INTERNATIONAL STANDARD

ISO 21806-6

First edition
2020-09

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 6: Data link layer

Véhicules routiers — Système de transport axé sur les médias —

Partie 6: Couche de liaison de données

Reference number
ISO 21806-6:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 22, *Road vehicles*, Subcommittee SC 31, *Data communication*.

A list of all parts in the ISO 21806 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The Media Oriented Systems Transport (MOST) communication technology was initially developed at the end of the 1990s in order to support complex audio applications in cars. The MOST Cooperation was founded in 1998 with the goal to develop and enable the technology for the automotive industry. Today, MOST[1)] enables the transport of high quality of service (QoS) audio and video together with packet data and real-time control to support modern automotive multimedia and similar applications. MOST is a function-oriented communication technology to network a variety of multimedia devices comprising one or more MOST nodes.

Figure 1 shows a MOST network example.

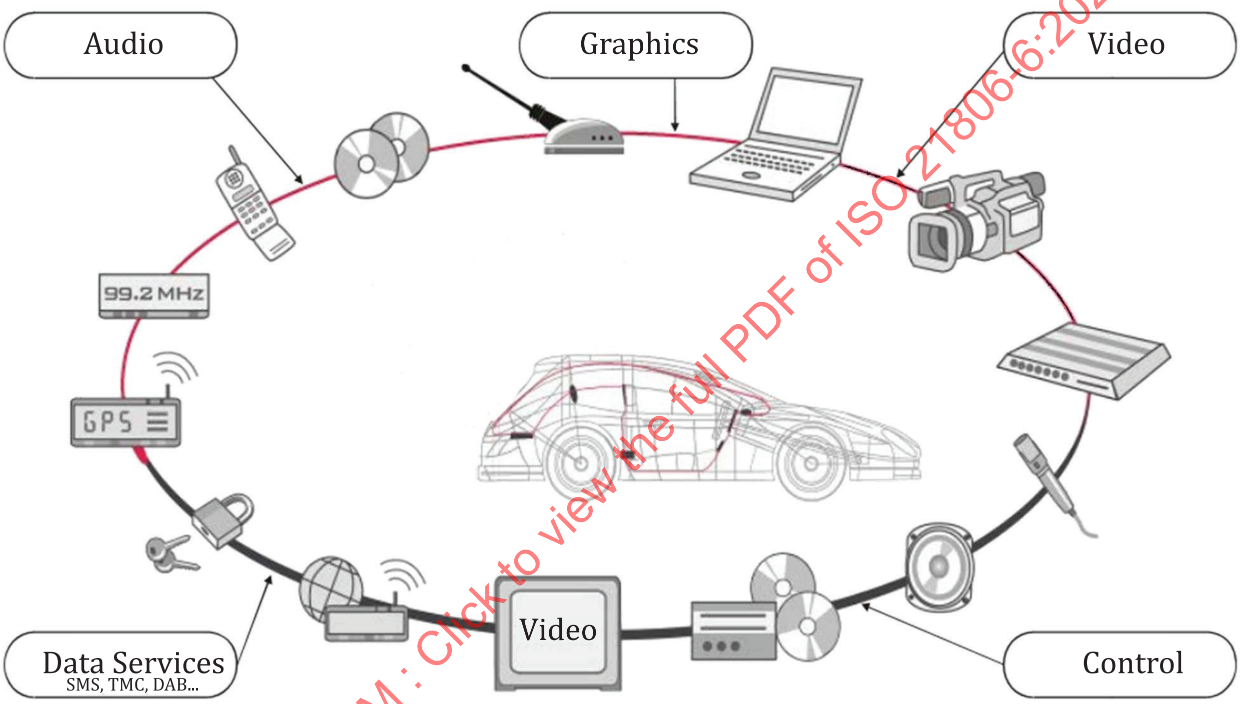


Figure 1 — MOST network example

The MOST communication technology provides

— synchronous and isochronous streaming,

— small overhead for administrative communication control,

— a functional and hierarchical system model,

— API standardization through a function block (FBlock) framework,

— free partitioning of functionality to real devices,

— service discovery and notification, and

— flexibly scalable automotive-ready Ethernet communication according to ISO/IEC/IEEE 8802-3[4].

MOST is a synchronous time-division-multiplexing (TDM) network that transports different data types on separate channels at low latency. MOST supports different bit rates and physical layers. The network clock is provided with a continuous data signal.

1) MOST® is the Registered Trademark of Microchip Technology Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO.

Within the synchronous base data signal, the content of multiple streaming connections and control data is transported. For streaming data connections, bandwidth is reserved to avoid interruptions, collisions, or delays in the transport of the data stream.

MOST specifies mechanisms for sending anisochronous, packet-based data in addition to control data and streaming data. The transmission of packet-based data is separated from the transmission of control data and streaming data. None of them interfere with each other.

A MOST network consists of devices that are connected to one common control channel and packet channel.

In summary, MOST is a network that has mechanisms to transport the various signals and data streams that occur in multimedia and infotainment systems.

The ISO standards maintenance portal (https://standards.iso.org/iso/) provides references to MOST specifications implemented in today's road vehicles because easy access via hyperlinks to these specifications is necessary. It references documents that are normative or informative for the MOST versions 4V0, 3V1, 3V0, and 2V5.

The ISO 21806 series has been established in order to specify requirements and recommendations for implementing the MOST communication technology into multimedia devices and to provide conformance test plans for implementing related test tools and test procedures.

To achieve this, the ISO 21806 series is based on the open systems interconnection (OSI) basic reference model in accordance with ISO/IEC 7498-1[1] and ISO/IEC 10731[2], which structures communication systems into seven layers as shown in Figure 2. Stream transmission applications use a direct stream data interface (transparent) to the data link layer.

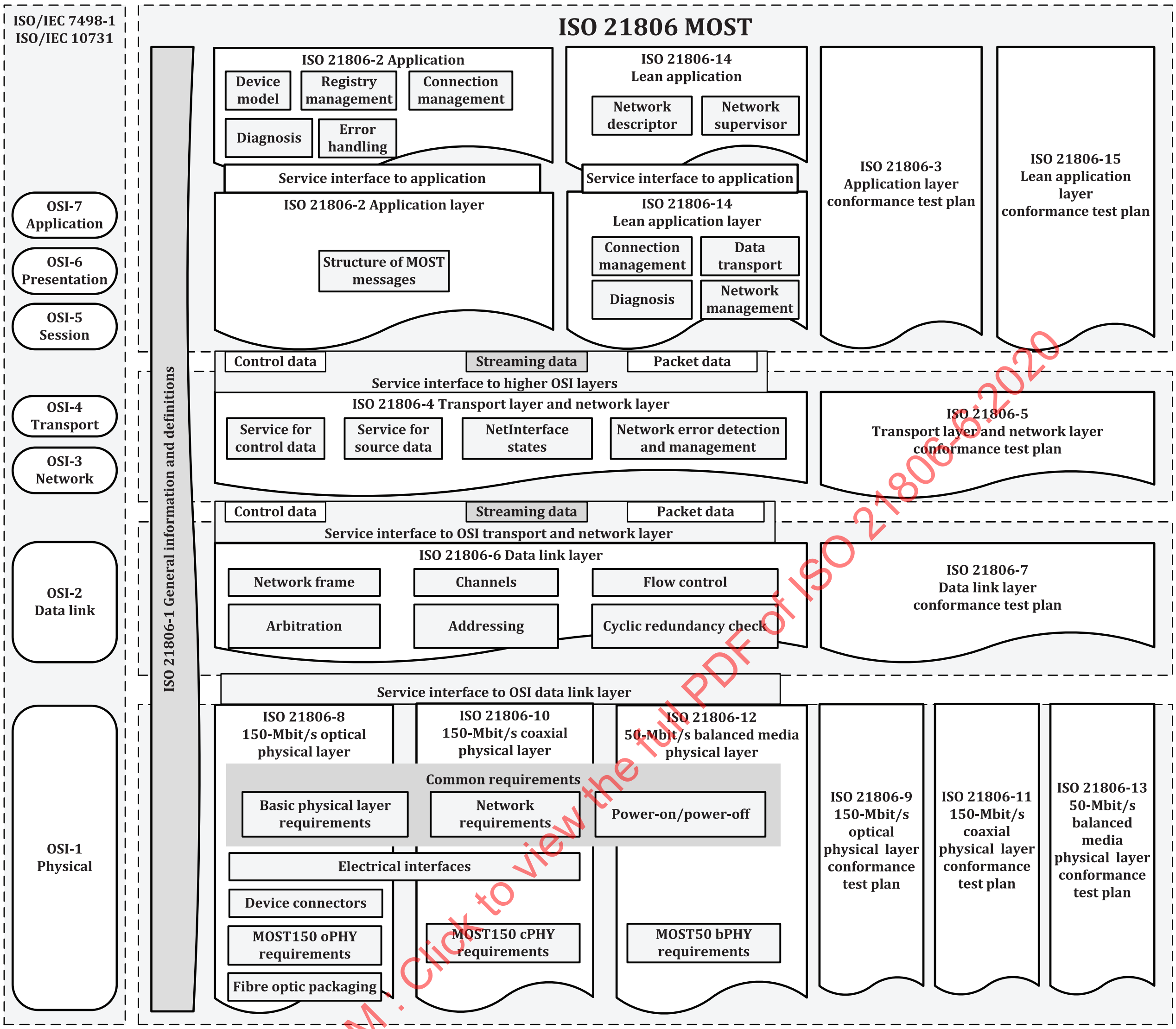


Figure 2 — The ISO 21806 series reference according to the OSI model

The International Organization for Standardization (ISO) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO. Information may be obtained from the patent database available at www.iso.org/patents.

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

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 6: Data link layer

1 Scope

This document specifies technical requirements related to the MOST data link layer functionality.

A MOST network is comprised of two or more nodes connected through a physical layer. The data link layer functionality is provided by each node. On each network, all nodes are synchronised and one node provides the system clock. This node is the TimingMaster, while all other nodes are TimingSlaves. The timing configuration of the node (TimingMaster or TimingSlave) determines the tasks that need to be performed on the data link layer.

The data link layer specifies the following subjects:

— the service interface to the network layer;

— the network frame, its areas and indicators;

— the different network channels;

— the different flow control mechanisms;

— the load-adaptive arbitration and the round-robin arbitration;

— the different addressing options;

— the different cyclic redundancy checks, their usage and the CRC acknowledge;

— the frame indicators.

2 Normative references

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

ISO 21806-1, *Road vehicles — Media Oriented Systems Transport (MOST) — Part 1: General information and definitions*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 21806-1 and the following apply.

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
diagnosis flag
flag that determines whether diagnosis is active

3.2
END
indicator for the end of a channel frame

3.3
Ethernet frame
frame according to ISO/IEC/IEEE 8802-3

3.4
isochronous frame
frame that consists of isochronous data

3.5
new allocation flag
flag that is set for newly allocated bytes

3.6
packet frame
frame that transports packet data with 16-bit addressing

3.7
PREAMBLE
indicator for the start of the network frame

3.8
protected system channel
channel that transports network status information

3.9
ring lock flag
flag that is set when the TimingMaster locks onto the incoming data stream

3.10
START
indicator for the start of a channel frame

3.11
static master flag
flag that determines whether the TimingMaster continuously sends network frames

3.12
synchronous frame
frame that is synchronous to the network clock and consists of unformatted data

3.13
timestamp channel
channel that is used to transport a CRC-protected timestamp

3.14
WAIT
indicator that is used for different purposes

4 Symbols and abbreviated terms

4.1 Symbols

<...>	range of bits, e.g. bit 7 to bit 0: <7:0>
...	and so on
---	empty cell/undefined
N_{PBC}	packet bandwidth control
$N_{\mathrm{PBC\_max}}$	maximum value of packet bandwidth control
$N_{\mathrm{SDBPFREST}}$	remaining number of source data bytes per frame
N_{TNABPF}	total number of administrative bytes per frame
N_{TNBPF}	total number of bytes per frame
N_{TNSDBPF}	total number of source data bytes per frame

4.2 Abbreviated terms

alloc	allocation
Arb[X]	8-bit arbitration value
ARBVAL	arbitration value
CACK	CRC acknowledge
CF	channel frame
CL	connection label
CPos	calculated position of the END indicator
CRC	cyclic redundancy check
DLL	data link layer
LSb	least significant bit
MOST	Media Oriented Systems Transport
MSb	most significant bit
NC	node counter (used in tables and figures)
NF	network frame (used in tables and figures)
NOFFAD	number of frames for auto deblock
PACK	pre-emptive acknowledge
PDU	protocol data unit
SDBPF	source data bytes per frame

SOAF start of allocation frame

TM TimingMaster (used in figures)

TS TimingSlave (used in figures)

5 Conventions

This document is based on OSI service conventions as specified in ISO/IEC 10731[2].

6 DLL — Service interface to upper layers

6.1 DLL — Overview

The DLL service interface defines the abstract interface to the OSI transport layer and network layer (see ISO 21806-4[3]).

Figure 3 shows the service interface to upper layers.

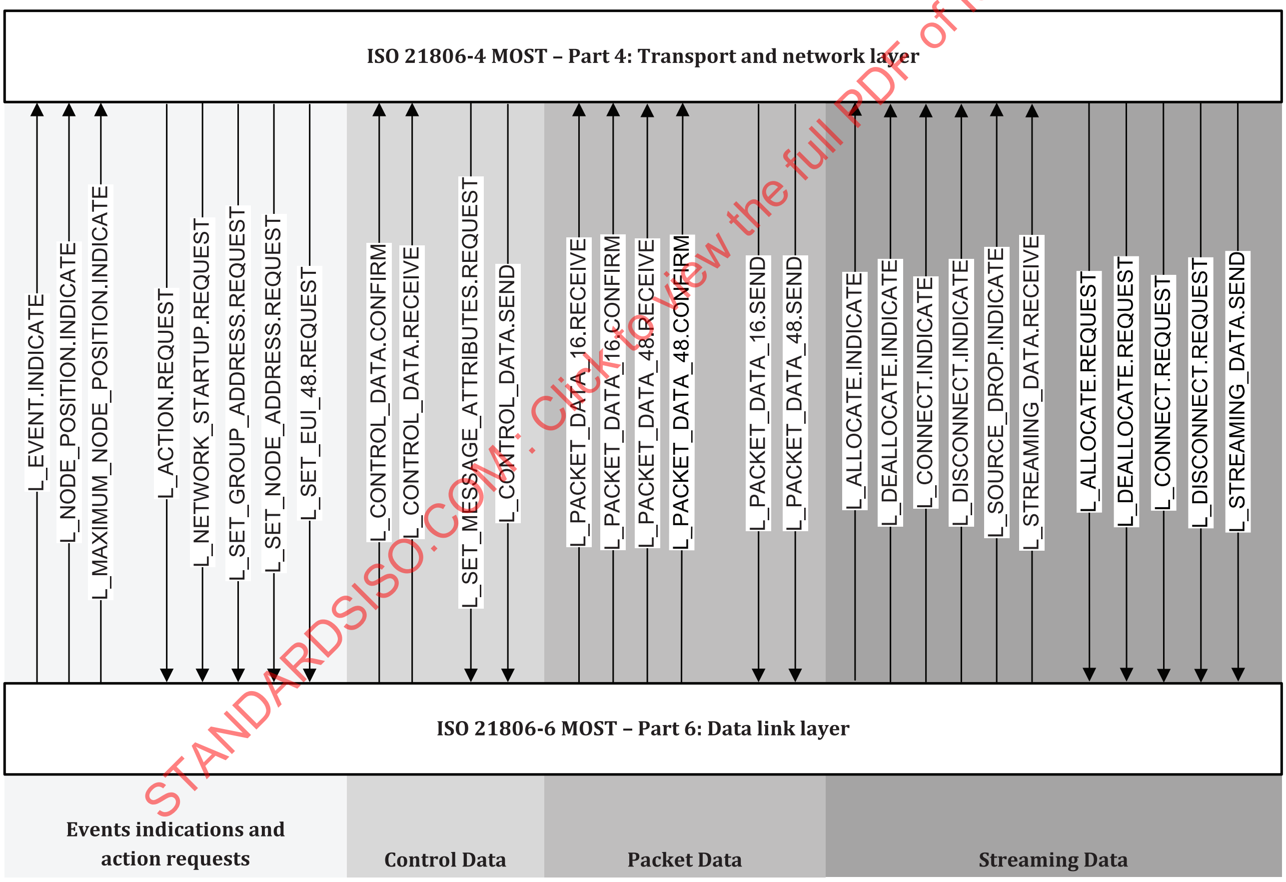


Figure 3 — Service interface to upper layers

6.2 DLL — Data type definitions

REQ	2.1 Service interface – DLL – Data type definitions
The data types shall be in accordance to:	

REQ	2.1 Service interface – DLL – Data type definitions
— Enum: 8-bit enumeration; — Unsigned Byte: 8-bit unsigned numeric value; — Unsigned Word: 16-bit unsigned numeric value; — EUI-48: 48-bit address value; — Byte Array: sequence of 8-bit aligned data.	

6.3 DLL — Parameters

6.3.1 DLL — Parameters – DLL to TL/NL

6.3.1.1 DLL — Overview

Table 1 provides an overview of the parameters that are used in the specified service interface and passed from DLL to TL/NL.

Table 1 — Parameters passed from DLL to TL/NL

Parameter	Data type	Description
Network_Event	Enum {Unlock, Lock, Lock_Flag, Network_Change_Event, Shutdown_Flag, MOST_Output_Off, Network_Activity}	An event that is reported to TL/NL.
Node_Position	Unsigned Byte	Node counter
Maximum_Position	Unsigned Byte	Visible nodes
Transmission_Status	Enum {Success, Buffer_Full, CRC_Error, Wrong_Target}	Transmission status that is reported back to the sender.

6.3.1.2 DLL — Network_Event

The Network_Event lists events that are used to notify TL/NL about changes in the DLL, which require no additional information.

REQ	2.2 Service interface – DLL – Parameters – DLL to TL/NL – DLL – Network_Event
The Network_Event parameter shall be of data type Enum and shall use the values defined in Table 2.	

Table 2 — Network_Event values

Enum value	Description
Unlock	Unlock event occurred
Lock	Lock reached
Lock_Flag	Lock flag detected
Network_Change_Event	The visible nodes value that is distributed by the TimingMaster has changed. Consequently, a network change event (NCE) is generated.
Shutdown_Flag	Shutdown flag detected
Network_Activity_End	Network activity ended
Network_Activity	Network activity detected

6.3.1.3 DLL — Node_Position

REQ	2.3 Service interface – DLL – Parameters – DLL to TL/NL – DLL – Node_Position
The `Node_Position` parameter shall be of data type `Unsigned Byte` and shall contain the node counter (NC) value of the respective node.	

6.3.1.4 DLL — Maximum_Position

REQ	2.4 Service interface – DLL – Parameters – DLL to TL/NL – DLL – Maximum_Position
The `Maximum_Position` parameter shall be of data type `Unsigned Byte` and shall contain the valid number of nodes visible in the MOST network.	

6.3.1.5 DLL — Transmission_Status

The `Transmission_Status` lists the possible outcomes of a transmission.

REQ	2.5 Service interface – DLL – Parameters – DLL to TL/NL – DLL – Transmission_Status
The `Transmission_Status` values shall be of data type `Enum` and shall contain the value specified according to Table 3.	

Table 3 — Transmission_Status values

Enum value	Description
`Success`	The node is ready to receive and a valid CRC is received.
`Buffer_Full`	The node is not ready to receive.
`CRC_Error`	Incorrect CRC received.
`Wrong_Target`	There is no such target. The PACK and CACK bytes remain unaltered.

6.3.2 DLL — Parameters – TL/NL to DLL

6.3.2.1 DLL — Overview

Table 4 provides an overview of the parameters that are used in the specified service interface and passed from TL/NL to DLL.

Table 4 — Parameters passed from TL/NL to DLL

Parameter	Data type	Description
`Network_Request`	`Enum { cmd_Set_Shutdown_Flag, cmd_Set_Lock_Flag, cmd_Clear_Lock_Flag, cmd_MOST_Output_Off, cmd_MOST_Output_On, cmd_Emergency_Shutdown, cmd_Open_Bypass, cmd_Static_Master, cmd_Ring_Lock }`	A request from TL/NL
`Network_Startup_Type`	`Enum { TimingMaster, TimingSlave }`	Determines how a node starts up.
`Number_Of_Retries`	`Unsigned Byte`	The number of low-level retries to perform on a control frame.
`Priority`	`Unsigned Byte`	The priority for a control frame

Table 4 *(continued)*

Parameter	Data type	Description
Group_Address	Unsigned Word	A group address
Node_Address	Unsigned Word	A logical node address
EUI_48	EUI-48	A 48-bit address
Bandwidth	Unsigned Word	Required bandwidth

6.3.2.2 DLL — Network_Request

The Network_Request lists actions that may be requested from the DLL. These requests require no additional information.

REQ	2.6 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Network_Request
The Network_Request values shall be of data type Enum and shall use the value specified according to Table 5.	

Table 5 — Network_Request values

Enum value	Description
cmd_Set_Shutdown_Flag	Setting the shutdown flag requested. By default, the shutdown flag is not set.
cmd_Set_Lock_Flag	Setting the lock flag requested. By default, the lock flag is cleared.
cmd_Clear_Lock_Flag	Clearing the lock flag requested. By default, the lock flag is cleared.
cmd_MOST_Output_Off	Switching off the MOST output requested. By default, the MOST output is off.
cmd_MOST_Output_On	Switching on the MOST output requested. By default, the MOST output is off.
cmd_Emergency_Shutdown	Emergency shutdown requested. By default, emergency shutdown is not active.
cmd_Open_Bypass	Opening the bypass requested. By default, the bypass is closed.
cmd_Static_Master	Static master mode requested. By default, the TimingMaster is in standard TimingMaster mode.
cmd_Ring_Lock	Ring lock requested. By default, the TimingMaster is not locked.

6.3.2.3 DLL — Network_Startup_Type

The Network_Startup_Type lists the two different startup types, which can be chosen for the node.

REQ	2.7 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Network_Startup_Type
The Network_Startup_Type values shall be of data type Enum and shall use the value specified according to Table 6.	

Table 6 — Network_Startup_Type values

Enum value	Description
TimingMaster	For startup, configure the node as TimingMaster.
TimingSlave	For startup, configure the node as TimingSlave.

6.3.2.4 DLL — Number_Of_Retries

REQ	2.8 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Number_Of_Retries
The Number_Of_Retries parameter shall be of data type Unsigned Byte and shall use the maximum permissible number of low-level retries for a particular transmission.	

6.3.2.5 DLL — Waiting_Period

REQ	2.9 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Waiting_Period
The Waiting_Period parameter shall be of data type Unsigned Byte and shall use the waiting period between low-level retries for a particular transmission.	

6.3.2.6 DLL — Priority

REQ	2.10 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Priority
The Priority parameter shall be of data type Unsigned Byte and shall use the priority for a particular transmission.	

6.3.2.7 DLL — Group_Address

REQ	2.11 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Group_Address
The Group_Address parameter shall be of data type Unsigned Word and shall use the group address of the node.	

6.3.2.8 DLL — Node_Address

REQ	2.12 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Node_Address
The Node_Address parameter shall be of data type Unsigned Word and shall use the logical node address.	

6.3.2.9 DLL — EUI_48

REQ	2.13 Service interface – DLL – Parameters – TL/NL to DLL – DLL – EUI_48
The EUI_48 parameter shall be of data type EUI-48 and shall use the MAC address of the node.	

6.3.2.10 DLL — Bandwidth

REQ	2.14 Service interface – DLL – Parameters – TL/NL to DLL – DLL – Bandwidth
The Bandwidth parameter shall be of data type Unsigned Word and shall use the value to request the allocation of the corresponding number of bytes in the network frame.	

6.3.3 DLL — Parameters – DLL to TL/NL and TL/NL to DLL

6.3.3.1 DLL — Overview

Table 7 provides an overview of the parameters that are used in the specified service interface for both directions.

Table 7 — Parameters used both ways

Parameter	Data type	Description
Media_Interface_ID	Unsigned Word	An identifier for media data output or input
Length	Unsigned Word	Length of the data field that is used in the same service interface.
Data	Byte Array	A data field, whose length is determined by the Length parameter.
Session_ID	Unsigned Word	A session identifier to correlate confirmations to send operations.
Target_Address	Unsigned Word	A 16-bit target address
Source_Address	Unsigned Word	A 16-bit source address

Table 7 *(continued)*

Parameter	Data type	Description
`Destination_MAC_Address`	`EUI-48`	A 48-bit target address
`Source_MAC_Address`	`EUI-48`	A 48-bit source address

6.3.3.2 DLL — Media_Interface_ID

REQ	2.15 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Media_Interface_ID
The `Media_Interface_ID` parameter shall be of data type `Unsigned Word` and shall be used to unambiguously identify a media interface as source or target of streaming data.	

6.3.3.3 DLL — Length

REQ	2.16 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Length
The `Length` parameter shall be of data type `Unsigned Word` and shall be used to provide the size of the data field that is used in the same service interface.	

6.3.3.4 DLL — Data

REQ	2.17 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Data
The `Data` parameter shall be of data type `Byte Array` and shall be used as a wrapper for payload that requires no interpretation in the context of the service interface that contains it.	

6.3.3.5 DLL — Session_ID

REQ	2.18 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Session_ID
The `Session_ID` parameter shall be of data type `Unsigned Word` and shall be used to unambiguously identify an instance of a control frame that is passed down. The session ID is used when determining the outcome of a transmission attempt.	

6.3.3.6 DLL — Target_Address

REQ	2.19 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Target_Address
The `Target_Address` parameter shall be of data type `Unsigned Word` and shall be used to fill the target address field for 16-bit addressing for control data and packet data.	

6.3.3.7 DLL — Source_Address

REQ	2.20 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Source_Address
The `Source_Address` parameter shall be of data type `Unsigned Word` and shall be used to fill the source address field for 16-bit addressing for control data and packet data.	

6.3.3.8 DLL — Destination_MAC_Address

REQ	2.21 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Destination_MAC_Address
The `Destination_MAC_Address` parameter shall be of data type `EUI-48` and shall be used to fill the destination address field for 48-bit addressing.	

6.3.3.9 DLL — Source_MAC_Address

REQ	2.22 Service interface – DLL – Parameters – DLL to TL/NL and TL/NL to DLL – DLL – Source_MAC_Address
The Source_MAC_Address parameter shall be of data type EUI-48 and shall be used to fill the source address field for 48-bit addressing.	

6.4 DLL — Event indications and action requests

6.4.1 DLL — L_EVENT.INDICATE

REQ	2.23 Service interface – DLL – Event indications and action requests – DLL – L_EVENT.INDICATE – Structure

The L_EVENT.INDICATE shall be passed from DLL to TL/NL to indicate that an event has occurred.
L_EVENT.INDICATE shall have the following structure:
L_EVENT.INDICATE{
    Network_Event
}	

6.4.2 DLL — L_NODE_POSITION.INDICATE

REQ	2.24 Service interface – DLL – Event indications and action requests – DLL – L_NODE_POSITION.INDICATE – Structure

The L_NODE_POSITION.INDICATE shall be passed from DLL to TL/NL to indicate the node position.
L_NODE_POSITION.INDICATE shall have the following structure:
L_NODE_POSITION.INDICATE{
    Node_Position
}	

6.4.3 DLL — L_MAXIMUM_NODE_POSITION.INDICATE

REQ	2.25 Service interface – DLL – Event indications and action requests – DLL – L_MAXIMUM_NODE_POSITION.INDICATE – Structure

L_MAXIMUM_NODE_POSITION.INDICATE shall be passed from DLL to TL/NL to indicate the maximum node position.
L_MAXIMUM_NODE_POSITION.INDICATE shall have the following structure:
L_MAXIMUM_NODE_POSITION.INDICATE{
    Maximum_Position
}	

6.4.4 DLL — L_ACTION.REQUEST

REQ	2.26 Service interface – DLL – Event indications and action requests – DLL – L_ACTION.REQUEST – Structure

L_ACTION.REQUEST shall be passed from TL/NL to DLL to trigger the execution of a request.
L_ACTION.REQUEST shall be specified according to the following structure:
L_ACTION.REQUEST {
    Network_Request
}	

6.4.5 DLL — L_NETWORK_STARTUP.REQUEST

REQ	2.27 Service interface – DLL – Event indications and action requests – DLL – L_NETWORK_STARTUP.REQUEST – Structure

`L_NETWORK_STARTUP.REQUEST` shall be passed from TL/NL to DLL to trigger the network start up.
`L_NETWORK_STARTUP.REQUEST` shall have the following structure:
`L_NETWORK_STARTUP.REQUEST{`
`    Network_Startup_Type`
`}`	

6.4.6 DLL — L_SET_GROUP_ADDRESS.REQUEST

REQ	2.28 Service interface – DLL – Event indications and action requests – Structure

`L_SET_GROUP_ADDRESS.REQUEST` shall be passed from TL/NL to DLL to set the group address of the node.
`L_SET_GROUP_ADDRESS.REQUEST` shall have the following structure:
`L_SET_GROUP_ADDRESS.REQUEST{`
`    Group_Address`
`}`	

6.4.7 DLL — L_SET_NODE_ADDRESS.REQUEST

REQ	2.29 Service interface – DLL – Event indications and action requests – DLL – L_SET_NODE_ADDRESS.REQUEST – Structure

`L_SET_NODE_ADDRESS.REQUEST` shall be passed from TL/NL to DLL to set the logical node address.
`L_SET_NODE_ADDRESS.REQUEST` shall have the following structure:
`L_SET_NODE_ADDRESS.REQUEST{`
`    Node_Address`
`}`	

6.4.8 DLL — L_SET_EUI_48.REQUEST

REQ	2.30 Service interface – DLL – Event indications and action requests – DLL – L_SET_EUI_48.REQUEST – Structure

`L_SET_EUI_48.REQUEST` shall be passed from TL/NL to DLL to set the EUI-48 of the node.
`L_SET_EUI_48.REQUEST` shall have the following structure:
`L_SET_EUI_48.REQUEST{`
`    EUI_48`
`}`	

6.4.9 DLL — L_SET_TRANSMISSION_ATTRIBUTES.REQUEST

REQ	2.31 Service interface – DLL – Event indications and action requests – DLL – L_SET_TRANSMISSION_ATTRIBUTES.REQUEST – Structure

L_SET_TRANSMISSION_ATTRIBUTES.REQUEST shall be passed from TL/NL to DLL to set the attributes for one transmission.

L_SET_TRANSMISSION_ATTRIBUTES.REQUEST shall have the following structure:

```
L_SET_TRANSMISSION_ATTRIBUTES.REQUEST{
    Session_ID
    Number_Of_Retries
    Waiting_Period
    Priority
}
```

6.5 DLL — Control Data

6.5.1 DLL — L_CONTROL_DATA.RECEIVE

REQ	2.32 Service interface – DLL – L_CONTROL_DATA.RECEIVE – Structure

L_CONTROL_DATA.RECEIVE shall be passed from DLL to TL/NL to receive control data.

L_CONTROL_DATA.RECEIVE shall have the following structure:

```
L_CONTROL_DATA.RECEIVE{
    Length
    Target_Address
    Source_Address
    Data
}
```

6.5.2 DLL — L_CONTROL_DATA.CONFIRM

REQ	2.33 Service interface – DLL – L_CONTROL_DATA.CONFIRM – Structure

L_CONTROL_DATA.CONFIRM shall be passed from DLL to TL/NL to provide the transmission status for the control data sent.

L_CONTROL_DATA.CONFIRM shall have the following structure:

```
L_CONTROL_DATA.CONFIRM{
    Session_ID
    Transmission_Status
}
```

NOTE It is created after successful transmission or after all low-level retries have expired.

6.5.3 DLL — L_CONTROL_DATA.SEND

REQ	2.34 Service interface – DLL – L_CONTROL_DATA.SEND – Structure

`L_CONTROL_DATA.SEND` shall be passed from TL/NL to DLL to send control data.
`L_CONTROL_DATA.SEND` shall have the following structure:
`L_CONTROL_DATA.SEND{`
`Length`
`Session_ID`
`Target_Address`
`Data`
`}`	

6.6 DLL — Packet data

6.6.1 DLL — 16-bit addressing

6.6.1.1 DLL — L_PACKET_DATA_16.RECEIVE

REQ	2.35 Service interface – DLL – L_PACKET_DATA_16.RECEIVE – Structure

`L_PACKET_DATA_16.RECEIVE` shall be passed from DLL to TL/NL to receive packet data with 16-bit addressing.
`L_PACKET_DATA_16.RECEIVE` shall have the following structure:
`L_PACKET_DATA_16.RECEIVE{`
`Length`
`Target_Address`
`Source_Address`
`Data`
`}`	

6.6.1.2 DLL — L_PACKET_DATA_16.CONFIRM

REQ	2.36 Service interface – DLL – L_PACKET_DATA_16.CONFIRM – Structure

`L_PACKET_DATA_16.CONFIRM` shall be passed from DLL to TL/NL to provide the transmission status for packet data that is sent with 16-bit addressing.
`L_PACKET_DATA_16.CONFIRM` shall have the following structure:
`L_PACKET_DATA_16.CONFIRM{`
`Session_ID`
`Transmission_Status`
`}`	

6.6.1.3 DLL — L_PACKET_DATA_16.SEND

REQ	2.37 Service interface – DLL – L_PACKET_DATA_16.SEND – Structure

L_PACKET_DATA_16.SEND shall be passed from TL/NL to DLL to send packet data with 16-bit addressing.
L_PACKET_DATA_16.SEND shall have the following structure:
L_PACKET_DATA_16.SEND{
    Length
    Session_ID
    Target_Address
    Data
}	

6.6.2 DLL — 48-bit addressing

6.6.2.1 DLL — L_PACKET_DATA_48.RECEIVE

REQ	2.38 Service interface – DLL – L_PACKET_DATA_48.RECEIVE – Structure

L_PACKET_DATA_48.RECEIVE shall be passed from DLL to TL/NL to receive packet data with 48-bit addressing.
L_PACKET_DATA_48.RECEIVE shall have the following structure:
L_PACKET_DATA_48.RECEIVE{
    Length
    Destination_MAC_Address
    Source_MAC_Address
    Data
}	

6.6.2.2 DLL — L_PACKET_DATA_48.CONFIRM

REQ	2.39 Service interface – DLL – L_PACKET_DATA_48.CONFIRM – Structure

L_PACKET_DATA_48.CONFIRM shall be passed from DLL to TL/NL to provide the transmission status for packet data that is sent with 48-bit addressing.
L_PACKET_DATA_48.CONFIRM shall have the following structure:
L_PACKET_DATA_48.CONFIRM{
    Session_ID
    Transmission_Status
}	

6.6.2.3 DLL — L_PACKET_DATA_48.SEND

REQ	2.40 Service interface – DLL – L_PACKET_DATA_48.SEND – Structure

L_PACKET_DATA_48.SEND shall be passed from TL/NL to DLL to send packet data with 48-bit addressing.
L_PACKET_DATA_48.SEND shall have the following structure:
L_PACKET_DATA_48.SEND{
    Length
    Session_ID
    Destination_MAC_Address
    Data
}	

6.7 DLL — Streaming data

6.7.1 DLL — L_ALLOCATE.INDICATE

REQ	2.41 Service interface – DLL – Packet data – DLL – L_ALLOCATE.INDICATE – Structure

L_ALLOCATE.INDICATE shall have the following structure:
L_ALLOCATE.INDICATE{
    Session_ID
    Media_Interface_ID
}	

REQ	2.42 Service interface – DLL – Packet data –DLL – L_ALLOCATE.INDICATE – Purpose
L_ALLOCATE.INDICATE shall be passed from DLL to TL/NL to indicate the allocation of bandwidth for source data. The Session_ID parameter shall be set to the value provided in L_ALLOCATE.REQUEST.	

6.7.2 DLL — L_DEALLOCATE.INDICATE

REQ	2.43 Service interface – DLL – Packet data – DLL – L_DEALLOCATE.INDICATE – Structure

L_DEALLOCATE.INDICATE shall have the following structure:
L_DEALLOCATE.INDICATE{
    Session_ID
    Media_Interface_ID
}	

REQ	2.44 Service interface – DLL – Packet data – DLL – L_DEALLOCATE.INDICATE – Purpose
L_DEALLOCATE.INDICATE shall be passed from DLL to TL/NL to indicate the completion of deallocation of bandwidth. If L_DEALLOCATE.INDICATE is not related to L_DEALLOCATE.REQUEST, the Session_ID parameter shall be set to $FFFF_{16}$. If L_DEALLOCATE.INDICATE is caused by L_DEALLOCATE.REQUEST, the Session_ID parameter shall be set to the value provided in L_DEALLOCATE.REQUEST.	

6.7.3 DLL — L_CONNECT.INDICATE

REQ	2.45 Service interface – DLL – Packet data – DLL – L_CONNECT.INDICATE – Structure

L_CONNECT.INDICATE shall have the following structure:
L_CONNECT.INDICATE{
    Session_ID
    Media_Interface_ID
}	

REQ	2.46 Service interface – DLL – Streaming data – DLL – L_CONNECT.INDICATE – Purpose
L_CONNECT.INDICATE shall be passed from DLL to TL/NL to indicate that a connection is established between a sink and a data stream provided by a source. The Session_ID parameter shall be set to the value provided in L_CONNECT.REQUEST.	

6.7.4 DLL — L_DISCONNECT.INDICATE

REQ	2.47 Service interface – DLL – Streaming data – DLL – L_DISCONNECT.INDICATE – Structure

L_DISCONNECT.INDICATE shall have the following structure:
L_DISCONNECT.INDICATE{
    Session_ID
    Media_Interface_ID
}	

REQ	2.48 Service interface – DLL – Streaming data – DLL – L_DISCONNECT.INDICATE – Purpose
L_DISCONNECT.INDICATE shall be passed from DLL to TL/NL to indicate that a sink is disconnected from a data stream provided by a source. If L_DISCONNECT.INDICATE is not related to L_DISCONNECT.REQUEST, the Session_ID parameter shall be set to $FFFF_{16}$. If L_DISCONNECT.INDICATE is caused by L_DISCONNECT.REQUEST, the Session_ID parameter shall be set to the value provided in L_DISCONNECT.REQUEST.	

6.7.5 DLL — L_SOURCE_DROP.INDICATE

REQ	2.49 Service interface – DLL – Streaming data – DLL – L_SOURCE_DROP.INDICATE – Structure

L_SOURCE_DROP.INDICATE shall have the following structure:
L_SOURCE_DROP.INDICATE{
    Media_Interface_ID
}	

REQ	2.50 Service interface – DLL – Streaming data – DLL – L_SOURCE_DROP.INDICATE – Purpose
L_SOURCE_DROP.INDICATE shall be passed from DLL to TL/NL to indicate a source malfunction.	

6.7.6 DLL — L_STREAMING_DATA.RECEIVE

REQ	2.51 Service interface – DLL – Streaming data – DLL – L_STREAMING_DATA.RECEIVE – Structure

L_STREAMING_DATA.RECEIVE shall have the following structure:
L_STREAMING_DATA.RECEIVE{
Length
Media_Interface_ID
Data
}	

REQ	2.52 Service interface – DLL – Streaming data – DLL – L_STREAMING_DATA.RECEIVE – Purpose
L_STREAMING_DATA.RECEIVE shall be passed from DLL to TL/NL to receive streaming data.	

6.7.7 DLL — L_ALLOCATE.REQUEST

REQ	2.53 Service interface – DLL – Streaming data – DLL – L_ALLOCATE.REQUEST – Structure

L_ALLOCATE.REQUEST shall have the following structure:
L_ALLOCATE.REQUEST{
Session_ID
Bandwidth
}	

REQ	2.54 Service interface – DLL – Streaming data – DLL – L_ALLOCATE.REQUEST – Purpose
L_ALLOCATE.REQUEST shall be passed from TL/NL to DLL to allocate bandwidth for source data.	

6.7.8 DLL — L_DEALLOCATE.REQUEST

REQ	2.55 Service interface – DLL – Streaming data – DLL – L_DEALLOCATE.REQUEST – Structure

L_DEALLOCATE.REQUEST shall have the following structure:
L_DEALLOCATE.REQUEST{
Session_ID
Media_Interface_ID
}	

REQ	2.56 Service interface – DLL – Streaming data – DLL – L_DEALLOCATE.REQUEST – Purpose
L_DEALLOCATE.REQUEST shall be passed from TL/NL to DLL to deallocate bandwidth. The Session_ID parameter shall not be set to $FFFF_{16}$.	

6.7.9 DLL — L_CONNECT.REQUEST

REQ	2.57 Service interface – DLL – Streaming data – DLL – L_CONNECT.REQUEST – Structure

L_CONNECT.REQUEST shall have the following structure:
L_CONNECT.REQUEST{
Session_ID
Media_Interface_ID
}	

REQ	2.58 Service interface – DLL – Streaming data – DLL – L_CONNECT.REQUEST – Purpose
L_CONNECT.REQUEST shall be passed from TL/NL to DLL to connect a sink to a data stream provided by a source.	

6.7.10 DLL — L_DISCONNECT.REQUEST

REQ	2.59 Service interface – DLL – Streaming data – DLL – L_DISCONNECT.REQUEST – Structure

L_DISCONNECT.REQUEST shall have the following structure:
L_DISCONNECT.REQUEST{
Session_ID
Media_Interface_ID
}	

REQ	2.60 Service interface – DLL – Streaming data – DLL – L_DISCONNECT.REQUEST – Purpose
L_DISCONNECT.REQUEST shall be passed from TL/NL to DLL to disconnect a sink from a data stream provided by a source. Session_ID parameter shall not be set to $FFFF_{16}$.	

6.7.11 DLL — L_STREAMING_DATA.SEND

REQ	2.61 Service interface – DLL – Streaming data – DLL – L_STREAMING_DATA.SEND – Structure

L_STREAMING_DATA.SEND shall have the following structure:
L_STREAMING_DATA.SEND{
Length
Media_Interface_ID
Data}
}	

REQ	2.62 Service interface – DLL – Streaming data – DLL – L_STREAMING_DATA.SEND – Purpose
L_STREAMING_DATA.SEND shall be passed from TL/NL to DLL to send streaming data.	

7 DLL — Network frame

7.1 DLL — General

Depending on the bit rate, the length of the network frame, that is, the total number of bytes per frame (N_{TNBPF}), varies.

The MOST network configurations shall be in accordance with Annex A.

Figure 4 shows the network frame.

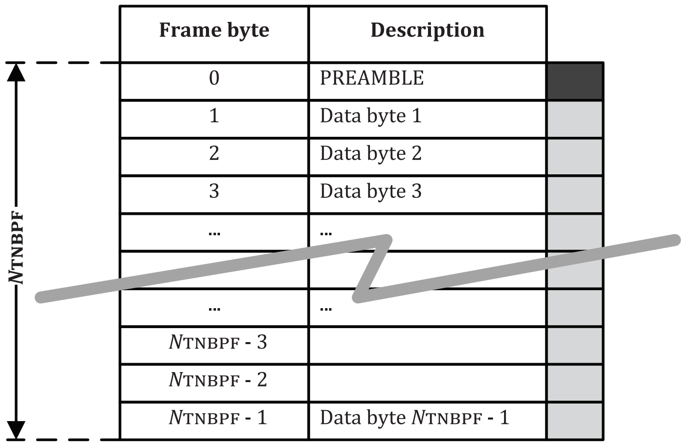


Figure 4 — Network frame

REQ	2.63 DLL – Network frame – PREAMBLE
A network frame shall consist of a network indicator of type PREAMBLE, indicating the start of the network frame, followed by a number of data bytes, see Figure 4.	

REQ	2.64 DLL – Network frame – General bit order
A node shall transmit the data bytes in the network frame except the node counter with the MSb first.	

REQ	2.65 DLL – Network frame – Node counter bit order
A node shall transmit the node counter with the LSb first.	

Different channels can transport data of different data formats. Some channels support network administration, while others transport source data (application-related data).

The total number of source data bytes per frame is called N_{TNBPF}.

The total number of administrative bytes per frame is called N_{TNABPF}.

Formula (1) specifies the N_{TNBPF}.

$$N_{\text{TNBPF}} = 1 + N_{\text{TNABPF}} + N_{\text{TNSDBPF}} \tag{1}$$

Figure 5 shows the network frame structure. The PREAMBLE is followed by the administrative area and the source data area.

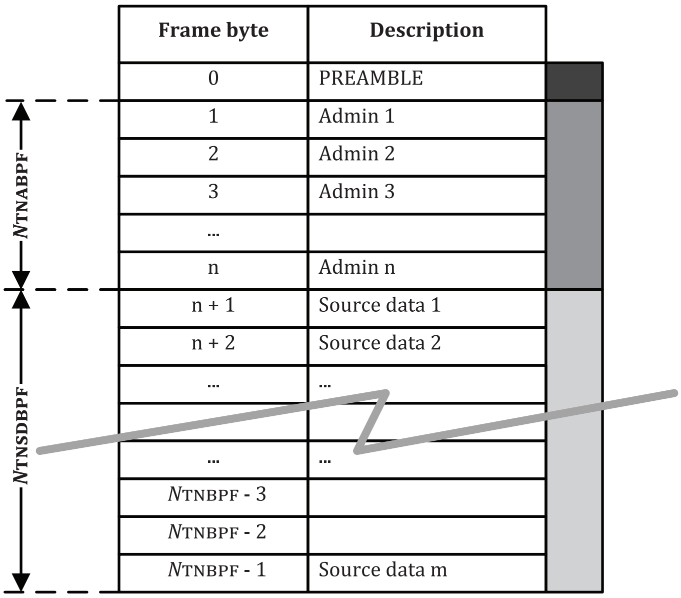


Figure 5 — Network frame — Structure

7.2 DLL — Administrative area

Channels in the administrative area transport data that serve network administration purposes and application-related data in case of the control channel.

The administrative area may consist of the channels as specified in Table 8.

Table 8 — Channels in the administrative area

Channel name	Comment
Allocation channel	See 8.1.
Control channel	See 8.2.
Protected system channel	See 8.3.
Timestamp channel	See 8.4.

7.3 DLL — Source data area

The channels in the source data area transport different kinds of payload data, such as streams of audio data or packetized data. The number of channels on a MOST network is variable; some channel types might not be present.

The following channels are specified for the source data area:

— packet channel,

— synchronous channel, and

— isochronous channel.

Table 9 specifies the channels in the source data area.

Table 9 — Channels in the source data area

Channel name	Comment
Packet channel	See 8.5.
Synchronous channel	See 8.6.
Isochronous channel	See 8.7.

7.4 DLL — Indicators

Indicators are located in the bit stream of the network frame. They allow the transmission of non-data information in the MOST network bit stream for advanced flow control. Indicators are transmitted on the physical layer in a unique way. Besides the PREAMBLE indicator that designates the first byte of a network frame, the START, END and WAIT indicators are used for, for example identifying start or end of a channel frame (CF) within the network frame. A channel frame is a frame that is transported on a channel within the network frame.

The indicators shall be generated as specified in Annex B.

8 DLL — Channels

8.1 DLL — Allocation channel

8.1.1 DLL — General

The allocation channel is used to allocate or de-allocate bandwidth in the source data area.

Whenever a node intends to make use of network frame bytes in the source data area, it allocates those bytes in an allocation request. An arbitration mechanism allows getting access to bytes that are not yet in use.

8.1.2 DLL — Allocation frame structure

The allocation channel consists of two consecutive bytes of the network frame. These two bytes together transport the allocation word.

Table 10 specifies the allocation word general format.

Table 10 — Allocation word general format

Bit	Content
<15:0>	The allocation word has different formats, depending on the current state of the allocation process. In its general format, it acts as a plain 16-bit value.

REQ	**2.66 DLL – Allocation channel – Allocation words**
Each byte of the network frame shall be represented by two subsequent allocation words in the allocation channel.	
NOTE Thus, the complete network frame is processed in an allocation frame that takes N_{TNBPF} × 2 network frames (2 network frames for each of the N_{TNBPF} network frame bytes).	

The start of the allocation frame is indicated by an allocation word in SOAF (start of allocation frame) format.

Figure 6 shows the allocation words of subsequent network frames forming the allocation frame.

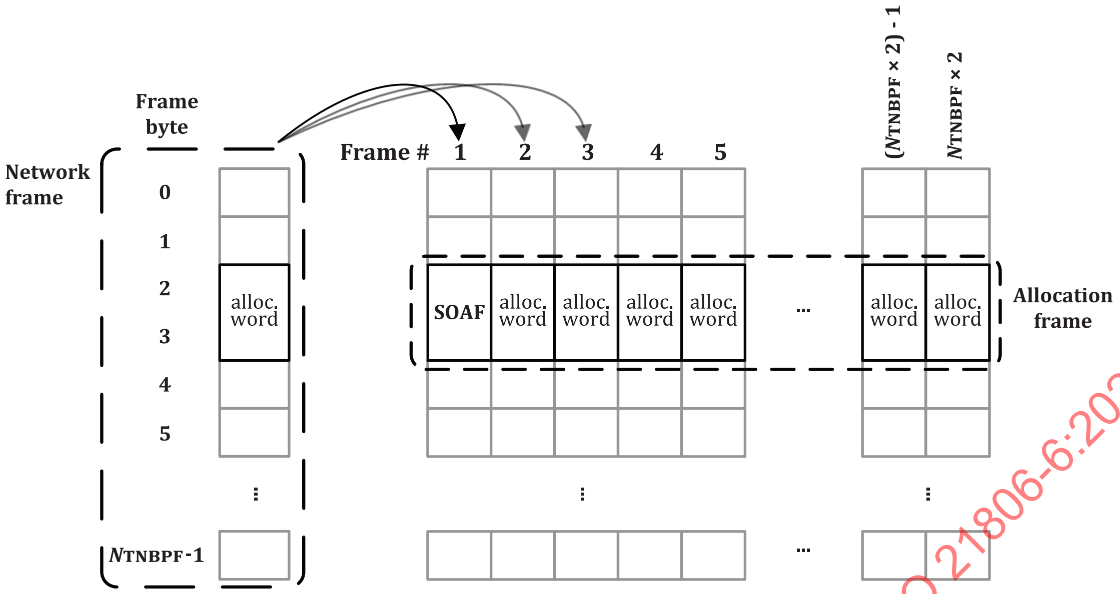


Figure 6 — Allocation words of subsequent network frames forming the allocation frame

REQ	2.67 DLL – Allocation channel – SOAF format
The structure of the allocation word - SOAF format shall be in accordance with Table 11.	

Table 11 — Allocation word - SOAF format

Bit	Content
<15:8>	START indicator
<7:0>	Reserved

Figure 7 shows the allocation words representing bytes of network frame.

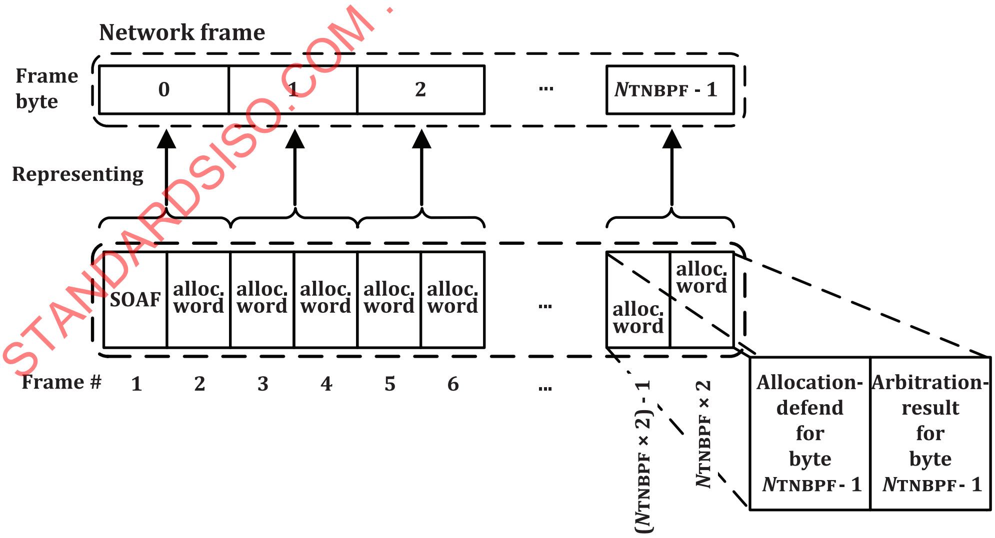


Figure 7 — Allocation words representing bytes of network frame

The first network frame that represents a particular byte is called the allocation-defend frame of this particular byte. The following network frame is named arbitration-result frame of the particular byte.

For communicating various details on allocated network frame bytes, a special format for the allocation word is specified. It is named the reporting format. In the reporting format, the allocation word is split up into several bit fields.

REQ	2.68 DLL – Allocation channel – Allocation word – Reporting format
The structure of the allocation word in reporting format shall be in accordance with Table 12 and 12.2.	

Table 12 — Allocation word — Reporting format

Bit	Content
<15:12>	4-bit CRC protecting bits 11 to 0 of the allocation word in reporting format
<11:10>	Reserved (default is zero)
9	New allocation flag (default is 0) 0_2: New allocation flag cleared/not set 1_2: New allocation flag indicates recent allocation
<8:0>	Connection label

8.1.3 DLL — Common allocation channel related subjects

All nodes supervise the allocation frame.

REQ	2.69 DLL – Allocation channel – Administrative area
A node shall not attempt to allocate bytes in the administrative area of the network frame.	

REQ	2.70 DLL – Allocation channel – Connection label (CL)
A node shall use a connection label to identify groups of bytes that are allocated during a single allocation request. NOTE 1 The connection label is unique. NOTE 2 If single bytes are allocated, they also receive a unique connection label.	

REQ	2.71 DLL – Allocation channel – Last allocated byte as connection label
After allocating all network frame bytes, which the node arbitrates for, it shall use the position of the network frame byte that is allocated last as connection label.	

REQ	2.72 DLL – Allocation channel – Send allocation word in allocation-defend frame
After successful allocation of a byte of the network frame, a node shall defend that byte in the corresponding allocation-defend frame.	

REQ	2.73 DLL – Allocation channel – Allocation word for defending a byte
The allocation word that a node uses to defend a network frame byte shall be calculated as such: 1. After allocating the byte, the allocation word shall be 0002_{16}. 2. After allocating all requested bytes, the allocation word in reporting format shall include the connection label and the new allocation flag set. After the timeout for the new allocation flag, the allocation word in reporting format shall include the connection label and the new allocation flag cleared.	

REQ	2.74 DLL – Allocation channel – Insufficient bandwidth
If during an allocation request, a node cannot allocate all required bytes, it shall cancel the allocation request.	

8.1.4 DLL — Allocation channel related subjects for the TimingMaster

8.1.4.1 DLL — Allocation-defend frame

REQ	2.75 DLL – Allocation channel –TimingMaster sends SOAF
To begin an allocation frame, the TimingMaster shall send an allocation word in SOAF format. NOTE This serves the purpose of synchronising the TimingSlave nodes to the beginning of the allocation frame, see Figure 7.	

REQ	2.76 DLL – Allocation channel – TimingMaster clears the allocation word
If the TimingMaster is not defending a network frame byte or attempting to allocate the byte, it shall send 0000_{16} in the corresponding allocation-defend frame. NOTE This is referred to as the TimingMaster clearing the allocated word.	

REQ	2.77 DLL – Allocation channel – TimingMaster starts allocation
To start the allocation of a network frame byte, the TimingMaster shall send 0001_{16} in the corresponding allocation-defend frame.	

8.1.4.2 DLL — Arbitration-result frame

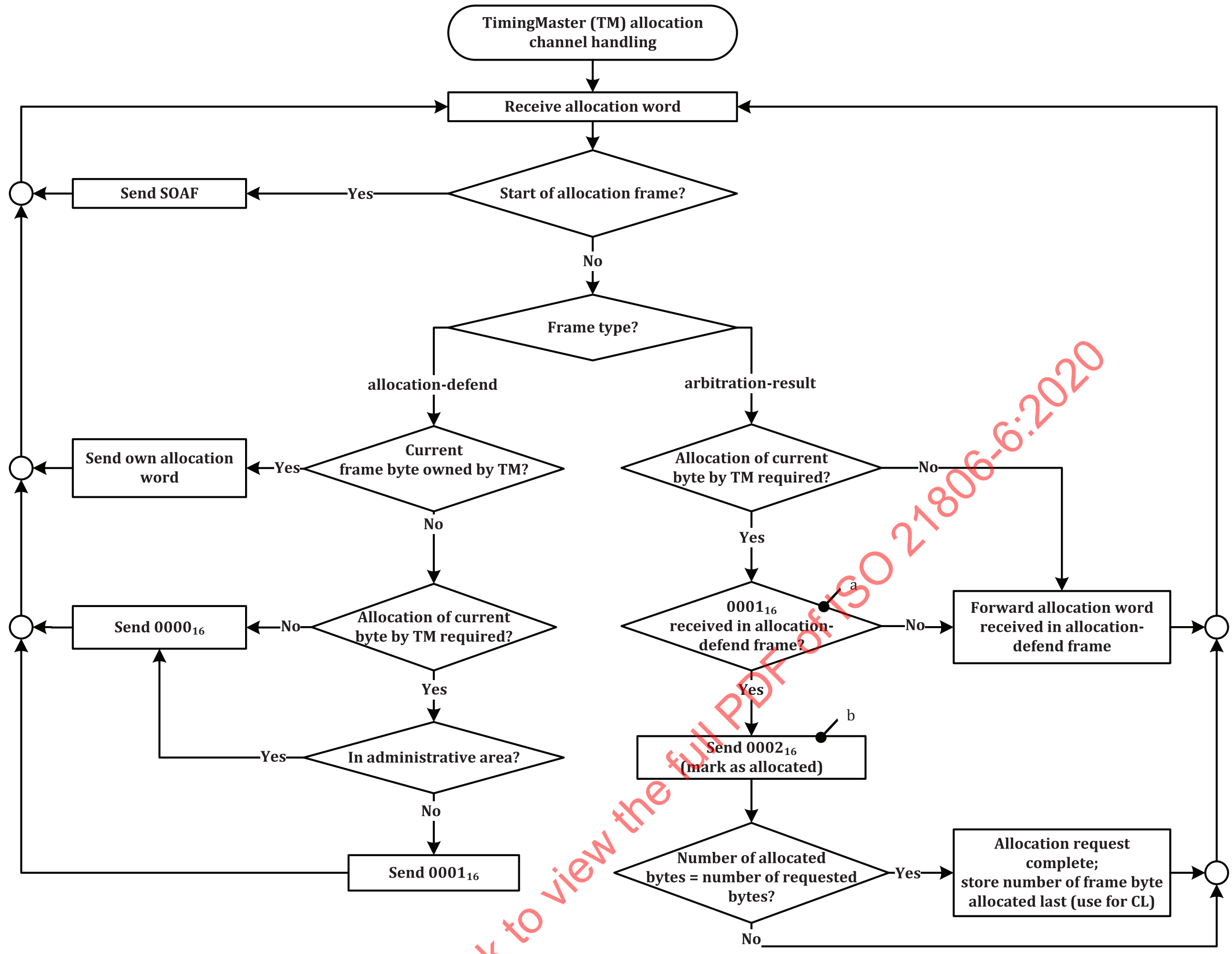
REQ	2.78 DLL – Allocation channel – TimingMaster not arbitrating
If the TimingMaster is not arbitrating for a network frame byte, in the arbitration-result frame, it shall forward the allocation word from the allocation-defend frame without altering it.	

REQ	2.79 DLL – Allocation channel – TimingMaster loses arbitration
For one network frame byte, if the TimingMaster does not receive 0001_{16} in the allocation-defend frame after sending 0001_{16} in the allocation-defend frame, the TimingMaster shall forward the allocation word received in the allocation-defend frame in the arbitration-result frame. NOTE Consequently, the TimingMaster loses arbitration for that byte.	

REQ	2.80 DLL – Allocation channel – TimingMaster wins arbitration
For one network frame byte, if the TimingMaster receives 0001_{16} in the allocation-defend frame after sending 0001_{16} in the allocation-defend frame, the TimingMaster shall send 0002_{16} in the arbitration-result frame. NOTE The TimingMaster wins arbitration for that byte and allocates it.	

8.1.4.3 DLL — Allocation channel handling

If no errors occur, the TimingMaster handles the allocation channel as shown in Figure 8.



a Arbitration lost.

b Arbitration won.

NOTE The byte is now owned by the TimingMaster. 0002_{16} is the temporary own allocation word until all bytes have been allocated.

Figure 8 — TimingMaster handles the allocation channel

8.1.5 DLL — Allocation channel related subjects for a TimingSlave

8.1.5.1 DLL — Allocation-defend frame

REQ	**2.81 DLL – Allocation channel – START**
If a TimingSlave receives a START indicator in the high byte of the respective allocation word, it shall regard the low byte of the respective allocation word as reserved.	

REQ	**2.82 DLL – Allocation channel – START and supervisory mechanisms**
A TimingSlave shall restart its supervisory mechanisms if it receives the START indicator.	

REQ	**2.83 DLL – Allocation channel – TimingSlave forwards allocation word**
If a TimingSlave has not allocated a network frame byte and is not attempting to allocate it, in the corresponding allocation-defend frame, it shall forward the received allocation word.	

REQ	2.84 DLL – Allocation channel – TimingSlave only arbitrates for unallocated bytes
If a TimingSlave does not receive 0000_{16} in the allocation-defend frame, it shall not start the allocation of the corresponding network frame byte.	

REQ	2.85 DLL – Allocation channel – TimingSlave starts allocation
To start the allocation of a network frame byte, a TimingSlave shall send 0001_{16} in the corresponding allocation-defend frame.	

8.1.5.2 DLL — Arbitration-result frame

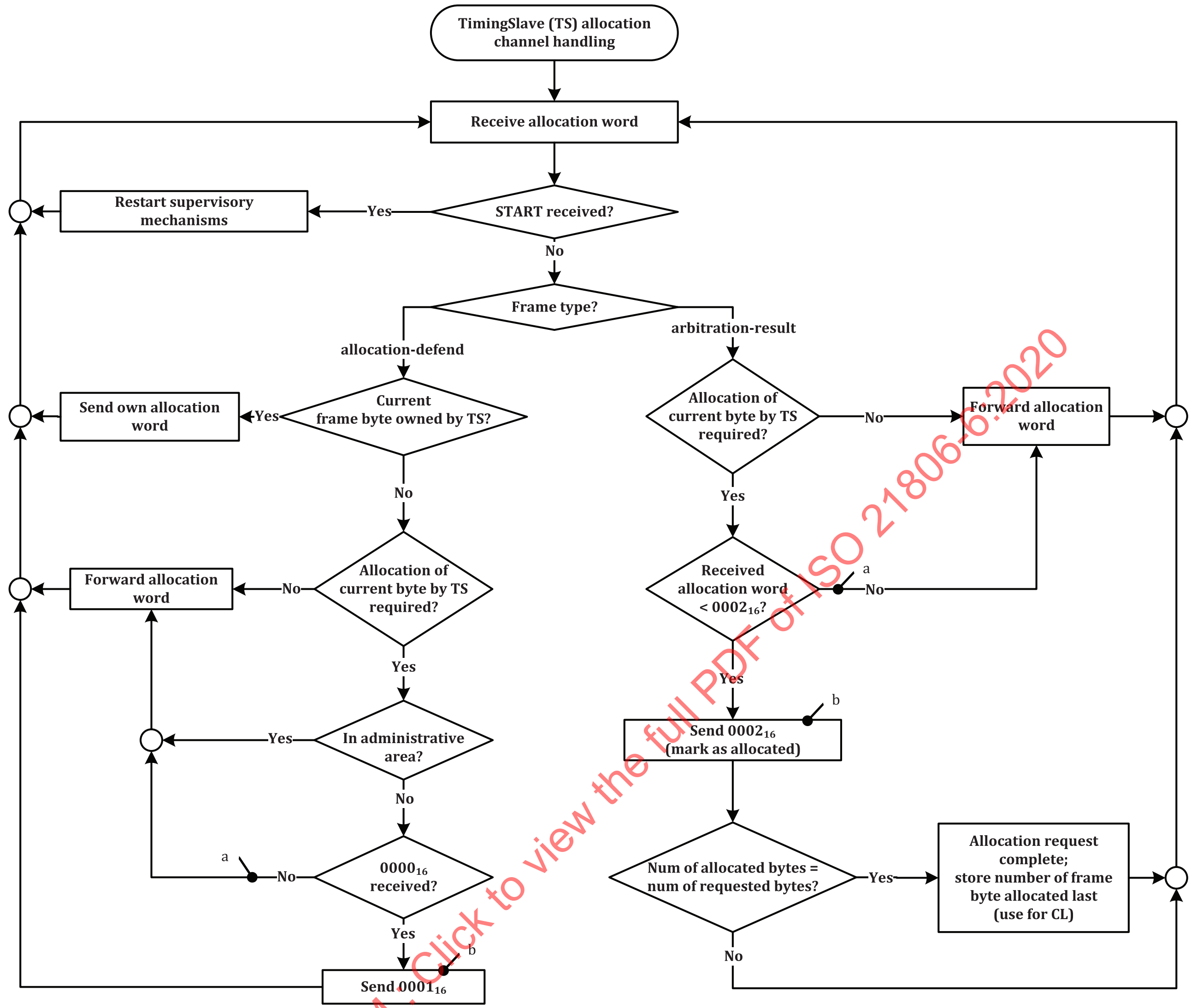
REQ	2.86 DLL – Allocation channel – TimingSlave not arbitrating
If a TimingSlave is not attempting to allocate, in the corresponding arbitration-result frame, it shall forward the received allocation word.	

REQ	2.87 DLL – Allocation channel – TimingSlave loses arbitration
For one network frame byte, the TimingSlave shall forward the allocation word received in the arbitration-result frame if the TimingSlave receives a value greater than 0001_{16} in the arbitration-result frame after sending 0001_{16} in the allocation-defend frame. NOTE 1 Consequently, the TimingSlave loses arbitration for that byte.	

REQ	2.88 DLL – Allocation channel – TimingSlave wins arbitration
For one network frame byte, the TimingSlave shall send 0002_{16} in the arbitration-result frame if the Timing-Slave receives a value less than 0002_{16} in the arbitration-result frame after sending 0001_{16} in the allocation-defend frame. NOTE 2 The TimingSlave wins arbitration for that byte and therefore allocates it.	

8.1.5.3 DLL — Allocation channel handling

If no error handling is necessary, the TimingSlave handles the allocation channel as shown in Figure 9.



a Arbitration lost.

b Allocation word is sent in the same network frame.

Figure 9 — TimingSlave handles the allocation channel

8.1.6 DLL — De-allocating

If a byte is no longer needed, a node stops defending the byte.

8.1.7 DLL — Source-drop recognition

8.1.7.1 DLL — General

If a source drops out of the MOST network (e.g. by general reset of the related MOST device), this node no longer defends its channels.

8.1.7.2 DLL — Unexpected connection label

A sink, connected to the channels of a node that drops out, recognizes a mismatch between the connection label(s) it expects and the connection label it receives. This is an indication for a source drop.

8.1.7.3 DLL — New allocation flag

The new allocation flag is set for newly allocated bytes. Therefore, the occurrence of the new allocation flag while being connected to particular bytes is an indication for a source drop that is followed by an immediate re-allocation of the respective bytes.

8.1.8 DLL — Error handling

8.1.8.1 DLL — 4-bit CRC

Connecting to a byte requires a valid 4-bit CRC and the matching connection label.

REQ	2.89 DLL – Allocation channel – 4-bit CRC
If a CRC error is detected for a particular allocation word, the result of counting free bytes in the network frame shall be ignored for the current allocation frame.	

In the case of a CRC error, source drops cannot be recognized for the allocation word.

8.1.8.2 DLL — Network synchronisation and bandwidth allocation

The synchronisation status of the MOST network influences the behaviour of both, the TimingMaster and the TimingSlaves, with respect to the allocation channel.

REQ	2.90 DLL – Allocation channel – Error handling
A TimingSlave node shall use the state transitions specified in Figure 10 for error handling.	

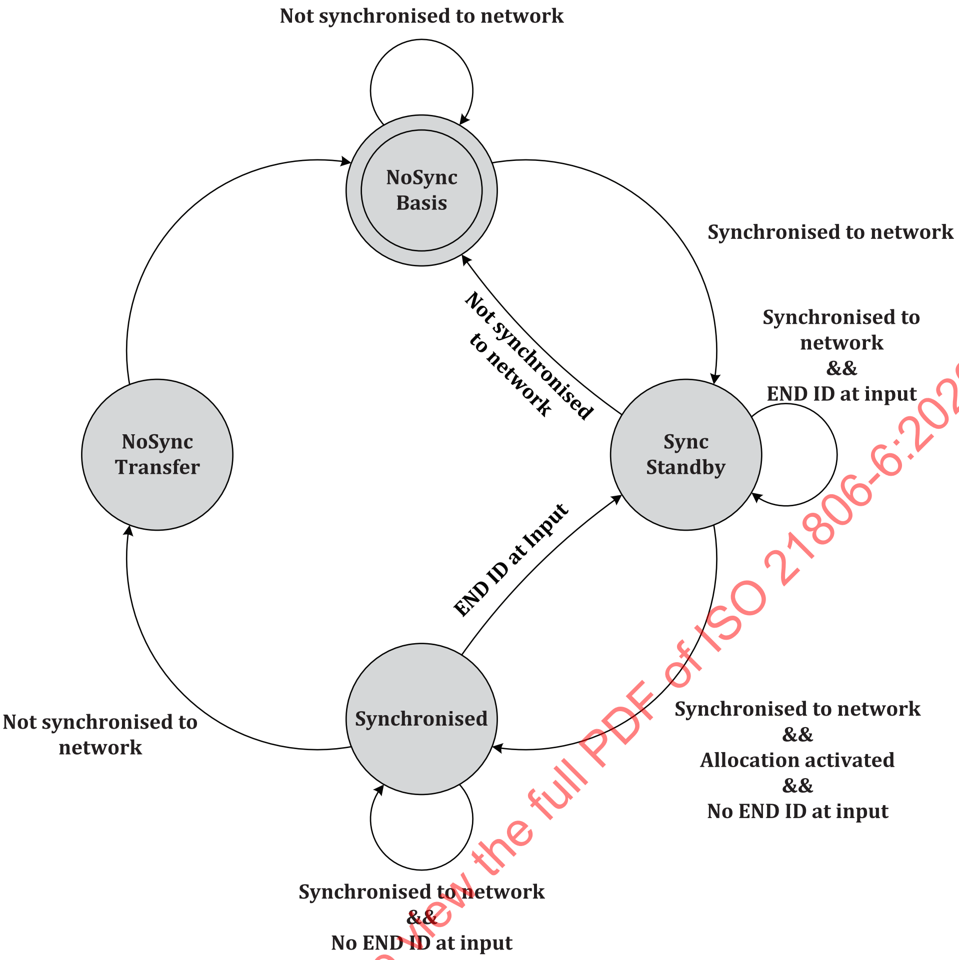


Figure 10 — Error handling of TimingMaster and TimingSlaves

REQ	2.91 DLL – Allocation channel – START ID filling
The structure of the allocation word for START ID filling shall be in accordance with Table 11.	

REQ	2.92 DLL – Allocation channel – END ID filling
The structure of the allocation word for END ID filling shall be in accordance with Table 13.	

Table 13 — Allocation word for END ID filling

Bit	Content
<15:8>	END indicator
<7:0>	END indicator

8.1.8.3 DLL — Specifics for TimingMaster

The error handling in TimingMaster nodes uses the state transitions shown in Figure 10.

Depending on the state of the allocation handling mechanism, the TimingMaster performs END ID filling or START ID filling.

REQ	2.93 DLL – Allocation channel – Filling mode
In filling mode, the TimingMaster shall replace the regular content of the allocation word with content as specified in Table 14 in each network frame.	

Table 14 specifies the synchronisation status and allocation word (TimingMaster).

Table 14 — Synchronisation status and allocation word (TimingMaster)

State	Allocation status	Allocation word
NoSync Basis	Standby	END ID filling
Sync Standby	Standby	START ID filling
Synchronised	Normal operation	As specified in 8.1.2 to 8.1.5
NoSync Transfer	Going to standby	END ID filling

8.1.8.4 DLL — Specifics for TimingSlave

The error handling in TimingSlave nodes uses the state transitions shown in Figure 10.

REQ	2.94 DLL – Allocation channel – Allocation word replacement
In filling mode, the TimingSlave shall replace the regular content of the allocation word with content as specified in Table 15 in each network frame.	

Table 15 specifies the synchronisation status and allocation word (TimingSlave).

Table 15 — Synchronisation status and allocation word (TimingSlave)

State	Allocation status	Allocation output
NoSync Basis	Standby	END ID filling
Sync Standby	Standby	Bypass content of allocation channel including END IDs.
Synchronised	Normal operation	As specified in 8.1.2 to 8.1.5, synchronise to incoming START IDs.
NoSync Transfer	Going to standby	END ID filling

8.2 DLL — Control channel

8.2.1 DLL — General

For data transmission, the control frame begins with a START indicator and ends with an END indicator. The structure of the control frame is described in Table 16.

REQ	2.95 DLL – Control channel – Long control frame
If the length of a control frame exceeds the available size of the control channel, the transmission shall span contiguous network frames without gaps in the transmitted control frame.	

Gaining exclusive write access to the control channel is governed by the load-adaptive arbitration mechanism (see 10.2). The arbitration mechanism uses the first two bytes (WAIT/START and Arb[X]) for arbitrating the control channel.

Only one node can send data at a time, whereas all other nodes can listen to the control channel at the same time. Thus, during a single write access, one or multiple nodes can be addressed.

REQ	2.96 DLL – Control channel – 16-bit addresses
The control channel shall use the 16-bit address types.	

Flow control is available, see Clause 9.

8.2.2 DLL — Control frame structure

REQ	**2.97 DLL – Control channel – Control frame structure**
The control frame structure shall be in accordance with Table 16.	

Table 16 — Control frame structure

Byte	Element	Description
1	WAIT/START	Network indicators (depend on arbitration, see 10.2)
2	Arb[X]	8-bit arbitration value (see 10.2)
3	TarLow	Low byte of 16-bit target address (see 11.2)
4	TarHigh	High byte of 16-bit target address (see 11.2)
5	PACK	Pre-emptive acknowledge (see 9.1)
6	LenHigh	Number of bytes that follow the LenLow byte. Counting starts at the Index byte and ends with the END indicator, which is included.
7	LenLow	
8	Index	Control frame index
9	SrcHigh	High byte of 16-bit logical node address of sender
10	SrcLow	Low byte of 16-bit logical node address of sender
11	PL1	Payload byte 1
12	PL2	Payload byte 2
:	:	:
k - 5	PLp - 1	Payload byte p - 1
k - 4	PLp	Payload byte p
k - 3	CRCHigh	CRC includes all bytes starting at TarLow up to the last payload byte. The PACK byte is assumed to be 00_{16} for the CRC calculation.
k - 2	CRCLow	
k - 1	CACK	See 12.6.
k	END	Network indicator

REQ	**2.98 DLL – Control channel – CRC calculation**
The CRC for the control frame shall be calculated according to 12.3.	

Table 17 shows an example of fitting control frame into a 4 bytes wide control channel.

Table 17 — Example — Fitting control frame into a 4 bytes wide control channel

Network frame	Control frame byte numbers	Control channel			
		Byte 1	Byte 2	Byte 3	Byte 4
1	1 to 4	WAIT/START	Arb[X]	TarLow	TarHigh
2	5 to 8	PACK	LenHigh	LenLow	Index

8.3 DLL — Protected system channel

8.3.1 DLL — General

The protected system channel conveys various kinds of low-level network status information through the protected system frame.

REQ	**2.99 DLL – Protected system channel – Size and location**
The protected system channel shall be one byte wide and located in the administrative area of the network frame.	

REQ	2.100 DLL – Protected system channel – TimingMaster's role
The TimingMaster shall drive the protected system frame.	

For a sample implementation, refer to Figure A.1.

8.3.2 DLL — Protected system frame structure

REQ	2.101 DLL – Protected system channel – Protected system frame structure
The protected system frame structure shall be in accordance with Table 18.	

Table 18 — Protected system frame structure

Byte	Content	Written by
1	Indicator	TimingMaster
2	Node counter	TimingMaster and TimingSlave
3	Visible nodes	TimingMaster
4	Byte4	TimingMaster
5	System flags	TimingMaster and TimingSlave
6	Byte6	TimingMaster
7	CRC High Byte	TimingMaster and TimingSlave
8	CRC Low Byte	TimingMaster and TimingSlave

8.3.2.1 DLL — Indicator

The START indicator generated by the TimingMaster indicates the start of the protected system frame.

REQ	2.102 DLL – Protected system channel – TimingSlave synchronisation
On receiving the START indicator, a TimingSlave shall synchronise its receiving mechanism for the protected system frame.	

REQ	2.103 DLL – Protected system channel – START not included in CRC
The START indicator itself shall not be part of CRC calculation.	

REQ	2.104 DLL – Protected system channel – Reaction on CRC error
If the CRC check of the incoming CRC performed by the TimingMaster node indicates an invalid value, the TimingMaster shall start the subsequent protected system frame by replacing the START indicator with a WAIT indicator.	

8.3.2.2 DLL — Node counter

REQ	2.105 DLL – Protected system channel – Node counting
Node counting shall start at the TimingMaster node.	

REQ	2.106 DLL – Protected system channel – Node position of the TimingMaster
The TimingMaster's position number shall be zero.	

REQ	2.107 DLL – Protected system channel – TimingMaster sets node counter
The TimingMaster shall set the node counter in the protected system frame to zero.	

Each TimingSlave node receives the node counter.

REQ	2.108 DLL – Protected system channel – TimingSlave increments node counter
A TimingSlave shall increment the received node counter by one, store the incremented counter value locally and transmit this incremented counter value to its direct neighbour node in downstream direction.	

REQ	2.109 DLL – Protected system channel – Handling of the node counter
A TimingSlave shall process each individual bit of the node counter on-the-fly.	

The incremented value represents the position of the node amongst the currently visible nodes.

REQ	2.110 DLL – Protected system channel – Active bypass prevents node counter incrementation
A TimingSlave node that has its bypass active shall not increment the node counter.	

The node counter value is valid and can only be used if the CRC for the current protected system frame is OK.

8.3.2.3 DLL — Visible nodes

The TimingMaster evaluates the node counter it receives.

REQ	2.111 DLL – Protected system channel – TimingMaster stores valid number of nodes
After sufficient time of continuous lock and if the CRC for the current protected system frame is OK, the TimingMaster shall increment the received node counter by one and store it as the valid number of nodes visible in the network.	

REQ	2.112 DLL – Protected system channel – Node counter validity
If there is not sufficient time of continuous lock or if the CRC for the current protected system frame is not OK, the TimingMaster shall ignore the received node counter.	

REQ	2.113 DLL – Protected system channel – Visible nodes value
The TimingMaster shall distribute the visible nodes value in the following protected system frame.	

REQ	2.114 DLL – Protected system channel – TimingSlave using visible nodes value
A TimingSlave that receives the visible nodes value shall use it if the protected system channel CRC for the current protected system frame is OK.	

REQ	2.115 DLL – Protected system channel – TimingSlave ignoring visible nodes value
A TimingSlave that receives the visible nodes value shall use the previous value (from the last protected system frame) if the protected system channel CRC for the current protected system frame is not OK.	

8.3.2.4 DLL — Customisation bytes

Byte4 and Byte6 are intended for customisation purposes.

8.3.2.5 DLL — System flags

REQ	2.116 DLL – Protected system channel – System flags
The structure of the system flags shall be in accordance with Table 19.	

Table 19 — System flags

Bit #	Content	Default	Written by	Comment
7	Reserved	0_2	TimingMaster	0_2: Reserved
6	Reserved	0_2	TimingMaster	0_2: Reserved
5	Reserved	1_2	TimingMaster	1_2: Reserved
4	Diagnosis flag	0_2	TimingMaster	0_2: Diagnosis is not active. 1_2: Diagnosis is active.
3	Static master flag	0_2	TimingMaster	0_2: Standard TimingMaster mode 1_2: TimingMaster is continuously sending network frames.
2	Ring lock flag	0_2	TimingMaster	0_2: TimingMaster is not locked. 1_2: TimingMaster is locked.
1	Shutdown flag	0_2	TimingMaster, TimingSlave	Set by TimingMaster, or bitwise disjunction by TimingSlave 0_2: Shutdown flag is cleared/not set. 1_2: Shutdown flag indicates regular shutdown.
0	Lock flag	0_2	TimingMaster	Set by TimingMaster 0_2: Lock flag is cleared/not set. 1_2: Lock flag indicates stable lock at TimingMaster.

8.3.2.6 DLL — CRC

REQ	2.117 DLL – Protected system channel – CRC calculation
The CRC for the protected system frame shall be calculated according to 12.3.	

8.4 DLL — Timestamp channel

8.4.1 DLL — General

On the timestamp channel, the TimingMaster outputs a CRC-protected timestamp frame to which TimingSlaves synchronise.

8.4.2 DLL — Timestamp frame structure

REQ	2.118 DLL – Timestamp channel – Timestamp frame structure
The structure of the timestamp frame shall be in accordance with Table 20.	

Table 20 — Timestamp frame structure

Byte	Content
0	START indicator
1	Timestamp frame counter, bits 31 to 24
2	Timestamp frame counter, bits 23 to 16
3	Timestamp frame counter, bits 15 to 8
4	Timestamp frame counter, bits 7 to 0
5	Reserved, write as 00_{16}
6	Reserved, write as 00_{16}
7	CRC high byte

Table 20 *(continued)*

Byte	Content
8	CRC low byte

8.4.3 DLL — Behaviour

REQ	2.119 DLL – Timestamp channel – Timestamp frame counter
The timestamp frame counter shall reside in the TimingMaster.	

REQ	2.120 DLL – Timestamp channel – Timestamp frame counter reset
After reset, the timestamp frame counter shall start at zero, counting upwards, incrementing once at each timestamp frame.	

REQ	2.121 DLL – Timestamp channel – Timestamp frame counter wraparound
When reaching `FFFF FFFF`$_{16}$, the timestamp frame counter shall wrap around to `0000 0000`$_{16}$.	

REQ	2.122 DLL – Timestamp channel – TimingSlave updates counter
When a TimingSlave node receives the timestamp frame with a valid CRC, the TimingSlave's timestamp frame counter shall be updated with the timestamp frame counter value received on the timestamp channel.	

REQ	2.123 DLL – Timestamp channel – Free-running counter
For timestamp frames, in which the timestamp frame is not finished or the CRC is invalid, the TimingSlave's timestamp frame counter shall be free-running.	

REQ	2.124 DLL – Timestamp channel – Incrementing the counter
When in free-running mode, the TimingSlave shall increment the timestamp frame counter by one at each network frame.	

REQ	2.125 DLL – Timestamp channel – End of free-running mode
The TimingSlave's timestamp frame counter shall take over the value as distributed on the MOST network again, once free-running mode ends.	

REQ	2.126 DLL – Timestamp channel – CRC calculation
The CRC for the timestamp frame shall be calculated according to 12.3.	

8.5 DLL — Packet channel

8.5.1 DLL — General

The packet channel is shared by all nodes to transfer burst-type an isochronous packet data.

The packet channel is used to transport packet frames and Ethernet data frames.

REQ	2.127 DLL – Packet channel – Long frame
If the length of a packet frame or Ethernet data frame exceeds the available size of the packet channel, the transmission shall span contiguous network frames without gaps in the transmitted packet frame or Ethernet data frame.	

REQ	2.128 DLL – Packet channel – More than one START condition
Back to back transmission – i.e. having more than one START condition in one network frame – shall not be performed.	

REQ	2.129 DLL – Packet channel – Exclusive write access
Gaining exclusive write access to the packet channel shall be governed by the round-robin arbitration mechanism (see 10.3).	

REQ	2.130 DLL – Packet channel – Address types
The packet channel shall use the 16-bit address types for packet frames and the 48-bit address types for Ethernet data frames.	

Flow control is available, see Clause 9.

8.5.2 DLL — Packet frame structure

REQ	2.131 DLL – Packet channel – Packet frame structure
The structure of the packet frame shall be in accordance with Table 21.	

Table 21 — Packet frame structure

Byte	Element	Content
1	START	---
2	LenHigh	Data length high byte: Bit 7 : 0_2 (packet frame identification) <6:3>: 0000_2 (reserved) <2:0>: bits 10 to 8 of the 11-bit data length
3	LenLow	Data length low byte: <7:0>: bits 7 to 0 of the 11-bit data length
4	TarLow	Target address low byte, see Clause 11.
5	TarHigh	Target address high byte, see Clause 11.
6	PACK	See 9.1.
7	Index	Packet frame index
8	SrcHigh	Source address high byte
9	SrcLow	Source address low byte
10	PL1	Payload byte 1
11	PL2	Payload byte 2
		...[a]
k - 5	PLp - 1	Payload byte p - 1
k - 4	PLp	Payload byte p
k - 3[b]	CRCHigh	CRC high byte of 16-bit CRC (CRC includes packet frame bytes 2 up to k - 4)
k - 2	CRCLow	CRC low byte of 16-bit CRC
k - 1	CACK	See 12.6.
k	END/WAIT	---

[a] Payload bytes p = k - 13 ≥ 1.

[b] Packet frame length = k - 3 ≥ 11.

REQ	2.132 DLL – Packet channel – Packet frame CRC calculation
The CRC for the packet frame shall be calculated according to 12.3.	

8.5.3 DLL — Ethernet data frame structure

REQ	2.133 DLL – Packet channel – Ethernet data frame structure
The structure of the Ethernet frame, according to ISO/IEC/IEEE 8802-3, shall be transported as specified in Table 22.	

Table 22 — Ethernet data frame structure

Byte	Element	Content
1	START	---
2	LenHigh	Data length high byte: Bit 7 : 1_2 (Ethernet frame identification) <6:3>: 0000_2 (reserved) <2:0>: bits 10 to 8 of the 11-bit data length
3	LenLow	Data length low byte: <7:0>: bits 7 to 0 of the 11-bit data length
4	DA1	Bits 47 to 40 of destination address
5	DA2	Bits 39 to 32 of destination address
6	DA3	Bits 31 to 24 of destination address
7	DA4	Bits 23 to 16 of destination address
8	DA5	Bits 15 to 8 of destination address
9	DA6	Bits 7 to 0 of destination address
10	PACK	See 9.1.
11	PL1	Payload byte 1
12	PL2	Payload byte 2
:	:	...[a]
k - 7	PLp - 1	Payload byte p - 1
k - 6	PLp	Payload byte p
k - 5	CRC1	Bits 31 to 24 of 32-bit CRC CRC includes Ethernet data frame bytes 4 up to k - 6; the PACK byte is explicitly excluded from CRC calculation.
k - 4	CRC2	Bits 23 to 16 of 32-bit CRC
k - 3[b]	CRC3	Bits 15 to 8 of 32-bit CRC
k - 2	CRC4	Bits 7 to 0 of 32-bit CRC
k - 1	CACK	See 12.6.
k	END/WAIT	---

[a] Payload bytes p = k - 16 ≥ 1.

[b] Data length = k - 3 ≥ 14.

REQ	2.134 DLL – Packet channel – Ethernet data frame CRC calculation
The CRC for the packet frame shall be calculated according to 12.4.	

8.5.4 DLL — Short packet frame or short Ethernet data frame

A packet frame or Ethernet data frame transported over the packet channel is called "short" if its length, including the START and END indicators, is less than or equal to the number of bytes allocated to the packet channel in the network frame.

REQ	2.135 DLL – Packet channel – Short frame
For a short packet frame or Ethernet data frame, the END indicator position (in network frame n + 1) shall match the START indicator position (in network frame n).	

REQ	2.136 DLL – Packet channel – Padding with zeros
For a short packet frame or Ethernet data frame, unused bytes between the CACK and the END indicator shall be padded with zeros.	

REQ	2.137 DLL – Packet channel – Removal of padded zeros
When receiving a short packet frame or Ethernet data frame, padded zeros shall be removed by the receiving node.	

A short packet frame or Ethernet data frame cannot have an earlyEND. The earlyEND is described in 9.2.

8.6 DLL — Synchronous channel

8.6.1 DLL — General

Synchronous data transfer uses a circuit-switched approach for streaming data, such as audio or video. As shown in 8.1, bandwidth allocation allows for reserving single bytes or groups of bytes in the source data area. Data put on such a byte/a group of bytes travels around the MOST network and thus can be read by all nodes. The MOST network DLL provides mechanisms to read data from or write data to a byte/a group of bytes on a frame-by-frame basis. This is independent of whether the bytes are arranged in a contiguous way or not.

8.6.2 DLL — Synchronous frame structure

The synchronous frame structure consists of bytes that carry unformatted data.

8.7 DLL — Isochronous channel

8.7.1 DLL — General

The isochronous channel provides transmission of isochronous data on the network frame by means of the isochronous frame.

8.7.2 DLL — Isochronous frame structure

REQ	2.138 DLL – Isochronous channel – Isochronous frame structure
The structure of the isochronous frame shall be in accordance with Table 23.	

Table 23 — Isochronous frame structure

Byte	Content
0	START indicator
1	Payload byte 1
2	Payload byte 2
3	Payload byte 3
...	
n	Payload byte n-1
n + 1	Payload byte n

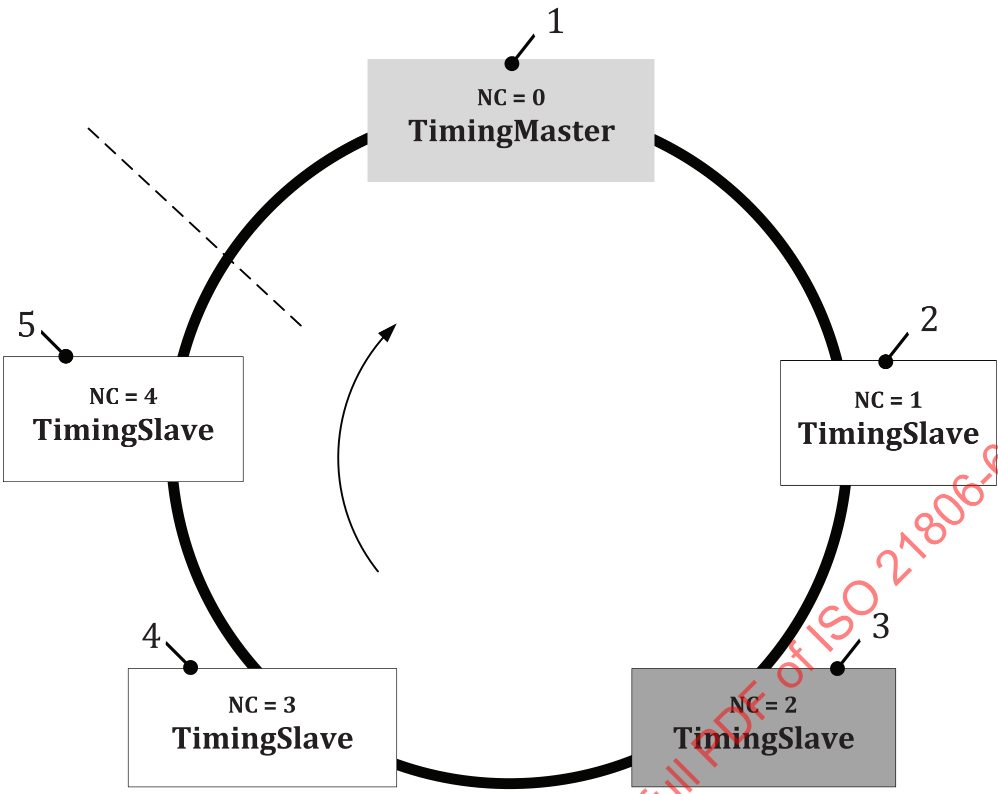
8.8 DLL — Channel frame delay

A network frame sent out by a node accumulates delay as it travels from node to node around the MOST network. When the network frame reaches the receiving section of the TimingMaster, it is no longer aligned to the network frame boundary transmitted by the TimingMaster.

REQ	2.139 DLL – Channel frame delay – Buffering
For the duration of one network frame, the TimingMaster shall buffer the received bytes.	
NOTE These are transmitted in the subsequent network frame.	

This buffering in the TimingMaster node differentiates nodes that are upstream of the sending node from nodes that are downstream of the sending node. Nodes between the node sending a channel frame and the TimingMaster are considered downstream, and nodes between the TimingMaster (incl. TimingMaster) and the node sending a channel frame are considered upstream. Due to the buffering in the TimingMaster, upstream nodes receive the most recent part of the channel frame shifted by one network frame, relative to downstream nodes.

Figure 11 shows upstream and downstream nodes.



Key

1 upstream node

2 upstream node

3 sending node or node initiating arbitration

4 downstream node

5 downstream node

Figure 11 — Upstream and downstream nodes

Figure 11 depicts an example of a five-node MOST network in which the node with node counter value two (NC = 2) is highlighted. This node is the sender of a channel frame. All nodes with a smaller node counter value are upstream nodes (relative to the node at position two). Those nodes having a node counter value greater than that of the sending node are downstream nodes. All upstream nodes see a delay of one network frame, while downstream nodes do not see that delay.

Table 24 — Example — Channel frame delay

Network frame	Downstream nodes	Upstream nodes
1	START|Admin1|Admin2|etc.	---
2	---	START|Admin1|Admin2|etc.
...	...	...
n	END	---
n + 1	---	END

Between START and END the channel is busy. After END is received, the channel is idle again. Table 24 shows that downstream nodes see the channel already busy in network frame 1 whereas upstream nodes see it one network frame later. When the sending node is finished in network frame ‘n’, downstream nodes see the channel idle again already in the same network frame whereas upstream nodes see the channel idle one network frame later, in network frame ‘n + 1’.

9 DLL — Flow control

9.1 DLL — Pre-emptive acknowledge byte

Channel frames that support pre-emptive acknowledgement use the PACK byte to communicate a pre-emptive acknowledge from one or multiple potential receiver(s) back to the originator. Thus, the originator is able to determine whether the channel frame can be received by all intended target nodes, by parts of them, or by no target node. In the latter case, the related transmission then can be aborted early to free up the related channel (see 9.2).

REQ	**2.140 DLL – Flow control – DLL – Pre-emptive acknowledge byte – Sending node behaviour**
A sending node shall transmit 00_{16} for the PACK byte.	

A node that potentially receives the channel frame transmits a modified version of the received PACK byte.

REQ	**2.141 DLL – Flow control – DLL – Pre-emptive acknowledge byte – Target address not matching**
A node, whose address does not match the target address of the channel frame, shall not modify the PACK byte.	

REQ	**2.142 DLL – Flow control – DLL – Pre-emptive acknowledge byte – Modification of the PACK byte**
Modification of the PACK byte shall be done through bitwise disjunction of 1_2 with the bits related to the particular status according to Table 25.	

If the PACK byte returns to the originator with bit 2 cleared (unaltered), then no node is ready to receive the current channel frame.

Table 25 — PACK byte structure

Bit	Default	Description
<7:3>	$0000\ 0_2$	Bits 7 to 3 shall be $0000\ 0_2$, except for debug PDUs, in which bit 7 shall be set to 1_2. Other values imply corruption of the PACK byte.
2	0_2	1_2 if node is ready to receive.
1	0_2	Reserved
0	0_2	1_2 if node is NOT ready to receive.

9.2 DLL — Early ending

REQ	**2.143 DLL – Flow control – DLL – Early ending – Aborting a channel frame**
If the PACK byte indicates the need to abort a channel frame transmission early, the sending node shall place END into the related channel. NOTE This case is referred to as earlyEND.	

REQ	**2.144 DLL – Flow control – DLL – Early ending – earlyEND**
In the case of an earlyEND, the position of the END indicator shall be determined according to these rules: 1. The position is the next channel byte that is located one network frame plus six network frame bytes after the PACK byte. 2. If this places the END indicator outside the given channel bytes, the position is the first channel byte in the following network frame after the PACK byte.	

Table 26 shows a hypothetical 7-byte packet channel, as an example for the first part of the earlyEND placement rule. Here the END indicator is placed one network frame plus six network frame bytes after the PACK byte.

Table 26 — Example — END inserted one network frame plus six bytes after PACK

Network frame	Packet frame byte numbers	Packet channel						
		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
1	1 to 7	START	Admin1	Admin2	Admin3	Admin4	Admin5	Admin6
2	8 to 14	PACK	Admin7	Admin8	Admin9	PL0	PL1	PL2
3	15 to 21	PL3	PL4	PL5	PL6	PL7	PL8	END

Table 27 shows another hypothetical example, in which the packet channel is 4 bytes wide only. When applying the first part of the earlyEND placement rule, the calculated position of the END indicator would be located outside the area of the given packet channel. As this is not allowed, the second part of the earlyEND placement rule is applied. Thus, the END indicator is contained in the first byte of the particular packet channel in the subsequent network frame.

Table 27 — Example — Insertion point of END moved

Network frame	Packet frame byte numbers	Packet channel			
		Byte 1	Byte 2	Byte 3	Byte 4
1	1 to 4	START	Admin1	Admin2	Admin3
2	5 to 8	Admin4	PACK	Admin5	Admin6
3	9 to 12	Admin7	PL0	PL1	PL2
4	13	END	Reserved	Reserved	Reserved

9.3 DLL — Low-level retries

If channels that support the low-level retry mechanism send channel frames that do not reach their intended target, for example due to a momentary overload at the target node or due to CRC errors, then low-level retries can help to overcome such a situation.

A low-level retry means resending the channel frame after a certain waiting period. The maximum number of low-level retry cycles is determined by L_SET_TRANSMISSION_ATTRIBUTES.REQUEST (see 6.4.9).

REQ	2.145 DLL – Flow control – DLL – Low-level retries – Low-level retry cycle
A low-level retry cycle shall consist of a waiting period and the resending of the channel frame.	

REQ	2.146 DLL – Flow control – DLL – Low-level retries – Termination
Low-level retries shall stop if either all intended recipients receive the channel frame or the specified maximum number of low-level retry cycles is used up.	

REQ	2.147 DLL – Flow control – DLL – Low-level retries – Sending mechanisms
The initial sending of a channel frame and the sending of retries of this channel frame shall use the same sending mechanisms, which explicitly includes arbitration.	

REQ	2.148 DLL – Flow control – DLL – Low-level retries – Waiting period unit
The unit of a waiting period shall be one network frame.	

10 DLL — Arbitration

10.1 DLL — General

Arbitration organizes access to channels on the MOST network. There are two major procedures for arbitration: load-adaptive arbitration and round-robin arbitration.

10.2 DLL — Load-adaptive arbitration

10.2.1 DLL — General

REQ	2.149 DLL – Arbitration – Start of arbitration
Arbitration shall start for a node if the channel is idle and the node is ready to transmit a channel frame. NOTE 1 The channel is idle for a node after reset or when no transmission is ongoing.	

REQ	2.150 DLL – Arbitration – Initiation
An arbitration cycle shall be initiated by issuing a WAIT indicator.	

REQ	2.151 DLL – Arbitration – Wait time
A node shall not start arbitration before one network frame passes after the reception of the END indicator. NOTE 2 It takes two or three network frames to complete arbitration, whereas the last network frame of the arbitration cycle already contains the first bytes of the actual channel frame.	

REQ	2.152 DLL – Arbitration – Only sending nodes participate
Only such nodes that intend to send a channel frame shall participate in the arbitration, all other nodes shall bypass the channel frame unaltered.	

REQ	2.153 DLL – Arbitration – ARBVAL
Each node shall maintain an 8-bit arbitration value (ARBVAL) as specified in Table 28.	

Table 28 — Node arbitration value (ARBVAL)

Nibble	Content
ARBVALLow	Low nibble of arbitration value (0_{16} to F_{16})
ARBVALHigh	High nibble of arbitration value. The values 0_{16} to 3_{16} can be used to implement a priority feature. The higher the number, the higher the priority. Value 1_{16} should be the default value.

REQ	2.154 DLL – Arbitration – ARBVALLow
The ARBVALLow value represents a four-bit counter that shall exist in each node.	

REQ	2.155 DLL – Arbitration – ARBVALLow initialisation
The ARBVALLow value shall be initialised to F_{16} every 255 network frames.	

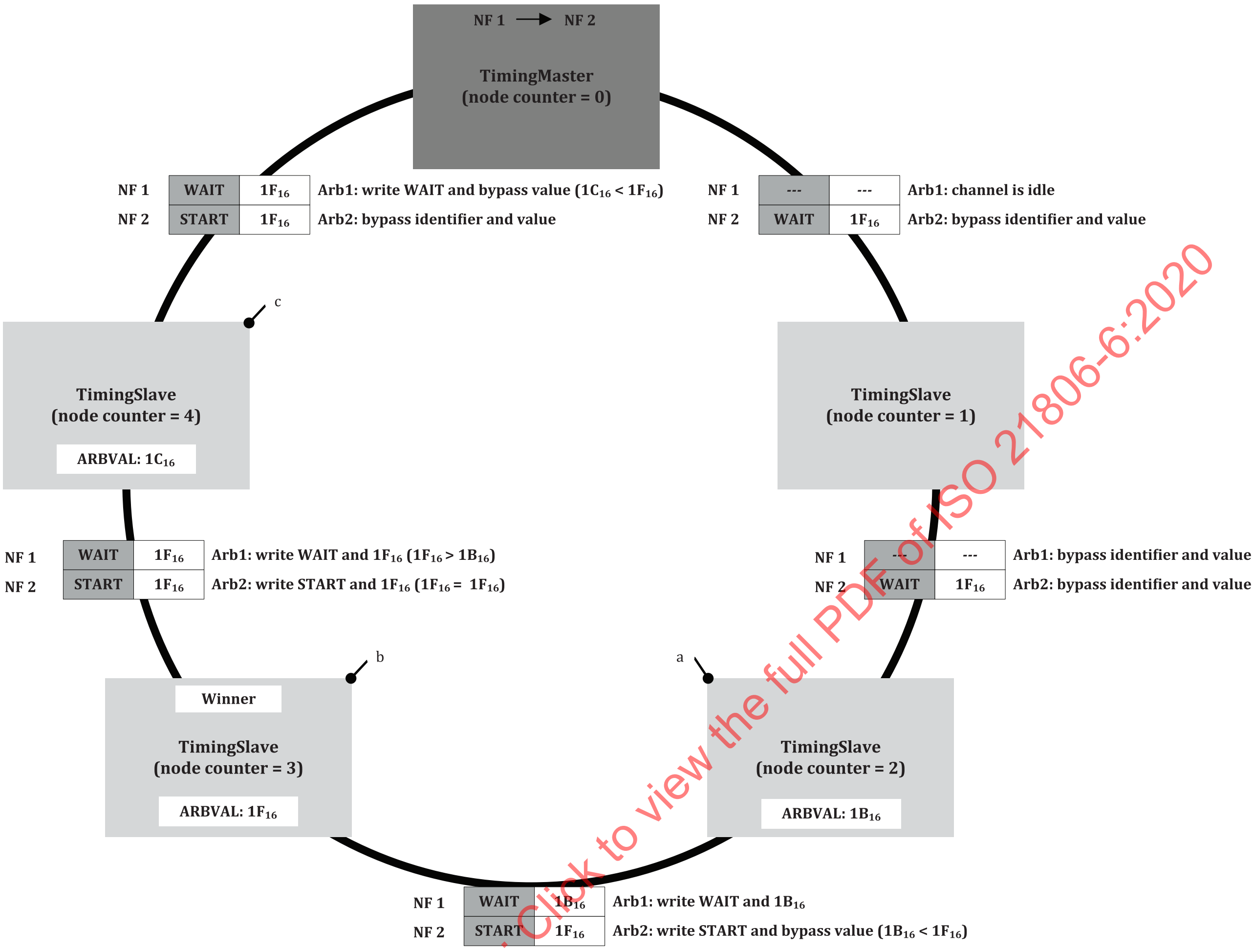
REQ	2.156 DLL – Arbitration – Node wins arbitration
When the node wins arbitration, it shall decrement the ARBVALLow counter by one.	

REQ	2.157 DLL – Arbitration – ARBVALLow minimum
The minimum value the ARBVALLow counter can reach shall be 0_{16}. NOTE 3 This allows other nodes (sending fewer channel frames and having greater ARBVAL values) a higher priority in the arbitration scheme.	

REQ	2.158 DLL – Arbitration – WAIT, START, and Arb[X]
WAIT or START and an Arb[X] value shall be transmitted in each channel frame. NOTE 4 In the network frame 1, the WAIT indicator is used to indicate the start of arbitration. In the network frame 2 and network frame 3, START is used to indicate the start of the channel frame. NOTE 4 START can only occur in the network frame 3 if an upstream node wins arbitration.	

For details on upstream and downstream nodes, refer to 8.8.

Figure 12 shows the arbitration value transport (only downstream).



[a] In the first network frame (NF), the initial identifier and value are replaced. The node waits for reception of the second network frame.

In the second network frame, the node sets START but the received Arb2 value is greater than ARBVAL; therefore, the node determines that it loses arbitration and as a result bypasses the Arb2 value.

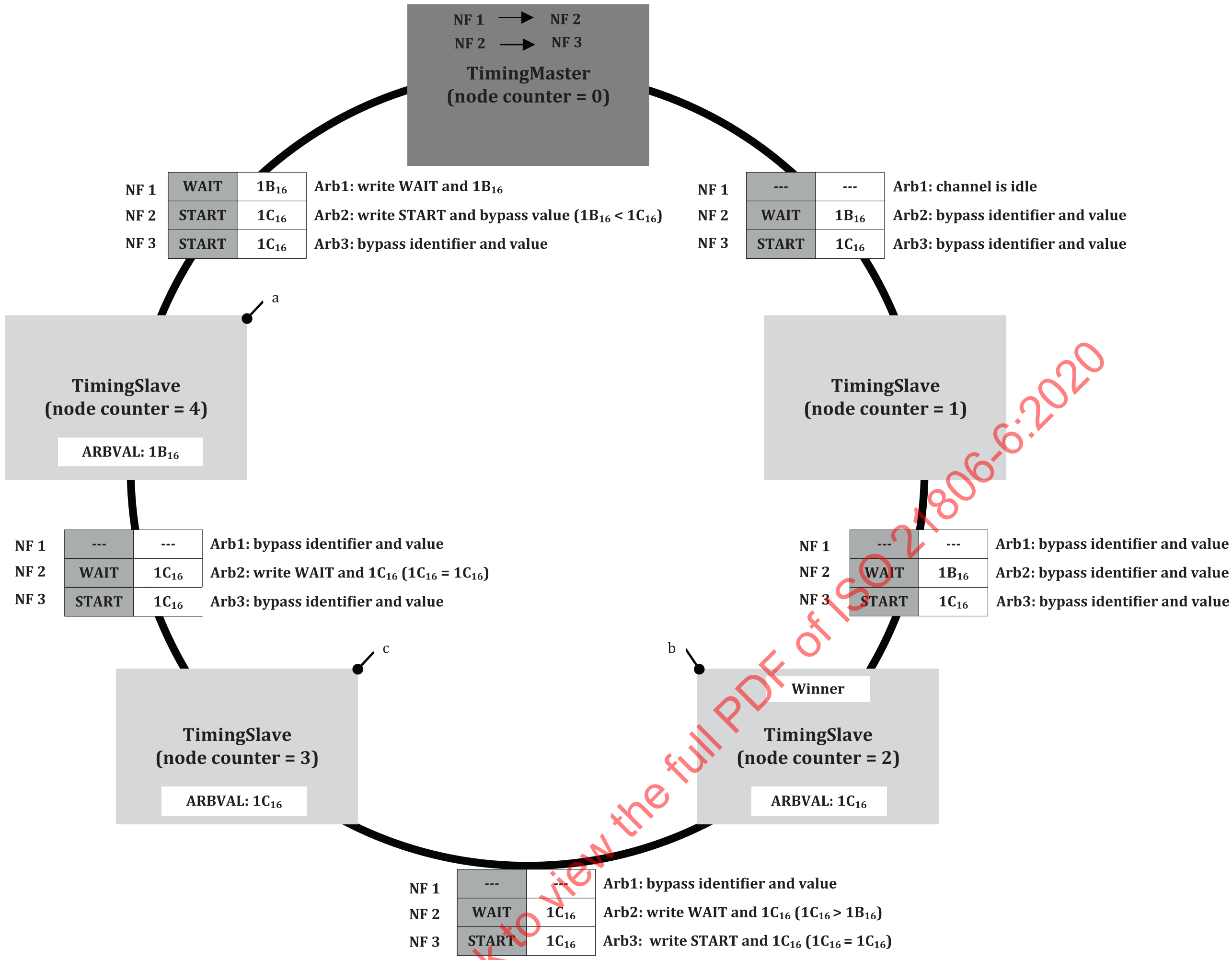
[b] In the first network frame, ARBVAL is greater than Arb1. The node waits for reception of Arb2.

In the second network frame, the received Arb2 value is equal to ARBVAL; therefore, the node determines that it wins arbitration. It starts transmitting a message with TarLow and TarHigh.

[c] In the first network frame, the received Arb1 value is greater than ARBVAL. The node determines that it loses arbitration and as a result bypasses the Arb1 value.

Figure 12 — Arbitration value transport (only downstream)

Figure 13 shows the arbitration value transport (upstream wins).



a In the first network frame (NF), the initial identifier and value are replaced. The node waits for reception of the second network frame.

In the second network frame, the received Arb2 value is greater than ARBVAL; therefore, the node determines that it loses arbitration and as a result bypasses the Arb2 value.

b In the first network frame, the node is idle.

In the second network frame, the received Arb2 value is less than ARBVAL. The node waits for Arb3.

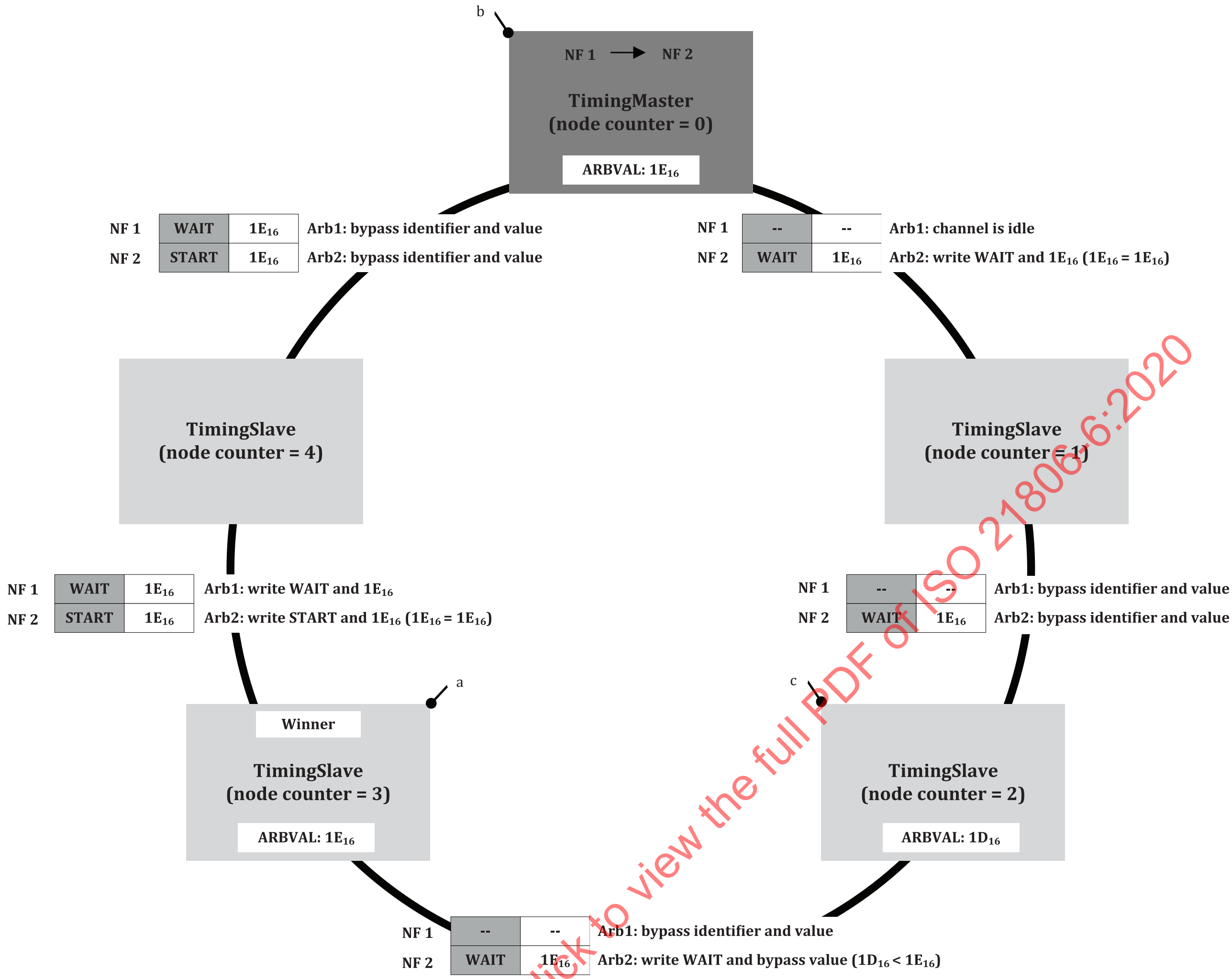
In the third network frame, the node determines that it wins arbitration. It starts transmitting a message with TarLow and TarHigh.

c In the first network frame, the node is idle.

In the second network frame, the received Arb2 value is equal to ARBVAL; therefore, the node determines that it loses arbitration.

Figure 13 — Arbitration value transport (upstream wins)

Figure 14 shows the arbitration value transport (upstream loses).



a In the first network frame (NF), the initial identifier and value are replaced. The node waits for reception of the second network frame.

In the second network frame, the received Arb2 value is equal to ARBVAL.

In the third network frame, the node determines that it wins arbitration. It starts transmitting a message with TarLow and TarHigh.

b In the first network frame, the node is idle.

In the second network frame, the received Arb2 value is equal to ARBVAL. The node determines that it loses arbitration.

c In the first network frame, the node is idle.

In the second network frame, the received Arb2 value is greater than ARBVAL; therefore, the node determines that it loses arbitration and bypasses Arb2.

Figure 14 — Arbitration value transport (upstream loses)

10.2.2 DLL — Downstream arbitration

This subclause covers downstream arbitration in the current network frame.

REQ	2.159 DLL – Arbitration – DLL – Downstream arbitration – No WAIT in same frame
During the transmission of network frame 1, if a node starts with WAIT to arbitrate and does not receive any WAIT in the same network frame, it shall put its ARBVAL directly into the next byte, Arb1.	

REQ	2.160 DLL – Arbitration – DLL – Downstream arbitration – Compare Arb1 and ARBVAL
During the transmission of network frame 1, if a node starts with WAIT to arbitrate and receives WAIT in the same network frame, it shall perform a comparison between the incoming Arb1 value and its own ARBVAL and issue the greater of both to the MOST network.	

REQ	2.161 DLL – Arbitration – DLL – Downstream arbitration – Bit-level comparison
The comparison of the incoming Arb1 value and the ARBVAL of a node and the following operation shall be performed on bit-level.	

REQ	2.162 DLL – Arbitration – DLL – Downstream arbitration – Arbitration part 1
The result for arbitration part 1 - network frame 1 (downstream) or network frame 2 (upstream) shall be determined according to Table 29.	

Table 29 — Arbitration part 1 — Network frame 1 (downstream) or network frame 2 (upstream) operation

OP #	Operation	Result
1	ARBVAL greater than incoming Arb1	Outgoing Arb1 = ARBVAL; overwriting. Arbitration decided in next network frame. Downstream node in current network frame or upstream node in next network frame can still have a higher ARBVAL.
2	ARBVAL equal or less than incoming Arb1	Outgoing Arb1 = Incoming Arb1; pass through arbitration lost, go to receive routine and bypass all following data.
3	No incoming Arb1	Outgoing Arb1 = ARBVAL; overwriting. Arbitration decided in next network frame. Downstream node in current network frame or upstream node in next network frame can still have a higher ARBVAL.

10.2.3 DLL — Downstream or upstream arbitration

This subclause describes downstream arbitration of the last network frame or upstream arbitration of the current network frame.

REQ	2.163 DLL – Arbitration – DLL – Downstream or upstream arbitration – Copy Arb1 into Arb2
In the network frame 2 of the arbitration flow, the TimingMaster shall copy the stored Arb1 value (from the network frame 1) into Arb2.	

REQ	2.164 DLL – Arbitration – DLL – Downstream or upstream arbitration – Copy WAIT
In the network frame 2 of the arbitration flow, the TimingMaster shall copy WAIT to the next network frame.	

New upstream nodes may enter arbitration in this network frame 2. They follow the rules that are described in 10.2.2, whereas Arb2 value now is used to decide who wins arbitration in the upstream nodes. Since they enter in the next network frame, they always see WAIT and they do a comparison on the Arb2 value.

The first node that started arbitration, and each downstream node still in the arbitration cycle, now issue START to begin transmitting the channel frame. Arb2 value is bypassed. The node that received back the same Arb2 value as its internal ARBVAL wins arbitration (no other node had a greater value) and starts transmitting the channel frame beginning with TarLow. All other nodes lose arbitration and switch to receive mode. For this example, it is assumed that none of the upstream nodes that entered arbitration can win this arbitration cycle.

Arbitration may end here if only downstream nodes are involved. If an upstream node starts arbitration in network frame 2 via issuing WAIT and it has a higher ARBVAL than the downstream nodes, it continues with network frame 3.

REQ	2.165 DLL – Arbitration – DLL – Downstream or upstream arbitration – Arbitration part 1
The result for arbitration part 2 – network frame 2 (downstream) or network frame 3 (upstream) shall be determined according to Table 30.	

Table 30 — Arbitration part 2 — Network frame 2 (downstream) or network frame 3 (upstream) operation

OP #	Operation	Result
1	ARBVAL equal to incoming Arb2	Arbitration won, prepare to start writing the channel frame. No further comparison required.
2	ARBVAL less than incoming Arb2	Arbitration lost, go to receive routine and bypass all following data.

10.2.4 DLL — Conditional upstream arbitration

REQ	2.166 DLL – Arbitration – DLL – Conditional upstream arbitration – Upstream node wins arbitration
If an upstream node wins arbitration, the network frame 3 shall be relevant. NOTE Otherwise, this frame contains channel frame data.	

REQ	2.167 DLL – Arbitration – DLL – Conditional upstream arbitration – Copy Arb2 into Arb3
If an upstream node wins arbitration, in the network frame 3 of the arbitration flow, the TimingMaster shall copy the stored Arb2 value (from the network frame 2) into Arb3.	

All downstream nodes and the node which started arbitration, have already lost arbitration.

REQ	2.168 DLL – Arbitration – DLL – Conditional upstream arbitration – Arb3 as relevant value
If an upstream node wins arbitration, the upstream nodes shall apply the rules as described in 10.2.3 to Arb3 to decide who wins.	

10.3 DLL — Round-robin arbitration

10.3.1 DLL — Basics

Round-robin arbitration of a channel is based on when and where within the network frame a node sets START relative to any START inserted by the other nodes in the MOST network.

Until an arbitrating node receives START, it cannot determine whether another node is arbitrating for the channel. When the arbitrating node sets START, it only receives START in the same frame byte of the next network frame (and thus win arbitration) if other nodes do not subsequently overwrite it; that is, if no downstream node sets START in a frame byte that is closer to the start of the current network frame and no upstream node sets START in a frame byte that is closer to the start of the next network frame.

A node receiving START cannot distinguish the difference between START set by an upstream node in the current network frame and START set by a downstream node in the previous network frame. Thus, potentially a node sets START before it receives START set by another node.

For example, a node sets START in frame byte 27 of the current network frame; previously, a downstream node sets START in frame byte 25 of the current network frame. The node that occupies byte 27 cannot detect the presence of START in byte 25 before it receives the next network frame.

Consequently, multiple nodes might set START in the channel in different network frame bytes at different times. Arbitration ensures that only one of these nodes wins arbitration.

While arbitrating, a node potentially receives partial channel frames, which are eventually overwritten by the data transmitted by the node that wins arbitration.

If a channel is currently in use and multiple nodes are waiting to transmit a channel frame, the nearest node in the network that is ready to transmit a channel frame wins arbitration once the channel becomes available.

The nearest node is either the closest downstream node to the current transmitter in the MOST network that is ready to transmit a channel frame, or if none of the downstream nodes are ready to transmit a channel frame, the furthest upstream node in the MOST network.

REQ	**2.169 DLL – Arbitration – DLL – Round-robin arbitration – Winning**
A node shall win arbitration if it receives START in the same frame byte of the current network frame that it set its START in the previous network frame.	

REQ	**2.170 DLL – Arbitration – DLL – Round-robin arbitration – Only set START when ready**
A node shall not set START if it is not ready to transmit a channel frame.	

REQ	**2.171 DLL – Arbitration – DLL – Round-robin arbitration – Only set START when idle**
A node shall not set START if the channel is not idle. NOTE 1 The channel is idle if no transmission is ongoing and at least one network frame is transmitted since the detection of END.	

REQ	**2.172 DLL – Arbitration – DLL – Round-robin arbitration – Mark channel on START**
If a node receives START while not arbitrating for a channel, it shall internally mark the channel as being in use. NOTE 2 Thus, this node is unable to arbitrate for the channel until it receives END.	

REQ	**2.173 DLL – Arbitration – DLL – Round-robin arbitration – Mark channel on losing arbitration**
A node that lost arbitration shall internally mark the channel as being in use.	

REQ	**2.174 DLL – Arbitration – DLL – Round-robin arbitration – Reset receiving mechanism on START**
If a node receives START, it shall reset its receiving mechanism to the beginning of a channel frame.	

REQ	**2.175 DLL – Arbitration – DLL – Round-robin arbitration – Ignore incoming STARTs**
After a node sets START, it shall transmit data for the remainder of the area of the network frame that belongs to the particular channel, ignoring any incoming START.	

REQ	**2.176 DLL – Arbitration – DLL – Round-robin arbitration – Losing arbitration**
If a node receives START in either the frame byte that it is scheduled to set START in, or in a frame byte that is closer to the start of the current network frame, the node shall lose arbitration, shall internally mark the channel as being in use, and shall forward the incoming START as well as the rest of the channel bytes it receives.	

10.3.2 DLL — Ensuring round-robin transmit order

REQ	2.177 DLL – Arbitration – DLL – Ensuring round-robin transmit order – Replace END with WAIT
If a node that is ready to transmit a channel frame receives END, the node shall replace it with a WAIT and transmit its channel frame. NOTE Thus, all the other nodes that are ready to transmit receive WAIT. This guarantees the round-robin transmit order.	

REQ	2.178 DLL – Arbitration – DLL – Ensuring round-robin transmit order – Forward WAIT
A node that receives WAIT shall forward the WAIT and regard the channel as being in use.	

10.3.3 DLL — Examples

Figure 15, section (a) to section (c) show examples, in which all nodes setting START do so in different frame bytes within the area of the network frame that belongs to the packet channel. The node setting START in the frame byte that is closest to the start of the network frame wins arbitration.

As shown in section (a), S1 loses arbitration to the downstream node S4, since in NF_N S4 sets its START closer to the start of the network frame than S1.

As shown in section (b), if S4 sets START in the frame byte in the previous network frame NF_{N-1} and S1 sets its START in a frame byte closer to the start of the network frame in NF_N, S1 wins arbitration.

Section (c) shows nodes (S1 and S4) losing arbitration to a node (S5) setting START in NF_{N-1}.

In section (d) all nodes are arbitrating the same network frame byte and the arbitration winner is determined by the node counter value in the MOST network.

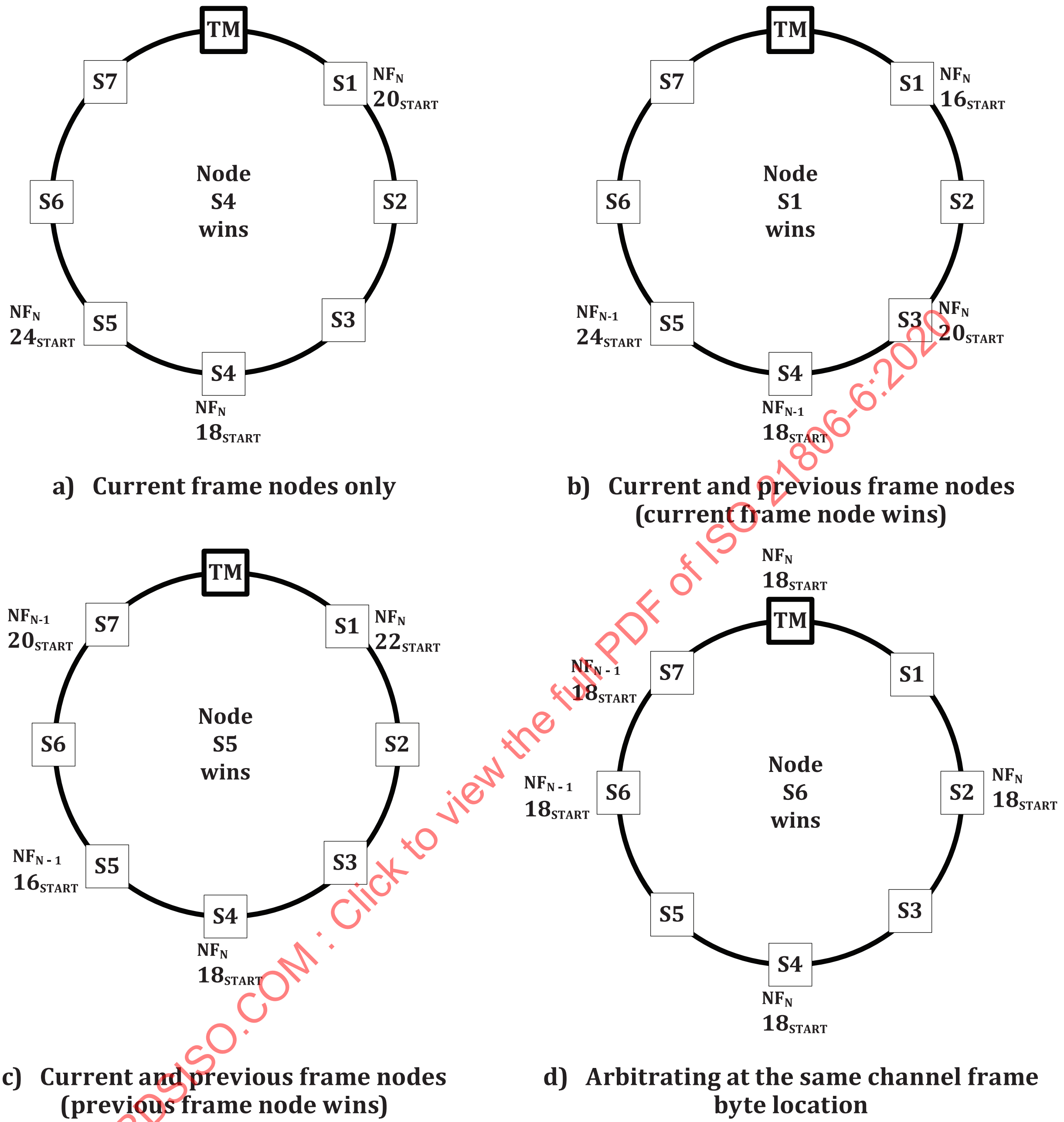


Figure 15 — Examples for arbitration on packet channel

11 DLL — Addressing

11.1 DLL — General

The 16-bit and 48-bit addressing mechanisms of the channels offer different addressing modes that direct a channel frame to one or multiple targets. Channel frames that support the PACK byte and the early ending (earlyEND) mechanism allow for Flow Control and bandwidth saving.

11.2 DLL — 16-bit address types

11.2.1 DLL — General

REQ	2.179 DLL – Addressing – DLL – 16-bit address types – Compare incoming target address
A node in receiving mode shall compare the incoming target address with its own local addresses.	

REQ	2.180 DLL – Addressing – DLL – 16-bit address types – Updating PACK byte
If one of its addresses matches the incoming target address, the receiving node shall check its readiness for receiving a channel frame and report the result by updating the PACK byte.	

REQ	2.181 DLL – Addressing – DLL – 16-bit address types – Address types
The address types for 16-bit addressing shall be in accordance with Table 31.	

Table 31 — Address types for 16-bit addressing

Value/Range	Address type	Addressing mode	Available for			
			single cast	multi-cast	control frame	packet frame
0000_{16}	Free-up address	Broadcast, blocking	no	yes	yes[a]	no[b]
0001_{16} up to $02FF_{16}$	Logical node address	Logical	yes	no	yes	yes
0300_{16} up to $03C7_{16}$	Group address	Groupcast	no	yes	yes	yes[c]
$03C8_{16}$	Blocking broadcast address	Broadcast, blocking	no	yes	yes	yes[b,c]
$03C9_{16}$ up to $03FE_{16}$	Group address	Groupcast	no	yes	yes	yes[c]
$03FF_{16}$	Non-blocking broadcast address	Broadcast, non-blocking	no	yes	yes	yes[c]
0400_{16} up to $04FF_{16}$ (0400_{16} + Pos)	Node position address	Physical	yes	no	yes	Yes
0500_{16} up to $0FEF_{16}$	Logical node address	Logical	yes	no	yes	Yes
$0FF0_{16}$	Debug address	Debug	yes	no	yes[a]	no
$0FF1_{16}$ up to $0FFF_{16}$	Logical node address	Logical	yes	no	yes	yes
1000_{16} up to $FFFF_{16}$	Reserved	Reserved	---	---	---	---

[a] Address reserved. Shall not be used as local logical node address by any node.

[b] For the packet frame, $03C8_{16}$ is working like $03FF_{16}$ (non-blocking broadcast). For the packet channel no free-up address is used.

[c] Multicast addressing is optional for packet frames.

11.2.2 DLL — Free-up address

The free-up address is part of the blocking broadcast mechanism, see 11.2.5.4.

11.2.3 DLL — Logical node address

Logical node addresses are address values for point-to-point transfers between nodes.

11.2.4 DLL — Group address

Nodes in the MOST network support an adjustable group address that allows sending channel frames to all members of a particular group.

11.2.5 DLL — Blocking broadcast address

11.2.5.1 DLL — General

Blocking broadcast helps to send a channel frame to all nodes in the MOST network.

REQ	2.182 DLL – Addressing – DLL – Blocking broadcast address – Node behaviour
Once a channel frame is sent using the blocking broadcast address, a node, which is not the sending node, shall not send channel frames.	

When a node sends channel frames to the blocking broadcast address, retries are performed, if necessary.

11.2.5.2 DLL — Entering blockage mode

REQ	2.183 DLL – Addressing – DLL – Blocking broadcast address – Entering blockage mode
As soon as a node receives the blocking broadcast address as target address, it shall enter blockage mode. NOTE This means, all attempts to send a channel frame are disabled.	

11.2.5.3 DLL — Receiving a blocking broadcast

REQ	2.184 DLL – Addressing – DLL – Blocking broadcast address – Frame handling
Reception of a blocking broadcast channel frame shall be handled in the same way as used for channel frames that are not sent in blockage mode. NOTE This includes handling of PACK, CACK and retries.	

11.2.5.4 DLL — Leaving blockage mode/free-up mechanism

REQ	2.185 DLL – Addressing – DLL – Blocking broadcast address – Free-up mechanism
For channel frames that support the free-up mechanism, a node in blockage mode shall leave blockage on any channel frame being sent on the MOST network that is not directed to $03C8_{16}$.	

A reserved logical node address (free-up address, see Table 31) is available for organizing deblocking accordingly.

REQ	2.186 DLL – Addressing – DLL – Blocking broadcast address – Do not use free-up address as local address
The free-up address shall not be used by any node in the MOST network as local address.	

REQ	2.187 DLL – Addressing – DLL – Blocking broadcast address – Sending the free-up frame
The free-up frame shall be sent by the sender of a blocking broadcast channel frame, whenever the sending procedure has ended.	

REQ	2.188 DLL – Addressing – DLL – Blocking broadcast address – End of sending procedure
The sending procedure shall end when a channel frame has either reached all target nodes, or when the last low-level retry is completed (independent of being successful or not).	

Table 32 shows an example of a free-up frame for a control channel that is 4 bytes wide.

REQ	2.189 DLL – Addressing – DLL – Blocking broadcast address – Timer-based free-up mechanism
In parallel to the releasing procedure based upon a free-up frame, there shall be an independent timer-based mechanism.	

REQ	2.190 DLL – Addressing – DLL – Blocking broadcast address – Supervisory timer
Each receiving node that enters blockage mode shall start a supervisory timer.	

REQ	2.191 DLL – Addressing – DLL – Blocking broadcast address – Unblock after NOFFAD
If no free-up frame is received NOFFAD (number of frames for auto deblock) network frames after entering the blockage mode, the receiving node shall leave blockage mode.	

The supervisory timer comes into effect once the originator of a blocking broadcast channel frame is not able to release the blocking by a free-up frame. This may happen, if the related node goes through a hardware reset during the sending procedure.

Table 32 — Example — Free-up frame on a 4 bytes wide control channel

Network frame	Control frame byte numbers	Control channel			
		Byte 1	Byte 2	Byte 3	Byte 4
1	1 to 4	START	Arb[X]	00_{16}	00_{16}
2	5 to 8	PACK	00_{16}	08_{16}	Index
3	9 to 12	SrcHigh	SrcLow	00_{16}	Reserved
4	13	END	Reserved	Reserved	Reserved

The free-up mechanism uses the earlyEND.

11.2.6 DLL — Non-blocking broadcast address

The non-blocking broadcast works like a groupcast but uses the predefined address $03FF_{16}$ and sends channel frames to all nodes.

11.2.7 DLL — Node position address

Node position addresses are based upon the node counter value as described in 8.3.2.2.

REQ	2.192 DLL – Addressing – DLL – Node position address – Obtaining
The node counter value shall be added to the base value 0400_{16} to obtain the node position address.	
NOTE It can vary due to nodes entering or leaving the MOST network.	

11.2.8 DLL — Debug address

The debug address is used for transmitting a debug PDU.

REQ	2.193 DLL – Addressing – DLL – Debug address – Do not use as local address
A node shall not use the debug address as local address.	

REQ	2.194 DLL – Addressing – DLL – Debug address – Target address $0FF0_{16}$
A node shall use target address $0FF0_{16}$ when sending debug PDUs.	

REQ	2.195 DLL – Addressing – DLL – Debug address – Low-level retries, PACK and CACK
When sending debug PDUs, a node shall not perform low-level retries and ignore PACK and CACK.	
NOTE See specifics for the PACK byte regarding debug PDUs in 9.1.	