de groupe de multidiffusion), par chaque nœud source et par chaque niveau de priorité de message.

b) Communication entre pairs

Chaque nœud possède les numéros de séquence de réception (R_SEQ) reçus en dernier et un numéro de version de réception (R_V_SEQ) reçu en dernier. Le R_SEQ doit être géré par chaque Dfn (numéro de champ de données), par la connexion pour chaque nœud et par chaque niveau de priorité de message.

Initialisation:

a) Communication à multidestination

Lors de l'ouverture d'un port de communication multidiffusion, R_V_SEQ et R_SEQ doivent être initialisés à 0.

b) Communication entre pairs

Lors de l'établissement de la connexion TCP, R_V_SEQ correspondant à la connexion et R_SEQ correspondant à la connexion doivent être initialisés à 0.

Réception:

A la réception d'un message, R_V_SEQ et R_SEQ sont récupérés par hd_sa, par hd_seq, par hd_pri et par hd_v_seq dans l'en-tête FALAR_N.

NOTE Tout message avec hd_v_seq = 0, hd_seq = 1, hd_cbn = 1 et hd_tbn = 1 est transmis à une application sans vérifier le numéro de séquence.

5.3.2.6.2.3 Vérification des numéros de séquence pour un message de réception

1) Si R_V_SEQ est égal à 0,

le message est reçu sans condition et hd_v_seq et hd_seq dans l'en-tête FALAR_N actualisent respectivement R_V_SEQ et R_SEQ.

2) Si R_V_SEQ n'est pas égal à 0 et hd_v_seq est égal à R_V_SEQ,

R_SEQ et hd_seq sont comparés pour détecter un doublon de message ou des messages perdus. Le Tableau 16 et la Figure 4 présentent les conditions de vérification des numéros de séquence pour un message de réception.

3) communication

a) Communication à multidestination

Si R_V_SEQ n'est pas égal à 0 et que hd_v_seq n'est pas égal à R_V_SEQ, les valeurs hd_v_seq et hd_seq mettent respectivement à jour les valeurs R_V_SEQ et R_SEQ. Lors de la détection d'un doublon de message, un nœud destinataire rejette le message reçu.

b) Communication entre pairs

Si R_V_SEQ n'est pas égal à 0 et hd_v_seq n'est pas égal à R_V_SEQ, un nœud destinataire déconnecte la connexion TCP et rejette le message reçu.

Tableau 16 – Conditions détaillées pour la vérification des numéros de séquence du message de réception

No.	Conditions	Résultat
1	R_SEQ != 0x7fffffff && R_SEQ + 1 == SEQ	Réception d'un message normal
2	R_SEQ == 0x7fffffff && SEQ == 1	Réception d'un message normal
3	R_SEQ > N1 && R_SEQ – N1 < SEQ <= R_SEQ	Réception d'un doublon de message
4	R_SEQ <= N1 && (0 < SEQ <= R_SEQ ‖ 0x7fffffff – (N1 – R_SEQ) < SEQ <= 0x7fffffff)	Réception d'un doublon de message
5	Autres conditions	Message manquant
Légende SEQ: valeur de hd_seq dans l'en-tête FALAR_N N1: numéro de séquence de transmission supposé être un doublon. Cette valeur est celle mise en œuvre.		

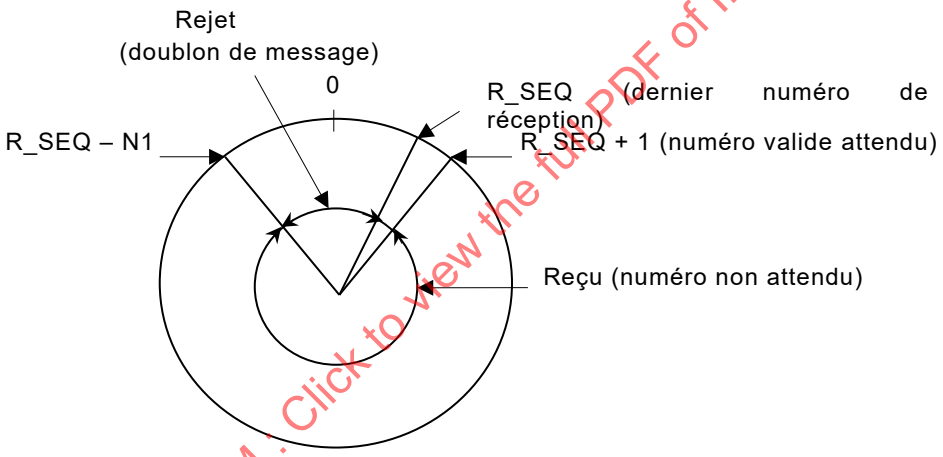


Figure 4 – Numéro de séquence valide pour un message de réception

5.3.2.7 hd_m_ctl

Ce champ doit spécifier des informations sur la commande de transmission de message décrite à la Figure 5.

0	1	2	3	4	5	6 31 bits
MLT	ONE	INQ	RPL	NIQ	MCR	Rsv

Légende

MLT: communication multidiffusion (= 0x80000000)

ONE: communication entre pairs (un à un) (= 0x40000000)

INQ: communication de demande (= 0x20000000)

RPL: communication de réponse (= 0x10000000)

NIQ: communication de N-demande (= 0x08000000)

MCR: communication multidiffusion avec commande de retransmission (= 0x04000000)

Rsv: reserved (réservées)

Figure 5 – hd_m_ctl

Le Tableau 17 représente les bits valides du champ hd_m_ctl dans chaque PDU.

Tableau 17 – Bits valides de hd_m_ctl

PDU	Bits valides de hd_m_ctl
CyclicData-PDU	0x80000000.
MulticastData-PDU	0x80000000 (type de communication multidiffusion)
	0x84000000. (Communication multidiffusion avec type de retransmission)
PtoPData-PDU	0x40000000.
Aliveinfo-PDU	0x80000000.
Aliveinfo6-PDU	0x80000000.
Inq-PDU	0xA0000000 (Type de communication multidiffusion)
	0x60000000 (Type de communication entre pairs)
Ninq-PDU	0x88000000.
Reply-PDU	0x50000000.
RetransEnq-PDU	0x84000000.
RetransConfirm-PDU	0x84000000.
RetransNak-PDU	0x84000000.

5.3.2.8 hd_inqid

Ce champ doit spécifier l'identifiant de demande/réponse, qui est constitué de inqid_inq_sa, inqid_tr_adr et inqid_id_seq. Ces trois champs sont respectivement décrits en 5.3.2.20, 5.3.2.21 et 5.3.2.22. Ce champ hd_inqid est valide dans Inq-PDU, Ninq-PDU et Reply-PDU.

5.3.2.9 hd_tcd

Ce champ doit spécifier le code de transaction. Toutes les transactions ont des codes de transaction attribués (TCDS) pour les identifier. Un TCD représente des données de transaction. Chaque programme d'application transmet des données avec un TCD spécifié dans un champ de données. D'autre part, chaque programme d'application reçoit des données avec un TCD indiquant un champ de données. Des TCD uniques doivent être définis pour chaque champ de données. Les TCD sont classés séparément selon qu'il s'agit de transactions utilisateur ou de transactions système, comme indiqué ci-dessous.

- TCD pour les transactions utilisateur: 1 à 59 999 (définis par l'utilisateur)
- TCD pour les transactions système: 60 000 à 65 534 (réservés au système)

Une transaction système est créée lorsque le système détecte un événement (tel qu'une erreur ou un changement d'état interne) et informe le programme utilisateur. Le Tableau 18 indique le TCD spécifié.

Tableau 18 – TCD spécifié

TCD spécifié	Description
60 003	Rapport d'état "alive"
60 061	Commande de retransmission
60 027	Transfert de fichier
60 056	Transfert de mémoire cyclique
60 058	Transfert de mémoire cyclique (type rapide)

5.3.2.10 hd_ver

Ce champ doit spécifier un identifiant de version de programme. Une valeur courante est 0.

5.3.2.11 hd_pkind

Ce champ doit spécifier un identifiant de transaction tel que représenté dans le Tableau 19. Ce champ est valide pour une communication un à un (PtoPData-PDU) à l'aide d'un réseau local duplex.

Tableau 19 – hd_pkind

Valeur	Description
1	Demande d'acquittement (ACK.req): demande au nœud de destination (Dfn, Lnn) d'informer un numéro de séquence de paquet de réception (R-PSEQ)
2	Réponse d'acquittement (ACK.inform): informe le numéro de séquence de paquet de réception (R-PSEQ) au nœud de demande d'acquittement.
0	Autres (NoACK)

5.3.2.12 hd_pseq

5.3.2.12.1 hd_pseq

Ce champ doit spécifier un numéro de séquence de paquet. La plage de valeurs est comprise entre 0x1 et 0x FFFFFFFF, et est utilisée de manière cyclique. Ce champ est efficace pour une communication multidiffusion avec retransmission, ainsi que pour une communication un à un à l'aide d'un réseau local duplex, comme indiqué dans le Tableau 20.

Tableau 20 – PDU avec un hd_pseq efficace

PDU	Efficacité	Description	Valeur de hd_pseq
MulticastData-PDU	Efficace	Communication multidiffusion avec retransmission	0x1 – 0x FFFFFFFF
PtoPData-PDU	Efficace	Communication un à un (PtoPData-PDU) à l'aide d'un réseau local duplex	0x1 – 0x FFFFFFFF
Autre PDU	Inefficace	Autres (y compris communication un à un (PtoPData-PDU) à l'aide d'un seul LAN)	0 (recommandé)

5.3.2.12.2 Gestion au niveau d'un nœud de transmission

Attribution:

Chaque nœud possède des informations de gestion (S_PSEQ) d'un numéro de séquence de paquets de transmission.

a) Communication multidiffusion avec retransmission

   Un S_PSEQ doit être attribué par Dfn (numéro de champ de données) et par Mgn (numéro de groupe de multidiffusion).

b) Communication un à un à l'aide d'un réseau local duplex

   Un S_PSEQ doit être attribué par Dfn (numéro de champ de données) et par la connexion TCP pour chaque nœud.

Initialisation:

a) Communication multidiffusion avec retransmission

   Lors de l'ouverture d'un port de communication multidiffusion, le S_PSEQ correspondant au port doit être défini sur 0.

b) Communication un à un à l'aide d'un réseau local duplex

   Lors de l'établissement de la connexion TCP, le S_PSEQ correspondant à la connexion doit être défini sur 0.

Transmission:

Lors de la transmission d'un paquet, le S_PSEQ mis à jour est défini sur hd_pseq dans l'en-tête FALAR_N.

S_PSEQ est mis à jour comme suit.

– Si S_PSEQ est différent de 0xFFFFFFFF, S_PSEQ est incrémenté de 1.

– Si S_PSEQ est égal à 0xFFFFFFFF, S_PSEQ est remis à 1 sans incrément.

NOTE Dans le cas d'une communication multidiffusion avec retransmission, ces trois PDU (RetransEnq-PDU, RetransConfirm-PDU, RetransNak-PDU) n'utilisent pas le champ hd_pseq.

5.3.2.12.2.1 Gestion au niveau d'un nœud destinataire

Attribution:

Chaque nœud possède des informations de gestion (R_PSEQ) d'un numéro de séquence de paquets reçu.

a) Communication multidiffusion avec retransmission

Un R_PSEQ doit être géré par Dfn (numéro de champ de données), par Mgn (numéro de groupe de multidiffusion) et par nœud source.

b) Communication un à un à l'aide d'un réseau local duplex

Un R_PSEQ doit être attribué par Dfn (numéro de champ de données) et par la connexion TCP pour chaque nœud.

Initialisation:

a) Communication multidiffusion avec retransmission

- Lors de l'ouverture d'un port de communication multidiffusion, le S_PSEQ correspondant au port doit être défini sur 0.
- En cas d'échec d'une demande de retransmission, le R_PSEQ correspondant au port doit être défini sur 0.

b) Communication un à un à l'aide d'un réseau local duplex

Lors de l'établissement de la connexion TCP, le R_PSEQ correspondant à la connexion doit être défini sur 0.

Réception:

A la réception d'un paquet, le R_PSEQ est récupéré par hd_pseq dans l'en-tête FALAR_N.

5.3.2.12.2.2 Vérification des numéros de séquence de paquets

a) Communication multidiffusion avec retransmission

A la réception d'un paquet, une comparaison entre R_PSEQ et hd_pseq dans l'en-tête FALAR_N est effectuée.

R_PSEQ est mis à jour comme suit.

1) Si R_PSEQ est égal à 0,

   le paquet est reçu sans condition et hd_pseq dans l'en-tête FALAR_N est défini sur R_PSEQ.

2) Si R_PSEQ est différent de 0,

   R_PSEQ et hd_pseq sont comparés pour détecter un doublon de paquet ou des paquets perdus. Le Tableau 21 et la Figure 6 illustrent les conditions de vérification des numéros de séquence de paquets.

Tableau 21 – Conditions détaillées de vérification des numéros de séquence de paquets de réception (communication multidiffusion avec retransmission)

No.	Conditions	Résultat
1	R_PSEQ != 0xffffffff && R_PSEQ + 1 == PSEQ	Réception de paquet normal (R_PSEQ = PSEQ)
2	R_PSEQ == 0xffffffff && PSEQ == 1	Réception de paquet normal (R_PSEQ = PSEQ)
3	R_PSEQ > N1 && R_PSEQ – N1 < PSEQ <= R_PSEQ	Réception d'un doublon de paquet
4	R_PSEQ <= N1 && (0 < PSEQ <= R_PSEQ || 0xffffffff – (N1 – R_PSEQ) < PSEQ <= 0xffffffff)	Réception d'un doublon de paquet
5	Autres conditions	Paquets perdus
PSEQ: valeur de hd_pseq dans l'en-tête FALAR_N. N1: numéro de séquence de paquet de transmission supposé être un doublon. Cette valeur est celle mise en œuvre.		

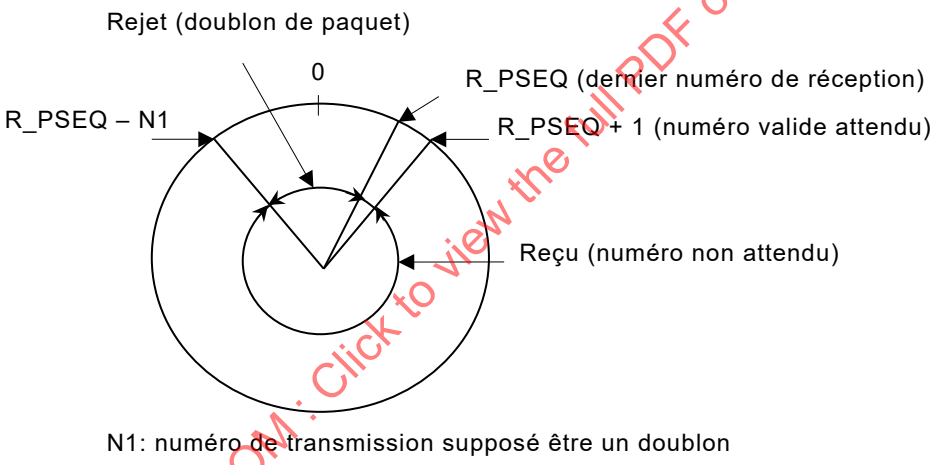


Figure 6 – Numéro de séquence de paquet de réception valide

b) Communication un à un à l'aide d'un réseau local duplex

1) Si R_PSEQ est égal à 0 et hd_pseq dans l'en-tête FALAR_N est égal à 1,
définir la valeur de hd_pseq sur R_PSEQ (premier paquet normal reçu).

2) Si R_PSEQ est égal à 0 et hd_pseq dans l'en-tête FALAR_N n'est pas égal à 1,
le nœud destinataire rejette le paquet reçu et déconnecte la connexion TCP.

3) Si R_PSEQ est différent de 0,
R_PSEQ et hd_pseq sont comparés pour détecter un doublon de paquet ou des paquets perdus. Le Tableau 22 représente les conditions de vérification des numéros de séquence de paquets.

Tableau 22 – Conditions détaillées de vérification des numéros de séquence de paquets

No.	Conditions	Résultat
1	R_PSEQ != 0xffffffff && R_PSEQ + 1== PSEQ	Réception de paquet normal (R_PSEQ = PSEQ)
2	R_PSEQ == 0xffffffff && PSEQ == 1	Réception de paquet normal (R_PSEQ = PSEQ)
3	Autres conditions	Réception d'un doublon de paquet ou de paquets perdus
Communication un à un à l'aide d'un réseau local duplex		

Lors de la détection d'un doublon de paquet ou de paquets perdus, le nœud destinataire rejette le paquet reçu.

5.3.2.13 hd_mode

5.3.2.13.1 Vue d'ensemble

Ce protocole permet d'ajouter un mode de nœud à un message pour indiquer si le nœud est en ligne ou soumis à essai. Sachant qu'un nœud peut commander la transmission/réception de messages de manière sélective en fonction du mode de discrimination (en ligne/essai), il peut éliminer tout message d'essai interruptif lorsqu'il est en cours d'essai.

Lorsqu'un nœud est soumis à un essai de simulation, il peut recevoir soit un message en ligne, soit un message d'essai.

5.3.2.13.2 Modes de message

Un mode de message est attribué par chaque message. Il existe deux types de modes de message:

1) Mode en ligne,
2) Mode d'essai.

5.3.2.13.3 Modes de nœud

Un mode de nœud est attribué par chaque Dfn (numéro de champ de données). Il existe deux types de modes de nœud:

1) Mode en ligne,
2) Mode d'essai.

Un nœud en mode en ligne doit définir le mode de message en ligne sur le champ hd_mode.

Un nœud en mode d'essai doit définir le mode de message d'essai sur le champ hd_mode.

5.3.2.13.4 Commande d'émission/de réception de message

Le Tableau 23 représente les relations entre les modes de message et les modes de nœud. La commande de message utilisant ces relations est exécutée.

Tableau 23 – Relation entre transmission/réception de messages

Mode de message	Mode de nœud			
	En ligne		Essai	
	Transmission	Réception	Transmission	Réception
En ligne	Efficace	Efficace	Inefficace	Efficace
Essai	Inefficace	Rejet	Efficace	Efficace
Efficace: transmission/réception d'un message; inefficace: pas de transmission de message				

5.3.2.13.5 hd_mode

Ce champ doit spécifier un mode de message tel que représenté dans le Tableau 24.

Tableau 24 – hd_mode

Valeur	Description
0	Mode en ligne
1	Mode d'essai

5.3.2.14 hd_pver

Ce champ doit spécifier une version de protocole. Une valeur courante est 1.

5.3.2.15 hd_pri

Ce champ doit spécifier un niveau de priorité de message tel que représenté dans le Tableau 25.

Tableau 25 – Niveau de priorité des messages

Niveau de priorité[a]	Description
0	Sans commande de priorité (aucune commande de priorité n'est mise en œuvre)
1	Message avec le niveau de priorité le plus élevé (1)
2	Message avec le niveau de priorité 2
3	Message avec le niveau de priorité 3
4	Message avec le niveau de priorité 4
5	Message avec le niveau de priorité 5
6	Message avec le niveau de priorité 6
7	Message avec le niveau de priorité le plus faible (7)
[a] Plus les chiffres sont élevés, moins le message est prioritaire.	

5.3.2.16 hd_cbn

5.3.2.16.1 Vue d'ensemble de la fragmentation et de l'assemblage des messages

Les spécifications limitent la taille maximale des messages utilisateur à 16 Ko pour la communication de type TCP/UDP, tandis que la taille maximale de transfert (MTU) pour les LAN est limitée à 1 500 octets selon la norme Ethernet.

Pour la communication de type TCP, les en-têtes IPv4 et TCP utilisent respectivement 20 octets et l'en-tête FALAR-N utilise 64 octets, de sorte que la taille de stockage de données libres disponible peut être calculée sous la forme (MTU – 104) octets.

D'autre part, l'en-tête IPv6 utilise 40 octets et l'en-tête FALAR-N utilise 64 octets, de sorte que la taille de stockage de données libres disponible peut être calculée sous la forme (MTU – 124) octets.

Pour la communication de type UDP, la taille de stockage de données libres disponible peut être calculée respectivement sous la forme (MTU – 92) octets ou (MTU – 112) octets.

NOTE La zone utilisateur libre maximale dans le segment TCP peut être inférieure à 1 460 octets ou inférieure à 1 440 octets par le biais d'une négociation sur la taille du segment TCP lors de l'établissement d'une connexion TCP.

Si une taille de message est supérieure à α octets, le message doit être divisé en plusieurs PDU avec des en-têtes FALAR-N ajoutés respectivement. Le Tableau 26 indique la valeur de α. Les PDU sont envoyées au champ de données dans l'ordre. Le nœud destinataire reçoit les PDU et les assemble en un seul message initial.

Tableau 26 – Valeur de α

Type de communication	Valeur de α
IPv4 + TCP	MTU – 104
IPv4 + UDP	MTU – 92
IPv6 + TCP	MTU – 124
IPv6 + UDP	MTU – 112

5.3.2.16.2 Informations d'en-tête pour la fragmentation et l'assemblage

Le Tableau 27 représente un exemple d'informations d'en-tête pour une fragmentation de message UDP.

Le Tableau 28 représente un exemple d'informations d'en-tête pour une fragmentation de message TCP. hd_cbn, hd_tbn et hd_bsize font référence à 5.3.2.16, 5.3.2.18 et 5.3.2.19 respectivement.

Tableau 27 – Exemple d'informations d'en-tête pour une fragmentation de message UDP

Exemple de fragmentation	Fragmenté en quatre paquets				Aucune fragmentation
	1er paquet	2e paquet	3e paquet	4e paquet	Paquet unique
hd_seq	100	100	100	100	101
hd_cbn / hd_tbn	1– 4	2– 4	3– 4	4– 4	1– 1
hd_bsize	1 472	1 472	1 472	340	1 024
hd_ml	4 564	4 564	4 564	4 564	1 024

Tableau 28 – Exemple d'informations d'en-tête pour une fragmentation de message TCP

Exemple de fragmentation	Fragmenté en quatre paquets				Aucune fragmentation
	1er paquet	2e paquet	3e paquet	4e paquet	Paquet unique
hd_seq	100	100	100	100	101
hd_cbn / hd_tbn	1– 4	2– 4	3– 4	4– 4	1– 1
hd_bsize	1 460	1 460	1 460	376	1 024
hd_ml	4 564	4 564	4 564	4 564	1 024

5.3.2.17 hd_cbn

Ce champ doit spécifier un numéro de bloc de PDU fragmentées en cours (supérieur à 1).

5.3.2.18 hd_tbn

Ce champ doit spécifier un nombre total de blocs de PDU fragmentées (supérieur à 1).

5.3.2.19 hd_bsize

Ce champ doit spécifier une taille de bloc de PDU (y compris la taille de l'en-tête FALAR-N).

5.3.2.20 inqid_inq_sa

Ce champ doit spécifier une adresse source. Cette valeur hd_inqid.inq_sa est spécifiée dans le Tableau 29.

Tableau 29 – Valeur inqid_inq_sa

Eléments du champ hd_inqid	Description		
	Inq-PDU, Ninq-PDU	Reply-PDU	Autre PDU
Inqid_inq_sa	Adresse source (= hd_sa)	Même valeur que pour inqid_inq_sa de l'Inq-PDU ou la Ninq-PDU reçue	0

5.3.2.21 inqid_tr_adr

Ce champ doit indiquer le numéro de séquence de demande-transaction du message de demande/N-demande. La plage de valeurs va de 0x00000001 à 0x0FFFFFFF, et elle est utilisée de manière cyclique. Lorsqu'un message de N-demande est transmis, cette valeur doit être incrémentée de 1.

Dans le cas d'une retransmission du message de N-demande, la même valeur doit être utilisée.

Cette valeur inqid_tr_adr est gérée séparément pour chaque inqid_id_seq. Voir 5.3.2.22.

Cette valeur inqid_ tr_adr est spécifiée dans le Tableau 30.

Tableau 30 – Valeur inqid_tr_adr

Eléments du champ hd_inqid	Description			
	Inq-PDU	Ninq-PDU	Reply-PDU	Autre PDU
Inqid_tr_adr	any	1 … 0x0FFFFFFF.	Même valeur que pour inqid_tr_adr de l'Inq-PDU ou la Ninq-PDU reçue	0

5.3.2.22 inqid_id_seq

Ce champ doit spécifier le numéro de séquence de demande-identification. Chaque message de demande/N-demande exige une valeur unique inqid_inq_seq dans le nœud source. Cette valeur inqid_id_seq est spécifiée dans le Tableau 31.

Tableau 31 – Valeur inqid_inq_seq

Eléments du champ hd_inqid	Description		
	Inq-PDU, Ninq-PDU	Reply-PDU	Autre PDU
Inqid_id_seq	1 … MaxIdSeq MaxIdSeq = 0x0400 (valeur par défaut = 0x0080) Ne pas utiliser la même valeur simultanément.	Même valeur que pour inqid_id_seq de l'Inq-PDU ou la Ninq-PDU reçue	0

Le Tableau 32 représente la relation entre inqid_id_seq et inqid_tr.

Tableau 32 – Relation entre inqid_id_seq et inqid_tr_adr

Eléments du champ hd_inqid	Description
Inqid_id_seq	**hd_inqid.tr_adr**
1	1 … 0x0FFFFFFF.
2	1 … 0x0FFFFFFF.
. .	1 … 0x0FFFFFFF.
128	1 … 0x0FFFFFFF.

5.3.3 CyclicData-PDU

5.3.3.1 Généralités

CyclicData-PDU transmet la mémoire de transfert cyclique dans un nœud à tous les nœuds appartenant à un groupe de multidiffusion spécifié, de manière cyclique. Cette PDU utilise un port UDP.

5.3.3.2 falArHeader

Voir 5.3.2.

5.3.3.3 tmid

La mémoire de transfert cyclique possède des identifiants de mémoire de transfert uniques (TMID) définis pour chaque champ de données. Ce champ doit indiquer un tmid dans un champ de données cible.

Valeur Tmid: 1 à 8 par champ de données.

5.3.3.4 blockNumber

Ce protocole transfère des unités de blocs de mémoire I de transfert cyclique, chacune ayant une taille présumée de 64 octets. Ce champ blockNumber doit indiquer le premier numéro de bloc de cyclicData dans la PDU.

5.3.3.5 blockCount

Ce champ doit indiquer un certain nombre de blocs de cyclicData dans la PDU.

5.3.3.6 cyclicData

Ce champ doit contenir des données de mémoire de transfert cyclique.

5.3.4 MulticastData-PDU

5.3.4.1 Généralités

MulticastData-PDU transmet une donnée de multidiffusion utilisateur à tous les nœuds appartenant à un groupe de multidiffusion spécifié. Cette PDU utilise un port UDP. Une communication multidiffusion avec retransmission utilise également cette MulticastData-PDU pour éviter la perte de messages de réception.

5.3.4.2 falArHeader

Voir 5.3.2.

5.3.4.3 multicastData

Ce champ contient les données de multidiffusion utilisateur.

5.3.5 PtoP Data-PDU

5.3.5.1 Généralités

PTop Data-PDU transmet une donnée utilisateur au nœud spécifié. Cette PDU utilise le port TCP.

5.3.5.2 falArHeader

Voir 5.3.2.

5.3.5.3 pointToPointData

Ce champ contient les données point à point de l'utilisateur.

5.3.6 Aliveinfo-PDU

5.3.6.1 Généralités

Dans l'environnement de réseau IPv4, AliveInfo-PDU transmet périodiquement l'état de son propre nœud à tous les nœuds appartenant à un champ de données spécifié. Cette PDU utilise le port UDP.

5.3.6.2 falArHeader

Voir 5.3.2.

5.3.6.3 al_nd_name

Ce champ doit spécifier un nom de nœud (une chaîne ASCII, jusqu'à dix caractères).

5.3.6.4 al_os_name

Ce champ doit spécifier un nom d'appareil fournisseur (une chaîne ASCII, jusqu'à dix caractères).

5.3.6.5 al_tm_out

Ce champ doit spécifier le temps d'intervalle de surveillance (en secondes) pour un message "alive". Cette surveillance redémarre chaque fois qu'un nouveau message "alive" arrive depuis un nœud surveillé. Si le nœud de surveillance n'a pas reçu de message "alive" de la part du nœud surveillé dans le délai spécifié par al_tm_out, le nœud de surveillance reconnaît qu'un nœud surveillé est hors tension.

5.3.6.6 al_msgserno

Ce champ doit spécifier le numéro de séquence de messages d'un message "alive". La plage de valeurs va de 0x0001 à 0x7FFF, et elle est utilisée de manière cyclique. Lors de la vérification de numéros de séquence pour un message "alive" reçu, hd_v_seq et al_msgserno sont tous deux efficaces.

5.3.6.7 al_mode

Ce champ doit spécifier le type d'un message "alive", comme représenté dans le Tableau 33.

Tableau 33 – Type d'un message "alive"

Valeur	Description	Remarque
1	Rapport normal	Obligatoire,
2	Notification (Arrêt)	Option Cause de l'arrêt de transmission d'un message "alive": Arrêt de l'équipement
3	Notification (Maintenance)	Option Cause de l'arrêt de transmission d'un message "alive": Maintenance de l'équipement

5.3.6.8 al_protocol

Ce champ doit spécifier le type d'un protocole de message "alive", comme représenté dans le Tableau 34.

Tableau 34 – Type d'un protocole de message "alive"

Valeur	Description	Remarque
1	Catégorie N_f	Protocole système décentralisé autonome dans la présente spécification
2 à 3 =	reserved (réservées)	Pour les autres protocoles existants
4	Catégorie N_l	Protocole système décentralisé autonome (type simplifié)
5 à 255 =	reserved (réservées)	Pour utilisation future

5.3.6.9 al_tg_cmn_cnt

Ce champ doit spécifier le nombre de fois qu'une information étendue a été modifiée (al_ExtentionInfo).

5.3.6.10 al_tg_cflag

Ce champ est réservé à un usage ultérieur.

5.3.6.11 al_tg_max

Ce champ doit spécifier le nombre maximal de tâches. La plage de valeurs va de 0x0000 à 0x0053.

5.3.6.12 al_tg_usecnt

Ce champ doit spécifier le nombre d'informations sur les tâches surveillées.

5.3.6.13 al_chg_time

Ce champ doit spécifier l'heure à laquelle l'état d'un nœud a été modifié. Il est recommandé de fixer le temps défini sous forme de nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00:00 GMT.

Si le réglage GMT n'est pas disponible, spécifier 0. Le temps de chaque al_mode est tel que représenté dans le Tableau 35.

Tableau 35 – Temps de chaque al_mode

Valeur d'al_mode	Description
1	Heure à laquelle l'état du nœud est passé de l'état "dead" à l'état "alive".
2	Heure à laquelle l'état du nœud est passé de l'état "alive" à l'état "acceptance of the shutdown request".
3	Heure à laquelle l'état du nœud est passé de l'état "alive" à l'état "acceptance of the maintenance request".

5.3.6.14 al_ipv4addr1.

Ce champ doit spécifier l'adresse IPv4 de LAN1.

5.3.6.15 al_ipv4addr2.

Ce champ doit spécifier l'adresse IPv4 de LAN2.

Spécifier 0 en cas de LAN unique (c'est-à-dire pas de LAN duplex).

5.3.6.16 al_ver

Ce champ doit spécifier la version de message "alive" (= 1).

5.3.6.17 al_ta_chgalvstat

Ce champ doit spécifier les changements d'état des tâches, comme représenté dans le Tableau 36.

Tableau 36 – Changement d'état des tâches

Valeur	Description
0	pas de changement
1	Changement d'état (de l'état "dead" à l'état "alive")
2	Changement d'état (de l'état "alive" à l'état "dead")

5.3.6.18 al_ta_chginfostat

Ce champ doit spécifier les changements de contenu des tâches, comme représenté dans le Tableau 37.

Tableau 37 – Changement de contenu des tâches

Valeur	Description
0	Pas de changement de contenu des tâches
1	Changement de contenu des tâches

5.3.6.19 al_ta_tid

Ce champ doit spécifier l'identification des tâches.

5.3.6.20 al_ta_data

Ce champ indique les informations d'état des tâches.

5.3.6.21 al_ExtentionInfo

Ce champ est une zone d'information étendue qui permet à l'utilisateur de définir librement des données.

5.3.7 Aliveinfo6-PDU

5.3.7.1 Généralités

Dans l'environnement de réseau IPv6, AliveInfo-PDU transmet périodiquement l'état de son propre nœud à tous les nœuds appartenant à un champ de données spécifié. Cette PDU utilise le port UDP.

5.3.7.2 falArHeader

Voir 5.3.2.

5.3.7.3 al_nd_name

Voir 5.3.6.3.

5.3.7.4 al_os_name

Voir 5.3.6.4.

5.3.7.5 al_tm_out

Voir 5.3.6.5.

5.3.7.6 al_msgserno

Voir 5.3.6.6.

5.3.7.7 al_mode

Voir 5.3.6.7.

5.3.7.8 al_protocol

Voir 5.3.6.8.

5.3.7.9 al_tg_cmn_cnt

Voir 5.3.6.9.

5.3.7.10 al_tg_cflag

Voir 5.3.6.10.

5.3.7.11 al_tg_max

Voir 5.3.6.11.

5.3.7.12 al_tg_usecnt

Voir 5.3.6.12.

5.3.7.13 al_chg_time

Voir 5.3.6.13.

5.3.7.14 al_ipv4addr1.

Ce champ fictif doit spécifier 0.

5.3.7.15 al_ipv4addr2.

Ce champ fictif doit spécifier 0.

5.3.7.16 al_ver

Ce champ doit spécifier la version du message "alive" (= 2).

5.3.7.17 al_ipv6addr1.

Ce champ doit spécifier l'adresse IPv6 de LAN1.

5.3.7.18 al_ipv6addr2.

Ce champ doit spécifier l'adresse IPv6 de LAN2.

Spécifier 0 en cas de LAN unique (c'est-à-dire pas de LAN duplex).

5.3.7.19 al_ta_chgalvstat

Voir 5.3.6.17.

5.3.7.20 al_ta_chginfostat

Voir 5.3.6.18.

5.3.7.21 al_ta_tid

Voir 5.3.6.19.

5.3.7.22 al_ta_data

Voir 5.3.6.20.

5.3.7.23 al_ExtentionInfo

Voir 5.3.6.21.

5.3.8 Inq-PDU

5.3.8.1 Généralités

Inq-PDU transmet un message de demande aux nœuds appartenant à un groupe de multidiffusion spécifié.

Chaque nœud destinataire transmet Reply-PDU au nœud émetteur. Deux utilisations sont possibles:

a) Demande à tous les nœuds utilisant le port UDP,
b) Demande au nœud spécifique utilisant le port TCP.

5.3.8.2 falArHeader

Voir 5.3.2.

5.3.8.3 inquirydata

Ce champ contient les données de demande utilisateur.

5.3.9 Ninq-PDU

5.3.9.1 Généralités

Ninq-PDU transmet un message de demande à tous les nœuds appartenant à un groupe de multidiffusion spécifié.

Chaque nœud spécifié dans cette PDU transmet une Reply-PDU à tous les nœuds. Cette PDU utilise un port UDP.

5.3.9.2 falArHeader

Voir 5.3.2.

5.3.9.3 node-list

Ce champ doit spécifier une liste de nœuds, comme représenté à la Figure 7. Chaque bit correspond à un numéro de nœud.

(Exemple)
Le premier bit (0) du premier octet (0) indique le numéro de nœud 1.
Le deuxième bit (1) du premier octet (0) indique le numéro de nœud 2.
Le huitième bit (7) du 512e octet (511) indique le numéro de nœud 4 096.

	0	1	2	3	4	5	6	7	bit
0	R	R	R	R	R	R	R	R	
1	R	R	R	R	R	R	R	R	
2	R	R	R	R	R	R	R	R	
.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	
511 Octet	R	R	R	R	R	R	R	R	

R = 1: demande au nœud cible de répondre, R = 0: aucune demande

Figure 7 – Liste des nœuds

5.3.9.4 ninquirydata

Ce champ contient les données ninquiry (demande un-à-N/multiréponse) de l'utilisateur.

5.3.10 Reply-PDU

5.3.10.1 Généralités

Reply-PDU répond à un nœud qui a envoyé une Inq-PDU ou une Ninq-PDU. Cette PDU utilise le port TCP.

5.3.10.2 falArHeader

Voir 5.3.2.

5.3.10.3 replydata

Ce champ contient des données de réponse utilisateur.

5.3.11 RetransEnq-PDU

5.3.11.1 Généralités

Dans la communication multidiffusion avec retransmission, RetransEnq-PDU transmet une demande de retransmission de messages spécifiés à tous les nœuds appartenant à un groupe de multidiffusion spécifié.

Cette PDU utilise le port UDP de la commande de retransmission.

Un nœud demandé pour la retransmission transmet tous les messages qui ont été envoyés, y compris le message spécifié.

Le temps de transmission d'une RetransEnq-PDU est le suivant:

- Temps que met un nœud destinataire pour détecter les messages perdus,
- Temps que met un nœud destinataire pour détecter la perte de messages indiquée par RetransConfirm-PDU.

5.3.11.2 falArHeader

Voir 5.3.2.

5.3.11.3 retransRequest

Ce champ doit spécifier le code de demande de retransmission (= 1).

5.3.11.4 retransRequestNode

Ce champ doit spécifier le numéro de nœud demandé pour retransmettre des messages.

5.3.11.5 retransNumberOfRequests

Ce champ doit spécifier le nombre de demandes contenues dans cette PDU. Une demande est constituée à la fois de retransMcg et de retransPseqNo.

5.3.11.6 retransMcg

Ce champ doit spécifier un numéro de groupe de multidiffusion (MCG) auquel le nœud de retransmission appartient.

5.3.11.7 retransPseqNo

Ce champ doit spécifier le numéro de séquence de paquets du premier paquet de retransmission.

5.3.12 RetransConfirm-PDU

5.3.12.1 Généralités

Dans la communication multidiffusion avec retransmission, RetransConfirm-PDU transmet une demande pour confirmer que le message spécifié est arrivé aux nœuds de destination appartenant à un groupe de multidiffusion spécifié.

Si un nœud destinataire confirme que le message spécifié n'est pas arrivé, il transmet une RetransEnq-PDU pour demander la retransmission d'un message.

Cette PDU utilise le port UDP de la commande de retransmission.

5.3.12.2 falArHeader

Voir 5.3.2.

5.3.12.3 retransRequest

Ce champ doit spécifier le code de confirmation de retransmission (= 2). Lorsqu'un paquet de transmission ne s'est pas produit dans le délai prédéfini, le nœud source transmet cette PDU pour confirmer que le paquet spécifié a atteint le nœud cible.

5.3.12.4 retransNumberOfRequests

Voir 5.3.11.5.

5.3.12.5 retransMcg

Ce champ doit spécifier le numéro de groupe de multidiffusion (MCG) auquel le nœud cible appartient.

5.3.12.6 retransPseqNo

Ce champ doit spécifier le numéro de séquence de paquets que le nœud cible est tenu de confirmer.

5.3.13 RetransNak-PDU

5.3.13.1 Généralités

Dans une communication multidiffusion avec retransmission, lorsqu'un nœud ne peut pas transmettre un message demandé par une RetransEnq-PDU, il transmet une RetransNak-PDU.

Cette PDU utilise le port UDP de la commande de retransmission.

5.3.13.2 falArHeader

Voir 5.3.2.

5.3.13.3 retransRequest

Ce champ doit spécifier le code de rejet de retransmission (= 3).

5.3.13.4 retransNumberOfRequests

Voir 5.3.11.5.

5.3.13.5 retransMcg

Ce champ doit spécifier un numéro de groupe de multidiffusion (MCG) auquel le nœud de demande de retransmission appartient.

5.3.13.6 retransPseqNo

Lorsque le nœud cible ne peut pas retransmettre le paquet spécifié dans RetransEnq-PDU, le nœud cible transmet cette RetransNak-PDU pour informer ce statut. Ce champ doit spécifier le numéro de séquence de paquets spécifié dans RetransEnq-PDU.

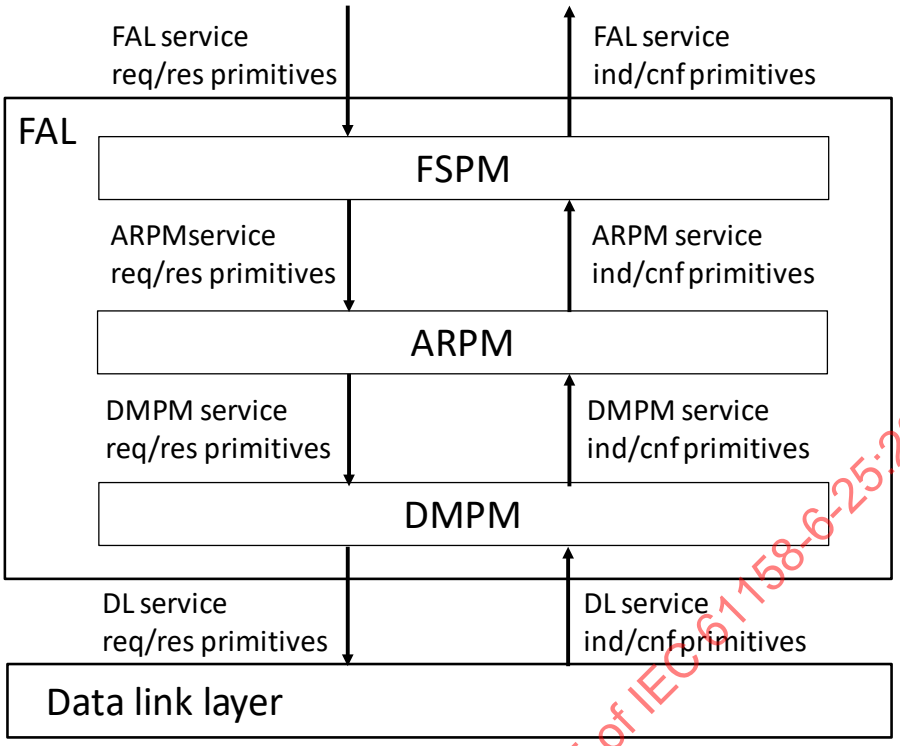
6 Structure du diagramme d'états de protocole FAL

Le diagramme d'états de protocole FAL comprend trois diagrammes d'états de protocole, tel qu'indiqué à la Figure 8. Ces trois diagrammes d'états de protocole, dans l'ordre depuis le côté couche de liaison de données, sont comme suit: un data link layer mapping protocol machine (DMPM, diagramme d'états de protocole de mapping de liaison de données), un application relationship protocol machine (ARPM, diagramme d'états de protocole de mapping d'application) et un FAL service protocol machine (FSPM, machine de protocole de services de la FAL).

Le rôle du FSPM consiste à recevoir des primitives de service des utilisateurs de la FAL et à convertir les primitives en primitives internes, à sélectionner un ARPM (diagramme d'états de protocole de mapping d'application), à recevoir les primitives internes de l'ARPM et à convertir ces dernières en primitives de service de la FAL et à les transmettre aux utilisateurs de la FAL.

Le rôle de l'ARPM consiste à convertir les primitives de service entre l'ARPM et le DMPM.

Le rôle du DMPM est de mapper dans la couche de liaison de données.



Anglais	Français
FAL service req/res primitives	Primitives de demande/réponse de service FAL
FAL service ind/cnf primitives	Primitives d'indication/de confirmation de service FAL
FAL	ASE
FSPM	FSPM
ARPM service req/res primitives	Primitives de demande/réponse de service ARPM
ARPM service ind/cnf primitives	Primitives d'indication/de confirmation de service ARPM
ARPM	ARPM
DMPM service req/res primitives	Primitives de demande/réponse de service DMPM
DMPM service ind/cnf primitives	Primitives d'indication/de confirmation de service DMPM
DMPM	DMPM
DL service req/res primitives	Primitives de demande/réponse de service DL
DL service ind/cnf primitives	Primitives d'indication/de confirmation de service DL
Data link layer	Couche liaison de données

Figure 8 – Relations entre les diagrammes de protocole

7 Machine de protocole FSPM

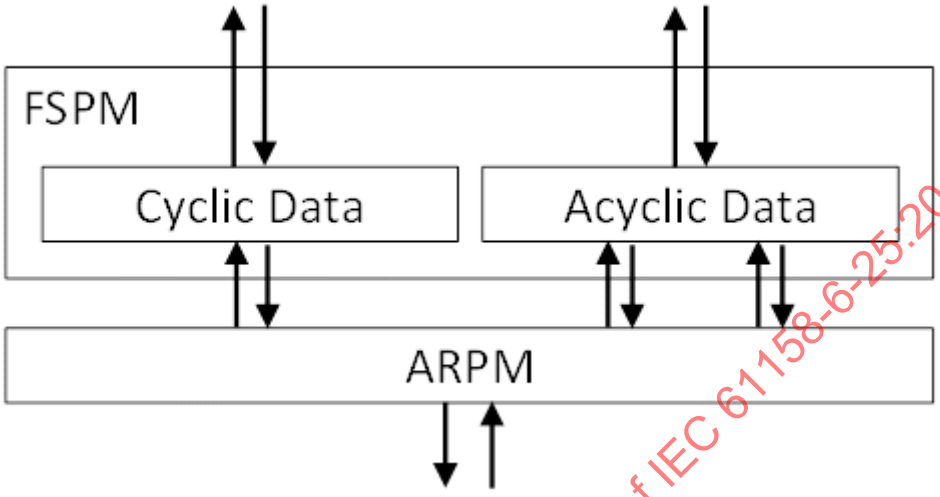
7.1 Vue d'ensemble

Le FSPM offre une interface aux utilisateurs de la FAL. Il procède au mapping entre les services des utilisateurs de la FAL et les services internes de la FAL.

7.2 Type S de FSPM

7.2.1 Vue d'ensemble

Le FSPM comprend trois diagrammes d'états de protocole: données cycliques, données acycliques et gestion. La relation entre les machines de protocole est indiquée à la Figure 9.



Anglais	Français
FSPM	FSPM
Cyclic Data	Données cycliques
Acyclic Data	Données acycliques
ARPM	ARPM

Figure 9 – Structure de type S de FSPM

Les primitives suivantes sont envoyées par l'utilisateur de la FAL au FSPM.

Put_cyclicdata.req
Get_cyclicdata.req
Ctl_cyclic.req
Send_ctldata.req
Send_infodata.req
Send_rmtctl.req

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur de la FAL.

Put_cyclicdata.cnf
Get_cyclicdata.cnf
Ctl_cyclic.cnf
Send_ctldata.ind
Send_ctldata.cnf
Send_infodata.ind
Send_infodata.cnf
Send_rmtctl.cnf

7.2.2 Interface de communication cyclique avec les utilisateurs de la FAL

Dans la communication cyclique, chaque nœud planifie indépendamment la transmission de la trame de communication cyclique à l'aide de son propre temporisateur. Pour gérer la communication cyclique pendant le cycle de transmission, le trafic de communication de chaque nœud est limité.

Dans le réseau de type S, la quantité de trafic de données appelée "Transmission factor" (Facteur de transmission) est inférieure à "Threshold" (Seuil). Le facteur de transmission est défini comme le nombre de blocs (64 octets/bloc) envoyés par cycle. La relation entre la durée de cycle, le nombre de blocs et le facteur de transmission est représentée dans l'équation ci-dessous.

(Facteur de transmission)
= (Nombre de blocs pour un cycle de 1 ms)/1 + (Nombre de blocs pour un cycle de 2 ms)/2
+ (Nombre de blocs pour un cycle de 5 ms)/5 + (Nombre de blocs pour un cycle de 10 ms)/10
+ (Nombre de blocs pour un cycle de 20 ms)/20 + (Nombre de blocs pour un cycle de 50 ms)/50
+ (Nombre de blocs pour un cycle de 100 ms)/100
≤ (Seuil)

La durée de cycle minimale est déterminée par le nombre de nœuds. Lorsqu'un réseau est composé de 16 nœuds, la durée de cycle minimale est de 1 ms. Composé de 16 à 32 nœuds, le réseau peut utiliser un cycle de 2 ms au minimum. De 33 à 80 nœuds, le cycle minimal est de 5 ms. De 81 à 127 nœuds, il utilise uniquement un cycle de plus de 10 ms. Par conséquent, le seuil est différent en fonction de la longueur totale du câble et de la durée de cycle minimale. Le Tableau 38 représente le seuil du facteur de transmission.

Tableau 38 – Seuil du facteur de transmission

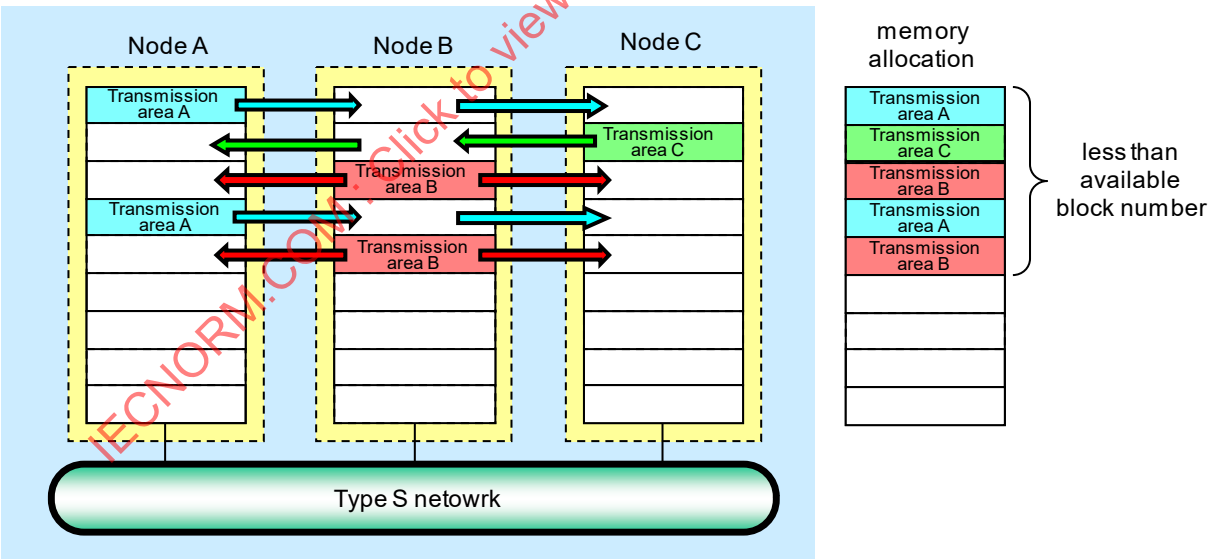
No.	Longueur totale du câble [km]	Durée de cycle minimale [ms]	Seuil
1	0 km à 10 km	1 s à 5 ms	256,0
2	0 km à 10 km	Plus de 10 ms	409,6
3	Plus de 10 km	1 s à 5 ms	128,0
4	Plus de 10 km	Plus de 10 ms	204,8

Les blocs utilisés pour la communication cyclique sont spécifiés par les utilisateurs de la FAL. Le Tableau 39 représente un exemple de menu de configuration de commande de trafic. Il donne le total des blocs disponibles en fonction du nombre de nœuds et de la durée de cycle utilisée dans un réseau de type S. Par exemple, le numéro 2 représente le cas de la connexion de 16 nœuds et l'utilisation de durées de cycle de 1 ms et 2 ms. Dans ce cas, le réseau peut utiliser 128 blocs dans un cycle de 1 ms et 256 blocs dans un cycle de 2 ms.

Tableau 39 – Exemple de menu de configuration de commande de trafic

No.	Nombre de nœuds	Nombre de blocs disponibles lors du cycle de transfert [ms] (64 octets/bloc)						
		1	2	5	10	20	50	…
1	16	256						
2		128	256					
3		128		640				
4		128		320	640			
5		128		320		1280		
6	32		512					
7			256	640				
8			256		1280			
9			256		640	1280		
10			256		640		3200	
:								

La Figure 10 représente un exemple d'attribution de mémoire partagée dans un réseau de type S. Chaque nœud a la mémoire pour la communication cyclique et alloue la zone de transmission pour stocker ses propres données à envoyer aux autres nœuds. Il planifie la communication cyclique en utilisant son propre temporisateur. Le nombre de blocs attribués est inférieur au nombre de blocs disponibles.



Anglais	Français
Node A / B / C	Nœud A / B / C
Transmission area A / B / C	Zone de transmission A / B / C
Type S network	Réseau de type S
Memory allocation	Attribution de mémoire
less than available block number	inférieure au nombre de blocs disponibles

Figure 10 – Attribution de mémoire partagée dans un réseau de type S

7.2.3 Diagramme d'états de FSPM

7.2.3.1 Données cycliques

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 40.

Tableau 40 – Table d'états des données cycliques

#	DEACT	Evénement /condition => action	état suivant
1	ACTIVE	Put_cyclicdata.req{CYC_ID,Block_ID,CYCdata} => CYC_WRITE.req{CYC_ID,Block_ID,CYCdata}	ACTIVE
2	ACTIVE	CYC_WRITE.cnf{CYC_ID,Status} => Put_cyclicdata.cnf{CYC_ID,Status}	ACTIVE
3	ACTIVE	Get_cyclicdata.req{CYC_ID,Block_ID} => CYC_READ.req{CYC_ID,Block_ID}	ACTIVE
4	ACTIVE	CYC_READ.cnf{CYC_ID,CYCdata,Status} => Get_cyclicdata.cnf{CYC_ID,CYCdata,Status}	ACTIVE
5	ACTIVE	Ctl_cyclic.req{CYC_ID,CYCctl,CYC-CTLdata} => CTL_CYCLIC.req{CYC_ID,CYCctl,CYC-CTLdata}	ACTIVE
6	ACTIVE	CTL_CYCLIC.cnf{CYC_ID,Status} => Ctl_cyclic.cnf{CYC_ID,Status}	ACTIVE

7.2.3.2 Données acycliques

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 41.

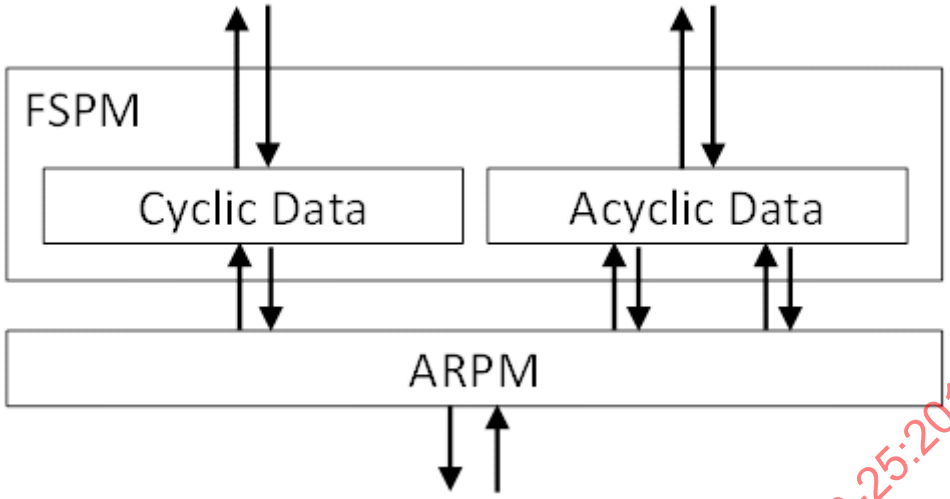
Tableau 41 – Table d'états des données acycliques (Acyclic data)

#	DEACT	Evénement /condition => action	état suivant
1	ACTIVE	Send_ctldata.req{D_add,S_add,CTLdata} => SendCTL.req{D_add,S_add,CTLdata}	ACTIVE
2	ACTIVE	SendCTL.cnf{Status} => Send_CTLdata.cnf{Status}	ACTIVE
3	ACTIVE	SendCTL.ind{D_add,S_add,CTLdata} => Send_ctldata.ind{D_add,S_add,CTLdata}	ACTIVE
4	ACTIVE	Send_infodata.req{D_add,S_add,INFOdata} => SendINFO.req{D_add,S_add,INFOdata};	ACTIVE
5	ACTIVE	SendINFO.cnf{Status} => Send_infodata.cnf{Status}	ACTIVE
6	ACTIVE	SendINFO.ind{D_add,S_add,INFOdata} => Send_infodata.ind{D_add,S_add,INFOdata}	ACTIVE
7	ACTIVE	Send_rmtctl.req{D_add,S_add,CMD,CMDdata} => SendRMTCTL.req{D_add,S_add,CMD,CMDdata}	ACTIVE
8	ACTIVE	SendRMTCTL.cnf{D_add,S_add,CMD,CMDdata} => Send_rmtctl.cnf{D_add,S_add,CMD,CMDdata}	ACTIVE

7.3 FSPM de type F

7.3.1 Vue d'ensemble

Le FSPM comprend trois diagrammes d'états de protocole: données cycliques, données acycliques et gestion. La relation entre les machines de protocole est indiquée à la Figure 11.



Anglais	Français
FSPM	FSPM
Cyclic Data	Données cycliques
Acyclic Data	Données acycliques
ARPM	ARPM

Figure 11 – Structure de type N de FSPM

Les primitives suivantes sont envoyées par l'utilisateur de la FAL au FSPM.

Put_cyclicdata.req
Get_cyclicdata.req
Put_message.req
Get_message.req
Put_inquiry message.req
Put_reply_message.req
Put_ninquiry_message.req
Send_aliveinfo.req
Receive_aliveinfo.req
Control_acyclic.req
Control_cyclic.req

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur de la FAL.

Get_cyclicdata.cnf
Put_message.ind
Get_message.cnf
Put_inquiry_message.ind
Put_reply_message.ind
Put_ninquiry_message.ind
Receive_aliveinfo.cnf

7.3.2 FSPM

7.3.2.1 Données cycliques

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 42.

Tableau 42 – Table d'états des données cycliques

#	DEACT	Evénement /condition => action	état suivant
1	ACTIVE	Put_cyclicdata.req => Write_cyclicdata.req	ACTIVE
2	ACTIVE	Get_cyclicdata.req => Get_cyclicdata.cnf	ACTIVE
3	ACTIVE	Write_cyclicdata.ind => Update cyclicdata	ACTIVE
4	ACTIVE	Control_cyclic.req => Ctl_cyclic.req	ACTIVE

7.3.2.2 Données acycliques

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 43.

Tableau 43 – Table d'états des données acycliques (Acyclic data)

#	DEACT	Evénement /condition => action	état suivant
1	ACTIVE	Put_message.req => Transmit_acyclicdata1.req.	ACTIVE
2	ACTIVE	Transmit_acyclicdata1.ind. / (CtlType == MLT) || (CtlType == ONE) => Put_message.ind	ACTIVE
3	ACTIVE	Get_message.req => Get_message.cnf	ACTIVE
4	ACTIVE	Put_inquiry_message.req => Transmit_acyclicdata1.req.	ACTIVE
5	ACTIVE	Transmit_acyclicdata1.ind. / CtlType == INQ => Put_inquiry_message.ind	ACTIVE

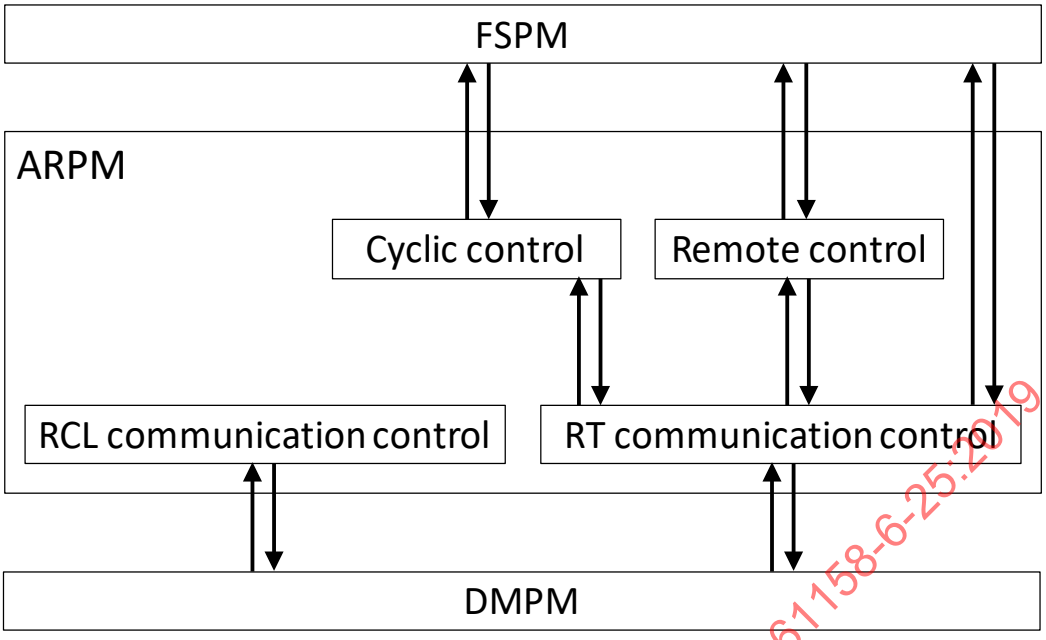
#	DEACT	Evénement /condition => action	état suivant
6	ACTIVE	Put_ninquiry_message.req => Transmit_acyclicdata1.req.	ACTIVE
7	ACTIVE	Transmit_acyclicdata1.ind. / CtlType == NIQ => Put_ninquiry_message.ind	ACTIVE
8	ACTIVE	Put_reply_message.req => Transmit_acyclicdata1.req.	ACTIVE
9	ACTIVE	Transmit_acyclicdata1.ind. / CtlType == RPL => Put_reply_message.ind	ACTIVE
10	ACTIVE	Send_aliveinfo.req => Transmit_acyclicdata2.req.	ACTIVE
11	ACTIVE	Receive_aliveinfo.req => Receive_aliveinfo.cnf	ACTIVE
12	ACTIVE	Control_acyclic.req => Ctl_acyclic.req	ACTIVE
13	ACTIVE	Transmit_acyclicdata2.ind. => Send_aliveinfo.req	ACTIVE

8 Machine protocolaire de relations entre applications (ARPM)

8.1 Type S d'ARPM

8.1.1 Vue d'ensemble

L'ARPM comprend quatre sous-protocoles. Commande cyclique, commande à distance, commande de communication RCL et commande de communication RT. La structure de l'ARPM est indiquée à la Figure 12.



Anglais	Français
FSPM	FSPM
ARPM	ARPM
Cyclic control	Commande cyclique
Remote control	Commande à distance
RCL communication control	Commande de communication RCL
RT communication control	Commande de communication RT
DMPM	DMPM

Figure 12 – Structure de type F d'ARPM

8.1.2 Commande cyclique

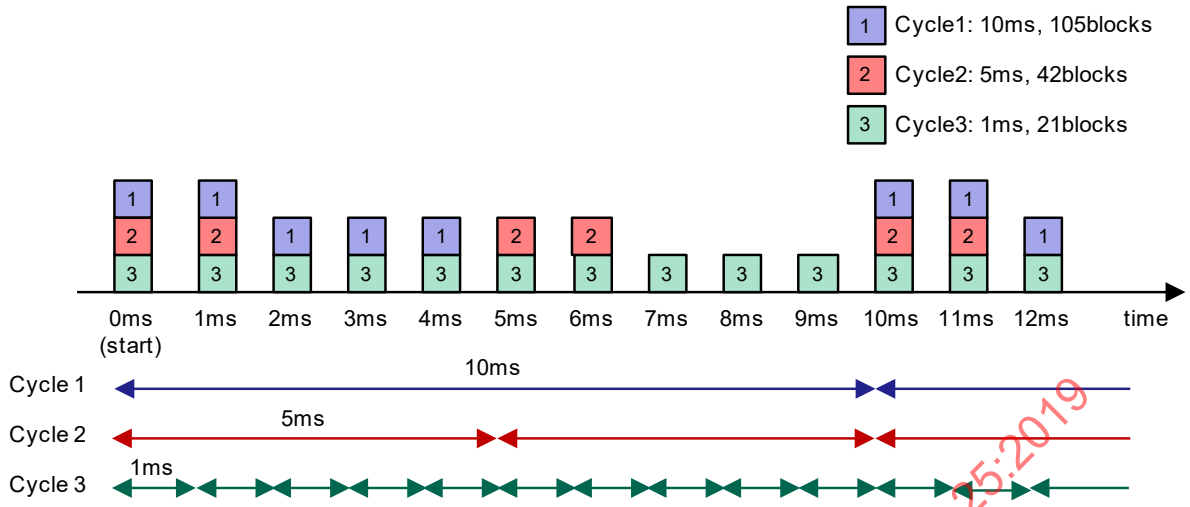
8.1.2.1 Menu de communication cyclique

La commande cyclique génère le menu de configuration de commande de trafic en fonction des données de configuration provenant de l'utilisateur de la FAL. Le menu est créé par chaque nœud à chaque durée de cycle. Le nombre de trames dans chaque nœud est spécifié.

La relation entre la durée de cycle n[ms], les blocs utilisés x[blocks] et le nombre de transmissions par cycle N_nc[blocks] est indiquée ci-dessous.

Durée de cycle n [ms] (n={1,2,5,10,20,50,100,200,500,1 000})
Le nombre de blocs x ($0 \leq x \leq 21 \times n$)
Le nombre de blocs par cycle $N\_nc = \min(21, x-21\times c)$ (c={0,1,⋯,⌈x/21⌉-1})

La Figure 13 représente la séquence de communication cyclique basée sur les formules ci-dessus. Dans la séquence, le cycle 1 transmet des trames avec une durée de cycle de 10 ms et utilise 105 blocs. Le cycle 2 transmet des trames avec une durée de cycle de 5 ms et utilise 42 blocs. Le cycle 3 transmet une trame avec une durée de cycle de 1 ms et utilise 21 blocs. Les cases de la Figure 13 désignent les trames envoyées, le nombre dans chaque case désigne respectivement les cycles 1, 2 et 3.



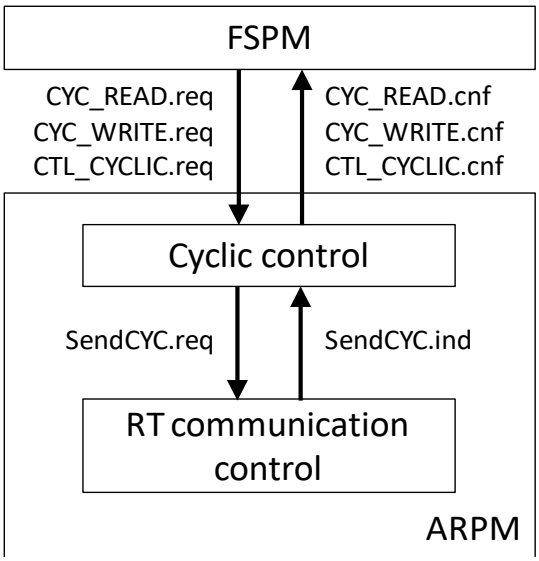
Anglais	Français
Cycle1: 10ms, 105blocks	Cycle1: 10ms, 105blocks
Cycle2: 5ms, 42blocks	Cycle2: 5ms, 42blocks
Cycle3: 1ms, 21blocks	Cycle3: 1ms, 21blocks
start	début
time	temps
ms	ms
Cycle 1 / 2 / 3	Cycle 1 / 2 / 3

Figure 13 – Séquence de communication cyclique

8.1.2.2 Définitions des primitives

La FSPM émet un service CYC_READ.req, CYC_WRITE.req ou CTL_CYCLIC.req pour la commande cyclique. La commande de communication RT émet un service SendCYC.ind pour la commande cyclique. La commande cyclique émet un service CYC_READ.cnf, CYC_WRITE.cnf ou CTL_CYCLIC.cnf pour la FSPM. La commande cyclique émet un service SendCYC.req pour la commande de communication RT.

Les primitives de commande cyclique sont représentées à la Figure 14.



Anglais	Français
FSPM	FSPM
CYC_READ.req	CYC_READ.req
CYC_WRITE.req	CYC_WRITE.req
CTL_CYCLIC.req	CTL_CYCLIC.req
CYC_READ.cnf	CYC_READ.cnf
CYC_WRITE.cnf	CYC_WRITE.cnf
CTL_CYCLIC.cnf	CTL_CYCLIC.cnf
Cyclic control	Commande cyclique
SendCYC.req	SendCYC.req
SendCYC.ind	SendCYC.ind
RT communication control	Commande de communication RT
ARPM	ARPM

Figure 14 – Primitives de commande cyclique

8.1.2.3 Tableau d'états de la commande cyclique

La description détaillée du diagramme d'états de commande cyclique est représentée dans le Tableau 44.

Tableau 44 – Tableau d'états de la commande cyclique

#	DEACT	Evénement /Condition => Action	état suivant
1	N'importe quel état	Mise sous tension ou Réinitialisation => CYCMenu_flag = 0 Current_cycle = 0	PRET
2	N'importe quel état	CYC_READ.req{CYC_ID,Block_ID} /CYC_rcheck(Block_ID) == True => CYCdata = Read_memory(Block_ID) Status:= "success" CYC_READ.cnf{CYC_ID,Block_ID, CYCdata,Status}	N'importe quel état (pas de changement)
3	PRET	CYC_WRITE.req{CYC_ID,Block_ID,CYCdata} /CYC_wcheck(Block_ID,CYCdata) == True => WRITE_Memory(Block_ID,CYCdata) Status:= "success" CYC_WRITE.cnf{CYC_ID,Status}	PRET
4	PRET	CTL_CYCLIC.req{CYC_ID,CYCctl,CYC-CTLdata} /Config_check(CYCctl,CYC-CTLdata) == True && CYCctl == 1 /* Démarrage */ => Make_CYCMenu(CYC-CTLdata) START_TIMER(CYCtask,CYC_Cycle) Status:= "success" CTL_CYCLIC.cnf{CYC_ID,Status}	ACTIVE
5	ACTIVE	CYC_WRITE.req{CYC_ID,Block_ID,CYCdata} /CYC_wcheck(Block_ID,CYCdata) == True => WRITE_Memory(Block_ID,CYCdata);	ACTIVE

6	ACTIVE	/EXPIRED_TIMER(CYCtask) == "True"
		=>
		Current_cycle++
		CYCdata= Send_CYCMenu(Current_cycle)
		D_add = Broadcast
		S_add = Myadd
		SendCYC.req{D_add,S_add,CYCdata}
		IF(Current_cycle == Max_CYCLE) THEN
		Current_cycle = 0
		ENDIF
			ACTIVE

#	DEACT	Evénement /Condition => Action	état suivant
7	ACTIVE	SendCYC.ind{D_add,S_add,CYCdata} / D_add = Myadd => BlockID = Get_BlockID(CYCdat) WRITE_Memory(Block_ID,CYCdata)	ACTIVE
8	ACTIVE	CTL_CYCLIC.req{CYC_ID,CYCCTL,CYC-CTLdata} /CYCCTL == 0 /* Arrêt */ => STOP_TIMER(CYCtask,CYC_Cycle) Status:= “success” CTL_CYCLIC.cnf{CYC_ID,Status}	PRET

8.1.2.4 Fonctions

Toutes les fonctions utilisées par la commande cyclique sont résumées dans le Tableau 45.

Tableau 45 – Fonctions de commande cyclique

Nom de fonction	Entrée	Sortie	Description
START_TIMER	CYCtask, CYC_Cycle	(<none>)	Le temporisateur CYCtask est défini par la valeur de CYC_Cycle et est activé.
EXPIRED_TIMER	CYCtask	Vrai/Faux	Lorsque le temporisateur CYCtask demandé a expiré, "True" est retourné; sinon, "False" est renvoyé.
CYC_rcheck	Block_ID	Vrai/Faux	Cette fonction est utilisée pour vérifier si le Block_ID est valide pour la lecture à partir de la mémoire partagée.
CYC_wcheck	Block_ID, CYCdata	Vrai/Faux	Cette fonction est utilisée pour vérifier si les Block_ID et CYCdata sont valides pour l’écriture dans la mémoire partagée.
Read_memory	Block_ID	CYCdata	Cette fonction charge les données cycliques du bloc dans le Block_ID spécifié par la mémoire partagée.
WRITE_Memory	Block_ID, CYCdata	Vrai/Faux	Cette fonction écrit des CYCdata dans le Block_ID spécifié dans la mémoire partagée.
Config_check	CYCctl, CYC-CTLdata	Vrai/Faux	Cette fonction est utilisée pour vérifier si le Block_ID est valide pour la lecture à partir de la mémoire partagée.
Make_CYCMenu	CYC-CTLdata	Vrai/Faux	Cette fonction permet au menu cyclique d’envoyer des trames de communication cyclique.
Send_CYCMenu	Current_Cycle	CYCdata	Cette fonction renvoie les CYCdata pour envoyer des trames cycliques. Cette fonction charge le tableau CYCdata[Current_Cycle] et renvoie les données de variable.

8.1.2.5 Variables

Toutes les variables utilisées par la commande cyclique sont résumées dans le Tableau 46.

Tableau 46 – Variables de commande cyclique

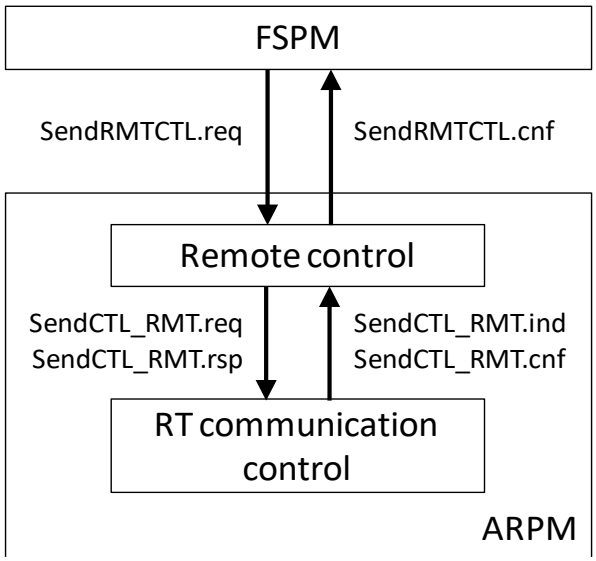
Name (Nom)	Description
Current_cycle	Cette variable indique le cycle courant pour envoyer des trames de communication cyclique. Si cette variable est 0, la communication cyclique n'est pas activée dans le nœud. Si la communication cyclique est activée, cette variable passe de 1 à Max_CYCLE.
Diffusion	Cette variable est l'adresse MAC de diffusion: FF-FF-FF-FF-FF-FF
Myadd	Cette variable indique l'adresse de son propre nœud.
Max_CYCLE	Cette variable indique le cycle maximal qui peut être utilisé pour la communication cyclique sur le réseau de type S.

8.1.3 Commande à distance

8.1.3.1 Définitions des primitives

La FSPM émet un service SendRMTCTL.req pour la commande à distance. La commande de communication RT émet un service SendCTL_RMT.ind et un service SendCTL_RMT.cnf pour la commande à distance. La commande à distance émet un service SendRMTCTL.cnf pour la FSPM. La commande à distance émet un service SendCTL_RMT.req et un service SendCTL_RMT.rsp pour la commande de communication RT.

Les primitives de la commande à distance sont représentées à la Figure 15.



Anglais	Français
FSPM	FSPM
SendRMTCTL.req	SendRMTCTL.req
SendRMTCTL.cnf	SendRMTCTL.cnf
Remote control	Commande à distance
SendCTL_RMT.req	SendCTL_RMT.req
SendCTL_RMT.rsp	SendCTL_RMT.rsp
SendCTL_RMT.ind	SendCTL_RMT.ind
SendCTL_RMT.cnf	SendCTL_RMT.cnf
RT communication control	Commande de communication RT
ARPM	ARPM

Figure 15 – Primitives de commande à distance

8.1.3.2 Tableau d'états de commande à distance

La description détaillée du diagramme d'états de commande à distance est représentée dans le Tableau 47.

Tableau 47 – Table d'états de commande à distance

#	DEACT	Evénement /condition => action	état suivant
1	N'importe quel état	Mise sous tension ou Réinitialisation =>	ACTIVE
2	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} /D_add != My_add => CTLdata = CreateData(CMD,CMDdata) SendCTL_RMT.req{D_add,S_add,CTLdata}	ACTIVE

3	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata}
		/D_add == My_add &&
		CMD_code(CMD) == "RMT_Read"
		=>
		D_add_rsp = S_add
		S_add_rsp = My_add
		R_add = CMD_address(CMD)
		R_size = CMD_size(CMD)
		CMDdata_rsp = Readmemory(R_add,R_size)
		if(CMDdata_rsp != NULL) THEN
		Status:= "success"
		SINON
		Etat = "Soft I/F Error"
		ENDIF
		CMD_rsp = MakeRSPCMD(CMD,Status)
		SendRMTCTL.cnf{D_add_rsp,S_add_rsp,
		CMD_rsp,CMDdata_rsp}
			ACTIVE

4	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} /D_add == My_add && CMD_code(CMD) == "RMT_Reset" => Reset()	-
5	ACTIVE	SendRMTCTL.req{D_add,S_add,CMD,CMDdata} / D_add == My_add && CMD != "RMT_Read" && CMD != "RMT_Reset" => D_add_rsp = S_add S_add_rsp = My_add Status = "Command parameter error" CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL SendRMTCTL.cnf{D_add_rsp,S_add_rsp, CMD_rsp,CMDdata_rsp}	ACTIVE

#	DEACT	Evénement /condition => action	état suivant
6	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} / D_add == My_add && CMD_check(CTLdata) == "RMT_Start" => Start_CPU() D_add_rsp = S_add S_add_rsp = My_add Status:= “success” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE
7	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /D_add == My_add && CMD_check(CTLdata) == "RMT_Stop" => Stop_CPU() D_add_rsp = S_add S_add_rsp = My_add Status:= “success” CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE

#	DEACT	Evénement /condition => action	état suivant
8	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /D_add == My_add && CMD_check(CTLdata) == "RMT_Write" => D_add_rsp = S_add S_add_rsp = My_add CMD = CTL2CMD(CTLdata) CMDdata = CTL2DATA(CTLdata) W_add = CMD_address(CMD) W_size = CMD_size(CMD) Status = Writememory(W_add,W_size) CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE

9	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata}
		/D_add == My_add &&
		CMD_check(CTLdata) == "RMT_Read"
		=>
		D_add_rsp = S_add
		S_add_rsp = My_add
		CMD = CTL2CMD(CTLdata)
		CMDdata = CTL2DATA(CTLdata)
		R_add = CMD_address(CMD)
		R_size = CMD_size(CMD)
		CMDdata_rsp = Readmemory(R_add,R_size)
		if(CMDdata_rsp != NULL) THEN
		Status:= “success”
		SINON
		Status = "Command parameter error"
		ENDIF
		CMD_rsp = MakeRSPCMD(CMD,Status)
		CTLdata_rsp = CreateData(CMD_rsp,
		CMDdata_rsp)
		SendCTL.rsp{D_add_rsp,S_add_rsp,
		CTLdata_rsp}
			ACTIVE

10	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /S_add == My_add && CMD_check(CTLdata) == "RMT_Reset" => Reset()	-

#	DEACT	Evénement /condition => action	état suivant
11	ACTIVE	SendCTL_RMT.ind{D_add,S_add,CTLdata} /S_add == My_add && CMD_check(CTLdata) == "Not_Valid" => D_add_rsp = S_add S_add_rsp = My_add Status = "Command parameter error" CMD_rsp = MakeRSPCMD(CMD,Status) CMDdata_rsp = NULL CTLdata_rsp = CreateData(CMD_rsp, CMDdata_rsp) SendCTL_RMT.rsp{D_add_rsp,S_add_rsp, CTLdata_rsp}	ACTIVE
12	ACTIVE	SendCTL_RMT.cnf{D_add,S_add,CTLdata} /CMD_check(CTLdata) != "Not_Valid" => CMD = CTL2CMD(CTLdata) CMDdata = CTL2DATA(CTLdata) SendRMTCTL.cnf{D_add,S_add,CMD,CMDdata}	ACTIVE

8.1.3.3 Fonctions

Toutes les fonctions utilisées par la commande à distance sont résumées dans le Tableau 48.

Tableau 48 – Fonctions de commande à distance

Nom de fonction	Entrée	Sortie	Description
CreateData	CMD CMDdata	CTLdata	Cette fonction crée des données pour envoyer une PDU de commande à distance.
CMD_code	CMD	CMD_type	Cette fonction récupère le type de commande à distance.
CMD_address	CMD	R_add ou W_add	Cette fonction obtient l'adresse pour l'accès à la mémoire.
CMD_size	CMD	R_size ou W_size	Cette fonction obtient la taille des données pour l'accès à la mémoire.
Readmemory	R_add, R_size	CMDdata_rsp	Cette fonction lit les données R_add et R_size spécifiées dans la mémoire du nœud.
MakeRSPCMD	CMD Statut	CMD_rsp	Cette fonction permet à la commande de réponse de recevoir la commande à distance.
Réinitialiser	(<none>)	Statut	Cette fonction est utilisée pour réinitialiser le nœud.
CMD_check	CTLdata	CMD_type	Cette fonction vérifie la commande provenant de CTLdata.
Start_CPU	(<none>)	Statut	Cette fonction démarre le réseau de type S connecté au CPU.

Nom de fonction	Entrée	Sortie	Description
Stop_CPU	(<none>)	Statut	Cette fonction arrête le réseau de type S connecté au CPU.
CTL2CMD.	CTLdata	CMD	Cette fonction effectue la commande à distance à partir de CTLdata.
CTL2DATA.	CTLdata	CMDdata	Cette fonction établit des données pour la commande à distance à partir de CTLdata.
Writememory	W_add, W_size	Statut	Cette fonction écrit les données dans la mémoire du nœud.

8.1.3.4 Variables

Toutes les variables utilisées par la commande à distance sont résumées dans le Tableau 49.

Tableau 49 – Variables de commande à distance

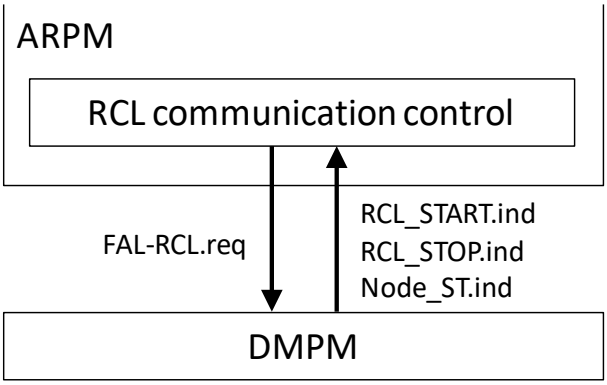
Nom	Description
CMD_type	Cette variable indique le type de commande pour la commande à distance, y compris "RMT_Read", "RMT_Reset", "RMT_Start", "RMT_Stop" ou "RMT_Write".
My_add	Cette variable indique l'adresse de son propre nœud.
R_add	Cette variable est utilisée pour spécifier l'adresse pour la lecture à partir de la mémoire du nœud.
R_size	Cette variable est utilisée pour spécifier la taille des données pour la lecture à partir de la mémoire du nœud.
NULL	Cette variable indique l'absence de données.
W_add	Cette variable est utilisée pour spécifier l'adresse pour l'écriture dans la mémoire de nœud.
W_size	Cette variable est utilisée pour spécifier la taille des données pour l'écriture dans la mémoire du nœud.

8.1.4 Commande de communication RCL

8.1.4.1 Définitions des primitives

La DMPM émet un service RCL_START.ind, RCL_STOP.ind ou Node_ST.ind pour la commande de communication RCL. La commande de communication RCL émet un service FAL-RCL.req pour la DMPM.

Les primitives de la commande de communication RCL sont représentées à la Figure 16.



Anglais	Français
ARPM	ARPM
RCL communication control	Commande de communication RCL
FAL-RCL.req	FAL-RCL.req
RCL_START.ind	RCL_START.ind
RCL_STOP.ind	RCL_STOP.ind
Node_ST.ind	Node_ST.ind
DMPM	DMPM

Figure 16 – Primitives pour la commande de communication RCL

8.1.4.2 Tableau d'états de la commande de communication RCL

Le Tableau50 représente le tableau d'états de la commande de communication RCL.

Tableau50 – Tableau d'états de la commande de communication RCL

#	DEACT	Evénement /Condition => Action	état suivant
1	N'importe quel état	Mise sous tension ou Réinitialisation => LCC_flag = "False" LCN_flag = "False" SCR_count = 0 START_TIMER(Fixedtask,Fixed_Cycle)	ACTIVE
2	ACTIVE	EXPIRED_TIMER(Fixedtask) == "True" => RCL_TIM_RHE() RCL_TIM_LCC() RCL_TIM_LCN() RCL_TIM_SCR()	ACTIVE

#	DEACT	Evénement /Condition => Action	état suivant
3	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCC => LCC_flag = "True" LCC_PDU = CreateLCCframe()	ACTIVE
4	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCA => LCA_PDU = CreateLCAframe(D_add) PortNum = RCLPort FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,LCA_PDU}	ACTIVE
5	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LCN => LCN_flag = "True" LCN_PDU = CreateLCNframe()	ACTIVE
6	ACTIVE	RCL_START.ind{RCLType,D_add, RCLPri,RCLPort} /RCLType == LNA => LNA_PDU = CreateLNAframe(D_add) PortNum = RCLPort FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,LNA_PDU}	ACTIVE
7	ACTIVE	RCL_STOP.ind{RCLType} /RCLType == LCC => LCC_flag = "False"	ACTIVE
8	ACTIVE	RCL_STOP.ind{RCLType} /RCLType == LCN => LCN_flag = "False"	ACTIVE
9	ACTIVE	Node_ST.ind{NodeST,PortAST,PortBST} => SCR_count = SCR_Sndnum SCR_PDU = CreateSCRframe(NodeST,PortAST, PortBST)	ACTIVE

8.1.4.3 Fonctions

Toutes les fonctions utilisées par la commande de communication RCL sont résumées dans le Tableau 51.

Tableau 51 – Fonctions de commande de communication RCL

Nom de fonction	Entrée	Sortie	Description
RCL_TIM_RHE	(<none>)	(<none>)	Fonction de cycle fixe pour l'envoi de RHE-PDU. RCL_TIM_RHE() est assemblée comme suit: RHE_PDU-A = CreateRHEframe(PortA) RHE_PDU-B = CreateRHEframe(PortB) S_add = Myadd RCLPri = VLAN_RCL RCLType = RHE PortNum = PortA FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,RHE_PDU-A} PortNum = PortB FAL-RCL.req{S_add,PortNum,RCLPri, RCLType,RHE_PDU-B}

RCL_TIM_LCC	(<none>)	(<none>)	Fonction de cycle fixe pour l'envoi de LCC-PDU.
			RCL_TIM_LCC() est assemblée comme suit:
			IF(LCC_flag == "TRUE") THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = LCC
			PortNum = Both
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			LCC_PDU}
			ENDIF

RCL_TIM_LCN	(<none>)	(<none>)	Fonction de cycle fixe pour l'envoi de LCN-PDU.
			RCL_TIM_LCN() est assemblée comme suit:
			IF(LCN_flag == "TRUE") THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = LCN
			PortNum = PortB
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			LCN_PDU}
			ENDIF

Nom de fonction	Entrée	Sortie	Description

RCL_TIM_SCR	(<none>)	(<none>)	Fonction de cycle fixe pour l'envoi de SCR-PDU.
			RCL_TIM_SCR() est assemblée comme suit:
			IF(SCR_count > 0) THEN
			S_add = Myadd
			RCLPri = VLAN_RCL
			RCLType = SCR
			PortNum = Both
			FAL-RCL.req{S_add,PortNum,
			RCLPri,RCLType,
			SCR_PDU}
			SCR_count--
			ENDIF

EXPIRED_TIMER	Fixedtask	Vrai/Faux	Lorsque le temporisateur demandé Fixedtask a expiré, "True" est renvoyé, sinon, "False" est renvoyé.
START_TIMER	Fixedtask, Fixed_Cycle	(<none>)	Le temporisateur Fixedtask est défini par la valeur de Fixed_Cycle et est activé.
CreateLCCframe	(<none>)	LCC_PDU	Générer la LCC-PDU
CreateLCAframe	D_add	LCA_PDU	Générer la LCA-PDU
CreateLCNframe	(<none>)	LCN_PDU	Générer la LCN-PDU
CreateLNAframe	D_add	LNA_PDU	Générer la LNA-PDU
CreateSCRframe	NodeST, PortAST, PortBST	SCR_PDU	Générer la SCR-PDU

8.1.4.4 Variables

Toutes les variables utilisées par la commande de communication RCL sont résumées dans le Tableau 52.

Tableau 52 – Variables de la commande de communication RCL

Nom	Description
LCC_flag	Ce fanion indique si l'envoi de LCC-PDU est en cours ou non.
LCN_Flag	Ce fanion indique si l'envoi de LCN-PDU est en cours ou non.
SCR_count	Ce compteur contient le nombre restant de transmissions de SCR-PDU.
SCR_Sndnum	Cette variable indique le nombre de transmissions de SCR-PDU par événement.
Myadd	Cette variable indique l'adresse de son propre nœud.
VLAN_RCL	Cette variable indique la priorité VLAN par rapport à la communication RCL.