PD IEC/PAS 62953:2015

BSI Standards Publication

Industrial communication networks — Fieldbus specifications

bsi.

...making excellence a habit.™

National foreword

This Published Document is the UK implementation of IEC/PAS 62953:2015.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2015

ISBN 978 0 580 88098 8
ICS 25.040.40; 35.100.01

Compliance with a British Standard cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 30 April 2015.

Amendments/corrigenda issued since publication

Date	Text affected

IEC PAS 62953

Edition 1.0 2015-04

PUBLICLY AVAILABLE SPECIFICATION

PRE-STANDARD

Industrial communication networks – Fieldbus specifications

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.01

ISBN 978-2-8322-2445-8

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS AND PROFILES – ADS-NET

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.

A PAS is a technical specification not fulfilling the requirements for a standard, but made available to the public.

IEC PAS 62953 has been processed by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this PAS is based on the following document:	This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document
Draft PAS	**Report on voting**
65C/787/PAS	65C/799/RVD

Following publication of this PAS, which is a pre-standard publication, the technical committee or subcommittee concerned may transform it into an International Standard.

This PAS shall remain valid for an initial maximum period of 3 years starting from the publication date. The validity may be extended for a single period up to a maximum of 3 years, at the end of which it shall be published as another type of normative document, or shall be withdrawn.

INTRODUCTION

This PAS contains the ADS-net fieldbus specifications and profiles (Communication profile CP 20/1 ADS-net/μΣNETWORK-1000, CP 20/2 ADS-net/NX) for inclusion in the IEC 61158 series as well as IEC 61784-2.

The intention is to make this technical content available immediately, while a corresponding new work item proposal (NP) has been launched. If the NP is accepted, this contents will be included in the next editions of the IEC 61158 series and of IEC 61784-2.

This PAS contains the relevant ADS-net (Type 25) elements for the DL and AL services and protocols and information with references to:

– IEC 61158-1:2014

– IEC 61158-2:2014

– IEC 61784-2:2014

The present IEC PAS is structured in the same way as the IEC 61158 series and IEC 61784-2:

– IEC PAS 62953-3-25 is intended to become a future IEC 61158-3-25;

– IEC PAS 62953-4-25 is intended to become a future IEC 61158-4-25;

– IEC PAS 62953-5-25 is intended to become a future IEC 61158-5-25;

– IEC PAS 62953-6-25 is intended to become a future IEC 61158-6-25;

– IEC PAS 62953-2 is intended to be merged into a revised version of IEC 61784-2:2014.

INCLUDED SUBPARTS

Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements

Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements

Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements

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

CPF20 input for IEC 61784-2: Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3

CONTENTS

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 3-25: Data-link layer service definition – Type 25 elements

1 Scope

1.1 General

This part of IEC PAS 62953 provides common elements for basic time-critical messaging communications between devices in an automation environment. 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 standard defines in an abstract way the externally visible service provided by the Type 25 fieldbus data-link layer in terms of

a) the primitive actions and events of the service;

b) the parameters associated with each primitive action and event, and the form which they take; and

c) the interrelationship between these actions and events, and their valid sequences.

The purpose of this standard is to define the services provided to

- the Type 25 fieldbus application layer at the boundary between the application and data-link layers of the fieldbus reference model;
- systems management at the boundary between the data-link layer and systems management of the fieldbus reference model.

1.2 Specifications

The principal objective of this standard is to specify the characteristics of conceptual data-link layer services suitable for time-critical communications, and thus supplement the OSI Basic Reference Model in guiding the development of data-link protocols for time-critical communications. A secondary objective is to provide migration paths from previously-existing industrial communications protocols.

This specification may be used as the basis for formal DL-Programming-Interfaces. Nevertheless, it is not a formal programming interface, and any such interface will need to address implementation issues not covered by this specification, including

a) the sizes and octet ordering of various multi-octet service parameters, and

b) the correlation of paired request and confirm, or indication and response, primitives.

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of data-link entities within industrial automation systems.

There is no conformance of equipment to this data-link layer service definition standard. Instead, conformance is achieved through implementation of the corresponding data-link protocol that fulfills the Type 25 data-link layer services defined in this standard.

2 Normative references

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

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

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

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications*

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

3 Terms, definitions, symbols and abbreviations

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

3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein:

3.1.1 DL-address **[ISO/IEC 7498-3]**

3.1.2 DL-address-mapping **[ISO/IEC 7498-1]**

3.1.3 called-DL-address **[ISO/IEC 7498-3]**

3.1.4 calling-DL-address **[ISO/IEC 7498-3]**

3.1.5 centralized multi-end-point-connection **[ISO/IEC 7498-1]**

3.1.6 DL-connection **[ISO/IEC 7498-1]**

3.1.7 DL-connection-end-point **[ISO/IEC 7498-1]**

3.1.8 DL-connection-end-point-identifier **[ISO/IEC 7498-1]**

3.1.9 DL-connection-mode transmission **[ISO/IEC 7498-1]**

3.1.10 DL-connectionless-mode transmission **[ISO/IEC 7498-1]**

3.1.11 correspondent (N)-entities **[ISO/IEC 7498-1]**
correspondent DL-entities (N=2)
correspondent Ph-entities (N=1)

3.1.12 DL-duplex-transmission **[ISO/IEC 7498-1]**

3.1.13 (N)-entity **[ISO/IEC 7498-1]**
DL-entity (N=2)
Ph-entity (N=1)

3.1.14 DL-facility **[ISO/IEC 7498-1]**

3.1.15 flow control **[ISO/IEC 7498-1]**

3.1.16 (N)-layer **[ISO/IEC 7498-1]**
DL-layer (N=2)
Ph-layer (N=1)

3.1.17 layer-management **[ISO/IEC 7498-1]**

3.1.18 DL-local-view **[ISO/IEC 7498-3]**

3.1.19 DL-name **[ISO/IEC 7498-3]**

3.1.20 naming-(addressing)-domain **[ISO/IEC 7498-3]**

3.1.21 peer-entities **[ISO/IEC 7498-1]**

3.1.22 primitive name **[ISO/IEC 7498-3]**

3.1.23 DL-protocol **[ISO/IEC 7498-1]**

3.1.24 DL-protocol-connection-identifier **[ISO/IEC 7498-1]**

3.1.25 DL-protocol-data-unit **[ISO/IEC 7498-1]**

3.1.26 DL-relay **[ISO/IEC 7498-1]**

3.1.27 reset **[ISO/IEC 7498-1]**

3.1.28 responding-DL-address **[ISO/IEC 7498-3]**

3.1.29 routing **[ISO/IEC 7498-1]**

3.1.30 segmenting **[ISO/IEC 7498-1]**

3.1.31 (N)-service **[ISO/IEC 7498-1]**

DL-service (N=2)
Ph-service (N=1)

3.1.32 (N)-service-access-point **[ISO/IEC 7498-1]**

DL-service-access-point (N=2)
Ph-service-access-point (N=1)

3.1.33 DL-service-access-point-address **[ISO/IEC 7498-3]**

3.1.34 DL-service-connection-identifier **[ISO/IEC 7498-1]**

3.1.35 DL-service-data-unit **[ISO/IEC 7498-1]**

3.1.36 DL-simplex-transmission **[ISO/IEC 7498-1]**

3.1.37 DL-subsystem **[ISO/IEC 7498-1]**

3.1.38 systems-management **[ISO/IEC 7498-1]**

3.1.39 DLS-user-data **[ISO/IEC 7498-1]**

3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

3.2.1 acceptor

3.2.2 asymmetrical service

3.2.3 confirm (primitive);

requestor.deliver (primitive)

3.2.4 deliver (primitive)

3.2.5 DL-confirmed-facility

3.2.6 DL-facility

3.2.7 DL-local-view

3.2.8 DL-mandatory-facility

3.2.9 DL-non-confirmed-facility

3.2.10 DL-provider-initiated-facility

3.2.11 DL-provider-optional-facility

3.2.12 DL-service-primitive;
primitive

3.2.13 DL-service-provider

3.2.14 DL-service-user

3.2.15 DLS-user-optional-facility

3.2.16 indication (primitive);
acceptor.deliver (primitive)

3.2.17 multi-peer

3.2.18 request (primitive);
requestor.submit (primitive)

3.2.19 requestor

3.2.20 response (primitive);
acceptor.submit (primitive)

3.2.21 submit (primitive)

3.2.22 symmetrical service

3.3 Terms and definitions

3.3.1
blocking
port state at which the port does not participate in frame communication

3.3.2
class
identifier that designates the communication range of the RCL frames and the other frames

3.3.3
control communication
acyclic data communication for high time-critical applications

3.3.4
cyclic communication
periodic data communication for real-time communication

3.3.5
DLCEP-address
DL-address which designates either

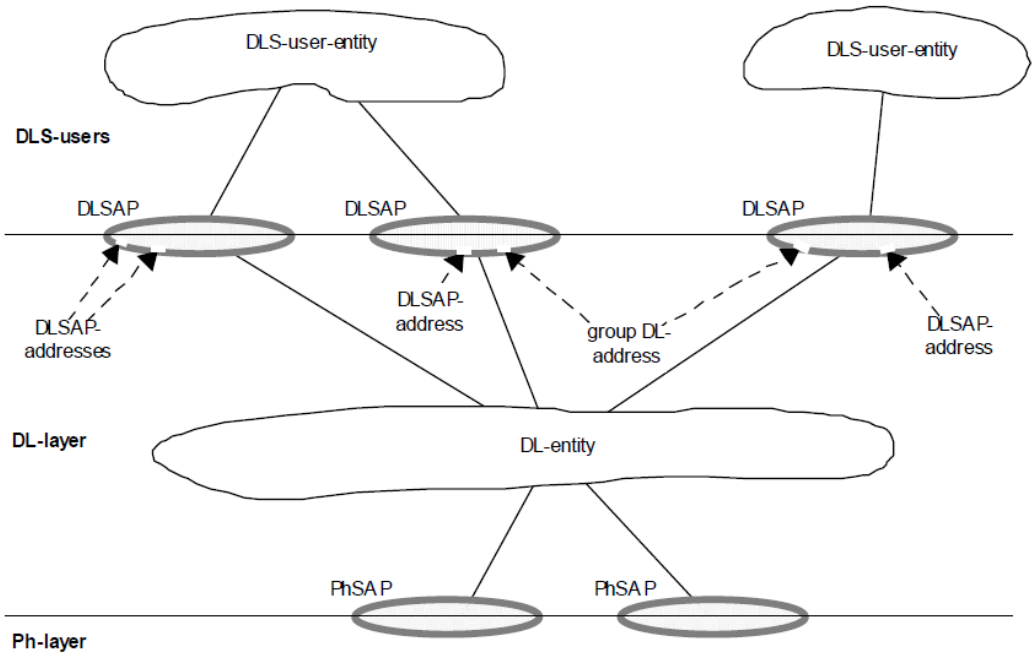
a) one peer DL-connection-end-point, or

b) one multi-peer publisher DL-connection-end-point and implicitly the corresponding set of subscriber DL-connection-end-points where each DL-connection-end-point exists within a distinct DLSAP and is associated with a corresponding distinct DLSAP-address

3.3.6
DL-segment, link, local link
single DL-subnetwork in which any of the connected DLEs may communicate directly, without any intervening DL-relaying, whenever all of those DLEs that are participating in an instance of communication are simultaneously attentive to the DL-subnetwork during the period(s) of attempted communication

3.3.7
DLSAP
distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity

Note 1 to entry: This definition, derived from ISO/IEC 7498-1, is repeated here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses.



NOTE 1 DLSAPs and PhSAPs are depicted as ovals spanning the boundary between two adjacent layers.

NOTE 2 DL-addresses are depicted as designating small gaps (points of access) in the DLL portion of a DLSAP.

NOTE 3 A single DL-entity may have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

Figure 1 – Relationships of DLSAPs, DLSAP-addresses and group DL-addresses

3.3.8
DL(SAP)-address
either an individual DLSAP-address, designating a single DLSAP of a single DLS-user, or a group DL-address potentially designating multiple DLSAPs, each of a single DLS-user

Note 1 to entry: This terminology is chosen because ISO/IEC 7498-3 does not permit the use of the term DLSAP-address to designate more than a single DLSAP at a single DLS-user.

3.3.9
(individual) DLSAP-address
DL-address that designates only one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple DLSAP-addresses associated with a single DLSAP.

3.3.10
extended link
DL-subnetwork, consisting of the maximal set of links interconnected by DL-relays, sharing a single DL-name (DL-address) space, in which any of the connected DL-entities may communicate, one with another, either directly or with the assistance of one or more of those intervening DL-relay entities

3.3.11
frame
denigrated synonym for DLPDU

3.3.12
group DL-address
DL-address that potentially designates more than one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple group DL-addresses associated with a single DLSAP. A single DL-entity also may have a single group DL-address associated with more than one DLSAP.

3.3.13
information communication
acyclic data communication for low time-critical applications

3.3.14
node
single DL-entity as it appears on one local link

3.3.15
receiving DLS-user
DL-service user that acts as a recipient of DLS-user-data

Note 1 to entry: A DL-service user can be concurrently both a sending and receiving DLS-user

3.3.16
ring control (RCL) communication
non-real-time communication for control and reconfiguration in type 25 DLL network using RCL frames

3.3.17
sending DLS-user
DL-service user that acts as a source of DLS-user-data

3.3.18
station
node

3.3.19
station address
identifier address that designates the node of type 25 network

3.4 Symbols and abbreviations

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

DL- Data-link layer (as a prefix)
DLC DL-connection
DLCEP DL-connection-end-point
DLE DL-entity (the local active instance of the data-link layer)
DLL DL-layer
DLM DL-management
DLME DL-management Entity (the local active instance of DL-management)
DLMS DL-management service
DLPCI DL-protocol-control-information
DLPDU DL-protocol-data-unit
DLS DL-service
DLSAP DL-service-access-point
DLSDU DL-service-data-unit
EGA Edge-A node
EGB Edge-B node
FIFO First-in first-out (queuing method)
ISL Isolate node
ITM Intermediate node
LCA Loop condition alert (Type 25 frame type)
LCC Loop condition check (Type 25 frame type)
LCN Loop condition notify (Type 25 frame type)
LLD Logical link down
LLU Logical link up
LNA Loop notify answer (Type 25 frame type)
NNB No neighborhood state
OSI Open systems interconnection
Ph- Physical layer (as a prefix)
PhE Ph-entity (the local active instance of the Physical layer)
PhL Ph-layer
PhS Ph-service
PLD Physical link down
PLU Port A (B) link up state
QoS Quality-of-service
RCL Ring control
RCLC RCL communication control
RHE Rapid hello (Type 25 frame type)
RT Real time
RTC RT communication control
SCR Station condition report (Type 25 frame type)
TRC Transmit/Receive control
WLU Wait link up state

3.5 Common conventions

This standard uses the descriptive conventions given in ISO/IEC 10731.

The service model, service primitives, and time-sequence diagrams used are entirely abstract descriptions; they do not represent a specification for implementation.

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

This standard uses a tabular format to describe the component parameters of the DLS primitives. The parameters that apply to each group of DLS primitives are set out in tables throughout the remainder of this standard. Each table consists of up to six columns, containing the name of the service parameter, and a column each for those primitives and parameter transfer directions used by the DLS:

– the request primitive's input parameters;

– the request primitive's output parameters;

– the indication primitive's output parameters;

– the response primitive's input parameters;

– the confirm primitive's output parameters.

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

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

M – parameter is mandatory for the primitive;

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

C – parameter is conditional upon other parameters or upon the environment of the DLS user;

(blank) – parameter is never present.

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

a) a parameter-specific constraint

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

b) an indication that some note applies to the entry

(n) indicates that the following note n contains additional information pertaining to the parameter and its use.

In any particular interface, not all parameters need be explicitly stated. Some may be implicitly associated with the DLSAP at which the primitive is issued.

In the diagrams which illustrate these interfaces, dashed lines indicate cause-and-effect or time-sequence relationships, and wavy lines indicate that events are roughly contemporaneous.

3.6 Additional Type 25 conventions

The following notation, a shortened form of the primitive classes defined in 3.2, is used in the figures.

req request primitive

ind indication primitive

cnf confirm primitive (confirmation)

rsp response primitive

4 DL services and concepts

4.1 Overview

This standard specifies the Type 25 data-link services for the ISO/IEC 8802-3 based real-time Ethernet. The Type 25 services extend Ethernet according to the ISO/IEC 8802-3 standard with mechanism to control frame flow with IEEE 802.1Q VLAN. The ring topology shall be use in type 25 network. Type 25 network controls the data traffic and without needs time sharing or node synchronization.

This standard specifies the data-link services that are the extension part of the ISO/IEC 8802-3 based data-link layer.

4.2 Types of DLS

4.2.1 General

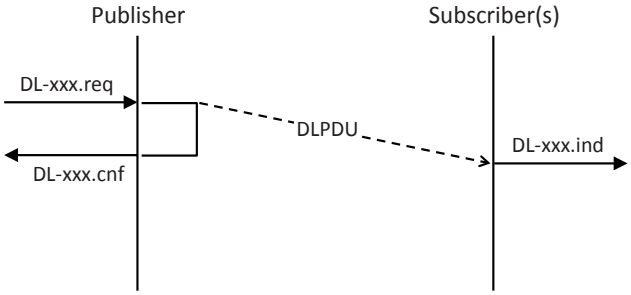
The DLS provides the following service primitives for data transmission and reception:

- RCL communication (for the ring network control and reconfiguration)
- RT communication (for the cyclic, control, and information communication)

The RCL communication controls and reconfigures the Type 25 network and it is non-real-time communication. The RT communication has three types of communication, cyclic, control, and information communication. In cyclic communication, nodes transmit periodically the cyclic communication frames and it guarantees the real-time communication among all nodes on Type 25 network. Control and Information communication are acyclic communication on Type 25 network. Control communication is used in high time-critical acyclic communication. Information communication is used in low time-critical acyclic communication. Control and information communication are non-real-time communication.

4.2.2 Primitive of the RCL communication and RT communication

The sequence of primitives for the RCL communication and RT communication services are shown in Figure 2.



NOTE The method by which a cnf primitive is correlated with its corresponding preceding req primitive is local matter. See 1.2.

Figure 2 – Sequence diagram of RCL communication and RT communication services

4.3 Detailed description of the RCL communication service

4.3.1 Sequence of primitives

These RCL communication service primitives and the parameters are summarized in Table 1, the primitive sequence is shown in Figure 2.

Table 1 – Primitives and parameters used on the RCL communication service

Function	Location	Primitive	Direction	Parameter
Transmit DLSDU	Publisher	DL-RCL.req	To DLE	S_add PortNum Frame_pri RCL_type DLSDU
Confirm DLSDU transmission	Publisher	DL-RCL.cnf	From DLE	Status
Receive DLSDU	Subscriber(s)	DL-RCL.ind	From DLE	S_add PortNum Frame_pri RCL_type DLSDU

4.3.2 Transmit / Receive DLSDU

4.3.2.1 Function

DL-RCL request allows the DLS-user to transfer data for ring control and reconfiguration.

4.3.2.2 Types of primitives and the parameters of the Transmit / Receive DLSDU

Table 2 indicates the parameters of transmit DLSDU service.

Table 2 – Transmit DLSDU primitives and parameters

DL-RCL	Request	Indication	Confirm
Parameter name	Input	Output	Output
S_add	M	M (=)	
Port_Num	M	M	
Frame_pri	M	M (=)	
RCL_type	M	M (=)	
DLSDU	M	M (=)	
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

4.3.2.2.1 S_add

The S_add (Source address) parameter specifies the DL-address of the publisher DLE.

4.3.2.2.2 PortNum

The PortNum parameter indicates the port identification for sending and receiving DLSDU. DLE have three types PortNum, A, B, and Both. In the case of PortNum is “Both”, the node transmits the DLSDU to both port A and port B. When node receives DLSDU, the parameter indicating the received port tells whether it is A or B.

4.3.2.2.3 Frame_pri

The Frame_pri parameter specifies the priority with IEEE 802.1Q VLAN on Type 25 network. In RCL communication service, this value is constant indicating the DLSDU for RCL communication.

4.3.2.2.4 RCL_type

The RCL type parameter specifies the frame type for the Type 25 network ring control and reconfiguration. For ring network control, Type 25 network has frames of several types. The frame types are distinguished by the function and communication area.

4.3.2.2.5 DLSDU

This parameter specifies the DLS-user data that is transferred by the DLE.

4.3.2.2.6 Status

This parameter specifies the DLS-user to determine whether the requested service was provided successfully, or failed due to a particular reason. The possible value conveyed in this parameter is as follows:

a) “success”;

b) “failure” .

4.4 Detailed description of the RT communication service

4.4.1 Sequence of primitives

The RT communication service primitives and the parameters are summarized in Table 3, the primitive sequence is shown in Figure 2.

Table 3 – Primitives and parameters used on the RT communication service

Function	Location	Primitive	Direction	Parameter
Transmit DLSDU	Publisher	DL-RTC.req	To DLE	D_add S_add Frame_pri DLSDU
Confirm DLSDU transmission	Publisher	DL-RTC.cnf	From DLE	Status
Receive DLSDU	Subscriber(s)	DL-RTC.ind	From DLE	D_add S_add Frame_pri DLSDU
NOTE In this table, time increases from top to bottom.				

4.4.2 Transmit / Receive DLSDU

4.4.2.1 Function

DL-RT request allows the DLS-user to transfer data based on ISO/IEC 8802-3. In RT communication, there are three communication types: cyclic, control, and information communication. Type 25 network distinguishes each type of communication by DLSDU parameters.

4.4.2.2 Types of primitives and the parameters of the Transmit / Receive DLSDU

Table 4 indicates the parameters of DLSDU transmission service.

Table 4 – Transmit DLSDU primitives and parameters

DL-RT	Request	Indication	Confirm
Parameter name	Input	Output	Output
D_add	M	M (=)	
S_add	M	M (=)	
Frame_pri	M	M (=)	
DLSDU	M	M (=)	
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

4.4.2.2.1 D_add

The D_add (Destination address) parameter specifies the DL-address of the subscriber DLE. It may be individual or multicast (including broadcast) address. Cyclic communication uses the dedicated multicast address indicating the cyclic communication.

4.4.2.2.2 S_add

The S_add (Source address) parameter specifies the DL-address of the publisher DLE.

4.4.2.2.3 Frame_pri

The Frame_pri parameter specifies the priority with IEEE 802.1Q VLAN on Type 25 network. This value differs among communication types, which are cyclic, control and information communication.

4.4.2.2.4 DLSDU

This parameter specifies the DLS-user data that is transferred by the DLE.

4.4.2.2.5 Status

See 4.3.2.2.6.

5 DL management services

5.1 General

This clause defines the interface between a DLE and a DL-management user (DLMS-user).

5.2 Facilities of the DLMS

DL-management organizes the initialization, configuration, event and error handling between the DLMS-user and the logical functions in the DLE. The following functions are provided to the DLMS-user:

a) Reset of the local DLE;
b) Request for and modification of the actual operating parameters and of the counters of the local DLE;
c) Notification of unexpected events, errors, and status changes, both local and remote;
d) Request for identification and for the DLSAP configuration of the local DLE.

5.3 Service of the DL-management

5.3.1 Overview

DL-management provides the following services to DLMS-user:

a) Reset;
b) Set value;
c) Get value;
d) RCL stop;
e) RCL start;
f) Node status;
g) Event.

Each service is defined as following subclauses.

NOTE DLMS-user may use this service only when the DLE adopts configurable time slot.

5.3.2 Reset

The DLMS-user utilizes this service to make DL-management to reset the DLE. A reset is equivalent to power on. The DLMS-user receives a confirmation thereof.

5.3.3 Set value

The DLMS-user utilizes this service to assign new values to the variables of the DLE. The DLMS-user receives confirmation that the specified variables have been set to the new values.

5.3.4 Get value

This service allows the DLMS-user to read variables of the DLE. The response of the DL management returns the actual value of the specified variables.

5.3.5 RCL stop

This service is an indicator to stop the designated type RCL frames to DLMS-user.

5.3.6 RCL start

This service is an indicator to start the designated type RCL frames to DLMS-user.

5.3.7 Node status

This service indicates the status of the node and ports to DLMS-user.

5.3.8 Event

DL-management utilizes this service to inform the DLMS-user about certain events or error in the DLL.

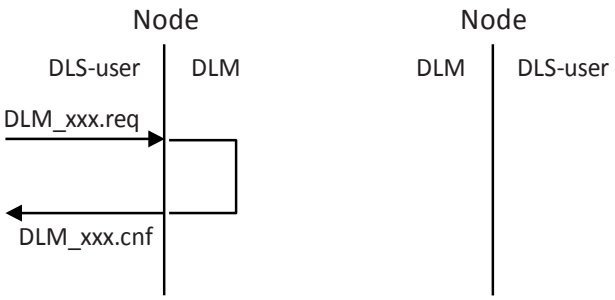
5.4 Overview of interactions

The DL-management services and their primitives are summarized in Table 5.

Table 5 – Transmit DLSDU primitives and parameters

Service	Primitive	Parameter
Reset	DLM_Reset.req	(<none>)
	DLM_Reset.cnf	(out DLM_status)
Set value	DLM_Set.req	(in Variable-name, Desired-value)
	DLM_Set.cnf	(out DLM_status)
Get value	DLM_Get.req	(in Variable-name)
	DLM_Get.cnf	(out DLM_status, Current-value)
RCL stop	DLM_RCL_STOP.ind	(out RCL_Indtype)
RCL start	DLM_RCL_START.ind	(out RCL_Indtype, RCL_IndDA, RCL_IndPri, RCL_IndPort)
Node status	DLM_Node_ST.ind	(out Node_ST, PortA_ST, PortB_ST)
Event	DLM_Event.ind	(out DLM_event identifier, Additional_info)

The temporal relationships of the DL-management primitives are shown in Figure 3 and Figure 4.



NOTE The method by which a cnf primitive is correlated with its corresponding preceding req primitive is local matter. See 1.2.

Figure 3 – Reset, Set value, and Get value services

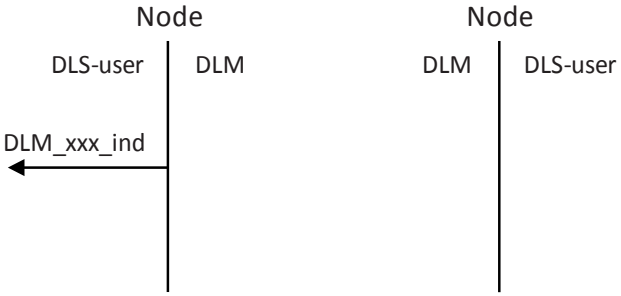


Figure 4 – Event service

5.5 Detail specification of service and interactions

5.5.1 Reset

5.5.1.1 Function

The DML_Reset request primitive causes DLMS to reset the DLE. The DLE is assumed in the “Offline” status, and the reset is carried out in the same manner as in the “power on” procedure; all the DLE variables are cleared. The DLMS user receives the DLM_Reset confirmation primitive along with the status of the result whether it was a success or a failure.

5.5.1.2 Type of primitives and parameters of the Reset

5.5.1.2.1 General

Table 6 indicates the primitives and parameters of the Reset service. This is a local service.

Table 6 – DLM_Reset primitives and parameters

DLM_Reset	Request	Confirm
Parameter name	Input	Output
DLM_Status		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is a local matter. See 1.2.		

5.5.1.2.2 DLM_status

This parameter allows the DLMS-user to determine whether the requested DLMS was provided successfully, or failed for the reason specified. The value conveyed in this parameter is as follows:

a) “OK – successfully completed”;

b) “Failure – terminated before completion”.

5.5.2 Set value

5.5.2.1 Function

This service is used to assign new values to the variables of the DLE. The DLMS-user receives confirmation that the specified variables have been set to the new values.

5.5.2.2 Type of primitives and parameters of the Set value

5.5.2.2.1 General

Table 7 indicates the primitives and parameters of the Set value service. This is a local service.

Table 7 – DLM_Set primitives and parameters

DLM_Set	Request	Confirm
Parameter name	Input	Output
Variable-name	M	
Desired-value	M	
DLM_status		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is a local matter. See 1.2.		

5.5.2.2.2 Variable-name

This parameter specifies the variable within the DLE whose value is set. The selectable variables and their permitted values or value ranges are defined in the corresponding part of IEC PAS 62953-4-25.

5.5.2.2.3 Desired-value

This parameter specifies the desired value for the selected variable.

5.5.2.2.4 DLM_status

This parameter allows the DLMS-user to determine whether the requested DLMS was provided successfully, or failed for the reason specified. The value conveyed in this parameter is as follows:

a) “OK – success – the variable could be updated";

b) “Failure – the variable does not exist or could not assume the new value”;

c) "Failure – invalid parameters in the request".

5.5.3 Get value

5.5.3.1 Function

This service can be used to read the value of a DLE variable. The response of the DLMS returns the actual value of the specified variable.

5.5.3.2 Type of primitives and parameters of the Set value

5.5.3.2.1 General

Table 8 indicates the primitives and parameters of the Get value service. This is a local service.

Table 8 – DLM_Get primitives and parameters

DLM_Get	Request	Confirm
Parameter name	Input	Output
Variable-name	M	
Current-value		M
DLM_status		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is a local matter. See 1.2.		

5.5.3.2.2 Variable-name

This parameter specifies the variable within the DLE whose value is set. The selectable variables and their permitted values or value ranges are defined in the corresponding part of IEC PAS 62953-4-25.

5.5.3.2.3 Current-value

This parameter is present when the status parameter indicates that the requested service was performed successfully. This parameter specifies the current value of the selected variable.

5.5.3.2.4 DLM_status

This parameter allows the DLMS-user to determine whether the requested DLMS was provided successfully, or failed for the reason specified. The value conveyed in this parameter is as follows:

a) “OK – success – the variable could be read";

b) “Failure – the variable does not exist or could not be read”;

c) "Failure – invalid parameters in the request".

5.5.4 RCL stop

5.5.4.1 Function

This service informs the DLMS-user to stop the RCL frames transmission.

5.5.4.2 Type of primitives and parameters of the RCL stop

5.5.4.2.1 General

Table 9 indicates the primitives and parameters of the RCL stop service. This is a local service.

Table 9 – DLM_RCL_STOP primitives and parameters

DLM_RCL_STOP	Indication
Parameter name	Output
RCL_Indtype	M

5.5.4.2.2 RCL_Indtype

This parameter specifies the RCL type of the requested RCL frames for the DLMS-user due to the change of the node status.

5.5.5 RCL start

5.5.5.1 Function

This service is used to inform the DLMS-user about starting the RCL frames.

5.5.5.2 Type of primitives and parameters of the RCL start

5.5.5.2.1 General

Table 10 indicates the primitives and parameters of the RCL start service. This is a local service.

Table 10 – DLM_RCL_START primitives and parameters

DLM_RCL_START	Indication
Parameter name	Output
RCL_Indtype	M
RCL_IndDA	M
RCL_IndPri	M
RCL_IndPort	M

5.5.5.2.2 RCL_Indtype

This parameter specifies the RCL type of the requested RCL frames for the DLMS-user due to the change of the node status.

5.5.5.2.3 RCL_IndDA

This parameter specifies the DL-address of the requested RCL frames for the DLMS-user due to the change of the node status.

5.5.5.2.4 RCL_IndPri

This parameter specifies the priority with VLAN of the requested RCL frames for the DLMS-user due to the change of the node status.

5.5.5.2.5 RCL_IndPort

This parameter specifies the transmission port of the requested RCL frames for the DLMS-user due to the change of the node status.

5.5.5.3 Node status

5.5.5.3.1 General

Table 11 indicates the primitives and parameters of the Node status service. This is a local service.

Table 11 – DLM_RCL_START primitives and parameters

DLM_Node_ST	Indication
Parameter name	Output
Node_ST	M
PortA_ST	M
PortB_ST	M

5.5.5.3.2 Node_ST

This parameter contains the node status (ISL/EGA/EGB/ITM) for the DLMS-user due to the change of the node status.

5.5.5.3.3 PortA_ST

This parameter contains the status of port A on own node for the DLMS-user due to the change of the node status.

5.5.5.3.4 PortB_ST

This parameter contains the status of port B on own node for the DLMS-user due to the change of the node status.

5.5.6 Event

5.5.6.1 Function

This service is used to inform the DLMS-user about certain internal events or errors to the DLS provider.

5.5.6.2 Type of primitives and parameters of the Event

5.5.6.2.1 General

Table 12 indicates the primitives and parameters of the Event service. This is a local service.

Table 12 – DLM_Event primitives and parameters

DLM_Event	Indication
Parameter name	Output
DLM_event identifier	M
Additional_information	C

5.5.6.2.2 DLM_event identifier

This parameter specifies the primitive or composite event within the DLE whose occurrence is being announced.

5.5.6.2.3 Additional_information

This field contains device profile or vendor specific additional information.

Bibliography

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEEE 802.1Q, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks; available at <http://www.ieee.org>*

ISO 15745-4:2003, *Industrial automation systems and integration – Open systems application integration framework – Part 4: Reference description for Ethernet-based control systems; available at <http://www.iso.org>*

CONTENTS

INDUSTRIAL COMMUNICATION NETWORKS - FIELDBUS SPECIFICATIONS –

Part 4-25: Data-link layer protocol specification - Type 25 elements

1 Scope

1.1 General

The data-link layer provides basic time-critical messaging communications between devices in an automation environment.

This protocol provides communication opportunities to all participating data-link entities

a) in a synchronously-starting cyclic manner, according to a pre-established schedule, and

b) in a cyclic or acyclic asynchronous manner, as requested each cycle by each of those data-link entities.

Thus this protocol can be characterized as one which provides cyclic and acyclic access asynchronously but with a synchronous restart of each cycle.

1.2 Specifications

This standard specifies

a) procedures for the timely transfer of data and control information from one data-link user entity to a peer user entity, and among the data-link entities forming the distributed datalink service provider;

b) procedures for giving communications opportunities to all participating DL-entities, sequentially and in a cyclic manner for deterministic and synchronized transfer at cyclic intervals up to one millisecond;

c) procedures for giving communication opportunities available for time-critical data transmission together with non-time-critical data transmission without prejudice to the time-critical data transmission;

d) procedures for giving cyclic and acyclic communication opportunities for time-critical data transmission with prioritized access;

e) procedures for giving communication opportunities based on standard ISO/IEC 8802-3 medium access control, with provisions for nodes to be added or removed during normal operation;

f) the structure of the fieldbus DLPDUs used for the transfer of data and control information by the protocol of this standard, and their representation as physical interface data units.

1.3 Procedures

The procedures are defined in terms of

a) the interactions between peer DL-entities (DLEs) through the exchange of fieldbus DLPDUs;

b) the interactions between a DL-service (DLS) provider and a DLS-user in the same system through the exchange of DLS primitives;

c) the interactions between a DLS-provider and a Ph-service provider in the same system through the exchange of Ph-service primitives.

1.4 Applicability

These procedures are applicable to instances of communication between systems which support time-critical communications services within the data-link layer of the OSI or fieldbus reference models, and which require the ability to interconnect in an open systems interconnection environment.

Profiles provide a simple multi-attribute means of summarizing an implementation's capabilities, and thus its applicability to various time-critical communications needs.

1.5 Conformance

This standard also specifies conformance requirements for systems implementing these procedures. This standard does not contain tests to demonstrate compliance with such requirements.

2 Normative references

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

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

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

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications*

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

IEEE 802.1D-2004, IEEE Standard for Local and metropolitan area networks – Media access control (MAC) Bridges, available at <http://www.ieee.org>

IEEE 802.1Q-2005, IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks, available at <http://www.ieee.org>

3 Terms, definitions, symbols and abbreviations

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

3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein:

3.1.1 DL-address **[ISO/IEC 7498-3]**

3.1.2 DL-address-mapping **[ISO/IEC 7498-1]**

3.1.3 called-DL-address **[ISO/IEC 7498-3]**

3.1.4 calling-DL-address **[ISO/IEC 7498-3]**

3.1.5 centralized multi-end-point-connection **[ISO/IEC 7498-1]**

3.1.6 DL-connection **[ISO/IEC 7498-1]**

3.1.7 DL-connection-end-point **[ISO/IEC 7498-1]**

3.1.8 DL-connection-end-point-identifier **[ISO/IEC 7498-1]**

3.1.9 DL-connection-mode transmission **[ISO/IEC 7498-1]**

3.1.10 DL-connectionless-mode transmission **[ISO/IEC 7498-1]**

3.1.11 correspondent (N)-entities **[ISO/IEC 7498-1]**

correspondent DL-entities (N=2)

correspondent Ph-entities (N=1)

3.1.12 DL-duplex-transmission **[ISO/IEC 7498-1]**

3.1.13 (N)-entity **[ISO/IEC 7498-1]**

DL-entity (N=2)

Ph-entity (N=1)

3.1.14 DL-facility **[ISO/IEC 7498-1]**

3.1.15 flow control **[ISO/IEC 7498-1]**

3.1.16 (N)-layer **[ISO/IEC 7498-1]**

DL-layer (N=2)

Ph-layer (N=1)

3.1.17 layer-management **[ISO/IEC 7498-1]**

3.1.18 DL-local-view **[ISO/IEC 7498-3]**

3.1.19 DL-name **[ISO/IEC 7498-3]**

3.1.20 naming-(addressing)-domain **[ISO/IEC 7498-3]**

3.1.21 peer-entities **[ISO/IEC 7498-1]**

3.1.22 primitive name **[ISO/IEC 7498-3]**

3.1.23 DL-protocol **[ISO/IEC 7498-1]**

3.1.24 DL-protocol-connection-identifier **[ISO/IEC 7498-1]**

3.1.25 DL-protocol-data-unit **[ISO/IEC 7498-1]**

3.1.26 DL-relay **[ISO/IEC 7498-1]**

3.1.27 reset **[ISO/IEC 7498-1]**

3.1.28 responding-DL-address **[ISO/IEC 7498-3]**

3.1.29 routing **[ISO/IEC 7498-1]**

3.1.30 segmenting **[ISO/IEC 7498-1]**

3.1.31 (N)-service **[ISO/IEC 7498-1]**

DL-service (N=2)

Ph-service (N=1)

3.1.32 (N)-service-access-point **[ISO/IEC 7498-1]**

DL-service-access-point (N=2)

Ph-service-access-point (N=1)

3.1.33 DL-service-access-point-address **[ISO/IEC 7498-3]**

3.1.34 DL-service-connection-identifier **[ISO/IEC 7498-1]**

3.1.35 DL-service-data-unit **[ISO/IEC 7498-1]**

3.1.36 DL-simplex-transmission **[ISO/IEC 7498-1]**

3.1.37 DL-subsystem **[ISO/IEC 7498-1]**

3.1.38 systems-management **[ISO/IEC 7498-1]**

3.1.39 DLS-user-data **[ISO/IEC 7498-1]**

3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

3.2.1 acceptor

3.2.2 asymmetrical service

3.2.3 confirm (primitive);

requestor.deliver (primitive)

3.2.4 deliver (primitive)

3.2.5 DL-confirmed-facility

3.2.6 DL-facility

3.2.7 DL-local-view

3.2.8 DL-mandatory-facility

3.2.9 DL-non-confirmed-facility

3.2.10 DL-provider-initiated-facility

3.2.11 DL-provider-optional-facility

3.2.12 DL-service-primitive;
primitive

3.2.13 DL-service-provider

3.2.14 DL-service-user

3.2.15 DLS-user-optional-facility

3.2.16 indication (primitive);
acceptor.deliver (primitive)

3.2.17 multi-peer

3.2.18 request (primitive);
requestor.submit (primitive)

3.2.19 requestor
response (primitive);
acceptor.submit (primitive)

3.2.20 submit (primitive)

3.2.21 symmetrical service

3.3 Terms and definitions

3.3.1
blocking
a port state which does not participate in frame communication

3.3.2
class
identifiers that designate communication range of the RCL frame and the other frames

3.3.3
control communication
non-real-time acyclic data communication for higher priority applications and node control communication

3.3.4
cyclic communication
periodic data communication for real-time communication

3.3.5
DLCEP-address
DL-address which designates either

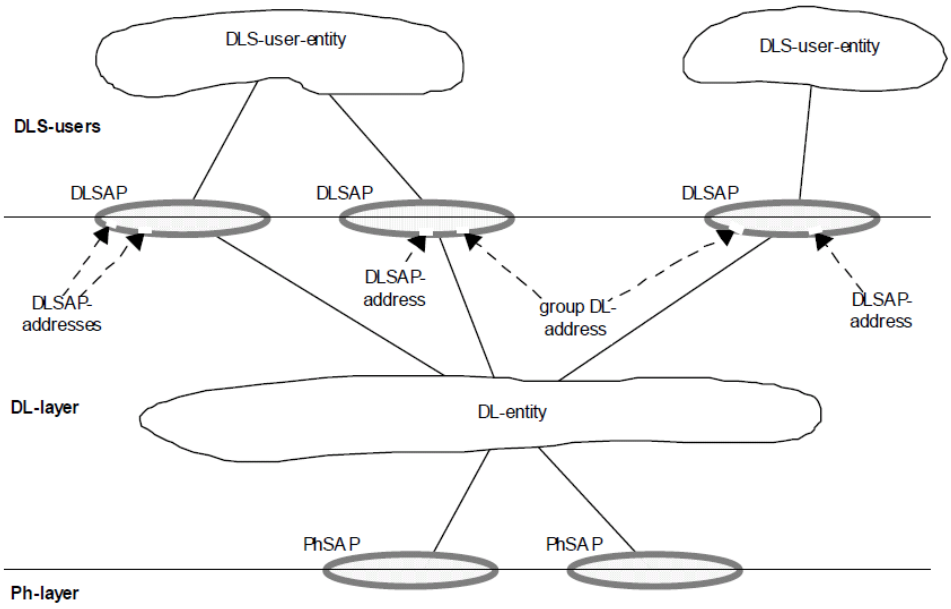
a) one peer DL-connection-end-point, or

b) one multi-peer publisher DL-connection-end-point and implicitly the corresponding set of subscriber DL-connection-end-points where each DL-connection-end-point exists within a distinct DLSAP and is associated with a corresponding distinct DLSAP-address.

3.3.6
DL-segment, link, local link
single DL-subnetwork in which any of the connected DLEs may communicate directly, without any intervening DL-relaying, whenever all of those DLEs that are participating in an instance of communication are simultaneously attentive to the DL-subnetwork during the period(s) of attempted communication

3.3.7
DLSAP
distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity

Note 1 to entry: This definition, derived from ISO/IEC 7498-1, is repeated here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses.



NOTE 1 DLSAPs and PhSAPs are depicted as ovals spanning the boundary between two adjacent layers.

NOTE 2 DL-addresses are depicted as designating small gaps (points of access) in the DLL portion of a DLSAP.

NOTE 3 A single DL-entity may have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

Figure 1 – Relationships of DLSAPs, DLSAP-addresses and group DL-addresses

3.3.8
DL(SAP)-address
either an individual DLSAP-address, designating a single DLSAP of a single DLS-user, or a group DL-address potentially designating multiple DLSAPs, each of a single DLS-user

Note 1 to entry: This terminology is chosen because ISO/IEC 7498-3 does not permit the use of the term DLSAP-address to designate more than a single DLSAP at a single DLS-user.

3.3.9
(individual) DLSAP-address
DL-address that designates only one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple DLSAP-addresses associated with a single DLSAP.

3.3.10
extended link
DL-subnetwork, consisting of the maximal set of links interconnected by DL-relays, sharing a single DL-name (DL-address) space, in which any of the connected DL-entities may communicate, one with another, either directly or with the assistance of one or more of those intervening DL-relay entities

3.3.11
frame
denigrated synonym for DLPDU

3.3.12
group DL-address
DL-address that potentially designates more than one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple group DL-addresses associated with a single DLSAP. A single DL-entity also may have a single group DL-address associated with more than one DLSAP.

3.3.13
information communication
non-real-time acyclic data communication for low priority applications

3.3.14
logical link down
link status at which the port is in a blocking state and does not communicate all kinds of frames except RCL frames

3.3.15
logical link up
link status at which the port communicates all kinds of frames

3.3.16
node
single DL-entity as it appears on one local link

3.3.17
physical link down
link status at which the port does not communicate the frames due to link down status defined in ISO/IEC 8802-3

3.3.18
receiving DLS-user
DL-service user that acts as a recipient of DLS-user-data

Note 1 to entry: A DL-service user can be concurrently both a sending and receiving DLS-user

3.3.19
ring control (RCL) communication
control communication of type 25 DLL ring network using RCL frames and non-real-ime

3.3.20
sending DLS-user
DL-service user that acts as a source of DLS-user-data

3.3.21
station
node

3.3.22
station address
identifier address that designates the node of type 25 network

3.4 Symbols and abbreviations

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

DL- Data-link layer (as a prefix)
DLC DL-connection
DLCEP DL-connection-end-point
DLE DL-entity (the local active instance of the data-link layer)
DLL DL-layer
DLM DL-management
DLME DL-management Entity (the local active instance of DL-management)
DLMS DL-management service
DLPCI DL-protocol-control-information
DLPDU DL-protocol-data-unit
DLS DL-service
DLSAP DL-service-access-point
DLSDU DL-service-data-unit
EGA Edge-A node
EGB Edge-B node
FIFO First-in first-out (queuing method)
ISL Isolate node
ITM Intermediate node
LCA Loop condition alert (Type 25 frame type)
LCC Loop condition check (Type 25 frame type)
LCN Loop condition notify (Type 25 frame type)
LLD Logical link down
LLU Logical link up
LNA Loop notify answer (Type 25 frame type)
NNB No neighborhood state
OSI Open systems interconnection
Ph- Physical layer (as a prefix)
PhE Ph-entity (the local active instance of the Physical layer)
PhL Ph-layer
PhS Ph-service
PLD Physical link down
PLU Port A (B) link up state
QoS Quality-of-service
RCL Ring control

RCLC RCL communication control

RHE Rapid hello (Type 25 frame type)

RT Real time

RTC RT communication control

SCR Station condition report (Type 25 frame type)

TRC Transmit/Receive control

WLU Wait link up state

3.5 Common conventions

This standard uses the descriptive conventions given in ISO/IEC 10731.

The service model, service primitives, and time-sequence diagrams used are entirely abstract descriptions; they do not represent a specification for implementation.

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

This standard uses a tabular format to describe the component parameters of the DLS primitives. The parameters that apply to each group of DLS primitives are set out in tables throughout the remainder of this standard. Each table consists of up to six columns, containing the name of the service parameter, and a column each for those primitives and parameter transfer directions used by the DLS:

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

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

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

M – parameter is mandatory for the primitive;

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

C – parameter is conditional upon other parameters or upon the environment of the DLS user;

(blank) – parameter is never present.

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

a) a parameter-specific constraint

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

b) an indication that some note applies to the entry

(n) indicates that the following note n contains additional information pertaining to the parameter and its use.

In any particular interface, not all parameters need be explicitly stated. Some may be implicitly associated with the DLSAP at which the primitive is issued.

In the diagrams which illustrate these interfaces, dashed lines indicate cause-and-effect or time-sequence relationships, and wavy lines indicate that events are roughly contemporaneous.

3.6 Additional Type 25 conventions

3.6.1 Primitive conventions

The following notation, a shortened form of the primitive classes defined in 3.2, is used in the figures.

req request primitive

ind indication primitive

cnf confirm primitive (confirmation)

rsp response primitive

3.6.2 State machine conventions

The protocol sequences are described by means of state machines.

In state diagrams, states are represented as boxes and state transitions are shown as arrows.

Names of states and transitions of the state diagram correspond to the names in the state table. The textual listing of the state transitions is structured as shown in Table 1.

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 must 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.
NOTE “*/ conditions” may be omitted	

The conventions used in the state machines are shown in Table 3.

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
++	Increment operator
--	Decrement operator
;	Breakpoint

4 Overview of the DL-protocol

4.1 General

A type 25 fieldbus is based on the ISO/IEC 8802-3 standard but extended towards real-time Ethernet. Type 25 network controls the data traffic and assigns frame priorities with IEEE 802.1Q VLAN.

NOTE Any standard Ethernet silicon, infrastructure component or test and measurement equipment such as a network analyzer is applicable to type 25 network.

4.2 Overview of the medium access control

4.2.1 General

Type 25 network provides mechanisms to achieve the following.

a) It assigns priorities to four types of frames (RCL/Cyclic/Control/Information).

b) The frames in type 25 network are guaranteed real-time of cyclic communication to control the data traffic.

c) It uses ISO/IEC 8802-3 standard communication without needs for special functions for RTE communication such as time sharing or node synchronization.

d) The ring topology is only permitted in type 25 network. In order to control the network, it categorizes the frames into three classes according to their communication ranges.

4.2.2 Network topology

The ring topology shall be use in type 25 network. The network shall be controled and reconfigured by RCL frames. Type 25 ring network does not need concept of master/slave model because each node in type 25 network reconfigures autonomously. To prevent frame loop, one section in the ring network is in the blocking state (IEEE 802.1D) and the blocked section terminates all kinds of frames except RCL frame (as illustrated in Table 2).

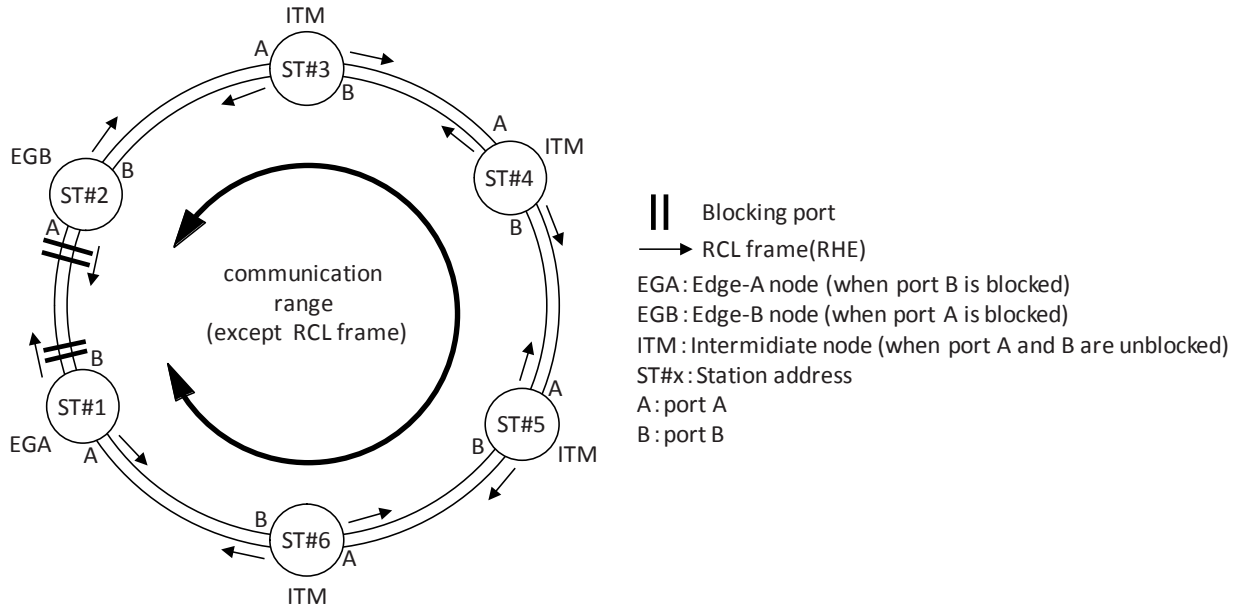


Figure 2 – Ring control in Type 25 network

Each node in the ring network has two ports A and B to connect to the neiboring nodes. Each node has its own station address. In type 25 network, each node has four different node states depending on the blocking status of their ports. Table 4 shows the characteristics of the node states.

Table 4 – Characteristics of the node states

Node states	Abbreviation	Description
Isolated node	ISL	Both port A and B are blocked in this state (e.g. not connect cables to other nodes) and the node is isolated from the type 25 network.
Edge-A node (when port B is blocked)	EGA	Port B is blocked, port A is unblocked. The node works with Edge-A. There is only one node in a Type 25 network.
Edge-B node (when port A is blocked)	EGB	Port A is blocked, port B is unblocked. The node works with Edge-B. There is only one node in a Type 25 network.
Intermediate node (when port A, B are unblocked)	ITM	Both port A and B are unblocked. The node works with Edge-A and Edge-B. In a Type 25 network, there are an Edge-A node and an Edge-B node. The other nodes are this state.

RCL frames are based on general ISO/IEC 8802-3 frames. These frames have different ranges of communication which are given by “class”. Table 5 shows the characteristics of the frame class and Figure 3 shows the communication ranges in Type 25 network.

Table 5 – Characteristic of the frame classes

Frame class	Description
Class 1	This frame is used to communicate with the neighboring node only. It passes through blocking ports. RHE frame are RCL frames in this class.
Class 2	This frame is used to communicate around the network. This frame is able to pass through blocking ports and the Edge-A node terminates the frames. LCC, LCA, LCN, LNA, and SCR frames are RCL frames in this class.
Class 3	This frame is general Ethernet frame and terminated by blocking ports. The RT frames (Cyclic, Control, Information frames) are in this class.

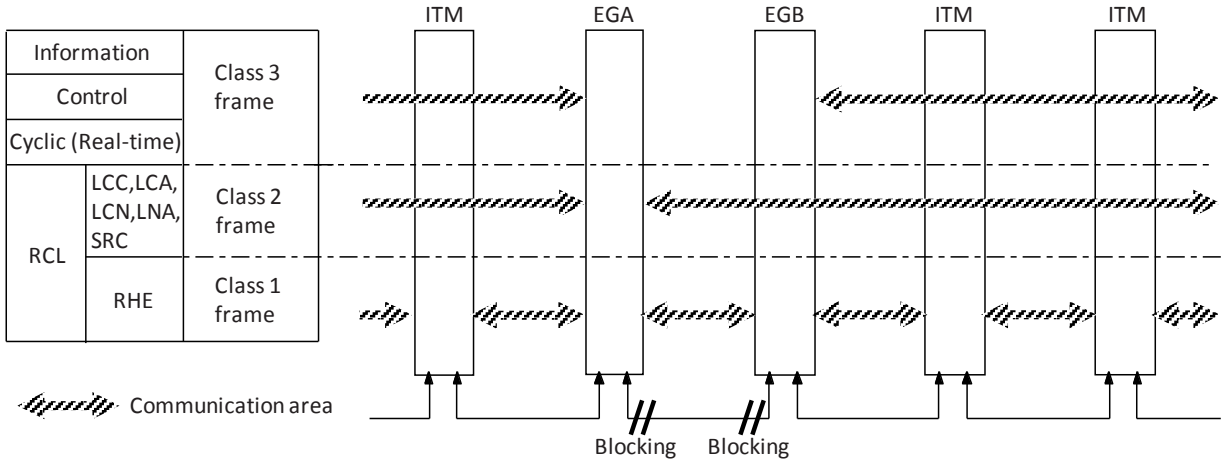


Figure 3 – Communication ranges of Type 25 frames

4.2.3 Priority control with VLAN

The Type 25 fieldbus has a medium access control with VLAN priority in order to provide the real-time communication. Figure 4 shows the mechanism of priority control with VLAN of the Type 25 fieldbus.

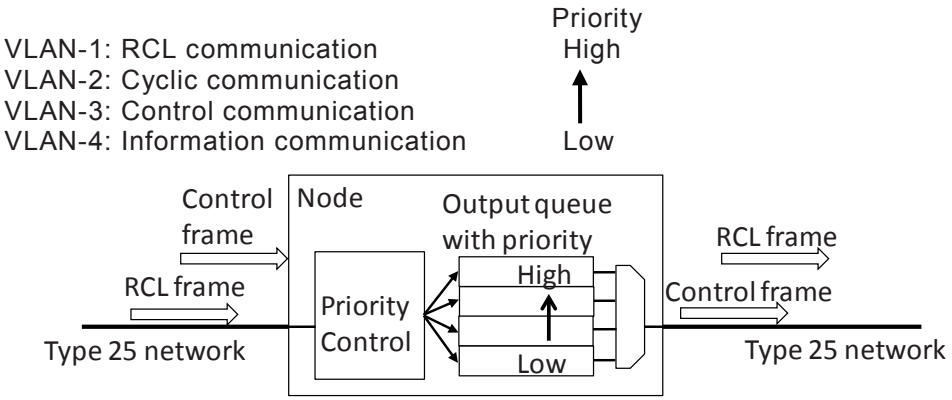


Figure 4 – Priority control with VLAN of Type 25 network

Each node has output queues with priority. The receiving frames are put into the output queues based on their frame priorities. The frame with higher priority will be forwarded neighboring node. In this mechanism, Type 25 network guarantees real-time cyclic communication.

4.2.4 The maximum delivery delay in Type 25 network

Type 25 network limits data traffic in the network to ensure real time communication. This maintains the communication delay of frames at an acceptable level. The traffic restriction of each node is determined by the total number of nodes in the network and cycle time using the node.

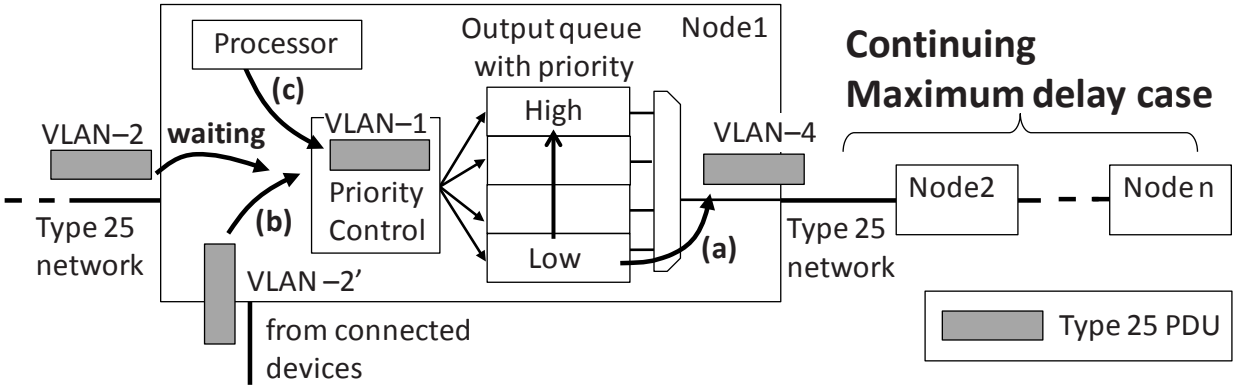


Figure 5 – The mechanism of transmission delay in a node

There are three causes of the delay regarding VLAN-2 (cyclic communication) frame in Type 25 network.

a) VLAN-2 frame arrives at the time when the node is sending another frame. The VLAN-2 frame is forced to wait until the transmission is finished. This delay is caused by any priority frames.

b) Before receiving a VLAN-2 frame, the node receives another VLAN-2 frame (VLAN-2' in Figure 5) from other ports. In this case, at the priority queue the VLAN2 frame is put after the VLAN-2' frame.

c) When the VLAN-2 frame is waiting for transmission due to the causes a) and b), the node receives higher priority frames (VLAN-1 frame). Since the node will transmit the highest priority frames first, the VLAN-2 frame is forced to wait until the transmission of higher priority frames are finished.

When three causes happen all at once in a node, the maximum delay occurs. The delivery time could be calculated in the case of the maximum delay occurring at all nodes in a Type 25 network.

4.2.5 Traffic control for real-time communication

Delivery time in Type 25 network is calculated based on the maximum delay time per node, the number of nodes, and the length of ring network.

The maximum delay of real-time communication (cyclic communication) between two nodes is calculated by taking into account the four delay causes shown in Figure 6. The detail of each cause is shown in IEC 61784-2 Profile 20/1.

1) Stacking delay in sending node:

   The delay caused by the transmission process. It depends on the implementation of the hardware and software.

2) Cable segment delay:

   The delay caused by transmission cable constructing ring network. It is proportional to the cable length.

3) Transmission delay in transit node

The delay caused by transmission delay per transit node. It is proportional to the number of nodes in the transmission path. The detail is shown in 4.2.4.

4) Stacking delay in receiving node:

The delay caused by the receiving process. It depends on the implementation of the hardware and software.

The delivery time is calculated by the following equation.

(delivery time) = (1) + (2) × cable length + (3) × (the number of nodes – 2) + (4)

The cycle time of cyclic communication shall be set to larger than the delivery time to guarantee real-time communication of cyclic communication frames in Type 25 network.

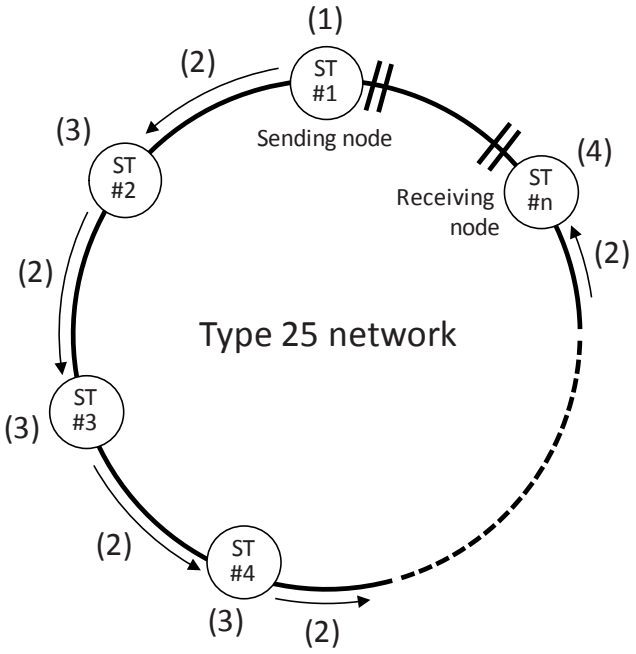


Figure 6 – The worst delay in Type 25 network

4.3 Service assumed from PhL

This subclause describes the assumed physical service (PhS) and the constraints used by the DLE. The Physical Service is assumed to provide the following service primitives specified by ISO/IEC 8802-3, Clause 2.

The assumed primitives of PhS are

– PhS-A_DATA.req,
– PhS-A_DATA.ind,
– PhS-B_DATA.req,
– PhS-B_DATA.ind.

4.4 DL Layer architecture

The Type 25 fieldbus DLL is modeled as a combination of control components of RCL communication control (RCLC), Real-time communication control (RTC), Transmit/Receive control (TRC) and DL-management (DLM).

The data-link layer is comprised of the components shown in Table 6.

Table 6 – Data-link layer components

Components	Description
RCL communication control (RCLC)	At RCLC, DLSDU for the ring control (RCL) data is stored and transferred between the DLS-user and the TRC.
Real-time communication control (RTC)	At RTC, control data/information data is stored and transferred between DLS-user and the TRC.
Transmit/Receive control (TRC)	At TRC, DLSDU is stored and transferred from the RCLC/RTC to Ph-Layer. DLPDU is stored and transferred from Ph-Layer to the RCLC/RTC.
DL-management (DLM)	The DL-management interface holds the station management variables that belong to the DLL, and manages synchronized changes of the link parameters.

The internal arrangement of these components, and their interfaces, is shown in Figure 7. The arrowheads illustrate the primary direction of the flow of data and control.

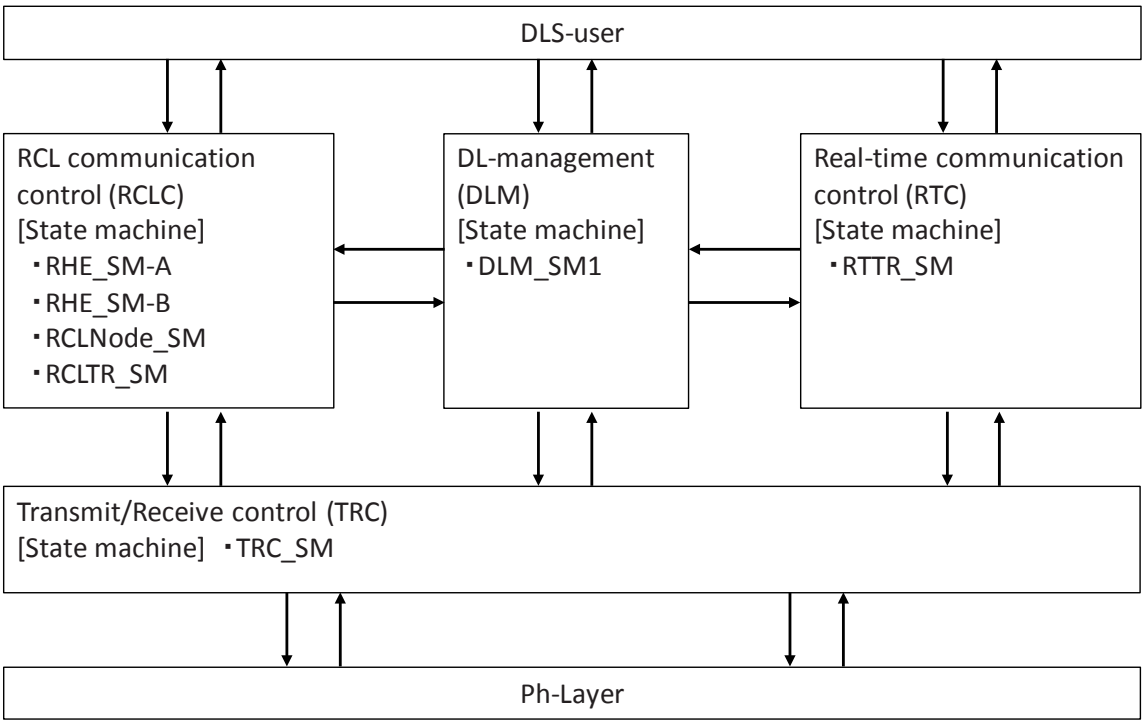


Figure 7 – Data-Link layer internal architecture

4.5 Local parameters and variables

4.5.1 Overview

This specification uses DLS-user request parameters P(...) and local variables V(...) as a means of clarifying the effect of certain actions and the conditions under which those actions are valid, local timers T(...) as a means of monitoring actions of the distributed DLS-provider and of ensuring a local DLE response to the absence of those actions, and local counters C(...) for performing rate measurement functions.

Unless otherwise specified, at the moment of their creation or of DLE activation:

a) all variables shall be initialized to their default value, or to their minimum permitted value if no default is specified;

b) all counters shall be initialized to zero;

c) all timers shall be initialized to inactive;

DL-management may change the values of configuration variables.

4.5.2 Variables, parameter, counter and timer

4.5.2.1 P(RHE_RxCK_NUM)

This parameter indicates the expiration time of not receiving RHE frames.

4.5.2.2 P(RHE_LKUP_NUM)

This parameter indicates the number of receiving RHE frames for logical linkup.

4.5.2.3 P(LCC_STOP_NUM)

This parameter indicates the threshold number to stopthe LCC frame transmission.

4.5.2.4 C(PA_RHE_LKUP), C(PB_RHE_LKUP)

These counters indicate the count of continuously receiving RHE frames at port A and port B, respectively.

4.5.2.5 C(PA_RHE_RxCK), C(PB_RHE_RxCK)

These counters are used to check whether continuously received or droped RHE frames. Each value is decreased in T(PA_RHE_TIME) or T(PB_RHE_TIME).

4.5.2.6 C(PA_Rx-RHE_NotMatch), C(PB_Rx-RHE_NotMatch)

These counters are used to detect the change from neighborhood node to another node.

4.5.2.7 C(Rx_Counter)

This counter indicates the number of receiving LCC frames by own node.

4.5.2.8 T(PA_RHE_TIME), T(PB_RHE_TIME)

T(PA_RHE_TIME) and T(PB_RHE_TIME) are used for fixed cycle to do RHE operation.

4.5.2.9 V(PA_Blocking), V(PB_Blocking)

These variables hold and designate the blocking status at the ports respectively. When the port is blocked, these variables are set to “True”, otherwise “False”.

4.5.2.10 V(LCN_Pri)

This variable is a unique value used to decide an Edge-A node in the network. This variable is used when the node sends LCN frames.

4.5.2.11 V(LNA_Pri)

This variable is a unique value used to decide an Edge-A node in the network. This variable is used when the node sends LNA frames.

4.5.2.12 V(PA_NBST_LKST), V(PB_NBST_LKST)

These variables hold and designate the link status of a neighborhood node. It takes three types value (NNB, WLU, and PLU). These values are updated by receiving the RHE frames.

4.5.2.13 V(PA_NBST_NDST), V(PB_NBST_NDST)

This variable holds and designates the node status of a neighborhood node. It takes four types value (ISL, EGA, EGB, and ITM) and updates the value by receiving the RHE frames.

4.5.2.14 V(PA_NBST_RCLADD), V(PB_NBST_RCLADD)

These variables hold and designate the address of a neighborhood node. These values are updated by receiving the RHE frames.

4.5.2.15 V(PA_RHE_CYCLE), V(PB_RHE_CYCLE)

These variables hold and designate the cycle time to do RHE operation.

4.5.2.16 V(PA_RHE_RxSeq), V(PB_RHE_RxSeq)

These variables hold and designate the sequence number of receiving RHE frames. If RHEframes from neighborhood node are received continuously, these values increase.

4.5.2.17 V(PA_STATUS), V(PB_STATUS)

These variables hold and designate the port link status of own node. They take three values (NNB, WLU, and PLU).

5 General structure and encoding of PhPDUs and DLPDU and related elements of procedure

5.1 Overview

The DLL and its procedures are necessary to provide the services offered to the DLS user by using the services available from the PhL. This clause describes the structure and semantics of DLPDU and the procedure, commonly used in this specification.

5.2 Common MAC frame structure, encoding and elements of procedure

5.2.1 MAC frame structure

Figure 8 shows the Type 25 fieldbus DLPDU.

Preamble (7)	SFD (1)	Destination address (6)	Source address (6)	VLAN tag (4)	Length/ Type (2)	DLS-user data	Padding	FCS (4)

(*)the number in parenthesis indicates length of the field in octets

Figure 8 – Type 25 fieldbus DLPDU frame format

5.2.2 Elements of the MAC frame

5.2.2.1 Preamble

The preamble field is identical to ISO/IEC 8802-3, Clause 3. The preamble field is a 56-bit field that is used to allow the physical signaling part circuitry to reach its steady state synchronization with the receiving frame timing.

The preamble pattern is:

“10101010 10101010 10101010 10101010 10101010 10101010 10101010.”

5.2.2.2 Start frame delimiter (SFD)

The Start Frame Delimiter (SFD) is identical to ISO/IEC 8802-3, Clause 3. The SFD field is the sequence of bit pattern “10101011”. It immediately follows the preamble pattern and indicates the start of a frame.

5.2.2.3 Destination address

The Destination address is identical to ISO/IEC 8802-3, Clause 3. This address field is a 48-bit in length. The frames received and sent at each node are classified into three class (class 1, 2, and 3) based on the communication range. The detail of the class is shown in 4.2.2.

The frame class is distinguished by destination address. Table 7 shows the destination address of three types of classes.

Table 7 – Destination address format

Frame class	Frame type	Destination address
class 1	RCL (RHE)	01-80-C2-00-00-0F
class 2	RCL (LCC/LCA/LCN/LNA/SCR)	01-80-C2-00-00-0E
class 3	Cyclic/Control/Information	Destination MAC address of the DLSDU

5.2.2.4 Source address

Source address shall contain the node address of the source DLE.

5.2.2.5 VLAN tag

VLAN tag uses IEEE 802.1Q VLAN tag, and indicates frame type (RCL, cyclic, control, and information) designated VID and frame priority designated PCP.

Table 8 – VLAN tag format

frame type	TPID (2 octets)	TCI (2 octets)		
		PCP (3 bit)	CFI (1 bit)	VID (12 bit)
RCL	0x8100	111	0	0xFFB
Cyclic	0x8100	101	0	0xFFC
Control	0x8100	011	0	0xFFD
Information	0x8100	001	0	0xFFE(*)
(*) Information frames may use other VID than 0xFFE(4094) from 1 to 50 and designate multiple VIDs.				

5.2.2.6 Length/Type

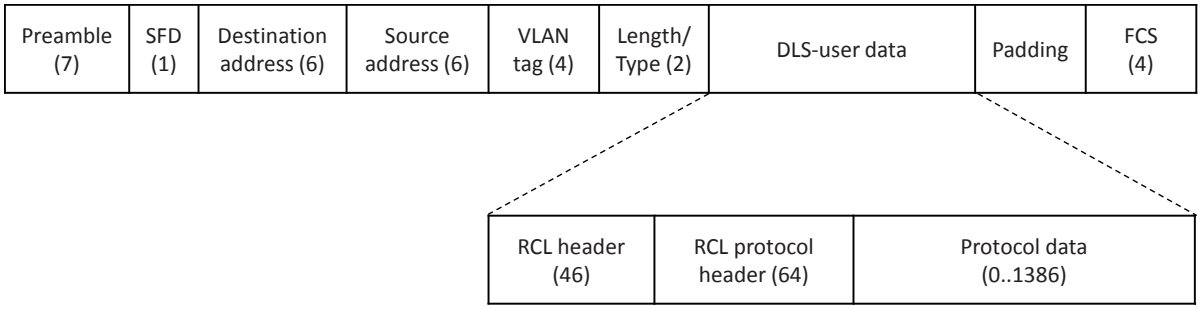
The Length/type is different for frame types.

– RCL frame is set to the frame length. It is different for the RCL_type.
– cyclic frame is set to 0x0800(designated IP frame).
– control and information frames are set to protocol ID that corresponds to PDU data.

5.2.2.7 DLS-user data field

5.2.2.7.1 RCL frame

RCL frame has three segments, RCL header, RCL protocol header, and Protocol data in DLS-user data. Each segment contains information which is used to control the ring network. The information differs according to RCL frame types.



(*)the number in parenthesis indicates length of the field in octets

Figure 9 – RCL frame format

5.2.2.7.2 RT frame

The RT frames are identical to the ISO/IEC 8802-3.

5.2.2.8 Padding

Padding is inserted into the frame so that the data length is always a multiple of 32 bits.

5.2.2.9 Field check sequence (FCS)

The frame check sequence (FCS) construction, polynomial and expected residual are identical to ISO/IEC 8802-3, Clause 3.

6 DLPDU-specific structure, encoding and elements of procedure

6.1 General

This clause defines the structure, contents and encoding of each type and format of the RCL communication DLPDU, and specifies elements of procedure for the DLPDU. The RT communication DLPDU is not described in this clause since its frame is identical to the ISO/IEC 8802-3.

6.2 Structure of the RCL DLPDU

Frame of class 1 is used to communicate to the neighboring node. Frame of class 2 is used to communicate between nodes around the network. For the RCL frame, there are one kind of class 1 frame and five kinds of class 2 frame. Table 9 shows the types and classes of RCL frames.

Table 9 – Types and classes of RCL frames

frame class	frame name	Description
Class 1	RHE (Rapid Hello)	The RHE frame confirms the neighboring node and transmission path.
Class 2	LCC (*) (Loop Condition Check)	The LCC frame is used to determine autonomously the single Edge-A node, which exists in the normal ring network.
Class 2	LCA (*) (Loop Condition Alert)	If a node's port B is physical or logical link down, the node's LCA frame declares that it has higher priority to be Edge-A node.
Class 2	LCN (*) (Loop Condition Notify)	A node's LCN frame declares that its own port B changes to physical link down state.
Class 2	LNA (*) (Loop Notify Answer)	The LNA frame is sent to the node which sent LNA frame. Only Edge-B node sends this frame.
Class 2	SCR (Station Condition Report)	The SCR frame notifies the event of change in network topology to all nodes in the ring network.
(*) LCC, LCA, LCN, and LNA have different priority to select an Edge-A node: LCN > LCA > LCC (high priority) > LCC (low priority)		

6.2.1 RCL header

6.2.1.1 General

The structure of RCL DLPDU is shown in Table 10.

Table 10 – Structure of RCL header

RCL header field		Size (Octet)
Class		2
Destination address	Priority	1
	Station address	1
	MAC address	6
Source address	Priority	1
	Station address	1
	MAC address	6
CMD		4
Sequence number		4
Reserved		20

6.2.1.2 Class

There are two kinds of class in RCL frame.

– Class 1: used to communicate to the neighboring node;
  Class 2: used to communicate between nodes around the network.

Table 11 – Class field format

frame class	Value(2 octets)
class 1	0x0001
class 2	0x0002

6.2.1.3 Destination address

Set value of priority, station address, MAC address, depending on RCL_type.

Table 12 – Destination address field format

item	RCL_type	
	RHE/LCC/LCN/SCR	LCA/LNA
Priority	0x00	set to the source address that receives RCL frame (LCA: received LCC frame, LNA: received LCN frame)
Station address	0xFF	
MAC address	FF-FF-FF-FF-FF-FF	

6.2.1.4 Source address

All RCL frames are set to the data shown in Table 13.

Table 13 –Source address field format

item	Value
Priority	0x00
Station address	Station address of the node
MAC address	MAC address of the node

6.2.1.5 CMD

The CMD is used to identify the RCL frame. This area is different according to the frame type (RCL_type).

Table 14 – CMD field format

Bit	item	Value					
		RHE	LCC	LCA	LCN	LNA	SCR
[3..0]	Reserved	1					
[7..4]	Unique number 1	0	0	1	2	3	0
[11..8]	Unique number 2	0	1	1	1	1	3
[15..12]	Reserved	0					
[23..16]	Frame class	1	2	2	2	2	2
[31..24]	Reserved	0					

6.2.1.6 Sequence number

Sequence number is used to detect loss and duplication of received frames. Its value ranges from 0x00000000 to 0xFFFFFFFF with an increment of 1.

7 DLE elements of procedure

7.1 Overview

In following subclause, the operation of RCLC, RTC, TRC, and DLM are described.

7.2 RCL communication control (RCLC)

7.2.1 General

RCLC provides the following functions.

– Control link status of communication with neighboring node using RHE frame;
– Control node status using class 2 RCL frames;
– Send and receive RCL frames.

7.2.2 Primitive definitions

7.2.2.1 Primitive definitions between RCLC and DLS-user

Table 15 summarizes all primitives exchanged between the RCLC and the DLS-user.

Table 15 – The primitives and parameters for DLS-user interface

Primitive name	Source	Associated parameters	Description
DL-RCL.req	DLS-user	(in S_add, PortNum Frame_pri, RCL_type, DLSDU)	Transmit a RCL frame from the DLS-user. The RCL frame is carried to TRC with the class parameter.
DL-RCL.cnf	RCLC	(out Status)	Respond to complete DLSDU transmission and report the status to DLS-user.
DL-RCL.ind	RCLC	(out S_add, PortNum Frame_pri, RCL_type, DLSDU)	Carries forward a received RCL frame to the DLS-user.

The parameters used with the primitives exchange between the RCLC and the DLS-user are described in Table 16.

Table 16 – Parameters used with primitives exchanged between RCLC and DLS-user

Parameter name	Description
S_add	The S_add parameter specifies the DL-address of the publisher.
PortNum	This parameter specifies the transmit/receive port of the publisher.
Frame_pri	This parameter specifies the priority with VLAN of the ring network.
RCL_type	This parameter specifies the RCL type of the RCL frame.
DLSDU	This parameter specifies the information of the RCL frame.
Status	This parameter allows DLMS-user to determine whether the requested DLS was provided successfully, or failed due to a particular reason. The value conveyed in this parameter is as follows: “OK – successfully completed”; “Failure – terminated before completion”

7.2.2.2 Primitive definitions between RCLC and TRC

Table 17 summarizes all primitives exchanged between the RCLC and the TRC.

Table 17 – The primitives and parameters for TRC interface

Primitive name	Source	Associated parameters	Description
RCLC-Tx.req	RCLC	D_add, S_add, Tx_port, Frame_pri, RCL_type, DLSDU	Transmit a RCL frame to TRC with associated parameters. The RCLC attaches the Tx_port parameter (port A, B or both).
RCLC-Rx.ind	TRC	D_add, S_add, Rx_port, Frame_pri, RCL_type, DLSDU	Carries forward a received RCL frame from the TRC.
RCLC-SetPT.req	RCLC	BLKport, Blocking	Requests to change the blocking status of a port to the TRC.

The parameters used with the primitives exchange between the RCLC and the TRC are described in Table 18.

Table 18 – Parameters used with primitives exchanged between RCLC and TRC

Parameter name	Description
D_add	The D_add parameter specifies the DL-address of the subscriber.
S_add	The S_add parameter specifies the DL-address of the publisher.
Tx_port	This parameter specifies the transmit port of the publisher.
Rx_port	This parameter specifies the receive port of the publisher.
Frame_pri	This parameter specifies the priority with VLAN of the ring network.
RCL_type	This parameter specifies the RCL type of the RCL frame.
DLSDU	This parameter specifies the information of the RCL frame.
BLKport	This parameter specifies the blocking port in itself.
Blocking	This parameter specifies the blocking status in itself.

7.2.2.3 Primitive definitions between RCLC and DLM

Table 19 summarizes all primitives exchanged between the RCLC and the DLM.

Table 19 – The primitives and parameters for DLM interface

Primitive name	Source	Associated parameters	Description
RCLC-Reset.ind	DLM	(none)	Indicates a reset of the RCLC from the DLM
RCL_STOP.ind	RCLC	RCL_Indtype	Indicates a stop of the RCL frame to DLM based on the ring control state machine.
RCL_START.ind	RCLC	RCL_Indtype, RCL_IndDA, RCL_IndPri, RCL_IndPort	Indicates a start of the RCL frame to DLM based on the ring control state machine.
Node_ST.ind	RCLC	Node_ST, PortA_ST, PortB_ST	Indicates a change of the node or port status to DLM based on the ring control state machine
RCLC-Event.ind	RCLC	DLM_event identifier, Additional_information	This service is used to inform the DLM about certain events or errors in the DLL

The parameters used with the primitives exchange between the RCLC and the DLM are described in Table 20.

Table 20 – Parameters used with primitives exchanged between RCLC and DLM

Parameter name	Description
RCL_Indtype	This parameter specifies the RCL type of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndDA	This parameter specifies the DL-address of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndPri	This parameter specifies the priority with VLAN of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndPort	This parameter specifies the transmission port of the requested RCL frames for the DLM due to the change of the node status.
Node_ST	This parameter contains the node status (ISL/EGA/EGB/ITM) for the DLM due to the change of the node status.
PortA_ST	This parameter contains the status of port A on own node for the DLM due to the change of the node status.
PortB_ST	This parameter contains the status of port B on own node for the DLM due to the change of the node status.
DLM_event identifier	This parameter specifies the primitive or composite event within the DLE whose occurrence is being announced.
Additional_information	This optional parameter provides event-specific additional information

7.2.3 RCLC state machine

7.2.3.1 RHE state machine of port A and B (RHE_SM-A/RHE_SM-B)

The RHE state machine is used to confirm the link status of communication with the neighboring node. There are two state machines on a node, one for port A (RHE_SM-A) and one for port B (RHE_SM-B). The state is updated when RHE is received and the link state between the node and the neighboring node is confirmed. These state machines have the following states.

– No_Neighborhood: The node does not connect to the neighboring node nor does it receive the RHE frame from the neighboring node.

– Wait_Linkup: The node waits to link up with the neighboring node. Although this state is “link up” status in ISO/IEC 8802-3, the node should not communicate by any frames except the RHE frame to the neighboring node.

– Port_Linkup: The node can communicate using any ISO/IEC 8802-3 frame to the neighboring node.

7.2.3.1.1 State machine RHE_SM-A

Figure 10 shows the state diagram of RHE_SM-A, and Table 21 is state table of port A.

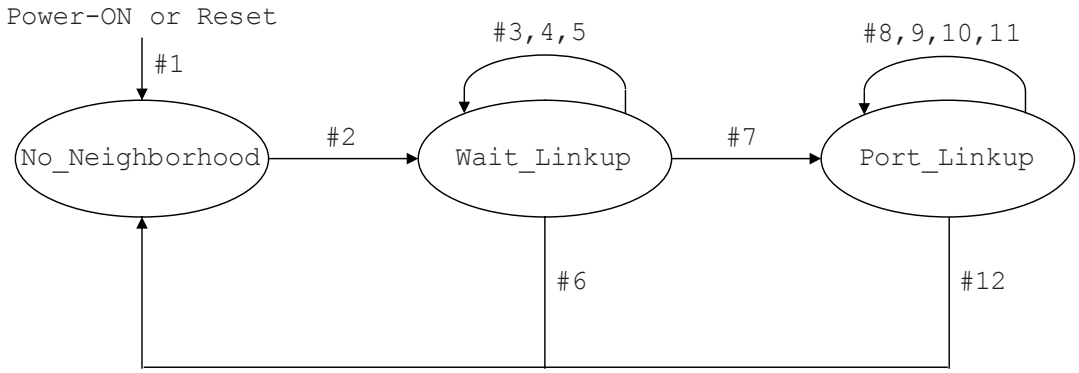


Figure 10 – State transition diagram of RHE_SM-A

Table 21 – Transitions of RHE_SM-A at RCL communication

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset => V(PA_STATUS) = NNB V(PA_NBST_LKST) = NNB V(PA_NBST_NDST) = ISL V(PA_NBST_RCLADD) = 0 C(PA_RHE_LKUP) = 0 V(PA_RHE_RxSeq) = 0 C(PA_RHE_RxCK) = 0 START_TIMER(T(PA_RHE_TIME), V(PA_RHE_CYCLE))	No_Neighborhood
2	No_Neighborhood	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A => V(PA_STATUS) = WLU C(PA_RHE_LKUP) = 1 C(PA_RHE_RxCK) = P(RHE_RxCK_NUM) V(PA_NBST_RCLADD) = S_add V(PA_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
3	Wait_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A && (V(PA_NBST_RCLADD) != S_add || (V(PA_RHE_RxSeq)+1) != RHE_Seq) => C(PA_RHE_LKUP) = 1 C(PA_RHE_RxCK) = P(RHE_RxCK_NUM) V(PA_NBST_RCLADD) = S_add V(PA_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
4	Wait_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A && V(PA_NBST_RCLADD) == S_add && V(PA_RHE_RxSeq)+1 == RHE_Seq => C(PA_RHE_LKUP)++ C(PA_RHE_RxCK) = P(RHE_RxCK_NUM) V(PA_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
5	Wait_Linkup	EXPIRED_TIMER(T(PA_RHE_TIME)) == “True” => C(PA_RHE_RxCK)--	Wait_Linkup

#	Current state	Event /condition => actions	Next state
6	Wait_Linkup	/C(PA_RHE_RxCK) == 0 => V(PA_STATUS) = NNB C(PA_RHE_LKUP) = 0 V(PA_NBST_RCLADD) = 0 V(PA_RHE_RxSeq) = 0 V(PA_NBST_LKST) = NNB V(PA_NBST_NDST) = ISL	No_Neighborhood
7	Wait_Linkup	/C(PA_RHE_LKUP) == P(RHE_LKUP_NUM) => V(PA_STATUS) = PLU C(PA_Rx-RHE_NotMatch) = 0 V(PA_NBST_LKST) = UPDATE_LKST(DLSDU) V(PA_NBST_NDST) = UPDATE_NDST(DLSDU)	Port_Linkup
8	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A && V(PA_NBST_RCLADD) != S_add => C(PA_Rx-RHE_NotMatch)++	Port_Linkup
9	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A && V(PA_NBST_RCLADD) != S_add && C(PA_Rx-RHE_NotMatch) >= 1 => C(PA_Rx-RHE_NotMatch)++ C(PA_RHE_RxCK) = P(RHE_RxCK_NUM) V(PA_NBST_RCLADD) = S_add V(PA_NBST_LKST) = UPDATE_LKST(DLSDU) V(PA_NBST_NDST) = UPDATE_NDST(DLSDU) V(PA_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Port_Linkup
10	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == A && V(PA_NBST_RCLADD) == S_add => C(PA_Rx-RHE_NotMatch) = 0 C(PA_RHE_RxCK) = P(RHE_RxCK_NUM) V(PA_NBST_LKST) = UPDATE_LKST(DLSDU) V(PA_NBST_NDST) = UPDATE_NDST(DLSDU) V(PA_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Port_Linkup

#	Current state	Event /condition => actions	Next state
11	Port_Linkup	EXPIRED_TIMER(T(PA_RHEcycle)) == “True” => C(PA_RHE_RxCK)--	Port_Linkup
12	Port_Linkup	/C(PA_RHE_RxCK) == 0 => V(PA_STATUS) = NNB C(PA_RHE_LKUP) = 0 V(PA_NBST_RCLADD) = 0 V(PA_RHE_RxSeq) = 0 V(PA_NBST_LKST) = NNB V(PA_NBST_NDST) = ISL	No_Neighborhood

7.2.3.1.2 State machine RHE_SM-B

Figure 11 shows the state diagram of RHE_SM-B, and Table 22 is state table of port B.

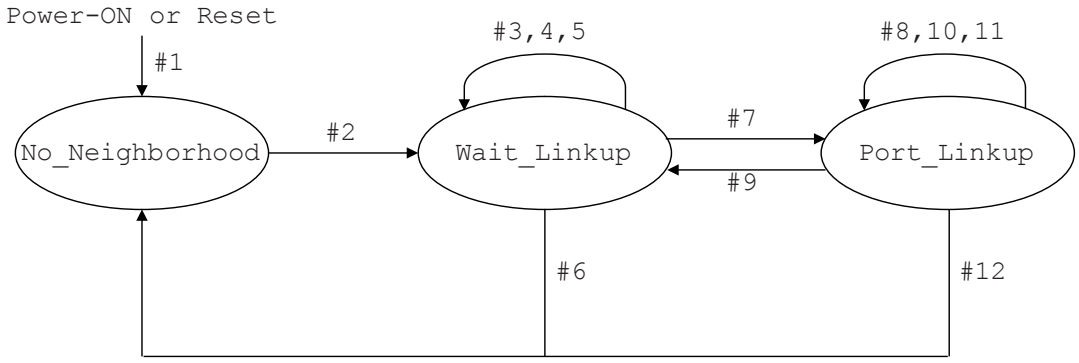


Figure 11 – State transition diagram of RHE_SM-B

Table 22 – Transitions of RHE_SM-B at RCL communication

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset => V(PB_STATUS) = NNB V(PB_NBST_LKST) = NNB V(PB_NBST_NDST) = ISL V(PB_NBST_RCLADD) = 0 C(PB_RHE_LKUP) = 0 V(PB_RHE_RxSeq) = 0 C(PB_RHE_RxCK) = 0 START_TIMER(T(PB_RHE_TIME), V(PB_RHE_CYCLE))	No_Neighborhood
2	No_Neighborhood	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B => V(PB_STATUS) = WLU C(PB_RHE_LKUP) = 1 C(PB_RHE_RxCK) = P(RHE_RxCK_NUM) V(PB_NBST_RCLADD) = S_add V(PB_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
3	Wait_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B && (V(PB_NBST_RCLADD) != S_add || V(PB_RHE_RxSeq)+1 != RHE_Seq) => C(PB_RHE_LKUP) = 1 C(PB_RHE_RxCK) = P(RHE_RxCK_NUM) V(PB_NBST_RCLADD) = S_add V(PB_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
4	Wait_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B && V(PB_NBST_RCLADD) == S_add && V(PB_RHE_RxSeq)+1 == RHE_Seq => C(PB_RHE_LKUP)++ C(PB_RHE_RxCK) = P(RHE_RxCK_NUM) V(PB_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
5	Wait_Linkup	EXPIRED_TIMER(T(PB_RHE_TIME)) == “True” => C(PB_RHE_RxCK)--	Wait_Linkup

#	Current state	Event /condition => actions	Next state
6	Wait_Linkup	/C(PB_RHE_RxCK) == 0 => V(PB_STATUS) = NNB C(PB_RHE_LKUP) = 0 V(PB_NBST_RCLADD) = 0 V(PB_RHE_RxSeq) = 0 V(PA_NBST_LKST) = NNB V(PA_NBST_NDST) = ISL	No_Neighborhood
7	Wait_Linkup	/C(PB_RHE_LKUP) == P(RHE_LKUP_NUM) => V(PB_STATUS) = PLU C(PB_Rx-RHE_NotMatch) = 0 V(PB_NBST_LKST) = UPDATE_LKST(DLSDU) V(PB_NBST_NDST) = UPDATE_NDST(DLSDU)	Port_Linkup
8	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B && V(PB_NBST_RCLADD) != S_add => C(PB_Rx-RHE_NotMatch)++	Port_Linkup
9	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B && V(PA_NBST_RCLADD) != S_add && C(PA_Rx-RHE_NotMatch) >= 1 => C(PB_Rx-RHE_NotMatch)++ C(PB_RHE_LKUP) = 1 C(PB_RHE_RxCK) = P(RHE_RxCK_NUM) V(PB_NBST_RCLADD) = S_add V(PB_NBST_LKST) = UPDATE_LKST(DLSDU) V(PB_NBST_NDST) = UPDATE_NDST(DLSDU) V(PB_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Wait_Linkup
10	Port_Linkup	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU} /RCL_type == RHE && Rx_port == B && V(NBST_RCLADD) == S_add => C(PB_Rx-RHE_NotMatch) = 0 C(PB_RHE_RxCK) = P(RHE_RxCK_NUM) V(PB_NBST_LKST) = UPDATE_LKST(DLSDU) V(PB_NBST_NDST) = UPDATE_NDST(DLSDU) V(PB_RHE_RxSeq) = UPDATE_RHESeq(DLSDU)	Port_Linkup

#	Current state	Event /condition => actions	Next state
11	Port_Linkup	EXPIRED_TIMER(T(PB_RHEcycle)) == “True” => C(PB_RHE_RxCK)--	Port_Linkup
12	Port_Linkup	/C(PB_RHE_RxCK) == 0 => V(PB_STATUS) = NNB C(PB_RHE_LKUP) = 0 V(PB_NBST_RCLADD) = 0 V(PB_RHE_RxSeq) = 0 V(PA_NBST_LKST) = NNB V(PA_NBST_NDST) = ISL	No_Neighborhood

7.2.3.2 RCL node status state machine (RCLNode_SM)

The RCLNode_SM selects an Edge-A node and an Edge-B node in the network. All of the nodes in Type 25 network use this state machine to decide the status of their own node.

The node controls the link status of ports by RCL_SM-A and RCL_SM-B. The RCLNode_SM uses these link status and informs to other nodes in the network by RCL frames. All nodes in the network select an Edge-A node and an Edge-B node using this state machine. This state machine has the following states.

– Node_ISL: Isolated node state, the port A and B are blocked.
– Edge-A_PLD: Edge-A physical link down state. At the state, the node is the Edge-A node, the port A is link up, and the port B is blocked due to physical factor (e.g. not connected to the cable).
– Edge-A_LLD: Edge-B logical link down state. At the state, the node is the Edge-A node, the port A is link up, the port B is blocked, and the ring network is normal.
– Edge-B_PLD: Edge-B physical link down state.
– Edge-B_LLD: Edge-B logical link down state.
– Node_ITM: Intermediate node state. The port A and B are link up.

Figure 12 and Table 23 show the state diagram and the state table of RCLNode_SM.

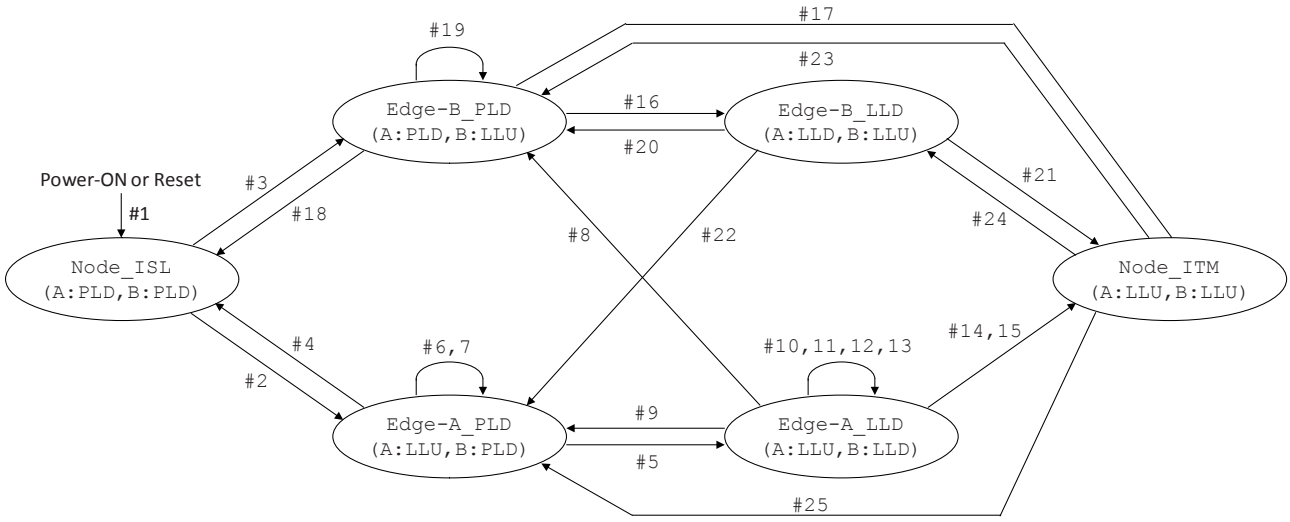


Figure 12 – The state diagram of RCLNode_SM

Table 23 – Transitions of RCLNode_SM at RCL communication

#	Current state	Event /condition => actions	Next state
1	Any states	Power-ON or Reset => Node_ST = ISL PortA_ST = PLD PortB_ST = PLD RCL_Indtype = LCN RCL_STOP.ind{RCL_Indtype} BLKport = A Blocking = True RCLC-SetPT.req{BLKport,Blocking} BLKport = B RCLC-SetPT.req{BLKport,Blocking} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ISL
2	Node_ISL	/V(PA_STATUS) == PLU => Node_ST = EGA PortA_ST = LLU BLKport = A Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_PLD
3	Node_ISL	/V(PB_STATUS) == PLU V(PB_NBST_NDST) != ISL => Node_ST = EGB PortB_ST = LLU BLKport = B Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_PLD
4	Edge-A_PLD	/V(PA_STATUS) == (NNB || WLU) => RCL_Indtype = LCN RCL_STOP.ind{RCL_Indtype} Node_ST = ISL PortA_ST = PLD BLKport = A Blocking = True RCLC-SetPT.req{BLKport,Blocking} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ISL

#	Current state	Event /condition => actions	Next state
5	Edge-A_PLD	/V(PB_STATUS) == PLU => RCL_Indtype = LCN RCL_STOP.ind{RCL_Indtype} C(Rx_Counter) = 0 RCL_Indtype = LCC RCL_IndDA = Brdcast RCL_IndPri = Normal RCL_IndPort = Both RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort} PortB_ST = LLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_LLD
6	Edge-A_PLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCC => RCL_Indtype = LCA RCL_IndDA = S_add RCL_IndPri = RCL_pri RCL_IndPort = Rx_port RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort}	Edge-A_PLD
7	Edge-A_PLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LNA && D_add == MyS_add => RCL_Indtype = LCN RCL_STOP.ind{RCL_Indtype}	Edge-A_PLD
8	Edge-A_LLD	/V(PA_STATUS) == (NNB || WLU) => RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} Node_ST = EGB PortA_ST = PLD BLKport = A Blocking = True RCLC-SetPT.req{BLKport,Blocking} BLKport = B Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_PLD

#	Current state	Event /condition => actions	Next state
9	Edge-A_LLD	/V(PB_STATUS) == (NNB || WLU) => RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} RCL_Indtype = LCN RCL_IndDA = Brdcast RCL_IndPri = V(LCN_Pri) RCL_IndPort = PortA RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort} PortB_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_PLD
10	Edge-A_LLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCC && RCL_IndPri < RCL_Pri => RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} C(Rx_Counter) = 0 RCL_IndDA = Brdcast RCL_IndPri = Normal RCL_IndPort = Both RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort}	Edge-A_LLD

#	Current state	Event /condition => actions	Next state
11	Edge-A_LLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCC && RCL_IndPri > RCL_Pri => RCL_Indtype = LCA RCL_IndDA = S_add RCL_IndPri = RCL_pri RCL_IndPort = Rx_port RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort} RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} C(Rx_Counter) = 0 RCL_Indtype = LCC RCL_IndDA = Brdcast RCL_IndPri = Normal RCL_IndPort = Both RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort}	Edge-A_LLD
12	Edge-A_LLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCC && RCL_IndPri == RCL_Pri => C(Rx_Counter)++	Edge-A_LLD
13	Edge-A_LLD	/C(Rx_Counter) >= P(LCC_STOP_NUM) => C(Rx_Counter) = 0 RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_LLD

#	Current state	Event /condition => actions	Next state
14	Edge-A_LLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCA && D_add == MyS_add => RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} BLKport = B Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST = ITM PortB_ST = LLU Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ITM
15	Edge-A_LLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCN => RCL_Indtype = LCC RCL_STOP.ind{RCL_Indtype} BLKport = B Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST = ITM PortB_ST = LLU Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ITM
16	Edge-B_PLD	/V(PA_STATUS) == PLU && V(PA_NBST_NDST) == EGA => PortA_ST = LLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_LLD
17	Edge-B_PLD	/V(PA_STATUS) == PLU && V(PA_NBST_NDST) == (EGB || ITM) => BLKport = A Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST = ITM PortA_ST = LLU Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ITM

#	Current state	Event /condition => actions	Next state
18	Edge-B_PLD	/V(PB_STATUS) == (NNB || WLU) => RCL_Indtype = LCN RCL_STOP.ind{RCL_Indtype} BLKport = B Blocking = True RCLC-SetPT.req{BLKport,Blocking} Node_ST = ISL PortB_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ISL
19	Edge-B_PLD	RCLC-Rx.ind{D_add,S_add,Rx_port, RCL_type,RCL_Pri,DLSDU} /RCL_type == LCN => RCL_Indtype = LNA RCL_IndDA = S_add RCL_IndPri = V(LNA_Pri) RCL_IndPort = PortB RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort}	Edge-B_PLD
20	Edge-B_LLD	/V(PA_STATUS) == (NNB || WLU) => PortA_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_PLD
21	Edge-B_LLD	/V(PA_STATUS) == PLU && V(PA_NBST_NDST) == (EGB || ITM) => BLKport = A Blocking = False RCLC-SetPT.req{BLKport,Blocking} Node_ST = ITM PortA_ST = LLU Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Node_ITM

#	Current state	Event /condition => actions	Next state
22	Edge-B_LLD	/V(PB_STATUS) == (NNB || WLU) => BLKport = B Blocking = True RCLC-SetPT.req{BLKport,Blocking} BLKport = A Blocking = False RCLC-SetPT.req{BLKport,Blocking} RCL_Indtype = LCN RCL_IndDA = Brdcast RCL_IndPri = V(LCN_pri) RCL_IndPort = PortA RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri} Node_ST = EGA PortA_ST = LLU PortB_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_PLD
23	Node_ITM	/V(PA_STATUS) == (NNB || WLU) => BLKport = A Blocking = True RCLC-SetPT.req{BLKport,Blocking} Node_ST = EGB PortA_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_PLD
24	Node_ITM	/V(PA_STATUS) == PLU && V(PA_NBST_NDST) == EGA => BLKport = A Blocking = True RCLC-SetPT.req{BLKport,Blocking} Node_ST = EGB PortB_ST = LLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-B_LLD

#	Current state	Event /condition => actions	Next state
25	Node_ITM	/V(PB_STATUS) == (NNB || WLU) => BLKport = B Blocking = True RCLC-SetPT.req{BLKport,Blocking} RCL_Indtype = LCN RCL_IndDA = Brdcast RCL_IndPri = V(LCN_pri) RCL_IndPort = PortA RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort} Node_ST = EGA PortB_ST = PLD Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Edge-A_PLD

7.2.3.3 RCL Transmit Receive state machine (RCLTR_SM)

The RCLTR_SM performs transit functions between the DLS-user and the TRC.

Figure 13 shows the state diagram of RCLT_SM, and Table 24 is state table of RCLTR_SM.

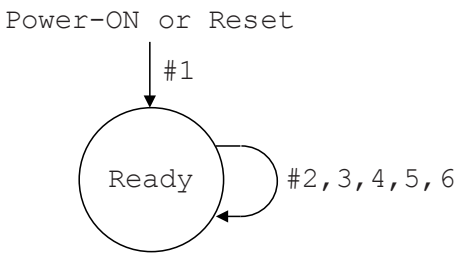


Figure 13 – The state diagram of RCLTR_SM

Table 24 – Transitions of RCLTR_SM at RCL communication

#	Current state	Event /condition => actions	Next state
1	Any state	Power-ON or Reset =>	Ready
2	Ready	DL-RCL.req{S_add,PortNum,Frame_pri, RCL_type,DLSDU} /RCL_type == RHE && CHK_TRCST() == “True” => Status = “Success” D_add = Class1_address Tx_port = PortNum RCLC-Tx.req{D_add,S_add,Tx_port, Frame_pri,RCL_type,DLSDU} DL-RCL.cnf{Status}	Ready
3	Ready	DL-RCL.req{S_add,PortNum,Frame_pri, RCL_type,DLSDU} /RCL_type == RHE && CHK_TRCST() != “True” => Status = “Failure” DL-RCL.cnf{Status}	Ready
4	Ready	DL-RCL.req{S_add,PortNum,Frame_pri, RCL_type,DLSDU} /RCL_type == (LCC || LCA || LCN || LNA || SCR) && CHK_TRCST() == “True” => Status = “Success” D_add = Class2_address Tx_port = PortNum RCLC-Tx.req{D_add,S_add,Tx_port, Frame_pri,RCL_type,DLSDU} DL-RCL.cnf{Status}	Ready
5	Ready	DL-RCL.req{S_add,PortNum,Frame_pri, RCL_type,DLSDU} /RCL_type == (LCC || LCA || LCN || LNA || SCR) && CHK_TRCST() != “True” => Status = “Failure” DL-RCL.cnf{Status}	Ready
6	Ready	RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_Pri,RCL_type,DLSDU} => PortNum = Rx_port DL-RCL.ind{S_add,PortNum,Frame_pri, RCL_type,DLSDU}	Ready

7.2.4 Function of RCLC

All the functions used by the RCLC are summarized in Table 25.

Table 25 – RCLC function table

Function name	Input	Output	Description and operation
UPDATE_LKST	DLSDU	Link Status (NNB/WLU/PLU)	Return link status of the neighboring node. The link status is included in the received DLSDU.
UPDATE_NDST	DLSDU	Node Status (EGA/EGB/ITM)	Return node status of the neighboring node. The node status included in the received DLSDU.
UPDATE_RHESeq	DLSDU	0x00000000 ~ 0xFFFFFFFF	Return 4 bytes of sequence number contained in DLSDU.
START_TIMER	TIM_ID, Val	(<none>)	Timer TIM_ID is set by value of Val and activated.
EXPIRED_TIMER	TIM_ID	True/False	When the requested timer TIM_ID has expired, “True” is returned, otherwise False is returned
CHK_TRCST	(<none>)	True/False	Return status of TRC. If TRC is able to send the frame due to absence of other frames in transmit buffer, it returns “True,” otherwise returns “False.”

7.3 Real-time communication control (RTC)

7.3.1 General

The RTC provides to send and receive the cyclic, control, and information communication frames.

7.3.2 Primitive definitions

7.3.2.1 Primitive definitions between RTC and DLS-user

Table 26 summarizes all primitives exchanged between the RTC and the DLS-user.

Table 26 – The primitives and parameters for DLS-user interface

Primitive name	Source	Associated parameters		Description
DL-RTC.req	DLS-user	(in	D_add, S_add, Frame_pri, DLSDU)	Transmit a RT frame from the DLS-user. The RT frame is carried to TRC with associated parameters.
DL-RTC.cnf	RTC	(out	Status)	Respond to complete DLSDU transmission and report the status to DLS-user.
DL-RTC.ind	RTC	(out	D_add, S_add, Frame_pri, DLSDU)	Carries forward a received RT frame to the DLS-user.

The parameters used with the primitives exchange between the RTC and the DLS-user are described in Table 27.

Table 27 – Parameters used with primitives exchanged between RTC and DLS-user

Parameter name	Description
D_add	The D_add parameter specifies the DL-address of the subscriber.
S_add	The S_add parameter specifies the DL-address of the publisher.
Frame_pri	This parameter specifies the priority with VLAN of the ring network.
DLSDU	This parameter specifies the information of the RT frame.
Status	This parameter allows DLMS-user to determine whether the requested DLS was provided successfully, or failed due to a particular reason. The value conveyed in this parameter is as follows: ”OK – successfully completed”; “Failure – terminated before completion”

7.3.2.2 Primitive definitions between RTC and TRC

Table 28 summarizes all primitives exchanged between the RTC and the TRC.

Table 28 – The primitives and parameters for TRC interface

Primitive name	Source	Associated parameters	Description
RTC-Tx.req	RTC	D_add, S_add, Tx_port Frame_pri, DLSDU	Transmit a RT frame to TRC with associated parameters. The RTC attaches the Tx_port “Both” parameter.
RTC-Rx.ind	TRC	D_add, S_add, Rx_port, Frame_pri, DLSDU	Carries forward a received RT frame from the TRC.

The parameters used with the primitives exchange between the RTC and the TRC are described inTable 29.

Table 29 – Parameters used with primitives exchanged between RTC and TRC

Parameter name	Description
D_add	The D_add parameter specifies the DL-address of the subscriber.
S_add	The S_add parameter specifies the DL-address of the publisher.
Tx_port	This parameter specifies the transmit port of the publisher.
Rx_port	This parameter specifies the receive port of the publisher.
Frame_pri	This parameter specifies the priority with VLAN of the ring network.
DLSDU	This parameter specifies the information of the RT frame.

7.3.2.3 Primitive definitions between RTC and DLM

Table 30 summarizes all primitives exchanged between the RTC and the DLM.

Table 30 – The primitives and parameters for DLM interface

Primitive name	Source	Associated parameters	Description
RTC-Reset.ind	DLM	(none)	Indicates a reset of the RTC from the DLS-user
RTC-Event.ind	RTC	DLM_event identifier, Additional_information	This service is used to inform the DLM about certain events or errors in the DLL

The parameters used with the primitives exchange between the RTC and the DLM are described in Table 31.

Table 31 – Parameters used with primitives exchanged between RTC and DLM

Parameter name	Description
DLM_event identifier	This parameter specifies the primitive or composite event within the DLE whose occurrence is being announced.
Additional_information	This optional parameter provides event-specific additional information.

7.3.3 RTC state machine

7.3.3.1 RT Transmit Receive state machine (RTTR_SM)

The RTTR_SM perform transit functions between the DLS-user and the TRC.

Figure 14 shows the state diagram of RTTR_SM, and Table 32 is state table of RTTR_SM.

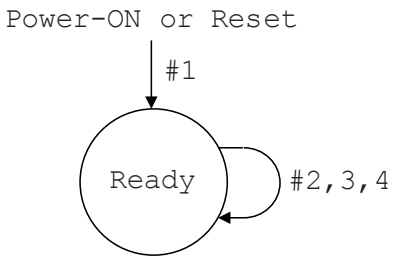


Figure 14 – The state diagram of RTTR_SM

Table 32 – Transitions of RTTR_SM at RT communication

#	Current state	Event /condition => actions	Next state
1	Any state	Power-ON or Reset =>	Ready
2	Ready	DL-RTC.req{D_add,S_add,Frame_pri,DLSDU} /CHK_TRCST() == “True” => Status = “Success” Tx_port = Both RTC-Tx.req{D_add,S_add,Tx_port, Frame_pri,RCL_type,DLSDU} DL-RTC.cnf{Status}	Ready
3	Ready	DL-RTC.req{D_add,S_add,Frame_pri,DLSDU} /CHK_TRCST() != “True” => Status = “Failure” DL-RTC.cnf{Status}	Ready
4	Ready	RTC-Rx.ind{D_add,S_add,Rx_port, Frame_Pri,DLSDU} => DL-RTC.ind{D_add,S_add,Frame_pri,DLSDU}	Ready

7.3.4 Function of RTC

All the functions used by the RTC are summarized in Table 33.

Table 33 – RTC function table

Function name	Input	Output	Description and operation
CHK_TRCST	(<none>)	True/False	Returns status of TRC. If TRC is able to send the frame due to without other frames in transmit buffer, it returns “True,” otherwise returns “False.”

7.4 Transmit/Receive control (TRC)

7.4.1 General

The TRC provides the following functions:

– Send and receive all the frames in Type 25 network;
– Control the port status at the instruction of RCLC and inform the port status to the RCLC;
– Transit the received frames from a port to another port.

7.4.2 Primitive definitions

7.4.2.1 Primitive definitions between TRC and DLM

Table 34 summarizes all primitives exchanged between the TRC and the DLM.

Table 34 – The primitives and parameters for DLM interface

Primitive name	Source	Associated parameters	Description
TRC-Reset.ind	DLM	(none)	Indicates a reset of the TRC from the DLS-user
TRC-Event.ind	TRC	DLM_event identifier, Additional_information	This service is used to inform the DLM about certain events or errors in the DLL

The parameters used with the primitives exchange between the TRC and the DLM are described in Table 35.

Table 35 – Parameters used with primitives exchanged between TRC and DLM

Parameter name	Description
DLM_event identifier	This parameter specifies the primitive or composite event within the DLE whose occurrence is being announced.
Additional_information	This optional parameter provides event-specific additional information.

7.4.3 TRC state machine

7.4.3.1 TRC state machine (TRC_SM)

The TRC_SM controls whether to send or to receive the RCL and RT frames according to the basis of blocking status determined by RCLC. The TRC_SM indicates the port status to the RCLC.

Figure 15 shows the state diagram of TRC_SM, and Table 36 is state table of TRC_SM.

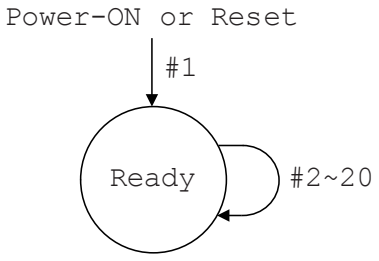


Figure 15 – The state diagram of TRC_SM

Table 36 – Transitions of TRC_SM

#	Current state	Event /condition => actions	Next state
1	Any state	Power-ON or Reset => V(PA_Blocking) = True V(PB_Blocking) = True	Ready
2	Ready	RCLC-SetPT.req{BLKport,Blocking} /BLKport == A => V(PA_Blocking) = Blocking	Ready
3	Ready	RCLC-SetPT.req{BLKport,Blocking} /BLKport == B => V(PB_Blocking) = Blocking	Ready
4	Ready	Phs-A_Data.ind{D_add,S_add, Frame_pri,DLSDU} /D_add == Class1frame => Rx_port = A RCL_type = GET_RCLTYPE(DLSDU) RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU}	Ready
5	Ready	Phs-B_Data.ind{D_add,S_add, Frame_pri,DLSDU} /D_add == Class1frame => Rx_port = B RCL_type = GET_RCLTYPE(DLSDU) RCLC-Rx.ind{D_add,S_add,Rx_port, Frame_pri,RCL_type,DLSDU}	Ready

6	Ready	Phs-A_Data.ind{D_add,S_add,
		Frame_pri,DLSDU}
		/D_add == Class2frame &&
		!(V(PA_Blocking) == True &&
		V(PB_Blocking) == True)
		=>
		Rx_port = A
		RCL_type = GET_RCLTYPE(DLSDU)
		RCLC-Rx.ind{D_add,S_add,Rx_port,
		Frame_pri,RCL_type,DLSDU}
		IF(V(PA_Blocking) == True ||
		V(PB_Blocking) == False) THEN
		Put_Queue-B(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
			Ready

#	Current state	Event /condition => actions	Next state

7	Ready	Phs-B_Data.ind{D_add,S_add,
		Frame_pri,DLSDU}
		/D_add == Class2frame &&
		!(V(PA_Blocking) == True &&
		V(PB_Blocking) == True)
		=>
		Rx_port = B
		RCL_type = GET_RCLTYPE(DLSDU)
		RCLC-Rx.ind{D_add,S_add,Rx_port,
		Frame_pri,RCL_type,DLSDU}
		IF(V(PA_Blocking) == True ||
		V(PB_Blocking) == False) THEN
		Put_Queue-A(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
			Ready

8	Ready	Phs-A_Data.ind{D_add,S_add,
		Frame_pri,DLSDU}
		/D_add != Class1frame &&
		D_add != Class2frame &&
		V(PA_Blocking) == False
		=>
		Rx_port = A
		RLC-Rx.ind{D_add,S_add,Rx_port,
		Frame_pri,DLSDU}
		IF(V(PB_Blocking) == False) THEN
		Put_Queue-B(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
			Ready

9	Ready	Phs-B_Data.ind{D_add,S_add,
		Frame_pri,DLSDU}
		/D_add != Class1frame &&
		D_add != Class2frame &&
		V(PB_Blocking) == False
		=>
		Rx_port = B
		RLC-Rx.ind{D_add,S_add,Rx_port,
		Frame_pri,DLSDU}
		IF(V(PA_Blocking) == False) THEN
		Put_Queue-A(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
			Ready

#	Current state	Event /condition => actions	Next state

10	Ready	RCLC-Tx.req{D_add,S_add,Tx_port,
		Frame_pri,RCL_type,DLSDU}
		/D_add == Class1frame
		=>
		IF(Tx_port == A) THEN
		Put_Queue-A(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
		IF(Tx_port == B) THEN
		PUT_Queue-B(D_add,S_add,Frame_pri,
		DLSDU)
		ENDIF
			Ready

11	Ready	RCLC-Tx.req{D_add,S_add,Tx_port,
		Frame_pri,RCL_type,DLSDU}
		/D_add == Class2frame &&
		!(V(PA_Blocking) == True &&
		V(PB_Blocking) == True)
		=>
		IF(Tx_port == Both || Tx_port == A) THEN
		Put_Queue-A(D_add,S_add,Frame_pri,
		DLSDU)
		IF(Tx_port == Both || Tx_port == B) THEN
		Put_Queue-B(D_add,S_add,Frame_pri,
		DLSDU)
			Ready

12	Ready	RTC-Tx.req{D_add,S_add,Tx_port,
		Frame_pri,DLSDU}
		/D_add != Class1frame &&
		D_add != Class2frame &&
		Tx_port == Both
		=>
		IF(V(PA_Blocking) == False) THEN
		Put_Queue-A(D_add,S_add,
		Frame_pri,DLSDU)
		ENDIF
		IF(V(PB_Blocking) == False) THEN
		Put_Queue-B(D_add,S_add,
		Frame_pri,DLSDU)
		ENDIF
			Ready

13	Ready	/Queue-A_Check(VLAN_RCL) == True => Send_Queuedata-A(VLAN_RCL)	Ready
14	Ready	/Queue-B_Check(VLAN_RCL) == True => Send_Queuedata-B(VLAN_RCL)	Ready
15	Ready	/Queue-A_Check(VLAN_CYC) == True && Queue-A_Check(VLAN_RCL) == False => Send_Queuedata-A(VLAN_CYC)	Ready

#	Current state	Event /condition => actions	Next state
16	Ready	/Queue-B_Check(VLAN_CYC) == True && Queue-B_Check(VLAN_RCL) == False => Send_Queuedata-B(VLAN_CYC)	Ready
17	Ready	/Queue-A_Check(VLAN_CTL) == True && Queue-A_Check(VLAN_RCL) == False && Queue-A_Check(VLAN_CYC) == False => Send_Queuedata-A(VLAN_CTL)	Ready
18	Ready	/Queue-B_Check(VLAN_CTL) == True && Queue-B_Check(VLAN_RCL) == False && Queue-B_Check(VLAN_CYC) == False => Send_Queuedata-B(VLAN_CTL)	Ready
19	Ready	/Queue-A_Check(VLAN_INFO) == True && Queue-A_Check(VLAN_RCL) == False && Queue-A_Check(VLAN_CYC) == False && Queue-A_Check(VLAN_CTL) == False => Send_Queuedata-A(VLAN_INFO)	Ready
20	Ready	/Queue-B_Check(VLAN_INFO) == True && Queue-B_Check(VLAN_RCL) == False && Queue-B_Check(VLAN_CYC) == False && Queue-B_Check(VLAN_CTL) == False => Send_Queuedata-B(VLAN_INFO)	Ready

7.4.4 Function of TRC

All the functions used by the TRC are summarized in Table 37.

Table 37 – TRC function table

Function name	Input	Output	Description and operation
GET_RCLTYPE	DLSDU	RCL_type (RHE/LCC/ LCA/LCN/ LNA/SCR)	Returns the RCL_type included in DLSDU.
Put_Queue-A	D_add, S_add, Frame_pri, DLSDU	(<none>)	Queues the input PDU into the port A transmit Queue appropriate to Frame_pri on a FIFO basis.
Put_Queue-B	D_add, S_add, Frame_pri, DLSDU	(<none>)	Queues the input PDU into the port B transmit Queue appropriate to Frame_pri on a FIFO basis.
Queue-A_Check	Frame_pri	True/False	Returns the “True” if the queue of port A specified by Frame_pri is not empty.
Queue-B_Check	Frame_pri	True/False	Returns the “True” if the queue of port B specified by Frame_pri is not empty.
Send_Queudata-A	Frame_pri	(<none>)	Dequeue from the queue of port A specified by Frame_pri on a FIFO basis and send the PDU from the port A. Send_Queudata-A(Frame_pri) is assembled as follows: PortNum = PortA D_add = dequeue_Dadd(PortNum, Frame_Pri) S_add = dequeue_Sadd(PortNum, Frame_Pri) DLSDU = dequeue_data(PortNum, Frame_Pri) Phs-A_Data.req{D_add,S_add, Frame_pri,DLSDU}
Send_Queudata-B	Frame_pri	(<none>)	Dequeue from the queue of port A specified by Frame_pri on a FIFO basis and send the PDU from the port A. Send_Queudata-B(Frame_pri) is assembled as follows: PortNum = PortB D_add = queue_Dadd(PortNum, Frame_Pri) S_add = queue_Sadd(PortNum, Frame_Pri) DLSDU = queue_data(PortNum, Frame_Pri) Phs-B_Data.req{D_add,S_add, Frame_pri,DLSDU}
dequeue_Dadd	PortNum, Frame_Pri	D_add	Dequeues the destination address from the queue specified by PortNum and Frame_Pri on a FIFO basis.
dequeue_Sadd	PortNum, Frame_Pri	S_add	Dequeues the source address from the queue specified by PortNum and Frame_Pri on a FIFO basis.
dequeue_data	PortNum, Frame_Pri	DLSDU	Dequeues the DLSDU from the queue specified by PortNum and Frame_Pri on a FIFO basis.

7.5 DLL management protocol (DLM)

7.5.1 Overview

The interface protocol between the DLM and the DLS-user is described in this subclause.

7.5.2 Primitive definitions

7.5.2.1 Primitive exchanged between DLM and DLS-user

Table 38 summarizes all primitives exchanged between the DLM and the DLS-user.

Table 38 – Primitives exchanged between DLM and DLS-user

Primitive name	Source	Associated parameters		Description
DLM_Reset.req	DLS-user	(<none>)		This request primitive causes DLMS to reset the DLE
DLM_Reset.cnf	DLM	(out	DLM_status)	This indicates the status of the reset
DLM_Set.req	DLS-user	(in	Variable-name, Desired-value)	This service is used to assign new values to the variables of the DLE
DLM_Set.cnf	DLM	(out	DLM_status)	The DLMS-user receives confirmation that the specified variable has been set to the new value
DLM_Get.req	DLS-user	(in	Variable-name)	This service is used to read the value of a DLE variable
DLM_Get.cnf	DLM	(out	DLM_status, Current-value)	This service returns the actual value of the specified variable
DLM_RCL_STOP.ind	DLM	(out	RCL_Indtype)	Indicates a stop of the RCL frame to DLS-user based on the ring control state machine.
DLM_RCL_START.ind	DLM	(out	RCL_Indtype, RCL_IndDA, RCL_IndPri, RCL_IndPort)	Indicates a start of the RCL frame to DLS-user based on the ring control state machine.
DLM_Node_ST.ind	DLM	(out	Node_ST, PortA_ST, PortB_ST)	Indicates a change of the node or port status to DLS-user based on the ring control state machine
DLM_Event.ind	DLM	(out	DLM_event identifier, Additional_information)	This service is used to inform the DLS-user about certain events or errors in the DLL

The parameters used with the primitives exchanged between the DLM and the DLS-user are described in Table 39..

Table 39 – Parameters used with primitives exchanged between DLM and DLS-user

Parameter name	Description
DLM_status	Status of the service execution
Variable-name	DL variable name to be addressed
Desired-value	DL desired value to be set
Current-value	Current DL variable value to be read out
RCL_Indtype	This parameter specifies the RCL type of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndDA	This parameter specifies the DL-address of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndPri	This parameter specifies the priority with VLAN of the requested RCL frames for the DLM due to the change of the node status.
RCL_IndPort	This parameter specifies the transmission port of the requested RCL frames for the DLM due to the change of the node status.
Node_ST	This parameter contains the node status (ISL/EGA/EGB/ITM) for the DLM due to the change of the node status.
PortA_ST	This parameter contains the status of port A on own node for the DLM due to the change of the node status.
PortB_ST	This parameter contains the status of port B on own node for the DLM due to the change of the node status.
DLM_event identifier	Event is being announced with related DL variable and the state changes
Additional_information	Optional parameter to provide event-specific additional information

7.5.3 DLM state machine (DLM_SM)

Figure 16 shows the state diagram of DLM_SM, and Table 40 is state table of DLM_SM.

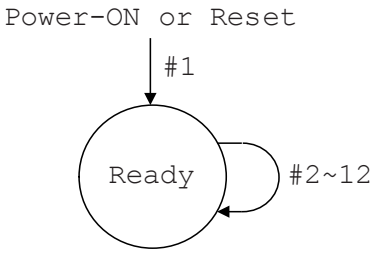


Figure 16 – The state diagram of DLM_SM

Table 40 – Transitions of DLM_SM

#	Current state	Event /condition => actions	Next state
1	Any state	Power-ON or Reset =>	Ready
2	Ready	DLM_Reset.req{} => DLM_Status = “Success” DLM-Reset.cnf{DLM_Status}	Ready
3	Ready	DLM_Set.req{Variable-name,Desired-value} /CHK_Value(Variable-name,Desired-value) == True => SET_Value(Variable-name,Desired-value) DLM_Status = “Success” DLM_Set.cnf{DLM_Status}	Ready
4	Ready	DLM_Set.req{Variable-name,Desired-value} /CHK_Value(Variable-name,Desired-value) == False => DLM_Status = “Failure” DLM_Set.cnf{DLM_Status}	Ready
5	Ready	DLM_Get.req{Variable-name} /CHK_VAR(Variable-name) == True => Current-value = GET_Value(Variable-name) DLM_Status = “Success” DLM_Get.cnf{DLM_Status,Current-value}	Ready
6	Ready	DLM_Get.req{Variable-name} /CHK_VAR(Variable-name) == False => Current-value = NIL DLM_Status = “Failure” DLM_Get.cnf{DLM_Status,Current-value}	Ready
7	Ready	RCL_STOP.ind{RCL_Indtype} / => DLM_RCL_STOP.ind{RCL_Indtype}	Ready
8	Ready	RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort} / => DLM_RCL_START.ind{RCL_Indtype,RCL_IndDA, RCL_IndPri,RCL_IndPort}	Ready
9	Ready	Node_ST.ind{Node_ST,PortA_ST,PortB_ST} / => DLM_Node_ST.ind{Node_ST,PortA_ST,PortB_ST}	Ready

#	Current state	Event /condition => actions	Next state
10	Ready	RCLC-Event.ind{DLM_eventID,Additional_Info} / => DLM_Event.ind{DLM_eventID,Additional_Info}	Ready
11	Ready	RTC-Event.ind{DLM_eventID,Additional_Info} / => DLM_Event.ind{DLM_eventID,Additional_Info}	Ready
12	Ready	TRC-Event.ind{DLM_eventID,Additional_Info} / => DLM_Event.ind{DLM_eventID,Additional_Info}	Ready

7.5.4 Function of DLM

All the functions used by the DLM are summarized in Table 41.

Table 41 – DLM function table

Function name	Input	Output	Description and operation
CHK_Value	Variable-name, Desired-value	True/False	Check if the requested variable with desired value is valid.
CHK_VAR	Variable-name,	True/False	Check if the requested variable is valid.

Bibliography

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEEE 802.1D-2004, *IEEE Standard for Local and metropolitan area networks – Media access control (MAC) Bridges*, available at http://www.ieee.org

IEEE 802.1Q, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks*; available at http://www.ieee.org

ISO 15745-4:2003, *Industrial automation systems and integration – Open systems application integration framework – Part 4: Reference description for Ethernet-based control systems;* available at <http://www.iso.org>

CONTENTS

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 5-25: Application layer service definition – Type 25 elements

1 Scope

1.1 Overview

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 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 standard defines in an abstract way the externally visible service provided by the different Types of the fieldbus Application Layer in terms of

a) an abstract model for defining application resources (objects) capable of being manipulated by users via the use of the FAL service,
b) the primitive actions and events of the service;
c) the parameters associated with each primitive action and event, and the form which they take; and
d) the interrelationship between these actions and events, and their valid sequences.

The purpose of this standard is to define the services provided to

a) the FAL user at the boundary between the user and the Application Layer of the Fieldbus Reference Model, and
b) Systems Management at the boundary between the Application Layer and Systems Management of the Fieldbus Reference Model.

This standard specifies the structure and services of the IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) 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 standard to provide access to the FAL to control certain aspects of its operation.

1.2 Specification

The principal objective of this standard is to specify the characteristics of conceptual application layer services suitable for time-critical communications, and thus supplement the OSI Basic Reference Model in guiding the development of application layer protocols for time-critical communications.

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 services standardized as the various Types of IEC 61158, and the corresponding protocols standardized in subparts of IEC 61158-6.

This specification may be used as the basis for formal Application Programming-Interfaces. Nevertheless, it is not a formal programming interface, and any such interface will need to address implementation issues not covered by this specification, including

a) the sizes and octet ordering of various multi-octet service parameters, and

b) the correlation of paired request and confirm, or indication and response, primitives.

1.3 Conformance

This standard 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 this application layer service definition standard. Instead, conformance is achieved through implementation of conforming application layer protocols that fulfill any given Type of application layer services as defined in this standard.

2 Normative references

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

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

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

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic ReferenceModel: Naming and addressing*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications*

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

IEEE 802.1D-2004, *IEEE Standard for Local and metropolitan area networks – Media accesscontrol (MAC) Bridges,* available at <http://www.ieee.org>

IEEE 802.1Q-2005, *IEEE Standard for Local and metropolitan area networks – Virtual BridgedLocal Area Networks,* available at <http://www.ieee.org>

3 Terms, definitions, symbols and abbreviations

3.1 Referenced 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 terms

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

a) object identifier;

b) type.

3.1.5 IEC 61158 terms

For the purposes of this document, the following terms given in IEC 61158 apply:

a) DLL mapping protocol machine;

b) fieldbus application layer;

c) FAL service protocol machine;

d) protocol data unit.

3.2 Additional terms and definitions for this IEC PAS 62953-5-25

3.2.1
ADP message
a message conveyed by an autonomous decentralized system protocol

3.2.2
alive-message
a 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 bytes

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 high 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
A memory which is allocated to each node in the data field, which each node transmits periodically for the purpose of logical sharing of this memory area

3.2.9
data field
a 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 (i.e. station, device, and server)

3.2.13
information communication
acyclic data communication for low 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)

3.2.16
multicast group
a group of nodes belonging to a data field to determine whether to receive the same data from another 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 (whether an online node or a test node)

3.2.19
primary LAN
a LAN used in normal state in a duplex LAN system

3.2.20
priority control
control the transmission order with VLAN priority to ensure the real-time communication in type S network

3.2.21
RCL communication
control ring network of type S network using RCL frames

3.2.22
Remote control
control the node based remote method invocation

3.2.23
secondary LAN
a 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
ASCII American Standard Code for Information Interchange
ASE Application Service Element
ASN.1 Abstract Syntax Notation 1
CP Communication Profile
CPF Communication Profile Family
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
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
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)
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 used in this document are defined in IEC 61158 Type 25 and IEC 61784-2 CPF 20.

3.4.2 Conventions for class definitions

Class definitions are defined using templates. Each template consists of a list of attributes and services for the class. The general form of the template is shown below:

FAL ASE:			**ASE Name**
CLASS:			**Class Name**
CLASS ID:			**#**
PARENT CLASS:			**Parent Class Name**
ATTRIBUTES:			
1	(o)	Key Attribute:	numeric identifier
2	(o)	Key Attribute:	name
3	(m)	Attribute:	attribute name (values)
4	(m)	Attribute:	attribute name (values)
4.1	(s)	Attribute:	attribute name (values)
4.2	(s)	Attribute:	attribute name (values)
4.3	(s)	Attribute:	attribute name (values)
5	(c)	Constraint:	constraint expression
5.1	(m)	Attribute:	attribute name (values)
5.2	(o)	Attribute:	attribute name (values)
6	(m)	Attribute:	attribute name (values)
6.1	(s)	Attribute:	attribute name (values)
6.2	(s)	Attribute:	attribute name (values)
SERVICES:			
1	(o)	OpsService:	service name
2	(c)	Constraint:	constraint expression
2.1	(o)	OpsService:	service name
3	(m)	MgtService:	service name

a) The "FAL ASE:" entry is the name of the FAL ASE that provides the services for the class being specified.

b) The "CLASS:" entry is the name of the class being specified. All objects defined using this template will be an instance of this class. The class may be specified by this standard, or by a user of this standard.

c) The "CLASS ID:" entry is a number that identifies the class being specified. This number is unique within the FAL ASE that will provide the services for this class. When qualified by the identity of its FAL ASE, it unambiguously identifies the class within the scope of the FAL. The value "NULL" indicates that the class cannot be instantiated. Class IDs between 1 and 255 are reserved by this standard to identify standardized classes. They have been assigned to maintain compatibility with existing national standards. CLASS IDs between 256 and 2048 are allocated for identifying user defined classes.

d) The "PARENT CLASS:" entry is the name of the parent class for the class being specified. All attributes defined for the parent class and inherited by it are inherited for the class being defined, and therefore do not have to be redefined in the template for this class.

NOTE The parent-class "TOP" indicates that the class being defined is an initial class definition. The parent class TOP is used as a starting point from which all other classes are defined. The use of TOP is reserved for classes defined by this standard.

e) The "ATTRIBUTES" label indicate that the following entries are attributes defined for the class.

1) Each of the attribute entries contains a line number in column 1, a mandatory (m) / optional (o) / conditional (c) / selector (s) indicator in column 2, an attribute type label in column 3, a name or a conditional expression in column 4, and optionally a list of enumerated values in column 5. In the column following the list of values, the default value for the attribute may be specified.

2) Objects are normally identified by a numeric identifier or by an object name, or by both. In the class templates, these key attributes are defined under the key attribute.

3) The line number defines the sequence and the level of nesting of the line. Each nesting level is identified by a period. Nesting is used to specify

i) fields of a structured attribute (4.1, 4.2, 4.3),

ii) attributes conditional on a constraint statement (5). Attributes may be mandatory (5.1) or optional (5.2) if the constraint is true. Not all optional attributes require constraint statements as does the attribute defined in (5.2),

iii) the selection fields of a choice type attribute (6.1 and 6.2).

f) The "SERVICES" label indicates that the following entries are services defined for the class.

1) An (m) in column 2 indicates that the service is mandatory for the class, while an (o) indicates that it is optional. A (c) in this column indicates that the service is conditional. When all services defined for a class are defined as optional, at least one has to be selected when an instance of the class is defined.

2) The label "OpsService" designates an operational service (1).

3) The label "MgtService" designates a management service (2).

4) The line number defines the sequence and the level of nesting of the line. Each nesting level is identified by a period. Nesting within the list of services is used to specify services conditional on a constraint statement.

The following clauses each represent additions in the context of their respective IEC document standards.

3.4.3 Conventions for service definitions

3.4.3.1 General

The service model, service primitives, and time-sequence diagrams used are entirely abstract descriptions; they do not represent a specification for implementation.

3.4.3.2 Service parameters

Service primitives are used to represent service user/service provider interactions (ISO/IEC 10731). They convey parameters which indicate information available in the user/provider interaction. In any particular interface, not all parameters need be stated explicitly.

The service specifications of this standard use a tabular format to describe the component parameters of the ASE service primitives. The parameters which apply to each group of service primitives are set out in tables. Each table consists of up to five columns for the

a) parameter name,

b) request primitive, (transmitted from the sender),

c) indication primitive, (transmitted to the receiver),

d) response primitive, (transmitted from the receiver) , and

e) confirm primitive (transmitted to the sender).

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

M parameter is mandatory for the primitive

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

C parameter is conditional upon other parameters or upon the environment of the service user.

– (blank) parameter is never present.

S parameter is a selected item.

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

1) a parameter-specific constraint

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

2) an indication that some note applies to the entry

   (n) indicates that the following note n contains additional information pertaining to the parameter and its use.

3.4.3.2.1 Service procedure

The procedures are defined in terms of:

- the interactions between application entities through the exchange of fieldbus Application Protocol Data Units, and
- the interactions between an application layer service provider and an application layer service user in the same system through the invocation of application layer service primitives.

These procedures are applicable to instances of communication between systems which support time-constrained communications services within the fieldbus application layer.

4 Concept

The basic concept of application layer services follows clause 9 of IEC 61158-1:2014.

The FAL defined herein has two primary deployment models. Type S network has characteristics of simple communication protocols, rapid cyclic communication. Type N network has characteristics of high scalability, expandability, and availability from the small network to the large-scale network. Type N network supports large capacity of the cyclic communication. Both types support a distributed memory model as well as client/server models.

5 Data type ASE

5.1 Overview

The overview of the data type ASE follows IEC 61158-1:2013 clause 10. The template is used to define the data type for the FAL.

5.2 Fixed length types

5.2.1 Numeric types

5.2.1.1 Integer types

5.2.1.1.1 Integer8

FAL ASE:			**Data type ASE**
CLASS:			**Data type**
CLASS ID:			**5**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	Data type numeric identifier	=	2
2	Data type name	=	Integer8
3	Format	=	Fixed length
4.1	Octet length	=	1

This integer type is a two’s complement binary number with a length of one octet.

5.2.1.1.2 Integer16

FAL ASE:			**Data type ASE**
CLASS:			**Data type**
CLASS ID:			**5**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	Data type numeric identifier	=	3
2	Data type name	=	Integer16
3	Format	=	Fixed length
4.1	Octet length	=	2

This integer type is a two’s complement binary number with a length of two octets.

5.2.1.1.3 Integer32

FAL ASE:			**Data type ASE**
CLASS:			**Data type**
CLASS ID:			**5**
PARENT CLASS:			**Top**

ATTRIBUTES:

1	Data type numeric identifier	=	4
2	Data type name	=	Integer32
3	Format	=	Fixed length
4.1	Octet length	=	4

This integer type is a two's complement binary number with a length of four octets.

5.2.1.2 Unsigned types

5.2.1.2.1 Unsigned8

FAL ASE: **Data type ASE**
CLASS: **Data type**
CLASS ID: **5**
PARENT CLASS: **Top**
ATTRIBUTES:

1	Data type numeric identifier	=	5
2	Data type name	=	Unsigned8
3	Format	=	Fixed length
4.1	Octet length	=	1

This type is a binary number with a length of one octet. No sign bit is included. The most significant bit of the most significant octet is always used as the most significant bit of the binary number.

5.2.1.2.2 Unsigned16

FAL ASE: **Data type ASE**
CLASS: **Data type**
CLASS ID: **5**
PARENT CLASS: **Top**
ATTRIBUTES:

1	Data type numeric identifier	=	6
2	Data type name	=	Unsigned16
3	Format	=	Fixed length
4.1	Octet length	=	2

This type is a binary number with a length of two octets. No sign bit is included. The most significant bit of the most significant octet is always used as the most significant bit of the binary number.

5.2.1.2.3 Unsigned32

FAL ASE: **Data type ASE**
CLASS: **Data type**
CLASS ID: **5**
PARENT CLASS: **Top**
ATTRIBUTES:

1	Data type numeric identifier	=	7
2	Data type name	=	Unsigned32

3	Format	=	Fixed length
4.1	Octet length	=	4

This type is a binary number with a length of four octets. No sign bit is included. The most significant bit of the most significant octet is always used as the most significant bit of the binary number.

5.3 String types

5.3.1 OctetString

FAL ASE: **Data type ASE**

CLASS: **Data type**

CLASS ID: **5**

PARENT CLASS: **Top**

ATTRIBUTES:

1	Data type numeric identifier	=	10
2	Data type name	=	OctetString
3	Format	=	string
4.1	Octet length	=	1 to n

This type is with a length of one to n octets. Octet 1 is referred to as the first octet.

6 Communication model specification

6.1 Communication model

6.1.1 General

Two types of communication models are used in FAL type 25: The cyclic communication model (n:n cyclic model) and the acyclic communication model (client server and push model).

6.1.2 Cyclic communication model

The cyclic communication model is shown in Figure 1 and uses an unconfirmed push model performed periodically.

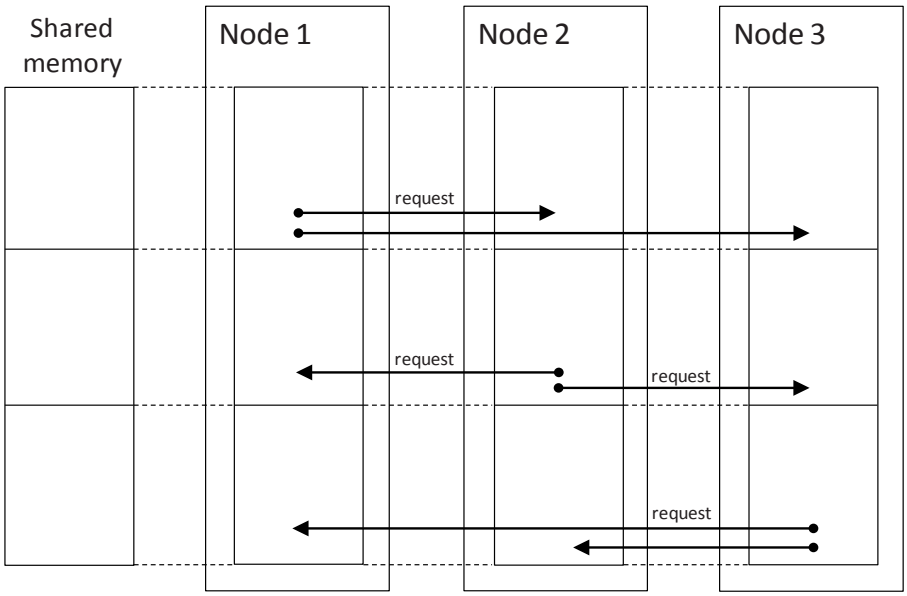


Figure 1 – Cyclic communication model (n:n communication with shared memory)

6.1.3 Acyclic communication model

The acyclic communication model is the client server model shown in Figure 2 and the push model shown in Figure 3.

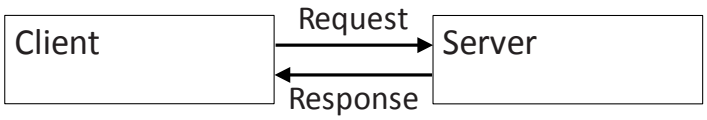


Figure 2 – Acyclic communication model (client server model)

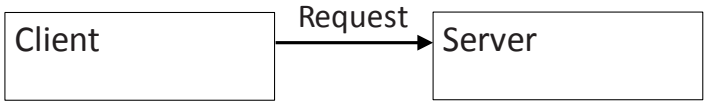


Figure 3 – Acyclic communication model (push model)

6.2 ASE type S

6.2.1 Overview

The structure of the ASE type S for FAL type 25 is shown in Figure 4.

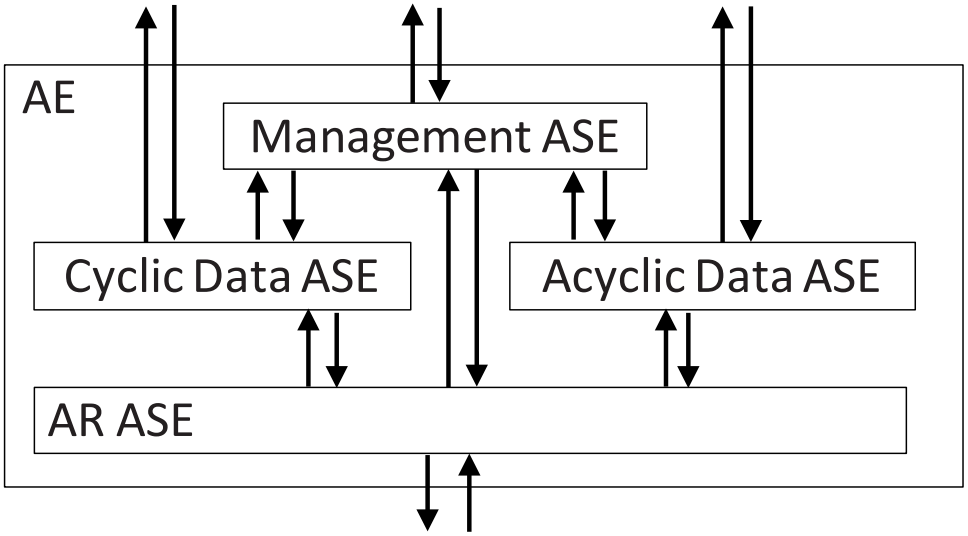


Figure 4 – Structure of ASE type S of FAL type 25

6.2.2 Cyclic data type S

6.2.2.1 Overview

The cyclic data ASE represents a distributed shared memory model which uses the cyclic data transmissions. The cyclic communication is performed to read and write data periodically.

6.2.2.2 Cyclic data class specification

6.2.2.2.1 Overview

This class is the parent class which realizes distributed shared memories.

6.2.2.2.2 Formal model

FAL ASE:			**Cyclic data ASE type S**
CLASS:			**Cyclic data class S**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Attribute:	CYC_ID
2	(m)	Attribute:	Block area
3	(m)	Attribute:	CYC_status
4	(m)	Attribute:	Station address
5	(m)	Attribute:	IP address
SERVICES:			
1	(m)	OpsService:	Put_cyclicdata

2 (m) OpsService: Get_cyclicdata
3 (m) OpsService: Ctl_cyclic

6.2.2.2.3 Attributes

6.2.2.2.3.1 CYC_ID

This parameter represents the identification of the cyclic operation.

6.2.2.2.3.2 Block area

This parameter represents the block area for transmission in the shared memory.

6.2.2.2.3.3 CYC_status

This parameter represents the status of cyclic operation.

6.2.2.2.3.4 Station address

This parameter represents the node (station) number of the node.

6.2.2.2.3.5 IP address

This parameter represents the IP address of the node.

6.2.2.2.4 Service specification

6.2.2.2.4.1 Put_cyclicdata

This service is used to write data into a specified address and specified size. Table 1 shows the parameters for this service.

Table 1 – Put_cyclicdata service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	M (=)
Block_ID	M	
CYCdata	M	
Result (+)		S
Status		M
Additional_info		C
Result (-)		S
Status		M
Additional_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

Block_ID

This parameter specifies the block area in the shared memory to write CYCdata.

CYCdata

This parameter specifies the data to write into the shared memory.

Result (+)

This selection type parameter indicates that the service request succeeded.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

6.2.2.2.4.2 Get_cyclicdata

This service is used to read data into a specified address and specified size. Table 2 shows the parameters for this service.

Table 2 – Get_cyclicdata service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	M (=)
Block_ID	M	
Result (+)		S
CYCdata		M
Status		M
Additioal_info		C
Result (-)		S
Status		M
Additioal_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

Block_ID

This parameter specifies the block area to read from the shared memory .

Result (+)

This selection type parameter indicates that the service request succeeded.

CYCdata

This parameter contains the value of the specified shared memory.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

6.2.2.2.4.3 Ctl_cyclic

This service controls the cyclic communication. It specifies parameters of the cyclic communication (cycle, sending area, etc) and starts (or stops) the cyclic communication. Table 3 shows the parameters for this service.

Table 3 – Ctl_cyclic service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	M (=)
CYCctl	M	
CYC-CTLdata	C	
Result (+)		S
Status		M
Additioal_info		C
Result (-)		S
Status		M
Additioal_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

CYCctl

This parameter is used to start and stop the cyclic communication.

CYC-CTLdata

This parameter contains control information for the cyclic communication. It specifies the cycle, sending area, and the header address.

Result (+)

This selection type parameter indicates that the service request succeeded.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor specific additional information.

6.2.3 Acyclic data ASE type S

6.2.3.1 Overview

Acyclic data ASE type S provides communication between nodes realized by the acyclic communication.

6.2.3.2 Acyclic Data class specification

6.2.3.2.1 Formal model

FAL ASE:			**Acyclic data ASE type S**
CLASS:			**Acyclic data S**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Attribute:	Station address
2	(m)	Attribute:	IP address
SERVICES:			
1	(m)	OpsService:	Send_ctldata
2	(m)	OpsService:	Send_infodata
3	(m)	OpsService	Send_rmtctl

6.2.3.2.2 Attributes

6.2.3.2.2.1 Station address

This parameter represents the node (station) number of the node.

6.2.3.2.3 Service specification

6.2.3.2.3.1 Send_ctldata

This service is used to send the messages by the control communication to other nodes. Table 4 shows the parameters for this service.

Table 4 – Send_ctldata service parameters

Parameter name	req	ind	cnf
Argument	M	M (=)	
D_add	M	M (=)	
S_add	M	M (=)	
CTLdata	M	M (=)	
Result (+)			S
Status			M
Result (-)			S
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CTLdata

The parameter contains the data sent by control communication.

Result (+)

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.2.3.2.3.2 Send_infodata

This service is used to send the messages by the information communication to other nodes. Table 5 shows the parameters for this service.

Table 5 – Send_infodata service parameters

Parameter name	req	ind	cnf
Argument	M	M (=)	
D_add	M	M (=)	
S_add	M	M (=)	
INFOdata	M	M (=)	
Result (+)			S
Status			M
Result (-)			S
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

INFOdata

The parameter contains the data sent by information communication.

Result (+)

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure

6.2.3.2.3.3 Send_rmtctl

This service is used to send the remote control frames to other nodes, and to respond the frames from the receiving nodes. The remote control frames are sent by control communication. Table 6 shows the parameters for this service.

Table 6 – Send_rmtctl service parameters

Parameter name	req	ind	rsp	cnf
Argument	M	M (=)	M	M (=)
D_add	M	M (=)	M	M (=)
S_add	M	M (=)	M	M (=)
CMD	M	M (=)	M	M (=)
CMDdata	C	C (=)		
Result (+)			S	S (=)
RSPdata			C	C (=)
Status			M	M (=)
Result (-)			S	S (=)
Status			M	M (=)

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CMD

This parameter is the command for the remote control operation.

CMDdata

The parameter contains the sent data of remote control command.

Result (+)

RSPdata

This parameter contains the respond data.

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.2.4 Management ASE type S

6.2.4.1 Overview

Management ASE provides the node management function.

6.2.4.2 Management class specification

6.2.4.2.1 Overview

This class provides the node management functions.

FAL ASE: **Management ASE**
CLASS: **Management S**
CLASS ID: **Not used**

PARENT CLASS: **Top**

ATTRIBUTES:

1 (m) Attribute Station address

SERVICES:

1 (m) OpsService: Set_attribute

2 (m) OpsService: Get_attribute

6.2.4.2.2 Attributes

6.2.4.2.2.1 Station address

This parameter represents the node (station) number of the node.

6.2.4.2.3 Service specification

6.2.4.2.3.1 Set_attribute

This service is used to set the attribute value of a class. Table 7 shows the parameters for this service.

Table 7 – Set_attribute service parameters

Parameter name	req	cnf
Argument	M	
Set_List	M	
Attribute	M	
Set_value	M	
Result (+)		S
Status		M
Result (-)		S
Status		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument conveys the service specific parameters of the service request.

Set_List

Specifies the attributes of the target whose values are requested.

Attribute

Specifies the attribute to be a set.

Set_value

Specifies the value to be set to the attribute.

Result (+)

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.2.4.2.3.2 Get_attribute

This service is used to set the attribute value of a class. Table 8 shows the parameters for this service.

Table 8 – Get_attribute service parameters

Parameter name	req	cnf
Argument	M	
Set_List	M	
Result (+)		S
List_of_Attribute values		M
Status		M
Result (-)		S
Status		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument conveys the service specific parameters of the service request.

Set_List

Specifies the attributes of the target whose values are requested.

Result (+)

List_of_AttributeValues

Contains the value of the target attributes.

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.3 ASE type N

6.3.1 Overview type N

The structure of the ASE type N for FAL type 25 is shown in Figure 5.

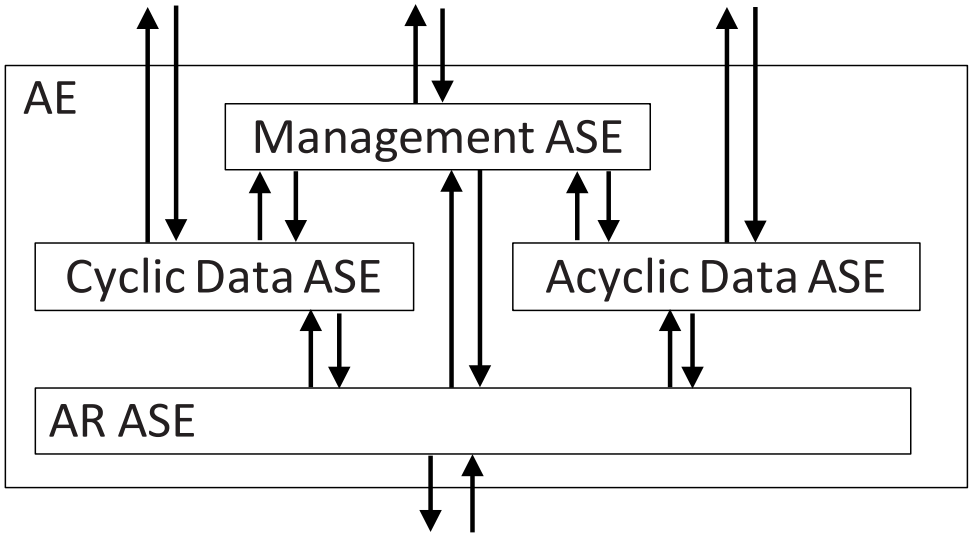


Figure 5 – Structure of ASE type N of FAL type 25

6.3.2 Cyclic data ASE type N

6.3.2.1 Overview

The cyclic data ASE represents a distributed shared memory model which is realized using the cyclic data transmissions. The cyclic communication is performed to read and write data periodically.

6.3.2.2 Cyclic data class specification

6.3.2.2.1 Overview

This class is the parent class which realizes distributed shared memories.

6.3.2.2.2 Formal model

FAL ASE:			**Cyclic data ASE type N**
CLASS:			**Cyclic data class N**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Key Attribute:	Dfn
2	(m)	Key Attribute:	Tmid
3	(m)	Attribute	Pri
4	(m)	Attribute	Memory address
5	(m)	Attribute	Memory size
6	(m)	Attribute	Block number
7	(m)	Attribute	Block count
SERVICES:			
1	(m)	OpsService:	Put_cyclicdata
2	(m)	OpsService:	Get_cyclicdata
3	(m)	OpsService:	Control_cyclic

6.3.2.2.3 Attributes

6.3.2.2.3.1 Dfn

This attribute represents the data field number to which the cyclic transfer memory belongs.

6.3.2.2.3.2 Tmid

This attribute represents the transfer memory identification number of the cyclic transfer memory.

6.3.2.2.3.3 Pri

This attribute represents the transmission priority of cyclic data.

6.3.2.2.3.4 **Memory** address

This attribute represents the starting address of the sending (or receiving) memory area.

6.3.2.2.3.5 Memory size

This attribute represents the size of the sending (or receiving) memory area

6.3.2.2.3.6 Block **number**

This attribute represents the starting number of the blocks.

6.3.2.2.3.7 Block **count**

This attribute represents the number of blocks.

6.3.2.2.4 Service specification

6.3.2.2.4.1 Put_cyclicdata

This service is used to place the cyclic data into the transfer memory which is specified by the transfer memory information. Table 9 shows the parameters for this service.

Table 9 – Put_cyclicdata service parameters

Parameter name	req
Argument	M
TargetDfn	M
Tmid	M
Pri	M
OffsetAddress	M
Size	M
CyclicData	M

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the targeted data field number to which the transfer memory recognized by the identifier (Tmid) belongs.

Tmid

This parameter specifies the transfer memory identification number of the transfer memory.

Pri

This parameter specifies the transmission priority of cyclic data.

OffsetAddress

This parameter specifies the distance from the beginning address of the transfer memory area to be transmitted CyclicData.

Size

This parameter specifies the size of CyclicData.

CyclicData

This parameter specifies the transmitting cyclic data.

6.3.2.2.4.2 Get_cyclicdata

This service is used to get cyclic data from the transfer memory area. Table 10 shows the parameters for this service.

Table 10 – Get_cyclicdata service parameters

Parameter name	req	cnf
Argument	M	
TargetDfn	M	
Tmid	M	
OffsetAddress	M	
Size	M	
Result(+)		S
Diagnosis		M
CyclicData		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the targeted data field number to which the transfer memory recognized by the identifier (Tmid) belongs.

Tmid

This parameter specifies the transfer memory identification number of the transfer memory.

OffsetAddress

This parameter specifies the distance from the beginning address of the transfer memory area to be read Cyclicdata.

Size

This parameter specifies the size of cyclicdata to get from the the transfer memory area.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

CyclicData

This parameter indicates cyclicdata get from the the transfer memory area.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.3.2.2.4.3 Control_cyclic

This service is used to control the cyclic transmission. This specifies parameters to start/ stop the cyclic transmission and to set the node-mode. Table 11 shows the parameters for this service.

Table 11 – Control_cyclic service parameters

Parameter name	req
Argument	M
TargetDfn	M
Tmid	M
CtlCmd	M
NodeMode	M

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the data field number to which the transfer memory associated with by the identifier (Tmid) belongs.

Tmid

This parameter specifies the transfer memory identification number of the transfer memory.

CtlCmd

This parameter specifies control command.

START

Cyclic transmission started.

STOP

Cyclic transmission stopped.

NodeMode

This parameter specifies a mode of own node belonging to the targeted data field. The mode indicates whether the node runs under the online state or under the test state.

ONLINE

Node to run under the online state (Online mode).

TEST

Node to run under the test state (Test mode).

6.3.3 Acyclic data ASE type N

6.3.3.1 Overview

Acyclic data ASE type N provides communication between nodes realized by the Acyclic communication.

6.3.3.2 Acyclic data class specification

6.3.3.2.1 Formal model

FAL ASE:			**Acyclic data ASE type N**
CLASS:			**Acyclic data class N**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Key Attribute:	Dfn
2	(m)	Key Attribute:	Node-address
2.1	(m)	Attribute:	Mgn
2.2	(m)	Attribute:	Lnn
3	(m)	Attribute:	Tcd
4	(m)	Attribute	Pri
5	(m)	Attribute::	Node-list
6	(m)	Attribute:	Inq-id
SERVICES:			
1	(m)	OpsService:	Put message
2	(m)	OpsService:	Get message
3	(m)	OpsService:	Put inquiry message
4	(m)	OpsService:	Put ninquiry message
5	(m)	OpsService:	Put reply message
6	(m)	OpsService:	Send aliveinfo
7	(m)	OpsService:	Receive aliveinfo
8	(m)	OpsService:	Control acyclic

6.3.3.2.2 Attributes

6.3.3.2.2.1 Dfn

This attribute represents the data field number to which the destination multicast group or the destination logical node belongs.

6.3.3.2.2.2 Node-address

This attribute represents the node address.

6.3.3.2.2.3 Mgn

This attribute represents the multicast group number to send/receive the acyclic data.

6.3.3.2.2.4 Lnn

This attribute represents the logical node number to send/receive the acyclic data.

6.3.3.2.2.5 **Tcd**

This attribute represents the transaction code corresponding to the characteristics of each acyclic data.

6.3.3.2.2.6 Pri

This attribute represents the transmission priority of acyclic data.

6.3.3.2.2.7 Node-list

This attribute represents the list of nodes which request to respond a reply message.

6.3.3.2.2.8 Inq-id

This attribute represents the inquiry identifier to match the inquiry message to the reply message.

6.3.3.2.3 Service specification

6.3.3.2.3.1 Put message

This service is used to put a message into the destination multicast group or destination node. Table 12 shows the parameters for this service.

Table 12 – Put message service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
Node-address	M	M(=)
DstMgn	S	S(=)
DstLnn	S	S(=)
Tcd	M	M(=)
Pri	M	M(=)
Data	M	M(=)
DataLength	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group or the destination logical node belongs.

Node-address

This parameter specifies the destination node address.

DstMgn

This parameter specifies the destination multicast group number to send a message.

DstLnn

This parameter specifies the destination logical node number to send a message.

Tcd

This parameter specifies the transaction code corresponding to the characteristics of each message.

Pri

This parameter specifies the transmission priority of message.

Data

This parameter specifies data to transmit.

DataLength

This parameter specifies the length of Data.

6.3.3.2.3.2 Get message

This service is used to get a message from the targeted multicast group or targeted node. Table 13 shows the parameters for this service.

Table 13 – Get message service parameters

Parameter name	req	cnf
Argument	M	
TargetDfn	M	
Node-address	M	
TargetMgn	S	
TargetLnn	S	
Result(+)		S
Diagnosis		M
TargetDfn		M
Node-address		M
TargetMgn		S
TargetLnn		S
Tcd		M
Pri		M
CtlType		M
Inq-id		C
Data		M
DataLength		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the data field number to which the targeted multicast group or the targeted logical node belongs.

Node-address

This parameter specifies the destination node address.

TargetMgn

This parameter specifies the destination multicast group number to send the acyclic data.

TargetLnn

This parameter specifies the destination logical node number to send the acyclic data.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

TargetDfn

This parameter indicates the data field number to which the destination multicast group or the destination logical node belongs.

Node-address

This parameter indicates the destination node address.

TargetMgn

This parameter indicates the destination multicast group number.

TargetLnn

This parameter indicates the destination logical node number.

Tcd

This parameter indicates the transaction code corresponding to the characteristics of each message.

Pri

This parameter indicates the transmission priority of a message.

CtlType

This parameter indicates the communication type of a receiving message.

MLT

This type indicates a multicast communication.

ONE

This type indicates a peer to peer (one to one) communication.

INQ

This type indicates an inquiry communication.

RPL

This type indicates a reply response communication.

NIQ

This type indicates an n-inquiry communication.

MCR

This type indicates a multicast communication with re-transmission control.

Inq-id

This parameter indicates the inquiry identifier to match the inquiry/ n-inquiry message to the reply message

A node which received the inquiry/ n-inquiry message shall respond with the reply message using the same Inq-id contained in the inquiry/ n-inquiry message. An Inq-id shall be unique in each node.

Data

This parameter indicates the receiving data.

DataLength

This parameter indicates the length of Data.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.3.3.2.3.3 Put inquiry message

This service is used to multicast (broadcast) an inquiry message to nodes which belong to the multicast group in the data field, or to send this message to the specified node in the data field. Each node which received this inquiry message responds with the reply message by using the put reply message service. Table 14 shows the parameters for this service.

Table 14 – Put inquiry message service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
Node-address	M	M(=)
DstMgn	S	S(=)
DstLnn	S	S(=)
Tcd	M	M(=)
Pri	M	M(=)
Data	M	M(=)
DataLength	M	M(=)
Inq-id		M

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group or the destination logical node belongs.

Node-address

This parameter specifies the destination node address.

DstMgn

This parameter specifies the destination multicast group number to send an inquiry message.

DstLnn

This parameter specifies the destination logical node number to send an inquiry message.

Tcd

This parameter indicates the transaction code corresponding to the characteristics of each inquiry message.

Pri

This parameter indicates the transmission priority of an inquiry message.

Data

This parameter specifies inquiry message (data) to transmit.

DataLength

This parameter specifies the length of Data.

Inq-id

This parameter specifies the inquiry identifier to match the inquiry message to the reply message. This parameter is locally generated. A node which received the inquiry message

shall respond with a reply message using the same Inq-id contained in the inquiry message. An Inq-id shall be unique in each node.

6.3.3.2.3.4 Put ninquiry message

This service is used to multicast (broadcast) an n-inquiry message to specified nodes which belong to the multicast group in the data field. Each node which received this inquiry message checks a node-list in the message. Nodes which are specified in the node-list respond with the reply message by using the put reply message service. Table 15 shows the parameters for this service.

Table 15 – Put ninquiry message service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
DstMgn	M	M(=)
Tcd	M	M(=)
Pri	M	M(=)
Data	M	M(=)
DataLength	M	M(=)
Node-list	M	M(=)
Inq-id		M

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group belongs.

DstMgn

This parameter specifies the destination multicast group number to send n-inquiry messages.

Tcd

This parameter indicates the transaction code corresponding to the characteristics of each n-inquiry message.

Pri

This parameter indicates the transmission priority of an n-inquiry message.

Data

This parameter specifies n-inquiry message (data) to transmit.

DataLength

This parameter specifies the length of Data.

Node-list

This parameter specifies the list of nodes which request to respond a reply message.

Inq-id

This parameter specifies the n-inquiry identifier to match the n-inquiry message to the reply message. This parameter is locally generated.

A node which received the n-inquiry message shall respond with the reply message using the same Inq-id contained in the n-inquiry message. An Inq-id shall be unique in each node.

6.3.3.2.3.5 Put reply message

This service is used to send a reply message to the destination node which sends an inquiry message or a n-inquiry message. Each node which received an inquiry message responds with the reply message by using the put reply message service. Nodes which are specified in the node-list within a n-inquiry message respond with the reply message by using the put reply message service. Table 16 shows the parameters for this service.

Table 16 – Put reply message service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
DstLnn	M	M(=)
Tcd	M	M(=)
Pri	M	M(=)
Data	M	M(=)
DataLength	M	M(=)
Inq-id	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination logical node belongs.

DstLnn

This parameter specifies the destination logical node number to send a reply messages. A destination logical node corresponds to the source node of an inquiry message or a ninquiry message.

Tcd

This parameter indicates the transaction code corresponding to the characteristics of each reply message.

Pri

This parameter indicates the transmission priority of a reply message.

Data

This parameter specifies a reply message (data) to transmit.

DataLength

This parameter specifies the length of Data.

Inq-id

This parameter specifies the inquiry/ n-inquiry identifier to match the inquiry/ n-inquiry message to the reply message. A node which received the inquiry/ n-inquiry message shall respond the reply message with the same Inq-id contained in the inquiry/ n-inquiry message. This parameter is locally generated.

6.3.3.2.3.6 Send aliveinfo

This service is used to send an alive-message into a specified multicast group periodically. Table 17 shows the parameters for this service.

Table 17 – Send aliveinfo service parameters

Parameter name	req	Ind
Argument	M	M(=)
DstDfn	M	M(=)
DstMgn	M	M(=)
AliveMode	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group belongs.

DstMgn

This parameter specifies the destination multicast group number to send an alive-message.

AliveMode

This parameter specifies the type of an alive-message.

Normal report (alive)

This type represents that a node is in active state.

Notice (Shutdown)

This type represents that a node will be scheduled to enter shutdown state.

Notice (Maintenance)

This type represents that a node will be scheduled to enter maintenance state.

6.3.3.2.3.7 Receive aliveinfo

This service is used to read information of alive-messages. This information has received periodically from a specified multicast group. Table 18 shows the parameters for this service.

Table 18 – Receive aliveinfo service parameters

Parameter name	req	cnf
Argument	M	
TargetDfn	M	
TargetMgn	M	
Result(+)		S
Diagnosis		M
TargetNodeAliveMode		N
Task-info1		C
Task-info2		C
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the data field number to which the targeted multicast group belongs.

TargetMgn

This parameter specifies the targeted multicast group number to read information of an alive-message.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

TargetNodeAliveMode

This parameter indicates an alive-related state receiving from the target node.

Normal report (alive)

This type represents that a node is in active state.

Notice (Shutdown)

This type represents that a node will be scheduled to enter shutdown state.

Notice (Maintenance)

This type represents that a node will be scheduled to enter maintenance state.

Task-info1

This parameter indicates list of state of tasks-information-1 on the target node. Each task-information-1 consists of 4 elements.

Ta_chgalvstat

This element represents status change on the task.

No change,

Status change (from “dead” state to “alive” state),

Status change (from “alive” state to “dead” state).

Ta_chginfostat

This element represents the presence or absence of change regarding information of the task.

No change,

Change.

Ta_tid

This element represents the identifier of a task.

Ta_data

This element represents the user defined data of a task.

Task-info2

This parameter indicates information allowing user to set data freely.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.3.3.2.3.8 Control acyclic

This service is used to control the acyclic transmission. This specifies parameters to activate/ inactivate the acyclic transmission, and to set the node-mode. Table 19 shows the parameters for this service.

Table 19 – Control_acyclic service parameters

Parameter name	req
Argument	M
TargetDfn	M
CtlCmd	M
NodeMode	M

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the targeted data field number.

CtlCmd

This parameter specifies control command.

ACTIVATE

Acyclic transmission inactivated.

INACTIVATE

Acyclic transmission inactivated.

NodeMode

This parameter specifies a mode of own node belonging to the targeted data field. The mode indicates whether the node runs under the online state or under the test state.

ONLINE

Node to run under the online state (Online mode).

TEST

Node to run under the test state (Test mode).

6.3.4 Management ASE type N

6.3.4.1 Overview

Management ASE provides the node management function.

6.3.4.2 Management class specification

6.3.4.2.1 Overview

This class provides the node management functions.

FAL ASE: **Management ASE type N**
CLASS: **Management N**
CLASS ID: **Not used**
PARENT CLASS: **Top**
ATTRIBUTES:
1 (m) Key Attribute: Dfn

SERVICES:

1 (m) MgtService: Get_attribute

2 (m) MgtService: Set_attribute

6.3.4.2.2 Attributes

6.3.4.2.2.1 Dfn

This attribute represents the data field number.

6.3.4.2.3 Service specification

6.3.4.2.3.1 Get_attribute

This service is used to get the attribute value of a class. Table 20 shows the parameters for this service.

Table 20 – Get attribute service parameters

Parameter name	req	cnf
Argument	M	
RequestList	M	
Result(+)		S
Diagnosis		M
List_of_Attribute Values		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

RequestList

This parameter specifies the attributes of the target whose values are requested.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

List_of_AttributeValues

This parameter indicates the list of attributes values.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.3.4.2.3.2 Set_attribute

This service is used to set the attribute value of a class. Table 21 shows the parameters for this service.

Table 21 – Set attribute service parameters

Parameter name	req	cnf
Argument	M	
SetList	M	
Attribute	M	
Set_value	M	
Result(+)		S
Diagnosis		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

SetList

This parameter specifies the list of attributes and values to be set.

Attribute

This parameter specifies the attribute to be set.

Set_value

This parameter specifies the value to be set.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.4 AR type S

AR consists of four classes: Cyclic control, Remote control, RCL communication control, and RT communication control. Figure 6 shows the structure.

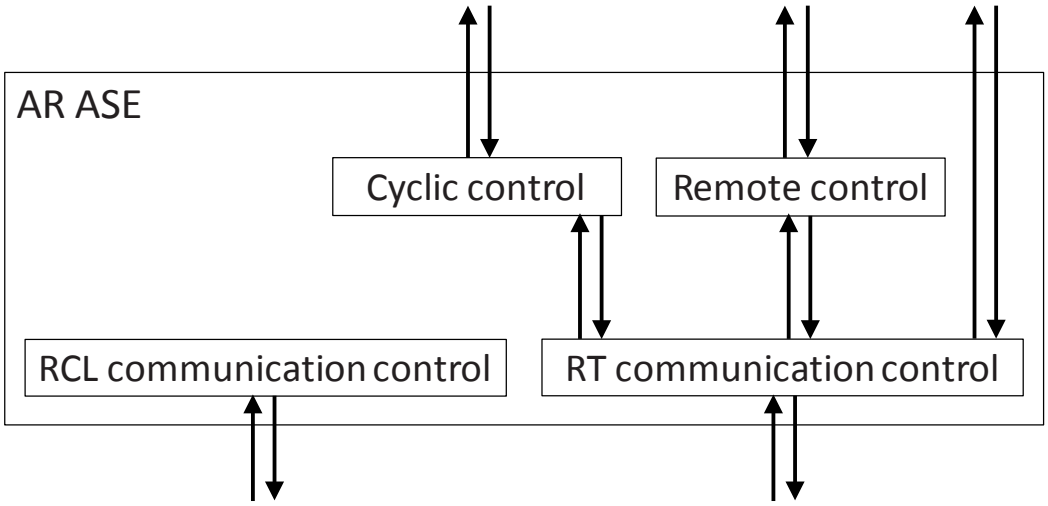


Figure 6 – Structure of AR type S of FAL type 25

There are two AR end points in each node for the RCL communication and the RT communication. These end points correspond to the RCL communication control class and the RT communication control class respectively. The Cyclic control class is a class relating to the cyclic communication. The Remote control class is a class relating to the remote device control.

6.4.1 Cyclic control type S

6.4.1.1 Cyclic control type S class specification

6.4.1.1.1 Formal model

FAL ASE: **AR ASE type S**

CLASS: **Cyclic control**

CLASS ID: **Not used**

PARENT CLASS: **Top**

ATTRIBUTES:

1	(m)	Attribute:	CYC_ID
2	(m)	Attribute:	Block area
3	(m)	Attribute:	CYC_status
4	(m)	Attribute:	Station address
5	(m)	Attribute:	Max cycletime

SERVICES:

1	(m)	OpsService:	CYC_WRITE
2	(m)	OpsService:	CYC_READ
3	(m)	OpsService	CTL_CYCLIC

6.4.1.1.2 Attributes

6.4.1.1.2.1 CYC_ID

This parameter represents the identification of the cyclic operation.

6.4.1.1.2.2 Block area

This parameter represents the block area for transmission in the shared memory.

6.4.1.1.2.3 CYC_status

This parameter represents the status of cyclic operation.

6.4.1.1.2.4 Station address

This parameter represents the node (station) number of own node.

6.4.1.1.2.5 Max cycletime

This parameter represent the maximum cycle that can be used in cyclic communication on Type S network.

6.4.1.1.3 Service specification

6.4.1.1.3.1 CYC_WRITE

This service is used to write data to the shared memory. Table 22 shows the parameters for this service.

Table 22 – CYC_WRITE service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	M (=)
Block_ID		
CYCdata		
Result (+)		S
Status		M
Additional_info		C
Result (-)		S
Status		M
Additional_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

Block_ID

This parameter specifies the block area in the shared memory to write CYCdata.

CYCdata

This parameter specifies the data to write into the shared memory.

Result (+)

This selection type parameter indicates that the service request succeeded.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Service not supported;

Command parameter error;

Configuration data error.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

6.4.1.1.3.2 CYC_READ

This service is used to read data from the shared memory. Table 23 shows the parameters for this service.

Table 23 – CYC_READ service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	M (=)
Block_ID	M	M (=)
Result (+)		S
CYCdata		M
Status		M
Additional_info		C
Result (-)		S
Status		M
Additional_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

Block_ID

This parameter specifies the block area to read the data from the shared memory.

Result (+)

This selection type parameter indicates that the service request succeeded.

CYCdata

This parameter contains the data of the specified Block_ID in shared memory.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Service not supported;

Command parameter error;

Configuration data error.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

6.4.1.1.3.3 CTL_CYCLIC

This service controls the cyclic communication. Table 24 shows the parameters for this service.

Table 24 – CTL_CYCLIC service parameters

Parameter name	req	cnf
Argument	M	
CYC_ID	M	
CYCctl	M	
CTLdata	M	
Result (+)		S
Status		M
Additional_info		C
Result (-)		S
Status		M
Additional_info		C
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

CYC_ID

This parameter represents the identification of this service request.

CYCctl

This parameter contains the command to control the cyclic communication.

CTLdata

This parameter contains the data to configure the cyclic control communication.

Result (+)

This selection type parameter indicates that the service request succeeded.

Status

This parameter indicates success.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

Service not supported;

Command parameter error;

Configuration data error.

Additional_info

This field contains the node profiles such as Station address and IP address, and vendor-specific additional information.

6.4.2 Remote control

6.4.2.1 Remote control specification

6.4.2.1.1 Formal model

FAL ASE:			**AR ASE type S**
CLASS:			**Remote control**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Attribute:	Station address
2	(m)	Attribute:	IP address
SERVICES:			
1	(m)	OpsService:	SendRMTCTL

6.4.2.1.2 Attributes

6.4.2.1.2.1 Station address

This parameter represents the node (station) number of the node.

6.4.2.1.2.2 IP address

This parameter represents the IP address of own node.

6.4.2.1.3 Service specification

6.4.2.1.3.1 SendRMTCTL

This service is used to write data to the shared memory. Table 25 shows the parameters for this service.

Table 25 – SendRMTCTL service parameters

Parameter name	req	cnf
Argument	M	M
D_add	M	M
S_add	M	M
CMD	M	
CMDdata	C	
Result (+)		S
CMD_rsp		M
CMDdata_rsp		C
Result (-)		S
CMD_rsp		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CMD

This parameter contains the command for remote control operation.

CMDdata

The parameter contains the data sent to another node to use remote control.

Result (+)

CMD_rsp

This parameter contains the response command from the node which received remote control command.

CMDdata_rsp

This parameter contains the response data from the node which received remote control command.

Result (-)

This selection type parameter indicates that the service request failed.

CMD_rsp

This parameter contains the response command to notify that the service request failed to receive remote control command.

6.4.3 RCL communication control

6.4.3.1 RCL communication control class specification

6.4.3.1.1 Formal model

FAL ASE: **AR ASE type S**

CLASS: **RCL communication control**

CLASS ID: **Not used**

PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Attribute:	Station address
2	(m)	Attribute:	MAC address
3	(m)	Attribute	Node status
4	(m)	Attribute	Port A status
5	(m)	Attribute	Port B status
6	(m)	Attribute:	DL-mapping-reference
SERVICES:			
1	(m)	OpsService:	RCL_START
2	(m)	OpsService:	RCL_STOP

6.4.3.1.2 Attributes

6.4.3.1.2.1 Station address

This parameter represents the node (station) number of the node.

6.4.3.1.2.2 MAC address

This parameter represents the MAC address of own node.

6.4.3.1.2.3 Node status

This parameter represents the node status of own node.

6.4.3.1.2.4 Port A status

This parameter represents the link status at the port A of own node.

6.4.3.1.2.5 Port B status

This parameter represents the link status at the port B of own node.

6.4.3.1.2.6 DL-mapping-reference

This attribute provides a reference to the underlying Data Link Layer mapping for the conveyance path for this AR ASE. DL mapping attributes for the data-link layer are specified in IEC PAS 62953-6-25.

6.4.3.1.3 Service specification

6.4.3.1.3.1 RCL_START

This service is used to start the specified RCL frame transmission. This is initiated by local action. Table 26 shows the parameters for this service.

Table 26 – RCL_START service parameters

Parameter name	ind
Argument	M
RCLType	M
D_add	M
RCLPri	M
RCLPort	M

Argument

The argument contains the parameters of the service indication from DLL.

RCLType

This parameter represents the RCL type of the requested RCL frames from DLL.

D_add

This parameter represents the MAC address of the requested RCL frames from DLL.

RCLPri

This parameter represents the priority with VLAN of the requested RCL frames from DLL.

RCLPort

This parameter represents the transmission port of the requested RCL frames from DLL.

6.4.3.1.3.2 RCL_STOP

This service is used to stop the specified RCL frames transmission. This is initiated by local action. Table 27 shows the parameters for this service.

Table 27 – RCL_STOP service parameters

Parameter name	ind
Argument	M
RCLType	M

Argument

The argument contains the parameters of the service indication from DLL.

RCLType

This parameter represents the RCL type of the requested RCL frames from DLL.

6.4.4 RT communication control

6.4.4.1 RT communication control class specification

6.4.4.1.1 Formal model

FAL ASE: **AR ASE type S**

CLASS: **RT communication control**

CLASS ID: **Not used**

PARENT CLASS: **Top**

ATTRIBUTES:

1 (m) Attribute: Station address

2 (m) Attribute: MAC address

3 (m) Attribute: DL-mapping-reference

SERVICES:

1 (m) OpsService: SendCTL
2 (m) OpsService: SendINFO
3 (m) OpsService: SendCTL_RMT
4 (m) OpsService: SendCYC

6.4.4.1.2 Attributes

6.4.4.1.2.1 Station address

This parameter represents the node (station) number of the node.

6.4.4.1.2.2 MAC address

This parameter represents the MAC address of own node.

6.4.4.1.2.3 DL-mapping-reference

This attribute provides a reference to the underlying Data Link Layer mapping for the conveyance path for this AR ASE. DL mapping attributes for the data-link layer are specified in IEC PAS 62953-6-25.

6.4.4.1.3 Service specification

6.4.4.1.3.1 SendCTL

This service is used to send the control communication frames other than remote control service. Table 28 shows the parameters for this service.

Table 28 – SendCTL service parameters

Parameter name	req	ind	cnf
Argument	M	M (=)	
D_add	M	M (=)	
S_add	M	M (=)	
CTLdata	M	M (=)	
Result (+)			S
Status			M
Result (-)			S
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CTLdata

The parameter contains the data sent by control communication.

Result (+)

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.4.4.1.3.2 SendINFO

This service is used to send the information communication frames. Table 29 shows the parameters for this service.

Table 29 – SendINFO service parameters

Parameter name	req	ind	cnf
Argument	M	M (=)	
D_add	M	M (=)	
S_add	M	M (=)	
INFOdata	M	M (=)	
Result (+)			S
Status			M
Result (-)			S
Status			M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

INFOdata

The parameter contains the sending information data.

Result (+)

Status

This parameter indicates success.

Result (-)

This selection type parameter indicates that the service request failed.

Status

This parameter indicates failure.

6.4.4.1.3.3 SendCTL_RMT

This service is used to send the control communication frames. Table 30 shows the parameters for this service.

Table 30 – SendCTL_RMT service parameters

Parameter name	req	ind	rsp	cnf
Argument	M	M (=)	M	M (=)
D_add	M	M (=)	M	M (=)
S_add	M	M (=)	M	M (=)
CTLdata	M	M (=)	M	M (=)

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CTLdata

The parameter contains the data sent by control communication.

6.4.4.1.3.4 SendCYC

This service is used to send the information communication frames. Table 31 shows the parameters for this service.

Table 31 – SendCYC service parameters

Parameter name	req	ind
Argument	M	M (=)
D_add	M	M (=)
S_add	M	M (=)
CYCdata	M	M (=)

Argument

The argument contains the parameters of the service request.

D_add

This parameter is the address of the destination node.

S_add

This parameter is the address of the source node.

CYCdata

The parameter contains the data sent by cyclic communication.

6.5 AR type N

AR consists of three classes: Cyclic transmission class, Acyclic transmission class and RT(Real time) communication class. Figure 7 shows the structure.

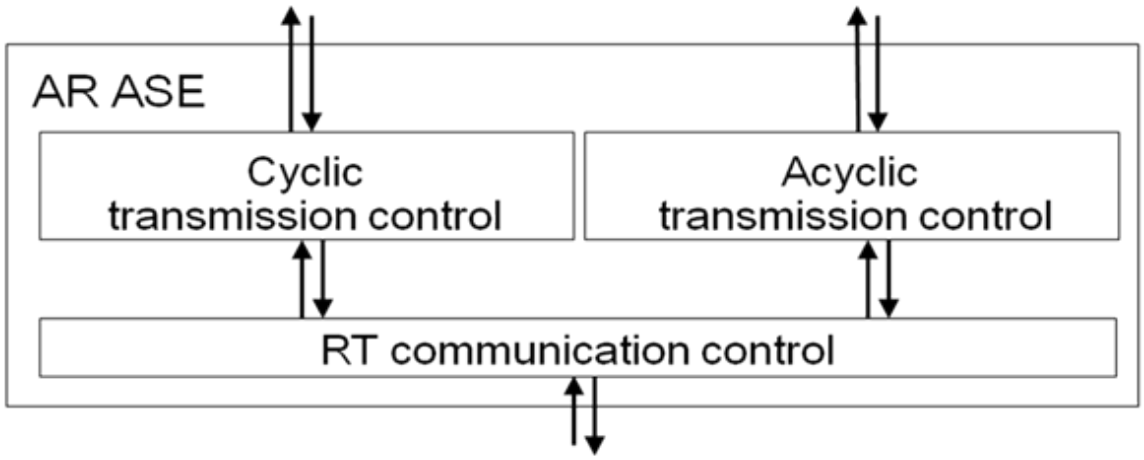


Figure 7 – Structure of AR type N of FAL type 25

There are two AR end points in each node for the cyclic transmission and the acyclic transmission. These end points correspond to the Cyclic transmission control class and the Acyclic transmission control class respectively. The RT communication control class is a class relating to communication paths control between end nodes and relating to communication priority control.

6.5.1 Cyclic transmission control

6.5.1.1 Cyclic transmission control class specification

6.5.1.1.1 Formal model

FAL ASE:			**AR ASE type N**
CLASS:			**Cyclic transmission control class**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Key Attribute:	Dfn
2	(m)	Key Attribute:	Tmid
3	(m)	Attribute	Pri
4	(m)	Attribute	Mgn
SERVICES:			
1	(m)	OpsService:	Write_cyclicdata
2	(m)	OpsService:	Read_cyclicdata
3	(m)	OpsService:	Ctl_cyclic

6.5.1.1.2 Attributes

6.5.1.1.2.1 Dfn

This attribute represents the data field number to which the transfer memory associated with the identifier (Tmid) belongs.

6.5.1.1.2.2 Tmid

This attribute represents the transfer memory identification number of the transfer memory.

6.5.1.1.2.3 Pri

This attribute represents the transmission priority of cyclic data.

6.5.1.1.2.4 Mgn

This attribute represents the multicast group number to send/receive the transfer memory.

6.5.1.1.3 Sevice specification

6.5.1.1.3.1 Write_cyclicdata

This service is used to write cyclic data into the transfer memory. Table 32 shows the parameters for this service.

Table 32 – Write_cyclicdata service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
DstMgn	M	M(=)
Tmid	M	M(=)
Pri	M	M(=)
OffsetAddress	M	M(=)
Size	M	M(=)
CyclicData	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the transfer memory associated with the identifier (Tmid) belongs.

DstMgn

This parameter specifies the multicast group number to send the transfer memory.

Tmid

This parameter specifies the transfer memory identification number of the transfer memory.

Pri

This parameter specifies the transmission priority of cyclic data.

OffsetAddress

This parameter specifies the distance from the beginning address of the transfer memory area for the CyclicData to be written.

Size

This parameter specifies the size of CyclicData.

CyclicData

This parameter specifies the transmitting cyclic data.

6.5.1.1.3.2 Ctl_cyclic

This service is used to control the cyclic transmission. Table 33 shows the parameters for this service.

Table 33 – Ctl_cyclic service parameters

Parameter name	req	cnf
Argument	M	
TargetDfn	M	
TargetMgn	M	
Tmid	M	
CtlCmd	M	
Result(+)		S
Diagnosis		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies the data field number to which the transfer memory associated with the identifier (Tmid) belongs.

TargetMgn

This parameter specifies the multicast group number to control the transfer memory.

Tmid

This parameter specifies the transfer memory identification number of the transfer memory.

CtlCmd

This parameter specifies control command.

START

Cyclic transmission started.

STOP

Cyclic transmission stopped.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.5.2 Acyclic transmission control

6.5.2.1 Acyclic transmission control class specification

6.5.2.1.1 Formal model

FAL ASE: **AR ASE type N**

CLASS: **Acyclic transmission control class**

CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Key Attribute:	Dfn
2	(m)	Key Attribute:	Node-address
2.1	(m)	Attribute:	Mgn
2.2	(m)	Attribute:	Lnn
3	(m)	Attribute:	Tcd
4	(m)	Attribute:	Pri
SERVICES:			
1	(m)	OpsService:	Transmit_acyclicdata1
2	(m)	OpsService:	Transmit_acyclicdata2
3	(m)	OpsService:	Ctl_cyclicdata

6.5.2.1.2 Attributes

6.5.2.1.2.1 Dfn

This attribute represents the data field number to which the multicast group belongs.

6.5.2.1.2.2 Node-address

This attribute represents the node address.

6.5.2.1.2.3 Mgn

This attribute represents the multicast group number to send/receive the acyclic data.

6.5.2.1.2.4 Lnn

This attribute represents the logical node number to send/receive the acyclic data.

6.5.2.1.2.5 Tcd

This attribute represents the transaction code corresponding to the characteristics of each acyclic data.

6.5.2.1.2.6 Pri

This attribute represents the transmission priority of acyclic data.

6.5.2.1.3 Service specification

6.5.2.1.3.1 Transmit_acyclicdata1

This service is used to transmit acyclic data (type 1) to the destination multicast group or destination node. Table 34 shows the parameters for this service.

Table 34 – Transmit_acyclicdata1 service parameters

Parameter name	req	ind	cnf
Argument	M	M(=)	
DstDfn	M	M(=)	
Node-address	M	M(=)	
DstMgn	S	S(=)	
DstLnn	S	S(=)	
Tcd	M	M(=)	
Pri	M	M(=)	
CtlType	M	M(=)	
Inq-id	C	C(=)	M
NodeList	C	C(=)	
Data	M	M	
DataLength	M	M	
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.			

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group or the destination logical node belongs.

Node-address

This parameter specifies the destination node address.

DstMgn

This parameter specifies the destination multicast group number to send the acyclic data.

DstLnn

This parameter specifies the destination logical node number to send the acyclic data.

Tcd

This parameter specifies the transaction code corresponding to the characteristics of each acyclic data.

Pri

This parameter specifies the transmission priority of acyclic data.

CtlType

This parameter specifies the communication type of a receiving message.

MLT

This type specifies a multicast communication.

ONE

This type specifies a peer to peer (one to one) communication.

INQ

This type specifies an inquiry communication.

RPL

This type specifies a reply response communication.

NIQ

This type specifies an n-inquiry communication.

MCR

This type specifies a multicast communication with re-transmission control.

Inq-id

This parameter specifies the inquiry identifier to match the inquiry/ n-inquiry communication message to the reply response communication message. Each node which received an inquiry communication message responds with the reply communication message. Nodes which are specified in the node-list respond with the reply communication message. An Inq-id shall be unique in each node.

NodeList

This parameter specifies the list of destination node required to response.

Data

This parameter specifies data to transmit.

DataLength

This parameter specifies the length of Data.

6.5.2.1.3.2 Transmit_acyclicdata2

This service is used to transmit alive message data into the destination multicast group. Table 35 shows the parameters for this service.

Table 35 – Transmit_acyclicdata2 service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
DstMgn	M	M(=)
AliveMode	M	M(=)
Task-info1	C	C(=)
Task-info2	C	C(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the destination multicast group belongs.

DstMgn

This parameter specifies the destination multicast group number to send alive messages.

AliveMode

This parameter specifies the type of an alive-message.

Normal report (alive)

This type represents that a node is in active state.

Notice (Shutdown)

This type represents that a node will be scheduled to enter shutdown state.

Notice (Maintenance)

This type represents that a node will be scheduled to enter maintenance state.

Task-info1

This parameter specifies list of state of tasks-information-1 on the own node. Each task-information-1 consists of 4 elements.

Ta_chgalvstat

This element represents status change on the task.

No change,

Status change (from "dead" state to "alive" state),

Status change (from "alive" state to "dead" state).

Ta_chginfostat

This element represents the presence or absence of change regarding information of the task.

No change,

Change.

Ta_tid

This element represents the identifier of a task.

Ta_data

This element represents the user defined data of a task.

Task-info2

This parameter specifies the task information that the user set.

6.5.2.1.3.3 Ctl_acyclic

This service is used to control acyclic transmission. Table 36 shows the parameters for this service.

Table 36 – Ctl_acyclic service parameters

Parameter name	req	cnf
Argument	M	
TargetDfn	M	
CtlCmd	M	
Result(+)		S
Diagnosis		M
Result(-)		S
Diagnosis		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

TargetDfn

This parameter specifies a targeted data field number.

CtlCmd

This parameter specifies control command.

START

Acyclic transmission started.

STOP

Acyclic transmission stopped.

Result(+)

This selection type parameter indicates that the service request succeeded.

Diagnosis

This parameter indicates success.

Result(-)

This selection type parameter indicates that the service request failed.

Diagnosis

This parameter indicates an error.

6.5.3 RT communication control type N

6.5.3.1 RT communication control type N class specification

6.5.3.1.1 Formal model

FA ASE:			**AR ASE type N**
CLASS:			**RT communication control class N**
CLASS ID:			**Not used**
PARENT CLASS:			**Top**
ATTRIBUTES:			
1	(m)	Key Attribute:	Dfn
2	(m)	Key Attribute:	Node-address
2.1	(m)	Attribute:	Mgn
2.2	(m)	Attribute:	Lnn
3	(m)	Attribute:	Tcd
SERVICES:			
1	(m)	OpsService:	Send_cyclicdata
2	(m)	OpsService:	Send_acyclicdata

6.5.3.1.2 Attributes

6.5.3.1.2.1 Dfn

This attribute represents the data field number to which the multicast group or logical node belongs.

6.5.3.1.2.2 Node-address

This attribute represents the node address.

6.5.3.1.2.3 Mgn

This attribute represents the multicast group number to send/receive data.

6.5.3.1.2.4 **Lnn**

This attribute represents the logical node number to send/receive data.

6.5.3.1.2.5 Tcd

This attribute represents the transaction code corresponding to the characteristics of each data.

6.5.3.1.3 Service specification

6.5.3.1.3.1 Send_cyclicdata

This service is used to send a cyclic PDU into the destination data field. Table 37 shows the parameters for this service.

Table 37 – Send_cyclicdata service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
DstMgn	M	M(=)
PDUdata	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This attribute specifies the data field number to which the multicast group belongs.

DstMgn

This parameter specifies the multicast group number to send a cyclic PDU.

PDUdata

This parameter specifies the data of a cyclic PDU.

6.5.3.1.3.2 Send_acyclicdata

This service is used to send an acyclic PDU into the destination data field or the destination logical node. Table 38 shows the parameters for this service.

Table 38 – Send_acyclicdata service parameters

Parameter name	req	ind
Argument	M	M(=)
DstDfn	M	M(=)
Node-address	M	M(=)
DstMgn	S	S(=)
DstLnn	S	S(=)
PDUdata	M	M(=)

Argument

The argument contains the parameters of the service request.

DstDfn

This parameter specifies the data field number to which the multicast group or logical node belongs.

Node-address

This parameter specifies the node address.

DstMgn

This parameter specifies the multicast group number to send/receive data.

DstLnn

This parameter specifies the logical node number to send/receive data.

PDUdata

This parameter specifies the data of an acyclic PDU.

Bibliography

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEEE 802.1Q, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks;* available at http://www.ieee.org

ISO 15745-4:2003, *Industrial automation systems and integration – Open systems application integration framework – Part 4: Reference description for Ethernet-based control systems;* available at <http://www.iso.org>

CONTENTS

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 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 12 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 standard 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; and.

The purpose of this standard is to define the protocol provided to

a) define the wire-representation of the service primitives defined in IEC PAS 62953-5-25, and
b) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) 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 standard to provide access to the FAL to control certain aspects of its operation.

1.2 Specification

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC PAS 62953-5-25. 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 standard 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, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25:Data-link layer service definition – Type 25 elements*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

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

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications*

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

IEEE 802.1D-2004, *IEEE Standard for Local and metropolitan area networks – Media access control (MAC) Bridges,* available at <http://www.ieee.org>

IEEE 802.1Q-2005, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks,* available at <http://www.ieee.org>

3 Terms, definitions, symbols and abbreviations

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 terms

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

a) object identifier;
b) type.

3.1.5 ISO/IEC 61158 terms

For the purposes of this document, the following terms given in IEC 61158 apply:

a) DLL mapping protocol machine;

b) fieldbus application layer;

c) FAL service protocol machine;

d) protocol data unit.

3.2 Additional terms and definitions for this IEC PAS 62953-6-25

3.2.1
ADP message
a message conveyed by an autonomous decentralized system protocol

3.2.2
alive-message
a 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 bytes

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
A 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
a 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 (i.e. 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)

3.2.16
multicast group
a 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 (whether an online node or a test node)

3.2.19
primary LAN
a LAN used in normal state in a duplex LAN system

3.2.20
priority control
control the transmission order with VLAN priority to ensure the real-time communication in type S network.

3.2.21
RCL communication
control ring network of type S network using RCL frames

3.2.22
Remote control
control the node based on remote method invocation

3.2.23
secondary LAN
a 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	
CP	Communication Profile	[IEC 61784-2]
CPF	Communication Profile Family	[IEC 61784-2]
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	
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 used in this document are defined in IEC 61158 Type 25 and IEC 61784-2 CPF 20.

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.

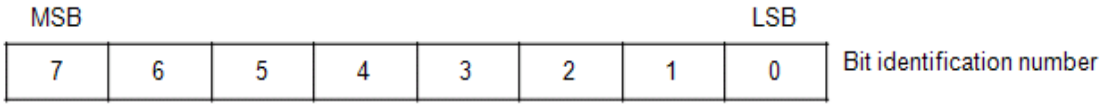


Figure 1 – Bit description in octets

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 must 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.
NOTE “*/ 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

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 TimeSend_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.

Al_ExtentionInfo::= OctString SIZE(0..664)
Al_chg_time::= Unsigned32
Al_ipv4addr1::= OctString SIZE(4)
Al_ipv4addr2::= OctString SIZE(4)
Al_ipv6addr1::= OctString SIZE(16)
Al_ipv6addr2::= OctString SIZE(16)
Al_mode::= Unsigned8
Al_msgsernot::= Unsigned16
Al_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

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	Bytes	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 x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1+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 byte 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

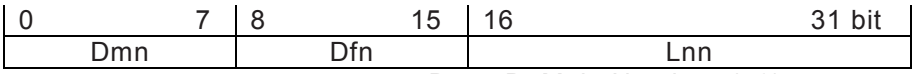
Value	PDU type
“NUXM”	PDUs transmitted in IPv4 Data field
“NUV6”	PDUs transmitted in IPv6 Data field

5.3.2.2 hd_ml

This field shall indicate the length of the total message including FALAR-N header length (= 64 bytes) 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.



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

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

– No data field can be set up across multiple LAN segments. Every LAN segment shall be assigned with Dfn.

– One LAN segment may 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

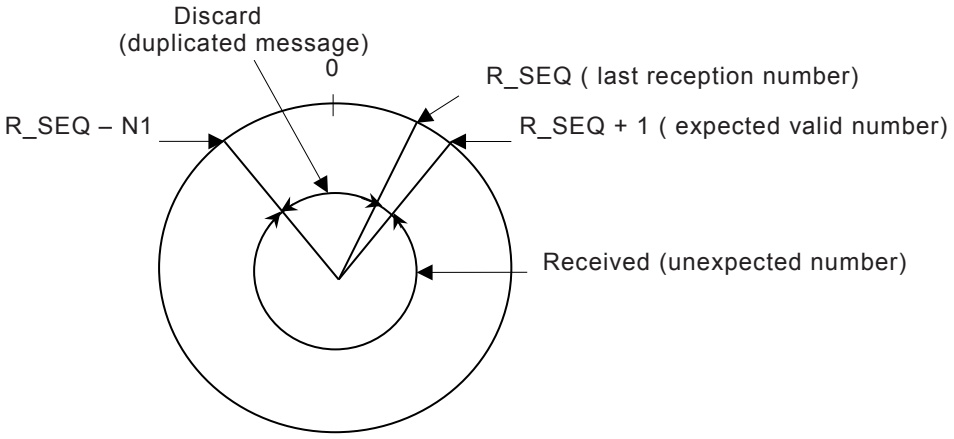
If R_V_SEQ is not 0 and hd_v_seq is not equal to R_V_SEQ, hd_v_seq and hd_seq values update the R_V_SEQ and R_SEQ values, respectively. On detecting a duplicated message, a receiving node discards the received message.

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
SEQ: Value of hd_seq in the FALAR_N Header N1: Transmission sequence number assumed to be duplicated. This value is implemented matter.		



N1: Transmission number assumed to be duplicated

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

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)
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 5.3.2.19, 5.3.2.20 and 5.3.2.21 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 59999 (user-definable)
- TCDs for system transactions: 60000 to 65534 (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
60003	Alive report
60061	Retransmission control
60027	File transfer
60056	Cyclic memory transfer
60058	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. 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 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.		

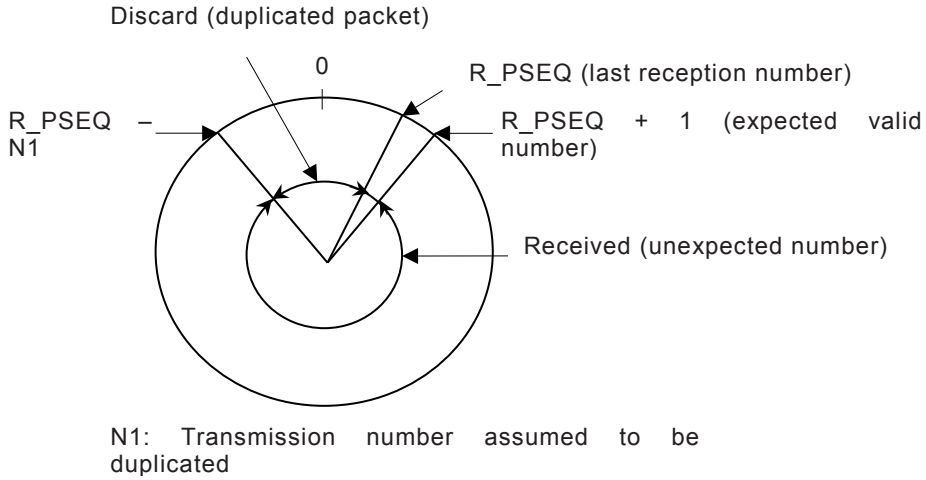


Figure 6 – Valid reception packet sequence number

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.

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

Table 22 – Detailed conditions for packet sequence number check (One to one communication using a duplex LAN)

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

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.

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 bytes under the Ethernet standard.

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

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

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

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

If a message size is more than α bytes, 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	1472	1472	1472	340	1024
hd_ml	4564	4564	4564	4564	1024

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	1460	1460	1460	376	1024
hd_ml	4564	4564	4564	4564	1024

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 bytes. 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:

c) Inquiry to all nodes using UDP port,
d) 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 byte (0) indicates node number 1.
The second bit (1) of the first byte (0) indicates node number 2.
The eighth bit (7) of the 512th byte (511) indicates node number 4096.

	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 Byte	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 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.

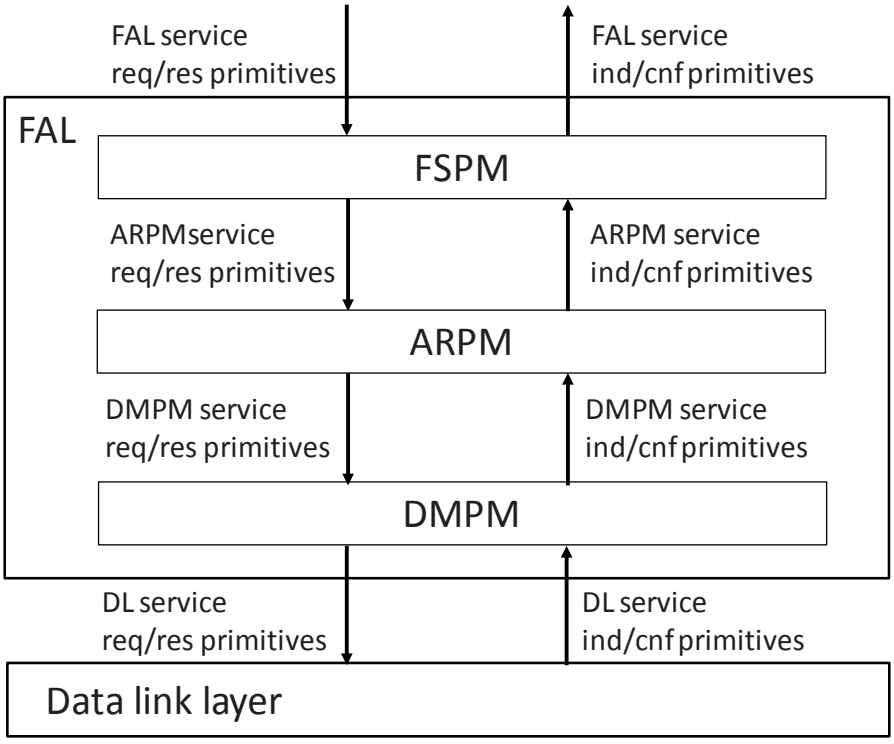


Figure 8 – Relationships between protocol machines

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.

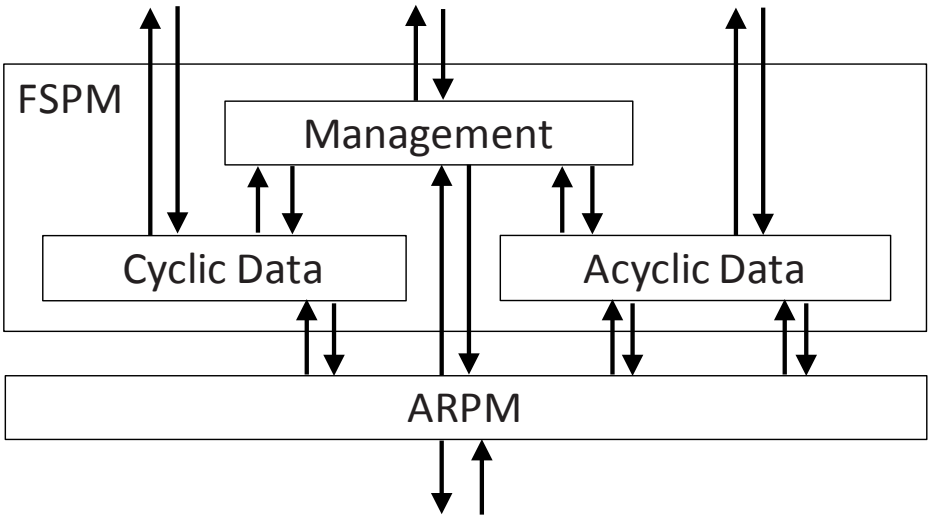


Figure 9 – Structure of FSPM type S

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
Set Attribute.req
Get Attribute.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

Set Attribute.cnf

Get Attribute.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 Byte/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	0km to 10km	1ms to 5ms	256.0
2	0km to 10km	Over 10ms	409.6
3	Over 10km	1ms to 5ms	128.0
4	Over 10km	Over 10ms	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 give 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.

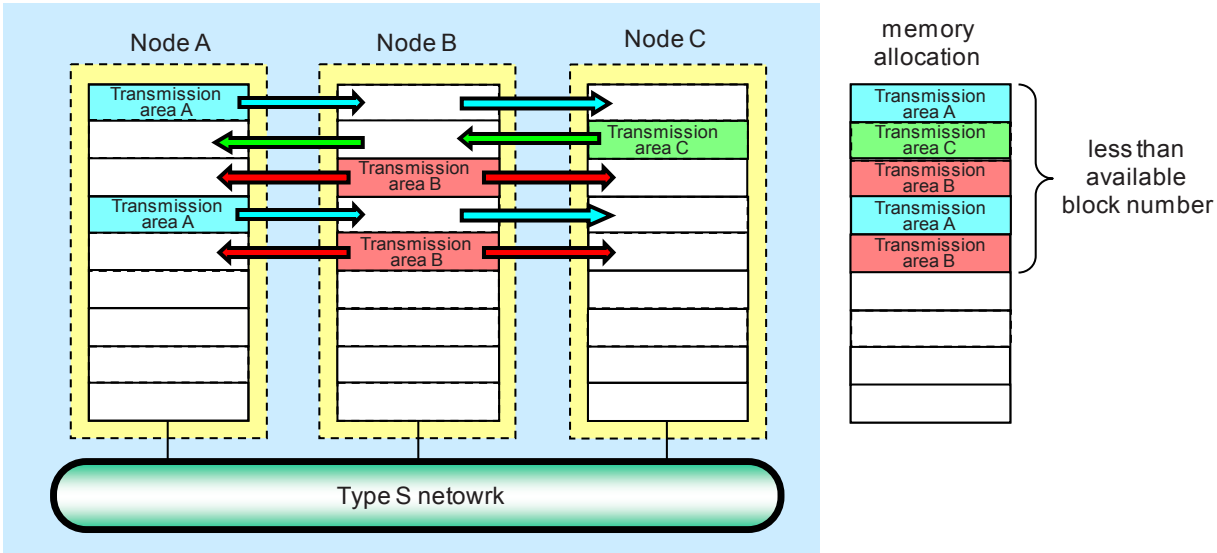


Figure 10 – Shared memory allocation in Type S network

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

7.2.3.3 Management

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

Table 42 – Management state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Set_attribute.req => Set_attribute.cnf	ACTIVE
2	ACTIVE	Get_attribute.req => Get_attribute.cnf	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.

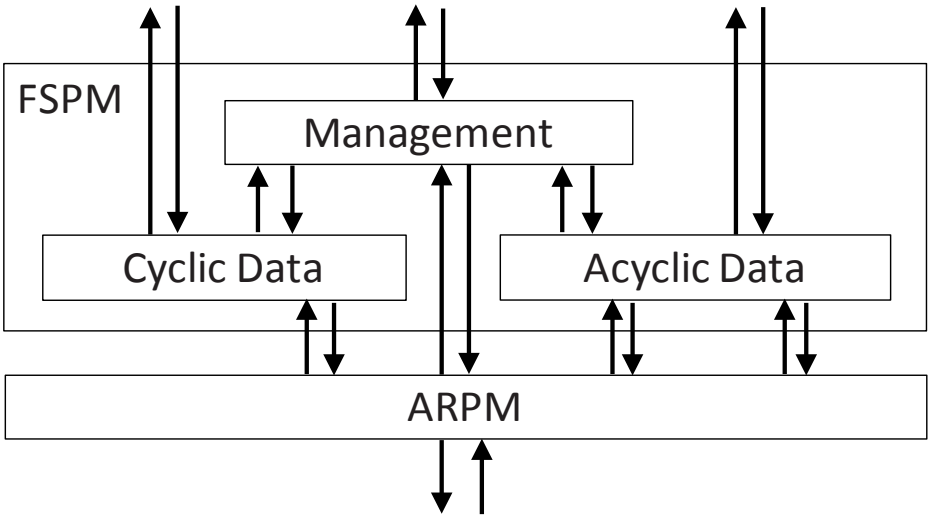


Figure 11 – Structure of FSPM type N

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
Set_attribute.req
Get_attribute.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

Set_attribute.cnf
Get_attribute.cnf

7.3.2 FSPM

7.3.2.1 Cyclic data

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

Table 43 – 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 44.

Table 44 – 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
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

#	Current state	Event /condition => actions	Next state
13	ACTIVE	Transmit_acyclicdata2.ind => Send_aliveinfo.req	ACTIVE

7.3.2.3 Management

Details of Cyclic data state machine are shown in Table 45.

Table 45 – Management state table

#	Current state	Event /condition => actions	Next state
1	ACTIVE	Set_attribute.req => Set_attribute.cnf	ACTIVE
2	ACTIVE	Get_attribute.req => Get_attribute.cnf	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.

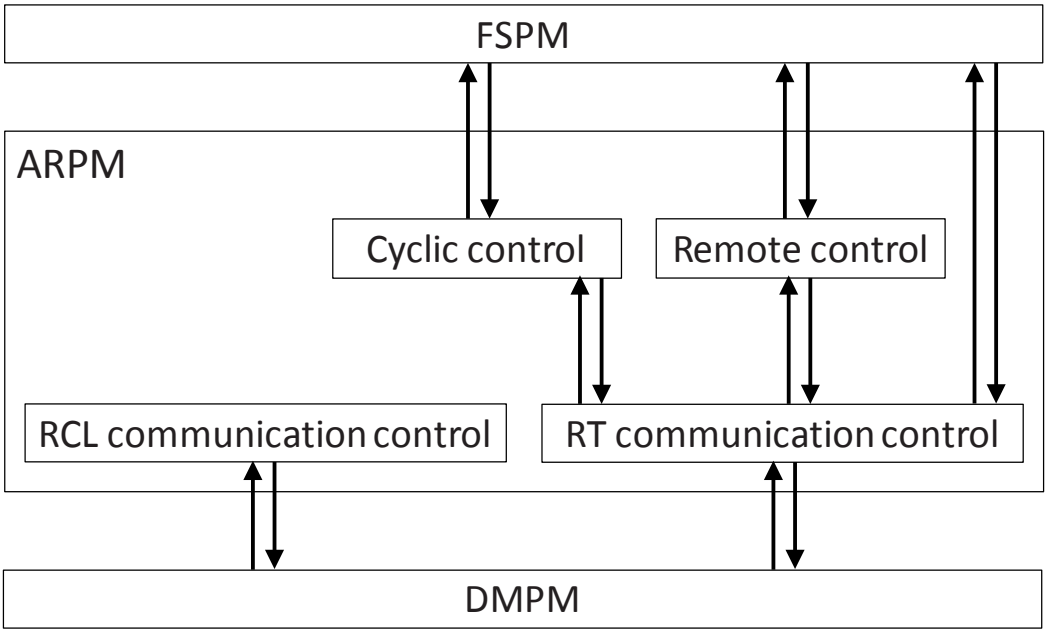


Figure 12 – Structure of ARPM type S

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={1,2,5,10,20,50,100,200,500,1000})

The number of blocks x ($0 \leq x \leq 21 \times n$)

The number of blocks per cycle $N\_nc = \min(21, x-21\times c)$ ($c=\{0,1,\cdots,[x/21]-1\}$)

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.

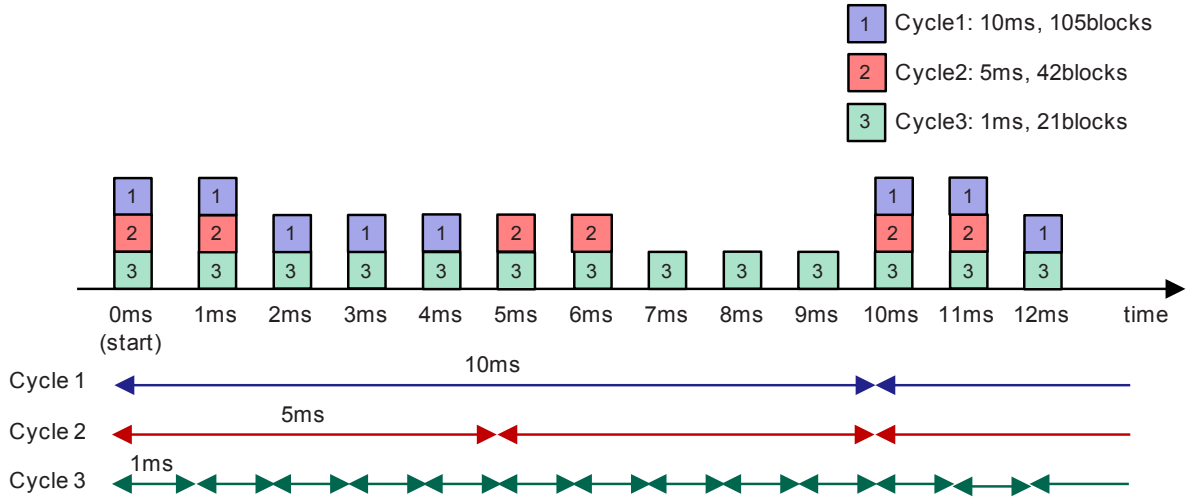


Figure 13 – Sequence of cyclic communication

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.

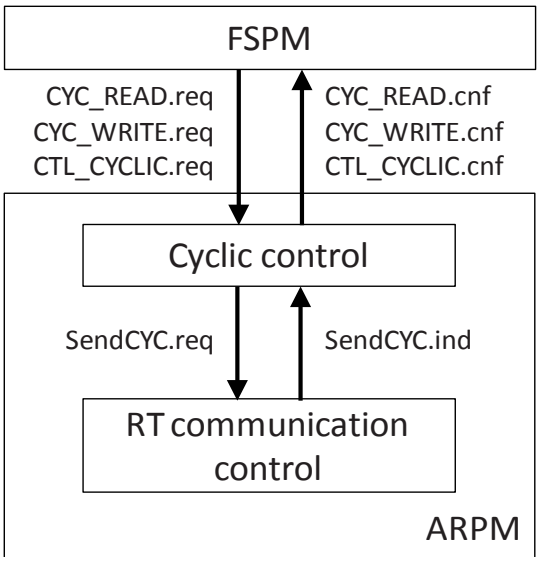


Figure 14 – The primitives for cyclic control

8.1.2.3 Cyclic control state table

Details of Cyclic control state machine are shown in Table 46.

Table 46 – 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

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 47.

Table 47 – 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 48.

Table 48 – 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.

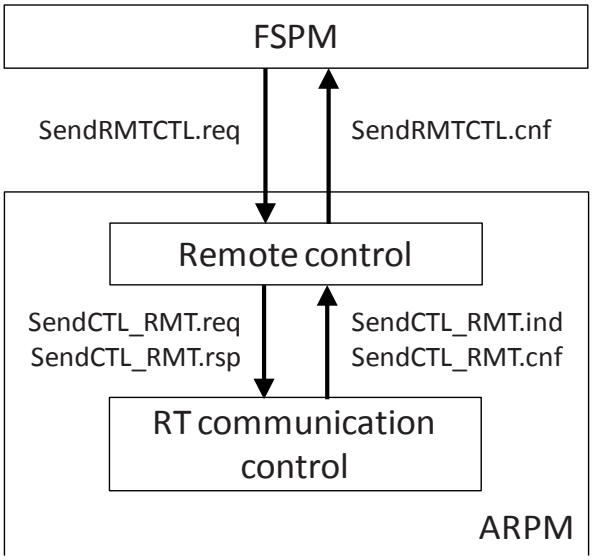


Figure 15 – The primitives for Remote control

8.1.3.2 Remote control state table

Details of Remote control state machine are shown in Table 49.

Table 49 – 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
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

#	Current state	Event /condition => actions	Next state
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
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 50.

Table 50 – 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.
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 51.

Table 51 – 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

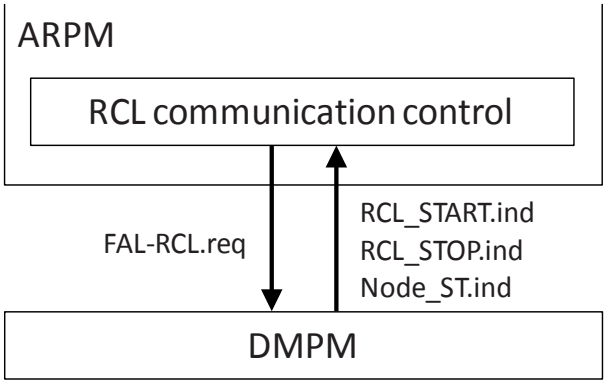


Figure 16 – The primitives for RCL communication control

8.1.4.2 RCL communication control state table

Table 52 shows the state table of the RCL communication control.

Table 52 – 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
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 53.

Table 53 – 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}

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

Function Name	Input	Output	Description

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
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 54.

Table 54 – 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.

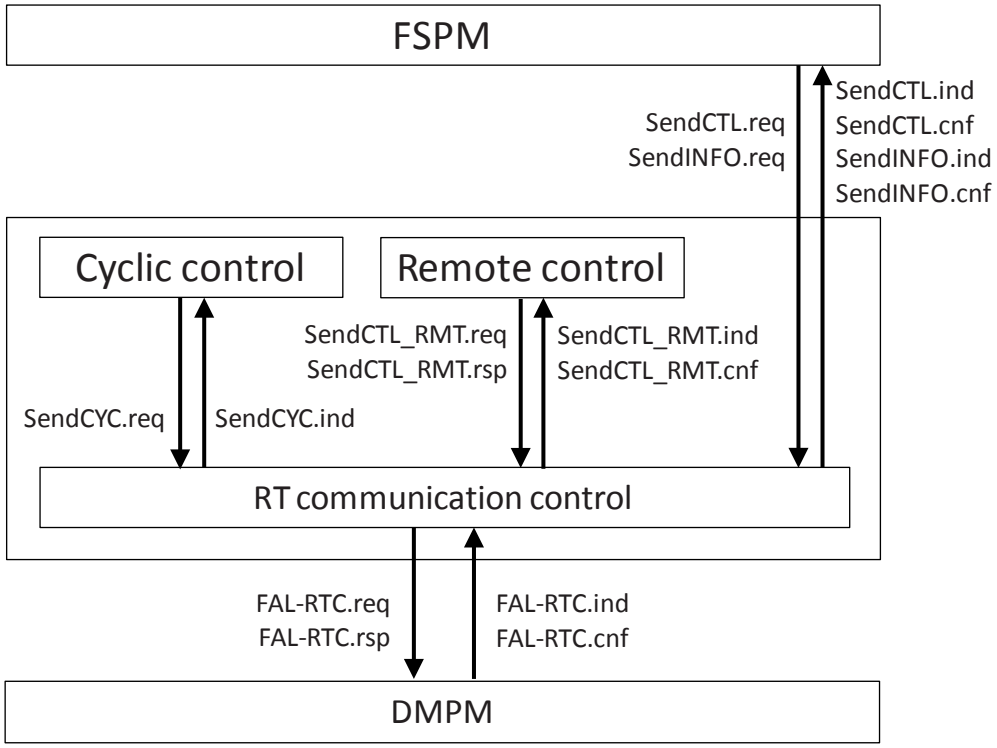


Figure 17 – The primitives for RT communication control

8.1.5.2 RT communication control state table

Table 55 shows the state table of the RT communication control.

Table 55 – 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
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

#	Current state	Event /condition => actions	Next state
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
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 56.

Table 56 – 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 57.

Table 57 – 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.

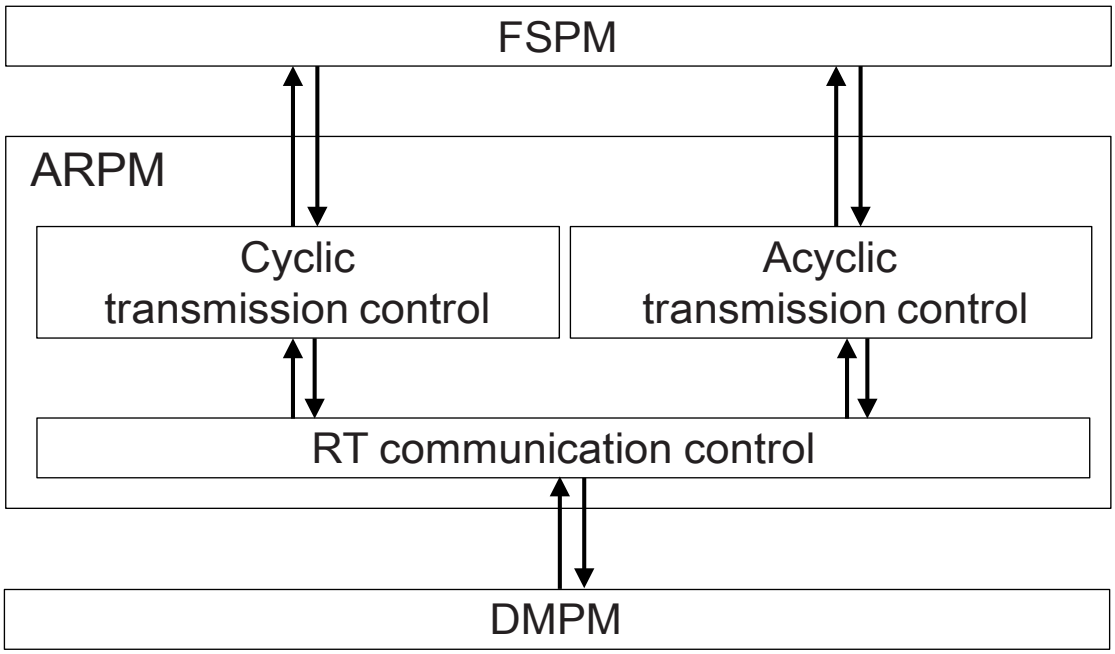


Figure 18 – Structure of ARPM type N

8.2.2 General control

8.2.2.1 Duplex LANs control

Type N protocol provides function to transmit messages using of 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 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 belongs 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 should recognize 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

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.

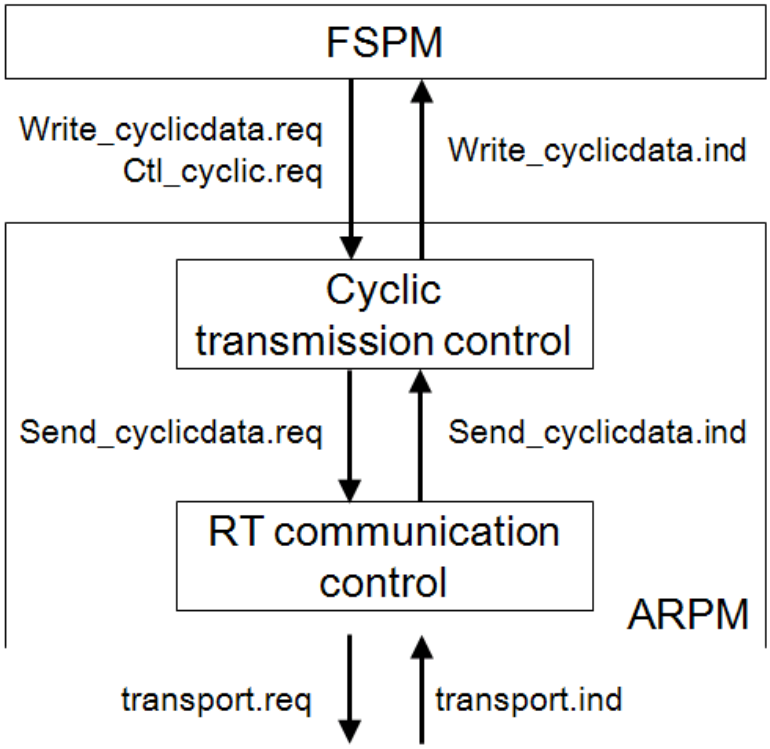


Figure 19 – Primitives of Cyclic transmission control

8.2.3.2 Cyclic transmission control state table

Details of Cyclic transmission control state machine are shown in Table 58.

Table 58 – 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

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 59.

Table 59 – 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.
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 60.

Table 60 – 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

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.

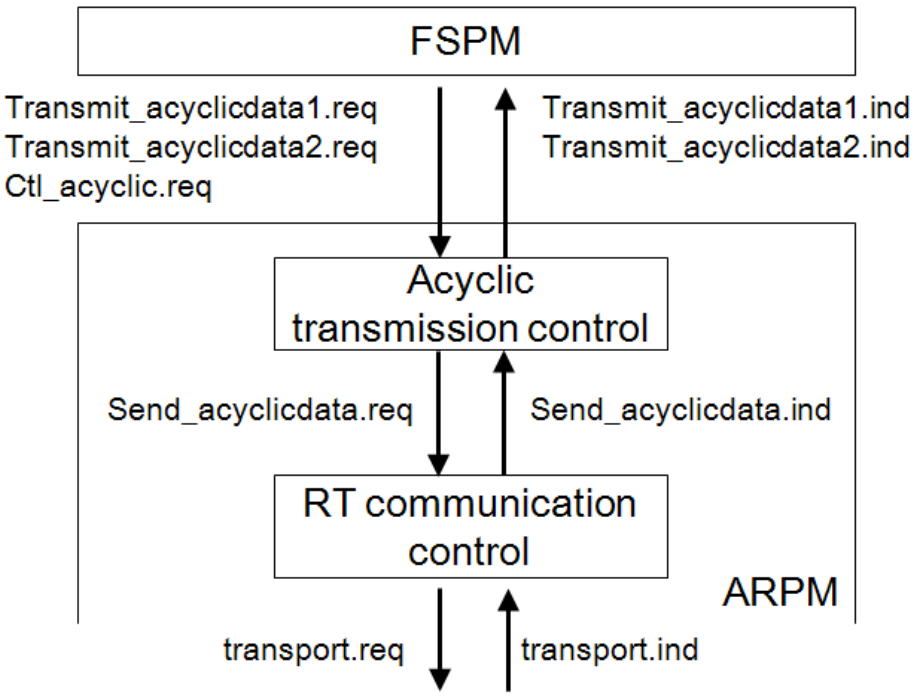


Figure 20 – Primitives of acyclic transmission control

8.2.4.2 Acyclic transmission control state table

Details of Acyclic transmission control state machine are shown in Table 61.

Table 61 – 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

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
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
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

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

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 62.

Table 62 – 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 63.

Table 63 – Acyclic transmission control variables

Name	Description
AliveMode	Mode of an alive-message (Alive 、Shutdown、Maintena
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 64 shows the state table of the RT communication control.

Table 64 – 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

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

#	Current state	Event /condition => actions	Next state

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

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

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

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

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

		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

#	Current state	Event /condition => actions	Next state

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 65.

Table 65 – 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 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 are 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 of that a RetransPduList is NULL or all PDUs have been extracted, sets False to a trans.

Name	Input	Output	Description
Get_transport_identifier	DstDfn, DstMg (Dst	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. (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_v_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)

Name	Input	Output	Description
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 of 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 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, returns True.

Name	Input	Output	Description
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
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)

Name	Input	Output	Description
Update_S_PSEQ	DstDfn, DstMgn (DstLnn)	UpdatedNumber	(1) To return a updated number after updating a sending sequence number (S-PEQ) of packet for multicast communication specified DstDfn and DstMgn. (2) To return a 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 66.

Table 66 – 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 te
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
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_MulticastWith Retrans	Receiving port number with function of retransmission

Name	Description
receive_acyclic_port_for_MulticastWith RetransControl	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 -2:tra 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 67 shows the relationships between ARPM and DL service.

Table 67 – 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

This sub clause 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 68 shows an assignment policy of communication ports.

Table 68 – 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 (Qos) for CP 20/2.

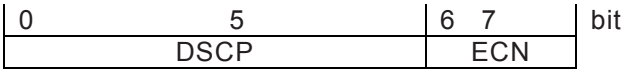
9.2.3.1 Overview

Quality of Service (QoS) is important for time sensitive applications. CP 20/2 defines requirements and recommendations for how CP 20/2 equipments use two standard QoS mechanisms: IEEE 802.1D/IEEE 802.1Q and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

Equipments 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 CP 20/2 traffic to DSCP and IEEE 802.1D/Q

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.



ECN: Explicit Congestion Notification

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.

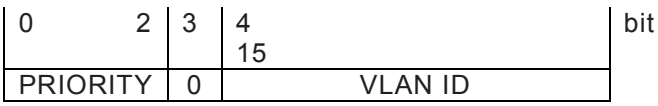


Figure 22 – IEEE 802.1Q tag frame format

Table 69 defines the default DSCP and IEEE 802.1D/Q priority mapping for CP20/2 traffic.

Table 69 – Default DSCP and IEEE 802.1D/Q priority mapping for CP20/2 traffic

CP20/2 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 CP 20/2 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 69, or the same as the values sent from the sending equipment.

When sending traffic with IEEE 802.1Q tagged frames, CP 20/2 equipments recommend to use the priority values specified in Table 69.

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

Bibliography

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

IEEE 802.1Q, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks; available at http://www.ieee.org*

ISO 15745-4:2003, *Industrial automation systems and integration – Open systems application integration framework – Part 4: Reference description for Ethernet-based control systems;* available at <http://www.iso.org>

IETF RFC 768, *User Datagram Protocol, available at http://www.ietf.org*

IETF RFC 793, *Transmission Control Protocol, available at http://www.ietf.org*

IETF RFC 1112, *Host Extensions for IP Multicasting, available at http://www.ietf.org*

IETF RFC 2460, *Internet Protocol, Version 6 (IPv6) Specification,* available at http://www.ietf.org

IETF RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,* available at http://www.ietf.org

IETF RFC 2597, *Assured Forwarding PHB Group, available at http://www.ietf.org*

IETF RFC 3140, *Per Hop Behavior Identification Codes, available at http://www.ietf.org*

IETF RFC 3246, *An Expedited Forwarding PHB (Per-Hop Behavior),* available at http://www.ietf.org

CONTENTS

INDUSTRIAL COMMUNICATION NETWORKS – PROFILES –

Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3

1 Scope

Chapter is identical in form and content for all CPFs – see IEC 61784-2:2014

2 Normative references

Additional references for CPF 20

IEC PAS 62953-3-25, *Industrial communication networks – Fieldbus specifications – Part 3-25: Data-link layer service definition – Type 25 elements*

IEC PAS 62953-4-25, *Industrial communication networks – Fieldbus specifications – Part 4-25: Data-link layer protocol specification – Type 25 elements*

IEC PAS 62953-5-25, *Industrial communication networks – Fieldbus specifications – Part 5-25: Application layer service definition – Type 25 elements*

IEC PAS 62953-6-25, *Industrial communication networks – Fieldbus specifications – Part 6-25: Application layer protocol specification – Type 25 elements*

3 Terms, definitions, symbols and abbreviations

3.1 Terms and definitions

Clause is identical in form and content for all CPFs – see IEC 61784-2:2014

3.2 Abbreviated terms and acronyms

Clause is identical in form and content for all CPFs – see IEC 61784-2:2014

3.3 Symbols

Additional sub-clause of the CPF 20

3.3.1 CPF 20 symbols

Symbol	Description	Unit
BW	Total bandwidth	%
BW_{NRTE}	Non-RTE bandwidth	%
BW_{RTE}	RTE bandwidth	%
CD	Cable segment delay	µs
CL	Cable length	km
DL_{CY}	Delay of RTE frame on sender node	µs
DL_{NCY}	Delay of Non-RTE frame on sender node	µs

Symbol	Description	Unit
DL_R	Delay of receiver node	µs
DL_S	Delay of sender node	µs
DT	Delivery time of RTE	µs
F_N	The number of frames in the RTE	-
M	The number of nodes sending RTE frame	-
MC_{RTE}	Common memory capacity	octet
N	The number of nodes between sending and receiving end-stations	-
PD	Cable propagation delay	µs
P_{OH}	Protocol overhead	µs
q	The number packets in the port transmit queue	µs
RD_{HD}	Delay of hardware receiving process	µs
SD_{FM}	Delay of sending process by firmware	µs
SD_{HD}	Delay of sending process by hardware	µs
SD_s	Sender stack delay	µs
SD_r	Receiver stack delay	µs
SL	Switch latency	µs
SPD	Switch processing delay	µs
ST_{RTE}	Communication cycle time	ms
T_{CI}	Transmit time of control and information communication packet	µs
T_{CN}	Transmit time of ring control communication packet	µs
T_{CY}	Transmit time of cyclic communication packet	µs
TR_{RTE}	Throughput RTE	µs
T_X	Transmit time of target packet	µs
T_{X_j}	Transmit time of packet j	µs

3.4 Conventions

Clause is identical in form and content for all CPFs – see IEC 61784-2:2014

4 Conformance to communication profiles

Chapter is identical in form and content for all CPFs – see IEC 61784-2:2014

5 RTE performance indicators

Chapter is identical in form and content for all CPFs – see IEC 61784-2:2014

6 Conformance tests

Chapter is identical in form and content for all CPFs – see IEC 61784-2:2014

7 Communication Profile Family 20 (ADS-net1) – RTE communication profiles

7.1 General overview

Communication Profile Family 20 defines profiles based on ISO/IEC 8802-3, IEC PAS 62953-3-25, IEC PAS 62953-4-25, IEC PAS 62953-5-25, and IEC PAS 62953-6-25 which specify the communication system protocols commonly known as ADS-net.

In this part of IEC 61784, the following communication profiles are specified for CPF 20:

– Profile 20/1

  A profile using ADS-net technology in a ring topology (ADS-net/μΣNETWORK-1000[1]),

– Profile 20/2

  A profile using ADS-net technology in a star / linear topology (ADS-net/NX[1]).

7.2 Profile 20/1

7.2.1 Physical layer

The physical layer of CP 20/1 is as specified in ISO/IEC 8802-3.

7.2.2 Data link layer

7.2.2.1 DLL service selection

DLL services are defined in IEC PAS 62953-3-25. Table 1 shows the subclauses included in this profile.

Table 1 – CP 20/1: DLL service selection

Clause	Header	Presence	Constraints
Whole document	Data link service definition (Type 25)	Yes	-

7.2.2.2 DLL protocol selection

DLL protocols are defined in IEC PAS 62953-4-25. Table 2 shows the subclauses included in this profile.

Table 2 – CP 20/1: DLL protocol selection

Clause	Header	Presence	Constraints
Whole document	Data link protocol specification (Type 25)	Yes	-

7.2.3 Application layer

7.2.3.1 AL service selection

Application Layer services are defined in IEC PAS 62953-5-25. Table 3 shows the subclauses included in this profile.

[1] ADS-net, ADS-net/μΣNETWORK-1000 and ADS-net/NX are used to describe the documents of IEC PAS 62953-3-25, IEC PAS 62953-4-25, IEC PAS 62953-5-25, IEC PAS 62953-6-25 and IEC 61784-2.

Table 3 – CP 20/1: AL service selection

Clause	Header	Presence	Constraints
1	Scope	Yes	-
2	Normative references	Yes	-
3	Terms, definitions, symbols and abbreviations	Partial	Used if needed
4	Concept	Yes	-
5	Data type ASE	Partial	Used if needed
6	Communication model specification	-	-
6.1	Communication model	Yes	-
6.2	ASE type S	Yes	-
6.3	ASE type N	No	-
6.4	AR type S	Yes	-
6.5	AR type N	No	-

7.2.3.2 AL protocol selection

Application Layer protocols are defined in IEC PAS 62953-6-25. Table 4 shows the subclauses included in this profile.

Table 4 – CP 20/1: AL protocol selection

Clause	Header	Presence	Constraints
1	Scope	Yes	-
2	Normative references	Yes	-
3	Terms, definitions, symbols and abbreviations	Partial	Used if needed
4	FAL syntax description	-	-
4.1	FALPDU type S abstract syntax	Yes	-
4.2	FALPDU type N abstract syntax	No	-
4.3	Data type assignments for type S	Yes	-
4.4	Data type assignments for type N	No	-
5	FAL transfer syntax	-	-
5.1	Encoding rules	Yes	-
5.2	FALPDU type S elements encoding	Yes	-
5.3	FALPDU type N elements encoding	No	-
6	Structure of the FAL protocol state machine	Yes	-
7	FAL service protocol machine (FSPM)	-	-
7.1	Overview	Yes	-
7.2	FSPM type S	Yes	-
7.3	FSPM type N	No	-
8	Application relationship protocol machine (ARPM)	-	-
8.1	ARPM type S	Yes	-
8.2	ARPM type N	No	-
9	DLL mapping protocol machine (DMPM)	-	-
9.1	DMPM type S	Yes	-
9.2	DMPM type N	No	-

7.2.4 Performance indicator selection

7.2.4.1 Performance indicator overview

Table 5 provides an overview of CP 20/1 performance indicators.

Table 5 – CP 20/1: performance indicator overview

Performance indicator	Applicable	Constraints
Delivery time	Yes	None
Number of end-stations	Yes	None
Basic network topology	Yes	Only ring topology is supported
Number of switches between end-stations	No	-
Throughput RTE	Yes	None
Non-RTE bandwidth	Yes	-
Time synchronization accuracy	No	-
Non-time-based synchronization accuracy	No	-
Redundancy recovery time	Yes	None

7.2.4.2 Performance indicator dependencies

7.2.4.2.1 Dependency matrix

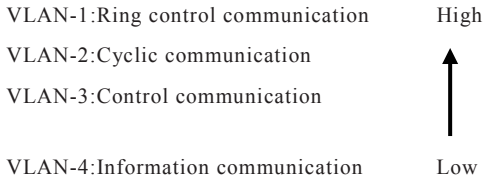
Table 6 shows the dependencies between performance indicators for CP 20/1.

Table 6 – CP 20/1: Performance indicator dependency matrix

Dependent PI	Influencing PI					
	Delivery time	Number of end-stations	Basic network topology	Throughput RTE	Non-RTE bandwidth	Redundancy recover time
Delivery time		Yes	No	Yes	Yes	No
Number of end-stations	Yes		No	Yes	Yes	No
Basic network topology	No	No		No	No	No
Throughput RTE	Yes	Yes	No		No	No
Non-RTE bandwidth	No	Yes	No	Yes		No
Redundancy recover time	No	No	No	No	No	

7.2.4.2.2 Delivery time

CP 20/1 traffic controls by using 4 VLAN as shown below.



The performance indicator “Delivery time” is related to the VLAN priority classes as shown above. The frame delivery time of each communication between any two end-nodes depends on multiple factors (e.g. frame buffering delay).

Cyclic communication time is calculated using formulae (1), (2), (3), (4), (5) and (6).

$$DT = DL_S + \sum_{i=1}^{M-2} DL_{CY} + \sum_{i=1}^{N-M} DL_{NCY} + DL_R + CD \quad (1)$$

$$DL_S = SD_{FM} + SD_{HD} + T_{CN} + T_{CI} + SPD + T_X \quad (2)$$

$$DL_{CY} = T_{CN} + T_{CY} + T_{CI} + SPD + T_X \quad (3)$$

$$DL_{NCY} = T_{CN} + T_{CI} + SPD + T_X \quad (4)$$

$$DL_R = T_{CN} + T_{CI} + SPD + T_X + RD_{HD} \quad (5)$$

$$CD = PD \cdot CL \quad (6)$$

where

DT is the delivery time of cyclic communication in microseconds (one frame/nodes sending);

DL_S is the delay of sender node (sending the cyclic communication packet);

DL_{CY} is the delay of cyclic frame sender node;

DL_{NCY} is the delay of non-cyclic frame sender node;

DL_R is the delay of receiver node (receiving the cyclic communication packet);

CD is the cable delay in microseconds;

SD_{FM} is the delay of sending process by firmware (firmware waiting time, depending on the selected hardware platform and the embededded software implementation);

SD_{HD} is the delay of sending process by hardware (depending on the selected hardware platform);

T_{CN} is the transmit time of ring control communication packet in microseconds;

T_{CI} is the transmit time of control and information communication packet in microseconds;

SPD is the switch processing delay in microseconds;

T_X is the transmit time of target packet in microseconds

T_{CY} is the transmit time of cyclic communication packet in microseconds;

RD_{HD} is the delay of hardware receiving process(depending on the selected hardware platform);

PD is the cable propagation delay (see parameter cable_delay in IEC 62411);

CL is the cable length in kilometers;

N is the number of nodes between sending and receiving end-nodes;

M is the number of nodes sending cyclic frame between sending and receiving end nodes.

7.2.4.2.3 Number of end-stations

The maximum number of end stations shall be up to 128.

7.2.4.2.4 Basic network topology

The network topology supported by this profile is of a loop (ring).

7.2.4.2.5 Throughput RTE

The throughput RTE is calculated according to formula (7).

$$TR_{RTE} = \frac{MC_{RTE}}{ST_{RTE}} \tag{7}$$

where

TR_{RTE} is the throughput RTE;

MC_{RTE} is the common memory capacity for RTE (cyclic communication);

ST_{RTE} is the communication cycle time by RTE packets.

7.2.4.2.6 Non-RTE bandwidth

The non-RTE bandwidth is calculated by using the Formulae (8) and (9).

$$BW_{NRTE} = BW - BW_{RTE}; \tag{8}$$

$$BW_{RTE} = \frac{(MC_{RTE} + F_N \cdot P_{OH}) \cdot 8}{ST_{RTE} \cdot 10^9} \tag{9}$$

where

BW_{NRTE} is the bandwidth used for non-RTE (control and information) communication in %;

BW is the total bandwidth in %, and the 100 % is 1 Gbps;

BW_{RTE} is the bandwidth used for the RTE communication in %;

F_N is the number of frames in the RTE communication;

P_{OH} is the protocol overhead.

7.2.4.2.7 Redundancy recovery time

CP 20/1 network (master less) is autonomously reconfigured with each node. Therefore CP 20/1 network configuration is simple and fast. The maximum time to become fully operational again from a failure, in case of a single permanent failure is 250 msec.

7.2.4.3 Consistent set of performance indicators

Table 7 shows the consistent set of the performance indicators for CP 20/1. The values in this table are maximum values of delivery time by calculated in 7.2.4.2.2.

Table 7 – CP 20/1: Consistent set of performance indicators

Performance indicator	Value	Constraints
Delivery time	1,7 ms	No failure, Total length = 10 km, 32 nodes
	9,1 ms	No failure, Total length = 500 km, 128 nodes
Number of end stations	32	Typical, Communication cycle time = 2 ms
	128	Max, Communication cycle time = 10 ms
Throughput RTE	16,4 M octets/s	Data=32 KB, 32 nodes
	26,1 M octets/s	Data=256 KB, 128 nodes
Non-RTE bandwidth	85,8 %	32 nodes
	77,2 %	128 nodes
Redundancy recovery time	250 ms	-

7.3 Profile 20/2

7.3.1 Physical layer

The physical layer of CP 20/2 is as specified in ISO/IEC 8802-3.

7.3.2 Data link layer

The data link layer of CP 20/2 is as specified in ISO/IEC 8802-3.

7.3.3 Application layer

7.3.3.1 AL service selection

Application Layer services are defined in IEC PAS 62953-5-25. Table 8 shows the subclauses included in this profile.

Table 8 – CP 20/2: AL service selection

Clause	Header	Presence	Constraints
1	Scope	Yes	-
2	Normative references	Yes	-
3	Terms, definitions, symbols and abbreviations	Partial	Used if needed
4	Concept	Yes	-
5	Data type ASE	Partial	Used if needed
6	Communication model specification	-	-
6.1	Communication model	Yes	-
6.2	ASE type S	No	-
6.3	ASE type N	Yes	-
6.4	AR type S	No	-
6.5	AR type N	Yes	-

7.3.3.2 AL protocol selection

Application Layer protocols are defined in IEC PAS 62953-6-25. Table 9 shows the subclauses included in this profile.

Table 9 – CP 20/2: AL protocol selection

Clause	Header	Presence	Constraints
1	Scope	Yes	-
2	Normative references	Yes	-
3	Terms, definitions, symbols and abbreviations	Partial	Used if needed
4	FAL syntax description	-	-
4.1	FALPDU type S abstract syntax	No	-
4.2	FALPDU type N abstract syntax	Yes	-
4.3	Data type assignments for type S	No	-
4.4	Data type assignments for type N	Yes	-
5	FAL transfer syntax	-	-
5.1	Encoding rules	Yes	-
5.2	FALPDU type S elements encoding	No	-
5.3	FALPDU type N elements encoding	Yes	-
6	Structure of the FAL protocol state machine	Yes	-
7	FAL service protocol machine (FSPM)	-	-
7.1	Overview	Yes	-
7.2	FSPM type S	No	-
7.3	FSPM type N	Yes	-
8	Application relationship protocol machine (ARPM)	-	-
8.1	ARPM type S	No	-
8.2	ARPM type N	Yes	-
9	DLL mapping protocol machine (DMPM)	-	-
9.1	DMPM type S	No	-
9.2	DMPM type N	Yes	-

7.3.4 Performance indicator selection

7.3.4.1 Performance indicator overview

Table 10 provides an overview of the CP 20/2 performance indicators.

Table 10 – CP 20/2: Performance indicator overview

Performance indicator	Applicable	Constraints
Delivery time	Yes	None
Number of end-stations	Yes	None
Basic network topology	Yes	None
Number of switches between end-stations	Yes	None
Throughput RTE	Yes	None
Non-RTE bandwidth	Yes	None
Time synchronization accuracy	No	-
Non-time-based synchronization accuracy	No	-
Redundancy recovery time	Yes	Duplex LANs

7.3.4.2 Performance indicator dependencies

7.3.4.2.1 Dependency matrix

Table 11 shows the dependencies between performance indicators for CP 20/2.

Table 11 – CP 20/2: Performance indicator dependency matrix

Dependent PI	Influencing PI						
	Delivery time	Number of end-stations	Basic network topology	Number of switches between end-stations	Throughput RTE	Non-RTE bandwidth	Redundancy recovery time
Delivery time		Yes	No	Yes	No	No	No
Number of end-stations	No		Yes	No	No	No	No
Basic network topology	No	No		No	No	No	No
Number of switches between end-stations	No	Yes	Yes		No	No	No
Throughput RTE	No	Yes	No	Yes		Yes	No
Non-RTE bandwidth	No	Yes	No	Yes	Yes		No
Redundancy recovery time	No	No	No	No	No	No	

7.3.4.2.2 Delivery time

CP 20/2 cyclic communication time is calculated using formulae (10), (11) and (12).

$$DT = SD_S + T_x + \sum_{i=1}^{N-1} CD_i + \sum_{k=1}^{N} SL_k + SD_r \tag{10}$$

$$CD_i = PD_i \times CL_i \tag{11}$$

$$SL_k = SPD_k + \sum_{j=1}^{q} T_{x\_j} + T_x \tag{12}$$

where

DT is the delivery time of cyclic communication in microseconds (one frame/nodes sending);

SD_s is the sender stack delay in microseconds (depending on the selected hardware platform and the embedded software implementation);

SD_r is the receiver stack delay in microseconds (depending on the selected hardware platform and the embedded software implementation);

CD is the cable segment delay in microseconds;

SL is the switch latency in microseconds;

N is the number of switches between sending and receiving end-stations;

PD is the cable propagation delay in nanoseconds per meter (depending on the characteristics of the selected cable);

CL is the cable segment length in meters;

SPD is the switch processing delay in microseconds (provided by the switch vendor instead of SL);

q is the number packets in the port transmit queue in front on of this packet;

T_{X_j} is the transmit time of packet j in microseconds;

T_X is the transmit time of the target packet in microseconds.

7.3.4.2.3 Number of end-stations

The maximum number of end stations shall be up to 65535.

7.3.4.2.4 Basic network topology

The network topology supported by this profile is a hierarchical star and linear or as a combination.

7.3.4.2.5 Number of switches between end-stations

The maximum number of switches between end stations shall be up to 256.

7.3.4.2.6 Throughput RTE

The throughput RTE is calculated by using the Formula (13).

$$TR_{RTE} = \frac{MC_{RTE}}{ST_{RTE}} \tag{13}$$

where

TR_{RTE} is the throughput RTE;

MC_{RTE} is the common memory capacity for RTE (cyclic communication);

ST_{RTE} is the communication cycle time by RTE packets.

7.3.4.2.7 Non-RTE bandwidth

The non-RTE bandwidth is calculated by using the Formulae (14) and (15).

$$BW_{NRTE} = BW - BW_{RTE}; \tag{14}$$

$$BW_{RTE} = \frac{(MC_{RTE} + F_N \cdot P_{OH}) \cdot 8}{ST_{RTE} \cdot 10^9} \tag{15}$$

where

BW_{NRTE} is the bandwidth used for non-RTE (control and information) communication in %;

BW is the total bandwidth in %, and the 100 % is 1 Gbps;

BW_{RTE} is the bandwidth used for the RTE communication in %;

F_N is the number of frames in the RTE communication;

P_{OH} is the protocol overhead.

7.3.4.2.8 Redundancy recovery time

The maximum time to become fully operational again from a failure in case of a single permanent failure is 0 msec. In CP 20/2 autonomous (master less) network, a sending node transmits the same data (message) into duplex LANs which consists of two physical independent network paths between end-nodes. Therefore, even if a single permanent network failure occurred, data transfer service can be continued.

7.3.4.3 Consistent set of performance indicators

Table 12 shows the consistent set of the performance indicators for CP 20/2. The values in this table are maximum values of delivery time by calculated in 7.3.4.2.2.

Table 12 – CP 20/2: Consistent set of performance indicators

Performance indicator	Value	Constraints
Delivery time	3,1 ms	No failure, 1 024 stations
	7,6 ms	No failure, 5 120 stations
Number of end stations	1024	Communication cycle time = 100 ms, 4 end stations are connected to one switch
	5120	Communication cycle time = 500 ms, 8 end stations are connected to one switch
Throughput RTE	10 M octets/s	Data: 1 MB, Communication cycle time: 100 ms
	2 M octets/s	Data: 1 MB, Communication cycle time: 500 ms
Non-RTE bandwidth	91 %	1 024 stations
	98 %	5 120 stations
Redundancy recovery time	0 ms	Sender: transmitting a message to Duplex LANs (Duplex LANs consists of two physical independent network paths between end-nodes.)

Annex A
(informative)

Performance Indicator calculation

A.1 CPF 20 – Performance indicator calculation

A.1.1 Profile 20/1

A.1.1.1 Delivery time

Using the formulae specified in 7.2.4.2.2, the maximum value of delivery time can be calculated based on the following assumptions:

– The maximum of RCL, control, and information communication frame length = 1 522 octets;
– The maximum cyclic communication frame length = 1 438 octets;
– The delay of sending process by firmware (SD_{FM}) = 50 µs;
– The delay of sending process by hardware (SD_{HD}) = 20 µs;
– The switch processing delay (SPD) = 3 µs;
– The delay of hardware receiving process (RD_{HD}) = 30 µs;
– The cable propagation delay (PD) = 5 ns/m;
– The cable length (CL) =10 km;
– The number of nodes between sending and receiving end-nodes (N) = 32;
– The number of nodes sending cyclic frame between sending and receiving end nodes (M) = 32;
– The transmit time of RCL frames (T_{CN}) = $\frac{(1522+20)\times 8}{10^9} = 12.3\,\mu s$,

  and the transmit time of the control and information frames (T_{CI}) are same as T_{CN};
– The transmit time of Cyclic frames (T_{CY}, T_X) = $\frac{(1438+20)\times 8}{10^9} = 11.7\,\mu s$;

The RTE communication cycle is 2 ms, and total volume of the cyclic data is 32 KB;

– No transmission errors.

$$DT = (50+20+12.3+12.3+3+11.7) + \sum_{i=1}^{30}(12.3+11.7+12.3+3+11.7) + (12.3+12.3+3+11.7)+30+50 \approx 1.7\text{ms}$$

A.1.1.2 RTE throughput

The maximum theoretical throughput RTE is calculated based on the following assumptions:

– the total of cyclic data size (MC_{RTE}) is 32 Koctets;
– the RTE communication cycle (ST_{RTE}) = 2 ms.

$$TR_{RTE} = \frac{MC_{RTE}}{ST_{RTE}} = \frac{32\cdot 1024}{2\cdot 10^{-3}} = 16.4\times 10^6 \text{ octets/s}$$

A.1.1.3 Non-RTE bandwidth

The maximum theoretical non-RTE bandwidth is calculated based on the following assumptions:

– the total of cyclic data size (MC_{RTE}) is 32 Koctets;
– all APDUs are of the same size, and APDU size = 1344 octets (maximum size);
– protocol overhead (P_{OH}) = 114 octets;
– the RTE communication cycle (ST_{RTE}) = 2 ms;
– Link data rate = 1 Gbps.

$$BW_{RTE} = \frac{(MC_{RTE} + F_N \cdot P_{OH}) \cdot 8}{ST_{RTE} \cdot 10^9} = \frac{\left(32 \cdot 1024 + \frac{32 \cdot 1024}{1344} \cdot 114\right) \cdot 8}{2 \cdot 10^{-3} \cdot 10^9} = 14.2\,\%$$

$$BW_{NRTE} = 100 - 14.2 = 85.8\,\%$$

A.1.2 Profile 20/2

A.1.2.1 Delivery time

Using the formulae specified in 7.3.4.2.2, the maximum value of delivery time can be calculated based on the following assumptions:

– SD_s = 50 µs;
– SD_r = 50 µs;
– APDU size = 1396 octets;
– Protocol overhead(P_{OH}) = 104 octets;
– Link data rate = 1 Gbit/s;
– All cable segments are of the same length and of the same cable type;
– Cable propagation delay (*PD*) = 5 ns/m;
– Cable length (*CL*) = 100 m;
– Switch processing delay (*SPD*) = 10 µs;
– Number of end stations = 1024
– Maximum number of switches between end-stations = 256;
– 4 end stations are connected to one switch
– No RTE packets in the transmit queue in front of this packet;
– No transmission errors.

$$CD_i = PD_i \times CL_i = 0.5\ \mu s$$

$$SL_k = SPD_k + T_x = 10 + \frac{(1396 + 104) \times 8}{10^9} = 11{,}2\ \mu s$$

$$DT = SD_S + T_x + \sum_{i=1}^{n-1} CD_i + \sum_{k=1}^{n} SL_k + SD_r = 50 + 12 + 255 \times 0.5 + 256 \times 11.2 + 50$$
$$= 3106.7 \mu s \approx 3.1\,ms$$

A.1.2.2 RTE throughput

The maximum theoretical throughput RTE is calculated based on the following assumptions:

– the total of cyclic data size (MC_{RTE}) is 1 Moctets;
– the RTE communication cycle (ST_{RTE}) = 100 ms.

$$TR_{RTE} = \frac{MC_{RTE}}{ST_{RTE}} = \frac{1}{100 \cdot 10^{-3}} = 10 \times 10^{6} \text{ octets/s}$$

A.1.2.3 Non-RTE bandwidth

The maximum theoretical non-RTE bandwidth is calculated based on the following assumptions:

– the total of cyclic data size (MC_{RTE}) is 1 Moctets;
– all APDUs are of the same size, and APDU size = 1396 octets (maximum size);
– protocol overhead (P_{OH}) = 104 octets;
– the RTE communication cycle (ST_{RTE}) = 100 ms;
– Link data rate = 1 Gbps.

– $$BW_{RTE} = \frac{\left(MC_{RTE} + F_N \cdot P_{OH}\right) \cdot 8}{ST_{RTE} \cdot 10^{9}} = \frac{\left(1 \cdot 1024 \cdot 1024 + \frac{1 \cdot 1024 \cdot 1024}{1396} \cdot 104\right) \cdot 8}{100 \cdot 10^{-3} \cdot 10^{9}} = 9\,\%$$

– $$BW_{NRTE} = 100 - 9 = 91\,\%$$

Bibliography

No additional bibliography of the CPF 20

This page deliberately left blank

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

bsi.

...making excellence a habit.™