INTERNATIONAL STANDARD

ISO 21806-4

First edition
2020-09

Road vehicles — Media Oriented Systems Transport (MOST) —

Part 4: **Transport layer and network layer**

Véhicules routiers — Système de transport axé sur les médias —

Partie 4: Couche de transport et couche réseau

Reference number
ISO 21806-4:2020(E)

Contents

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.

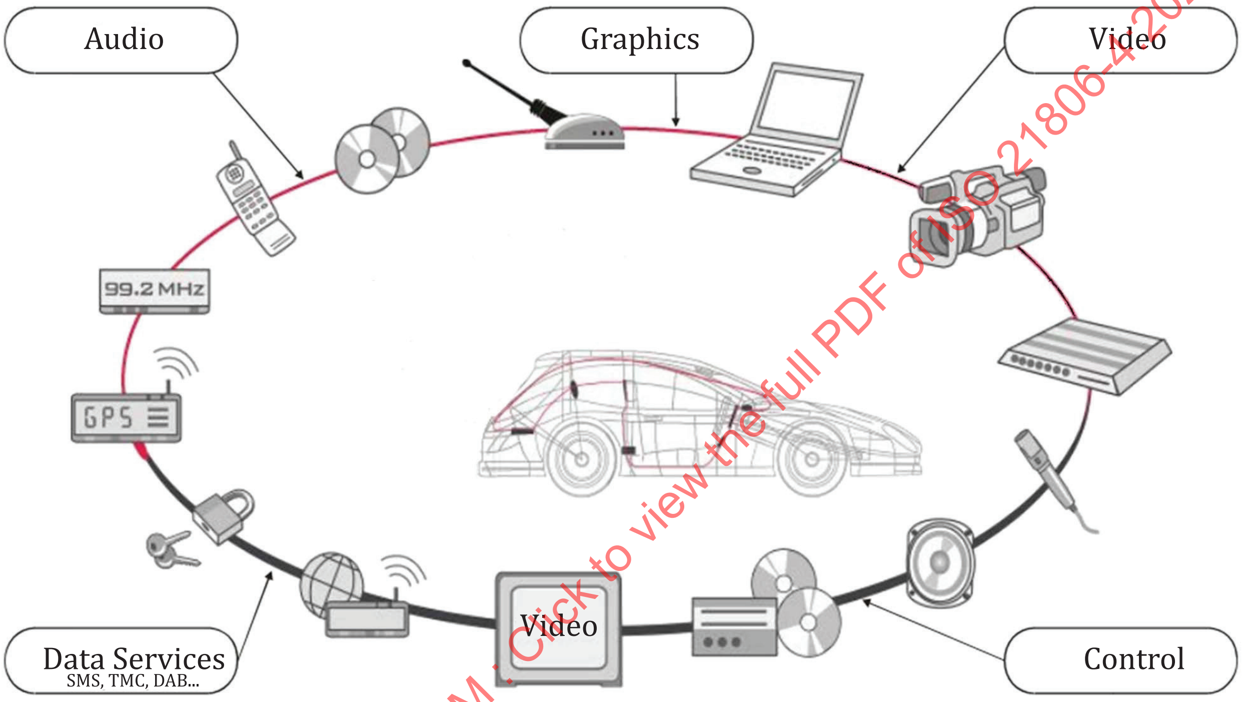


Figure 1 — MOST network example

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[6].

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[4], 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.

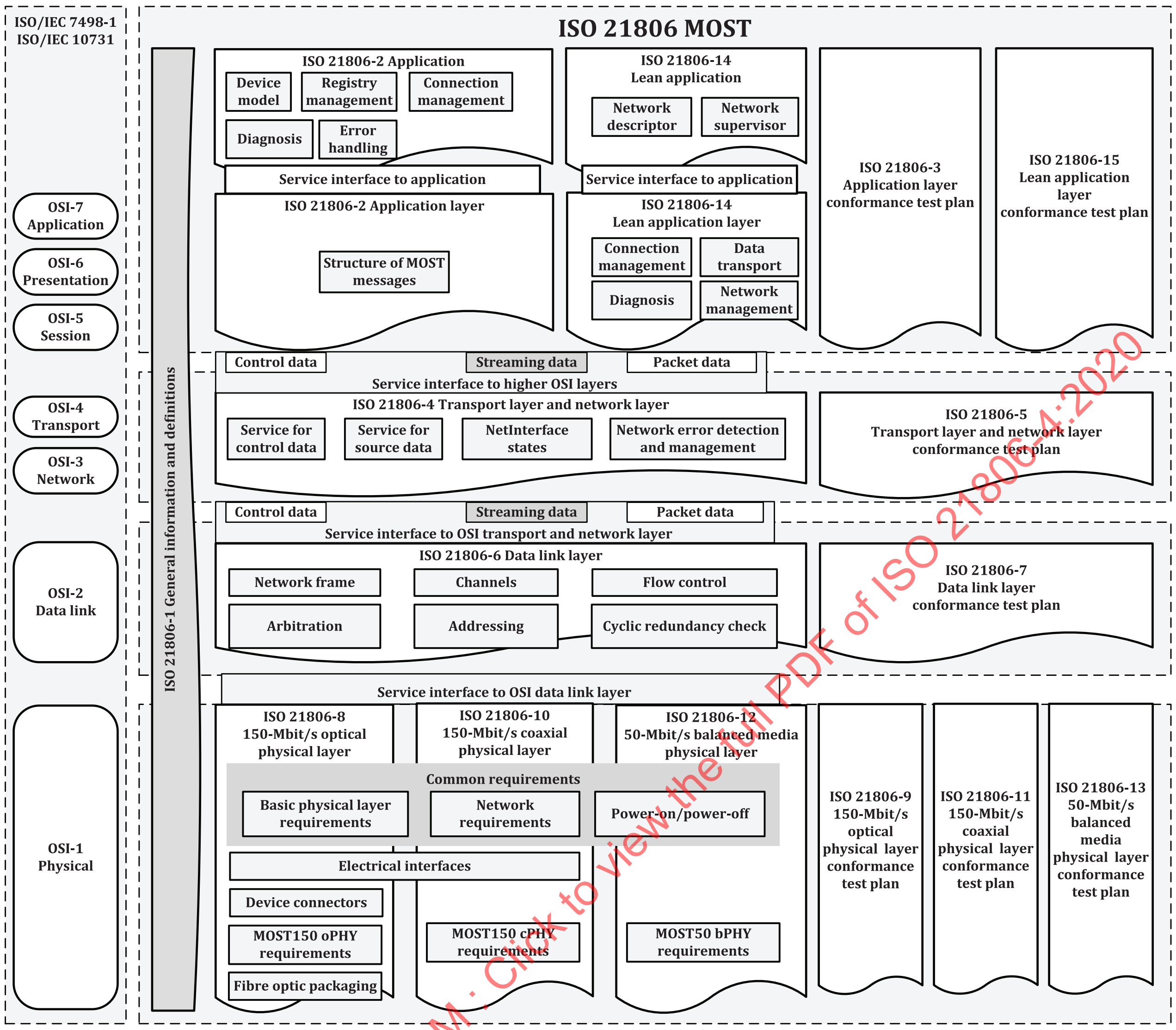

ISO/IEC 7498-1
ISO/IEC 10731
OSI-7 Application
OSI-6 Presentation
OSI-5 Session
OSI-4 Transport
OSI-3 Network
OSI-2 Data link
OSI-1 Physical
ISO 21806 MOST
ISO 21806-1 General information and definitions
ISO 21806-2 Application
Device model
Registry management
Connection management
Diagnosis
Error handling
Service interface to application
ISO 21806-2 Application layer
Structure of MOST messages
ISO 21806-14 Lean application
Network descriptor
Network supervisor
Service interface to application
ISO 21806-14 Lean application layer
Connection management
Data transport
Diagnosis
Network management
ISO 21806-3 Application layer conformance test plan
ISO 21806-15 Lean application layer conformance test plan
Control data
Streaming data
Packet data
Service interface to higher OSI layers
ISO 21806-4 Transport layer and network layer
Service for control data
Service for source data
NetInterface states
Network error detection and management
ISO 21806-5 Transport layer and network layer conformance test plan
Control data
Streaming data
Packet data
Service interface to OSI transport and network layer
ISO 21806-6 Data link layer
Network frame
Channels
Flow control
Arbitration
Addressing
Cyclic redundancy check
ISO 21806-7 Data link layer conformance test plan
Service interface to OSI data link layer
ISO 21806-8 150-Mbit/s optical physical layer
ISO 21806-10 150-Mbit/s coaxial physical layer
ISO 21806-12 50-Mbit/s balanced media physical layer
Common requirements
Basic physical layer requirements
Network requirements
Power-on/power-off
Electrical interfaces
Device connectors
MOST150 oPHY requirements
Fibre optic packaging
MOST150 cPHY requirements
MOST50 bPHY requirements
ISO 21806-9 150-Mbit/s optical physical layer conformance test plan
ISO 21806-11 150-Mbit/s coaxial physical layer conformance test plan
ISO 21806-13 50-Mbit/s balanced media physical layer conformance test plan

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 4: Transport layer and network layer

1 Scope

This document specifies technical requirements related to the MOST transport layer and network layer functionality:

— the service interface to application layer;

— the network layer services;

— the data transport mechanism;

— the dynamic behaviour of a node;

— the network error management.

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
active TimingSlave
TimingSlave that initiates a *network startup* (3.2)

3.2
network startup
network activity that commences so that all nodes in the MOST network change to `s_NetInterface_Normal_Operation`

3.3
network wake-up
process of all nodes in the MOST network exiting `s_NetInterface_Sleep`

3.4
network wake-up event
network activity or electrical wake-up line activity

3.5
passive TimingSlave
node that participates in a *network startup* (3.2), not initiating it

3.6
qualified local wake-up event
local trigger (one that does not affect all devices) that causes exiting `s_NetInterface_Sleep`

3.7
wake-up event
trigger for exiting `s_NetInterface_Sleep`

4 Symbols and abbreviated terms

4.1 Symbols

---	empty cell/undefined
L_{AMSmax}	maximum payload length for AMS
L_{CMmax}	maximum payload length for control data

4.2 Abbreviated terms

A:	action
AMS	application message service
C:	condition
`ev_`	prefix event name
`MsgID`	message identifier
NL	network layer
RBD	ring break diagnosis
`s_`	prefix state name
`SegCnt`	segment counter
`TelID`	telegram identifier
`TelLen`	telegram length
TL	transport layer

5 Conventions

This document is based on OSI service conventions as specified in ISO/IEC 10731[4].

6 Transport layer service interface to upper OSI layers

6.1 Overview of services

Figure 3 shows the transport layer (TL) service interface, which specifies the interface to the upper OSI layers, see ISO 21806-2[1].

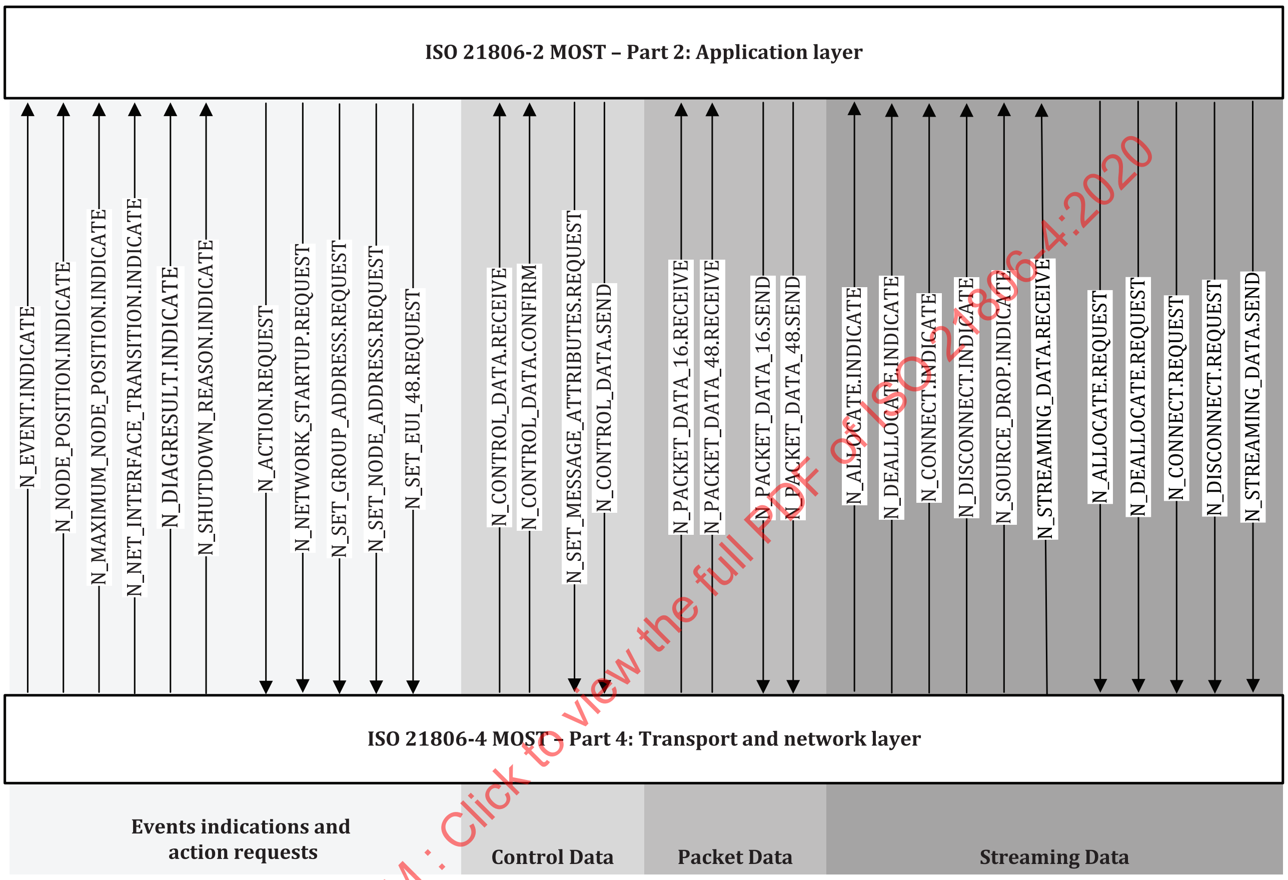


Figure 3 — Service interface to upper OSI layers

6.2 Data type definitions

REQ	4.1 Service interface – Data type definitions
The data types shall be in accordance to: — `Enum`: 8-bit enumeration; — `Unsigned Byte`: 8-bit unsigned numeric value; — `Unsigned Word`: 16-bit unsigned numeric value; — `Unsigned Long`: 32-bit unsigned numeric value; — `EUI-48`: 48-bit address value; — `Byte Array`: sequence of 8-bit aligned data.	

6.3 Parameters

6.3.1 Parameters — NL/TL to upper layers

6.3.1.1 Overview

Table 1 specifies the parameters that are used in the service interface and passed from NL/TL to upper layers.

Table 1 — Parameters passed from NL/TL to upper layers

Parameter	Data type	Description
`Application_Event`	`Enum { Unlock, Stable_Lock, Lock_Flag, Network_Change_Event, Shutdown_Flag, MOST_Output_Off, Network_Activity }`	An event that is reported to the application.
`NetInterface_Transition`	`Enum { ev_Wake_Up, ev_Sleep, ev_Start_Up, ev_Diagnosis_Start, ev_Init_Error_Shutdown, ev_Init_Diagnosis_Start, ev_Init_Ready, ev_Error_Shutdown, ev_Normal_Shutdown, ev_Diagnosis_Ready, ev_Diagnosis_End }`	A transition between NetInterface states
`DiagResult`	`Enum { No_Error, Ring_Break, Weak_Signal, Diagnosis_Inconclusive }`	DiagResult of the ring break diagnosis
`Relative_Position`	`Unsigned Byte`	Relative position of a node to a ring break
`Shutdown_Reason`	`Enum { No_Result_Available, No_Fault_Saved, Sudden_Signal_Off, Critical_Unlock }`	Shutdown reason
`Node_Position`	`Unsigned Byte`	Position of the node
`Maximum_Position`	`Unsigned Byte`	Maximum position information
`Transmission_Status`	`Enum { Success, Buffer_Full, CRC_Error, Wrong_Target }`	Transmission status that is reported back to the sender.

6.3.1.2 Application_Event

Application_Event corresponds to events that are used to notify the application about changes in lower layers, which require no additional information.

REQ	4.2 Service interface – Parameters – NL/TL to upper layers – Application_Event
The Application_Event parameter shall be of data type Enum and shall contain the value specified according to Table 2.	

Table 2 — Application_Event values

Enum value	Description
Unlock	Unlock event occurred.
Stable_Lock	Stable lock reached.
Lock_Flag	Lock flag detected.
Network_Change_Event	Network change event (NCE) occurred.
Shutdown_Flag	Shutdown flag detected.
Network_Activity_End	Network activity ends.
Network_Activity	Network activity detected.

6.3.1.3 NetInterface_Transition

NetInterface_Transition corresponds to events that are used to notify the application about transitions from one NetInterface state to another.

REQ	4.3 Service interface – Parameters – NL/TL to upper layers – NetInterface_Transition
The NetInterface_Transition values shall be of data type Enum and shall contain the value specified according to Table 3.	

Table 3 — NetInterface_Transition values

Enum value	Description
ev_Wake_Up	The ev_Wake_Up transition from s_NetInterface_Sleep to s_NetInterface_Off is taken.
ev_Sleep	The ev_Sleep transition from s_NetInterface_Off to s_NetInterface_Sleep is taken.
ev_Start_Up	The ev_Start_Up transition from s_NetInterface_Off to s_NetInterface_Init is taken.
ev_Diagnosis_Start	The ev_Diagnosis_Start transition from s_NetInterface_Off to s_NetInterface_Diagnosis is taken.
ev_Init_Error_Shutdown	The ev_Init_Error_Shutdown transition from s_NetInterface_Init to s_NetInterface_Off is taken.
ev_Init_Diagnosis_Start	The ev_Init_Diagnosis_Start transition from s_NetInterface_Init to s_NetInterface_Diagnosis is taken.
ev_Init_Ready	The ev_Init_Ready transition from s_NetInterface_Init to s_NetInterface_Normal_Operation is taken.
ev_Error_Shutdown	The ev_Error_Shutdown transition from s_NetInterface_Normal_Operation to s_NetInterface_Off is taken.
ev_Normal_Shutdown	The ev_Normal_Shutdown transition from s_NetInterface_Normal_Operation to s_NetInterface_Off is taken.
ev_Diagnosis_Ready	The ev_Diagnosis_Ready transition from s_NetInterface_Diagnosis to s_NetInterface_Normal_Operation is taken.

Table 3 *(continued)*

Enum value	Description
ev_Diagnosis_End	The ev_Diagnosis_End transition from s_NetInterface_Diagnosis to s_NetInterface_Off is taken.

6.3.1.4 DiagResult

DiagResult corresponds to the possible diagnosis to be provided to the application. The structure of DiagResult depends on the kind of diagnosis that is performed and should be adopted from the corresponding specification.

REQ	4.4 Service interface – Parameters – NL/TL to upper layers – DiagResult
The DiagResult values for ring break diagnosis, as specified in Annex A (informative), shall be of data type Enum and shall contain the value specified according to Table 4.	

Table 4 — DiagResult values for ring break diagnosis

Enum value	Description
No_Error	No error detected.
Ring_Break	Ring break detected. The result indicates the relative position of ring break in the Relative_Position parameter.
Weak_Signal	Excessive attenuation detected at the input.
Diagnosis_Inconclusive	The ring break diagnosis inconclusive.

6.3.1.5 Relative_Position

The content of the Relative_Position parameter is relevant if the DiagResult parameter contains the value Ring_Break.

REQ	4.5 Service interface – Parameters – NL/TL to upper layers – Relative_Position
The Relative_Position parameter shall be of data type Unsigned Byte and shall contain the relative position of the node to a ring break.	

6.3.1.6 Shutdown_Reason

Shutdown_Reason corresponds to the possible causes of a shutdown to be provided to the application.

REQ	4.6 Service interface – Parameters – NL/TL to upper layers – Shutdown_Reason
The Shutdown_Reason parameter shall be of data type Enum and shall contain the value specified according to Table 5.	

Table 5 — Shutdown_Reason values

Enum value	Description
No_Result_Available	Initial value of the shutdown reason.
No_Fault_Saved	Shutdown flag detected before network activity ceased.
Sudden_Signal_Off	Shutdown caused by a sudden signal off (SSO).
Critical_Unlock	Shutdown caused by a critical unlock.

6.3.1.7 Node_Position

Node_Position provides the current node position to the application.

REQ	4.7 Service interface – Parameters – NL/TL to upper layers – Node_Position
The Node_Position parameter shall be of data type Unsigned Byte and shall contain the current node position.	

6.3.1.8 Maximum_Position

Maximum_Position provides the maximum node position information to the application.

REQ	4.8 Service interface – Parameters – NL/TL to upper layers – Maximum_Position
The Maximum_Position parameter shall be of data type Unsigned Byte and shall contain the maximum node position information.	

6.3.1.9 Transmission_Status

Transmission_Status corresponds to the possible outcomes of a message transmission to be provided to the application.

REQ	4.9 Service interface – Parameters – NL/TL to upper layers – Transmission_Status
The Transmission_Status parameter shall be of data type Enum and shall contain the value specified according to Table 6.	

Table 6 — Transmission_Status values

Enum value	Description
Success	The message is transmitted successfully.
Buffer_Full	The receiver buffer is full.
CRC_Error	A CRC error occurred.
Wrong_Target	There is no such target.
Segmentation_Error_01	First segment missing, that is, the first telegram of a segmented transfer is not received.
Segmentation_Error_02	Target node does not provide enough buffers to handle a message of this size.
Segmentation_Error_03	Unexpected segment number
Segmentation_Error_04	Too many unfinished segmentation messages pending.
Segmentation_Error_05	Timeout while waiting for next segment.
Segmentation_Error_06	Node not capable of handling segmented transfers.
Segmentation_Error_07	Segmented transfer has not been finished before the arrival of another message with identical MsgID sent by the same node.

6.3.2 Parameters — Upper layers to TL/NL

6.3.2.1 Overview

Table 7 specifies the parameters that are used in the service interface and received from upper layers.

Table 7 — Parameters received from upper layers to TL/NL

Parameter	Data type	Description
`Application_Request`	`Enum { cmd_Network_Startup, cmd_Off_Request, cmd_Emergency_Shutdown, cmd_Start_Diagnosis, cmd_DiagResult, cmd_Shutdown_Reason }`	A request from the application; it is passed from upper layers to NL/TL.
`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 data transmission.
`Priority`	`Unsigned Byte`	The priority for a control data transmission
`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 Application_Request

`Application_Request` corresponds to actions that the application can request from lower layers. These requests require no additional information.

REQ	4.10 Service interface – Parameters – Upper layers to TL/NL – Application_Request
The `Application_Request` values shall be of data type `Enum` and shall contain the value specified according to Table 8.	

Table 8 — Application_Request values

Enum value	Description
`cmd_Network_Startup`	Network startup requested.
`cmd_Off_Request`	Off Request issued.
`cmd_Emergency_Shutdown`	Emergency shutdown requested.
`cmd_Start_Diagnosis`	Start of diagnosis requested. The network owner determines how the kind of diagnosis to be performed is selected, for example, through an additional action or by hard coding it.
`cmd_DiagResult`	`DiagResult` of the diagnosis requested.
`cmd_Shutdown_Reason`	Shutdown reason requested.

6.3.2.3 Network_Startup_Type

The `Network_Startup_Type` corresponds to the two different startup types, which can be chosen for the MOST network controller.

REQ	4.11 Service interface – Parameters – Upper layers to TL/NL – Network_Startup_Type
The `Network_Startup_Type` parameter shall be of data type `Enum` and shall contain the value specified according to Table 9.	

Table 9 — Network_Startup_Type values

Enum value	Description
`TimingMaster`	For startup, configure the MOST network controller as TimingMaster.
`TimingSlave`	For startup, configure the MOST network controller as TimingSlave.

6.3.2.4 Number_Of_Retries

`Number_Of_Retries` is used to set the maximum permissible number of low-level retries for a particular message.

REQ	4.12 Service interface – Parameters – Upper layers to TL/NL – Number_Of_Retries
The `Number_Of_Retries` parameter shall be of data type `Unsigned Byte` and shall contain the maximum permissible number of low-level retries for a particular message.	

6.3.2.5 Priority

`Priority` is used by the application to set the priority for a particular message.

REQ	4.13 Service interface – Parameters – Upper layers to TL/NL – Priority
The `Priority` parameter shall be of data type `Unsigned Byte` and shall contain priority for a particular message.	

6.3.2.6 Group_Address

`Group_Address` is an `Unsigned Word` parameter, which is used to set the group address of the node.

REQ	4.14 Service interface – Parameters – Upper layers to TL/NL – Group_Address
The `Group_Address` parameter shall be of data type `Unsigned Word` and shall contain the group address of the node.	

6.3.2.7 Node_Address

`Node_Address` is an `Unsigned Word` parameter, which is used to set the logical node address of the node.

REQ	4.15 Service interface – Parameters – Upper layers to TL/NL – Node_Address
The Node_Address parameter shall be of data type Unsigned Word and shall contain the logical node address of the node.	

6.3.2.8 EUI_48

`EUI_48` is used to set the MAC address of the node.

REQ	4.16 Service interface – Parameters – Upper layers to TL/NL – EUI_48
The `EUI_48` parameter shall be of data type `EUI-48` and shall contain the MAC address of the node.	

6.3.2.9 Bandwidth

`Bandwidth` is used to request the allocation of the corresponding number of bytes in the network frame.

REQ	4.17 Service interface – Parameters – Upper layers to TL/NL – Bandwidth
The `Bandwidth` parameter shall be of data type `Unsigned Word` and shall contain the number of bytes in the network frame that are requested for allocation.	

6.3.3 Parameters – NL/TL to upper layers and upper layers to TL/NL

6.3.3.1 Overview

Table 10 provides an overview of the parameters that are used in the specified service interface and used in both directions.

Table 10 — Parameters passed from NL/TL to upper layers and upper layers to TL/NL

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 contained in the same service interface.
`Data`	`Byte Array`	A data field, whose length is determined by the `Length` parameter.
`MsgID`	`Unsigned Long`	The MsgID field for control data
`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
`Destination_MAC_Address`	`EUI-48`	A 48-bit target address
`Source_MAC_Address`	`EUI-48`	A 48-bit source address

6.3.3.2 Media_Interface_ID

`Media_Interface_ID` is used to unambiguously identify a media interface as source or target of streaming data.

REQ	**4.18 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Media_Interface_ID**
The `Media_Interface_ID` parameter shall be of data type `Unsigned Word` and shall identify a media interface.	

6.3.3.3 Length

`Length` is used to provide the size of the data field that is contained in the same service interface.

REQ	**4.19 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Length**
The `Length` parameter shall be of data type `Unsigned Byte` and shall contain the size of the data field that is contained in the same service interface.	

6.3.3.4 Data

`Data` is used as a wrapper for payload that requires no interpretation in the context of the service interface that contains it.

REQ	**4.20 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Data**
The `Data` parameter shall be of data type `Byte Array`.	

6.3.3.5 MsgID

`MsgID` is used to identify the target of a control frame.

REQ	**4.21 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – MsgID**
The `MsgID` parameter shall be of data type `Unsigned Long` and shall identify the target of a control frame.	

6.3.3.6 Session_ID

`Session_ID` is an `Unsigned Word` parameter, which is used to unambiguously identify an instance of a control frame that is passed down by the application. The session ID is generated by the application and used when determining the outcome of a transmission attempt.

REQ	**4.22 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Session_ID**
The `Session_ID` parameter shall be of data type `Unsigned Word` and shall identify an instance of a control frame.	

6.3.3.7 Target_Address

`Target_Address` is used to fill the target address field for 16-bit addressing for control data and packet data.

REQ	**4.23 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Target_Address**
The `Target_Address` parameter shall be of data type `Unsigned Word` and shall contain the target address for 16-bit addressing for control data and packet data.	

6.3.3.8 Source_Address

`Source_Address` is used to fill the source address field for 16-bit addressing for control data and packet data.

REQ	**4.24 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Source_Address**
The `Source_Address` parameter shall be of data type `Unsigned Word` and shall contain the source address for 16-bit addressing for control data and packet data.	

6.3.3.9 Destination_MAC_Address

`Destination_MAC_Address` is used to fill the destination address field for 48-bit addressing.

REQ	**4.25 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Destination_MAC_Address**
The `Destination_MAC_Address` parameter shall be of data type `EUI-48` and shall contain the destination address for 48-bit addressing.	

6.3.3.10 Source_MAC_Address

`Source_MAC_Address` is used to fill the source address field for 48-bit addressing.

REQ	**4.26 Service interface – Parameters – NL/TL to upper layers and upper layers to TL/NL – Source_MAC_Address**
The `Source_MAC_Address` parameter shall be of data type `EUI-48` and shall contain the source address for 48-bit addressing.	

6.4 Event indications and action requests

6.4.1 N_EVENT.INDICATE

REQ	4.27 Service interface – Event indications and action requests – N_EVENT.INDICATE

The `N_EVENT.INDICATE` shall be passed from NL/TL to upper layers to indicate that an event has occurred.

`N_EVENT.INDICATE` shall comply with the following structure:

```
N_EVENT.INDICATE{
    Application_Event
}
```

6.4.2 N_NODE_POSITION.INDICATE

REQ	4.28 Service interface – Event indications and action requests – N_NODE_POSITION.INDICATE

The `N_NODE_POSITION.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate the node position.

`N_NODE_POSITION.INDICATE` shall comply with the following structure:

```
N_NODE_POSITION.INDICATE{
    Node_Position
}
```

6.4.3 N_MAXIMUM_NODE_POSITION.INDICATE

REQ	4.29 Service interface – Event indications and action requests – N_MAXIMUM_NODE_POSITION.INDICATE

The `N_MAXIMUM_NODE_POSITION.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate the maximum node position.

`N_MAXIMUM_NODE_POSITION.INDICATE` shall comply with the following structure:

```
N_MAXIMUM_NODE_POSITION.INDICATE{
    Maximum_Position
}
```

6.4.4 N_NET_INTERFACE_TRANSITION.INDICATE

REQ	4.30 Service interface – Event indications and action requests – N_NET_INTERFACE_TRANSITION.INDICATE

The `N_NET_INTERFACE_TRANSITION.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate that an event has occurred.

N `_NET_INTERFACE_TRANSITION.INDICATE` shall comply with the following structure:

```
N_EVENT.INDICATE{
    NetInterface_Transition
}
```

6.4.5 N_DIAGRESULT.INDICATE

The content of the `Relative_Position` parameter is relevant if the `DiagResult` parameter contains the value `Ring_Break`.

REQ | **4.31 Service interface – Event indications and action requests – N_DIAGRESULT.INDICATE**

The `N_DIAGRESULT.INDICATE` primitive shall be passed from NL/TL to upper layers to provide the `DiagResult`.

`N_DIAGRESULT.INDICATE` shall comply with the following structure:

```
N_DIAGRESULT.INDICATE{
    DiagResult
    Relative_Position
}
```

6.4.6 N_SHUTDOWN_REASON.INDICATE

`N_SHUTDOWN_REASON.INDICATE` is returned as response to `N_ACTION.REQUEST` when the `Application_Request` parameter contains the value `Shutdown_Reason`.

REQ | **4.32 Service interface – Event indications and action requests – N_SHUTDOWN_REASON.INDICATE**

The `N_SHUTDOWN_REASON.INDICATE` primitive shall be passed from NL/TL to upper layers to provide the shut-down reason.

`N_SHUTDOWN_REASON.INDICATE` shall comply with the following structure:

```
N_SHUTDOWN_REASON.INDICATE{
    Shutdown_Reason
}
```

6.4.7 N_ACTION.REQUEST

REQ | **4.33 Service interface – Event indications and action requests – N_ACTION.REQUEST**

The `N_ACTION.REQUEST` primitive shall be passed from upper layers to NL/TL to trigger execution of a request.

`N_ACTION.REQUEST` shall comply with the following structure:

```
N_ACTION.REQUEST{
    Application_Request
}
```

6.4.8 N_NETWORK_STARTUP.REQUEST

REQ | **4.34 Service interface – Event indications and action requests – N_NETWORK_STARTUP.REQUEST**

The `N_NETWORK_STARTUP.REQUEST` primitive shall be passed from upper layers to NL/TL to trigger network startup.

`N_NETWORK_STARTUP.REQUEST` shall comply with the following structure:

```
N_NETWORK_STARTUP.REQUEST{
    Network_Startup_Type
}
```

6.4.9 N_SET_GROUP_ADDRESS.REQUEST

REQ **4.35 Service interface – Event indications and action requests – N_SET_GROUP_ADDRESS.REQUEST**

The `N_SET_GROUP_ADDRESS.REQUEST` primitive shall be passed from upper layers to NL/TL to set the group address of the node.

`N_SET_GROUP_ADDRESS.REQUEST` shall comply with the following structure:

```
N_SET_GROUP_ADDRESS.REQUEST{
    Group_Address
}
```

6.4.10 N_SET_NODE_ADDRESS.REQUEST

REQ **4.36 Service interface – Event indications and action requests – N_SET_NODE_ADDRESS.REQUEST**

The `N_SET_NODE_ADDRESS.REQUEST` primitive shall be passed from upper layers to NL/TL to set the logical node address of the node.

`N_SET_NODE_ADDRESS.REQUEST` shall comply with the following structure:

```
N_SET_GROUP_ADDRESS.REQUEST{
    Group_Address
}
```

6.4.11 N_SET_EUI_48.REQUEST

REQ **4.37 Service interface – Event indications and action requests – N_SET_EUI_48.REQUEST**

The `N_SET_EUI_48.REQUEST` primitive shall be passed from upper layers to NL/TL to set the EUI-48 of the node.

`N_SET_EUI_48.REQUEST` shall comply with the following structure:

```
N_SET_EUI_48.REQUEST{
    EUI_48
}
```

6.5 Control data

6.5.1 N_CONTROL_DATA.RECEIVE

REQ **4.38 Service interface – Event indications and action requests – N_CONTROL_DATA.RECEIVE**

The `N_CONTROL_DATA.RECEIVE` primitive shall be passed from NL/TL to upper layers to receive control data.

`N_CONTROL_DATA.RECEIVE` shall comply with the following structure:

```
N_CONTROL_DATA.RECEIVE{
    Target_Address
    Source_Address
    MsgID
    Length
    Data
}
```

6.5.2 N_CONTROL_DATA.CONFIRM

REQ	4.39 Service interface – Event indications and action requests – N_CONTROL_DATA.CONFIRM

The N_CONTROL_DATA.CONFIRM primitive shall be passed from NL/TL to upper layers to provide the transmission status for sent control data.

N_CONTROL_DATA.CONFIRM shall comply with the following structure:

```
N_CONTROL_DATA.CONFIRM{
    Session_ID
    Transmission_Status
}
```

6.5.3 N_SET_MESSAGE_ATTRIBUTES.REQUEST

REQ	4.40 Service interface – Event indications and action requests – N_SET_MESSAGE_ATTRIBUTES.REQUEST

The N_SET_MESSAGE_ATTRIBUTES.REQUEST primitive shall be passed from upper layers to NL/TL to set the attributes for one control frame.

N_SET_MESSAGE_ATTRIBUTES.REQUEST shall comply with the following structure:

```
N_SET_MESSAGE_ATTRIBUTES.REQUEST{
    Session_ID
    Number_Of_Retries
    Priority
}
```

6.5.4 N_CONTROL_DATA.SEND

REQ	4.41 Service interface – Event indications and action requests – N_CONTROL_DATA.SEND

The N_CONTROL_DATA.SEND primitive shall be passed from upper layers to NL/TL to send control data.

N_CONTROL_DATA.SEND shall comply with the following structure:

```
N_CONTROL_DATA.SEND{
    Session_ID
    Target_Address
    MsgID
    Length
    Data
}
```

6.6 Packet data

6.6.1 16-bit addressing

6.6.1.1 N_PACKET_DATA_16.RECEIVE

REQ	4.42 Service interface – Event indications and action requests – N_PACKET_DATA_16.RECEIVE

The N_PACKET_DATA_16.RECEIVE primitive shall be passed from NL/TL to upper layers to receive packet data with 16-bit addressing.

N_PACKET_DATA_16.RECEIVE shall comply with the following structure:

```
N_PACKET_DATA_16.RECEIVE{
    Target_Address
    Source_Address
    Length
    Data
}
```

6.6.1.2 N_PACKET_DATA_16.SEND

REQ	4.43 Service interface – Event indications and action requests – N_PACKET_DATA_16.SEND

The N_PACKET_DATA_16.SEND primitive shall be passed from upper layers to NL/TL to send packet data with 16-bit addressing.

N_PACKET_DATA_16.SEND shall comply with the following structure:

```
N_PACKET_DATA_16.SEND{
    Target_Address
    Length
    Data
}
```

6.6.2 48-bit addressing

6.6.2.1 N_PACKET_DATA_48.RECEIVE

REQ	4.44 Service interface – Event indications and action requests – N_PACKET_DATA_48.RECEIVE

The N_PACKET_DATA_48.RECEIVE primitive shall be passed from NL/TL to upper layers to receive packet data with 48 bit addressing.

N_PACKET_DATA_48.RECEIVE shall comply with the following structure:

```
N_PACKET_DATA_48.RECEIVE{
    Destination_MAC_Address
    Source_MAC_Address
    Length
    Data
}
```

6.6.2.2 N_PACKET_DATA_48.SEND

REQ	4.45 Service interface – Event indications and action requests – N_PACKET_DATA_48.SEND

The N_PACKET_DATA_48.SEND primitive shall be passed from upper layers to NL/TL to send packet data with 48-bit addressing.

N_PACKET_DATA_48.SEND shall comply with the following structure:

```
N_PACKET_DATA_48.SEND{
    Destination_MAC_Address
    Length
    Data
}
```

6.7 Streaming data

6.7.1 N_ALLOCATE.INDICATE

REQ	4.46 Service interface – Event indications and action requests – N_ALLOCATE.INDICATE

The N_ALLOCATE.INDICATE primitive shall be passed from NL/TL to upper layers to indicate the allocation of bandwidth for source data. The Session_ID parameter shall be set to the value provided in N_ALLOCATE.REQUEST.

N_ALLOCATE.INDICATE shall comply with the following structure:

```
N_ALLOCATE.INDICATE{
    Session_ID
    Media_Interface_ID
}
```

6.7.2 N_DEALLOCATE.INDICATE

REQ	4.47 Service interface – Event indications and action requests – N_DEALLOCATE.INDICATE

The `N_DEALLOCATE.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate that allocated bandwidth is freed. If `N_DEALLOCATE.INDICATE` is not related to `N_DEALLOCATE.REQUEST`, the `Session_ID` parameter shall be set to $\texttt{FFFF}_{16}$.If `N_DEALLOCATE.INDICATE` is caused by `N_DEALLOCATE.REQUEST`, the `Session_ID` parameter shall be set to the value provided in `N_DEALLOCATE.REQUEST`.

`N_DEALLOCATE.INDICATE` shall comply with the following structure:

```
N_DEALLOCATE.INDICATE {
    Session_ID
    Media_Interface_ID
}
```

6.7.3 N_CONNECT.INDICATE

REQ	4.48 Service interface – Event indications and action requests – N_CONNECT.INDICATE

The `N_CONNECT.INDICATE` primitive shall be passed from NL/TL to upper layers 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 `N_CONNECT.REQUEST`.

`N_CONNECT.INDICATE` shall comply with the following structure:

```
N_CONNECT.INDICATE {
    Session_ID
    Media_Interface_ID
}
```

6.7.4 N_DISCONNECT.INDICATE

REQ	4.49 Service interface – Event indications and action requests – N_DISCONNECT.INDICATE

The `N_DISCONNECT.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate that a sink is disconnected from a data stream provided by a source. If `N_DISCONNECT.INDICATE` is not related to `N_DISCONNECT.REQUEST`, the `Session_ID` parameter shall be set to $\texttt{FFFF}_{16}$. If `N_DISCONNECT.INDICATE` is caused by `N_DISCONNECT.REQUEST`, the `Session_ID` parameter shall be set to the value provided in `N_DISCONNECT.REQUEST`.

`N_DISCONNECT.INDICATE` shall comply with the following structure:

```
N_DISCONNECT.INDICATE {
    Session_ID
    Media_Interface_ID
}
```

6.7.5 N_SOURCE_DROP.INDICATE

REQ	4.50 Service interface – Event indications and action requests – N_SOURCE_DROP.INDICATE

The `N_SOURCE_DROP.INDICATE` primitive shall be passed from NL/TL to upper layers to indicate a source malfunction.

`N_SOURCE_DROP.INDICATE` shall comply with the following structure:

```
N_SOURCE_DROP.INDICATE{
    Media_Interface_ID
}
```

6.7.6 N_STREAMING_DATA.RECEIVE

REQ | **4.51 Service interface – Event indications and action requests – N_STREAMING_DATA.RECEIVE**

The `N_STREAMING_DATA.RECEIVE` primitive shall be passed from NL/TL to upper layers to receive streaming data.

`N_STREAMING_DATA.RECEIVE` shall comply with the following structure:

```
N_STREAMING_DATA.RECEIVE{
    Media_Interface_ID
    Length
    Data
}
```

6.7.7 N_ALLOCATE.REQUEST

REQ | **4.52 Service interface – Event indications and action requests – N_ALLOCATE.REQUEST**

The `N_ALLOCATE.REQUEST` primitive shall be passed from upper layers to NL/TL to allocate bandwidth for source data.

`N_ALLOCATE.REQUEST` shall comply with the following structure:

```
N_ALLOCATE.REQUEST{
    Session_ID
    Bandwidth
}
```

6.7.8 N_DEALLOCATE.REQUEST

REQ | **4.53 Service interface – Event indications and action requests – N_DEALLOCATE.REQUEST**

The `N_DEALLOCATE.REQUEST` primitive shall be passed from upper layers to NL/TL to free allocated bandwidth. The `Session_ID` parameter shall not be set to `FFFF`$_{16}$.

`N_DEALLOCATE.REQUEST` shall comply with the following structure:

```
N_DEALLOCATE.REQUEST {
    Session_ID
    Media_Interface_ID
}
```

6.7.9 N_CONNECT.REQUEST

REQ | **4.54 Service interface – Event indications and action requests – N_CONNECT.REQUEST**

The `N_CONNECT.REQUEST` primitive shall be passed from upper layers to NL/TL to connect a sink to a data stream provided by a source.

`N_CONNECT.REQUEST` shall comply with the following structure:

```
N_CONNECT.REQUEST {
    Session_ID
    Media_Interface_ID
}
```

6.7.10 N_DISCONNECT.REQUEST

REQ	4.55 Service interface – Event indications and action requests – N_DISCONNECT.REQUEST
The `N_DISCONNECT.REQUEST` primitive shall be passed from upper layers to NL/TL to disconnect a sink from a data stream provided by a source. The `Session_ID` parameter shall not be set to `FFFF`$_{16}$.	
`N_DISCONNECT.REQUEST` shall comply with the following structure:	

`N_DISCONNECT.REQUEST {`
`    Session_ID`
`    Media_Interface_ID`
`}`	

6.7.11 N_STREAMING_DATA.SEND

REQ	4.56 Service interface – Event indications and action requests – N_STREAMING_DATA.SEND
The `N_STREAMING_DATA.SEND` primitive shall be passed from upper layers to NL/TL to send streaming data.	
`N_STREAMING_DATA.SEND` shall comply with the following structure:	

`N_STREAMING_DATA.SEND {`
`    Media_Interface_ID`
`    Length`
`    Data`
`}`	

7 TL — Transport layer

7.1 TL — Overview

This clause comprises the TL service interface to upper layers and data transport mechanism.

7.2 TL — Data transport mechanism

7.2.1 TL — Transport service

The MOST network service provides an interface to the application with the purpose of enabling access to the MOST network.

7.2.2 TL — Service for control data

7.2.2.1 TL — Application message service (AMS)

The AMS represents the API for control data transfer.

For MOST commands and reports (see ISO 21806-2), the AMS uses the TelID field to distinguish between single transfer and segmented transfer.

AMS payload (see Figure 4) that does not exceed $L_{\text{AMSmax}} = L_{\text{CMmax}} - 6$ bytes (of the L_{CMmax} available control data payload bytes, a total of 6 bytes is reserved for `MsgID`, `TelID` and `TelLen`) is transmitted as single telegram (single transfer).

AMS payload that exceeds L_{CMmax} bytes is transmitted in multiple telegrams (segmented transfer).

REQ	4.57 TL – Service for control data – Maximum payload
A node shall not send an application message with a payload that exceeds 65 535 bytes.	
NOTE The system-specific maximum length is specified by the network design.	

The AMS uses control data (see 7.2.2.3). Application messages are structured as outlined in the following subclauses.

7.2.2.2 TL — Message identifier (MsgID)

The `MsgID` is 4 bytes wide and used for dispatching messages.

7.2.2.3 TL — Control data

Control data is transmitted on the control channel.

7.2.2.4 TL — Telegram identifier (TelID)

The `TelID` identifies the kind of telegram as listed in Table 11.

Table 11 — Use of the TelID field

TelID	Description
0	Single transfer
1	Segmented transfer: 1st telegram
2	Segmented transfer: 2nd and following telegrams, not including the final telegram
3	Segmented transfer: final telegram
4	Size-prefix for segmented transfer

REQ	4.58 TL – Service for control data – TelID 0
Every MOST node or remote-controlled node shall be able to send and receive messages with `TelID 0`.	

REQ	4.59 TL – Service for control data – TelID 1, 2, 3, and 4
Every MOST node shall be able to send and receive messages with `TelID 1, 2, 3`, and `4`.	

REQ	4.60 TL – Service for control data – TelID greater than 4
If a node receives a telegram that contains a `TelID` outside the valid range (i.e. `TelID` greater than `4`), the node shall ignore the message; no error message is sent to the originator of the invalid message.	

7.2.2.5 TL – Telegram length (TelLen)

REQ	4.61 TL – Service for control data – TelLen content
`TelLen` (telegram length) shall contain the number of bytes that follow the `TelLen` field.	

REQ	4.62 TL – Service for control data – TelLen range
`TelLen` shall not exceed the range `0` to L_{AMSmax}.	

7.2.2.6 TL — Payload

A payload of size *n* consists of the data bytes Data 0 to Data *n* - 1.

7.2.2.7 TL — Single transfer

Figure 4 shows a single transfer.

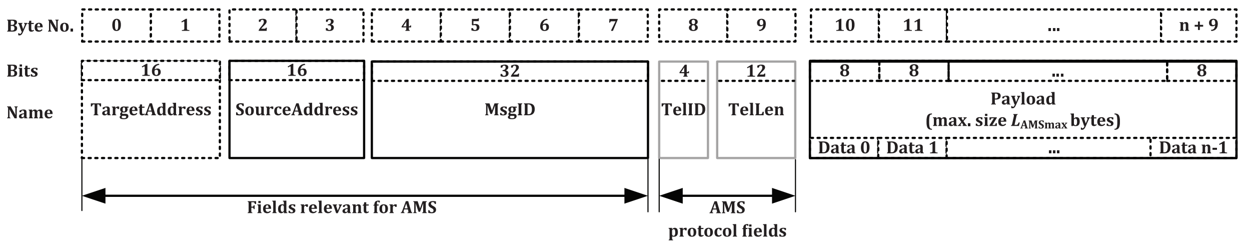


Figure 4 — Single transfer (n bytes payload)

7.2.2.8 TL — Segmented transfer

Table 12 shows an example of a segmented transfer with size-prefix.

Table 12 — Use of TelID and SegCnt in segmented transfer

Meaning	TelID	Byte 10	Byte 11
Size-prefix for segmented transfer	4	Message Size	
1st telegram	1	`SegCnt = 0`	Data 0
2nd telegram	2	`SegCnt = 1`	Data 0
...	2	...	Data 0
...	2	`SegCnt = 255`	Data 0
...	2	`SegCnt = 0`	Data 0
...	2	...	Data 0
(m - 1)th telegram	2	`SegCnt = (m - 2) mod 256`	Data 0
Final telegram	3	`SegCnt = (m - 1) mod 256`	Data 0

7.2.2.8.1 TL — Size-prefix for segmented transfer

7.2.2.8.1.1 TL — Purpose

A node announces a segmented transfer by transmission of a size-prefix, which is identified by `TelID 4`. The size-prefix is not part of the segmented transfer.

REQ	4.63 TL – Service for control data – TelID for size-prefix
Before starting a segmented transfer, a node shall send a message with `TelID 4` to the target of the segmented transfer.	

REQ	4.64 TL – Service for control data – Size-prefix content
The size-prefix shall not contain any payload apart from the message size (total payload) in bytes.	

Figure 5 shows the size-prefix for segmented transfer.

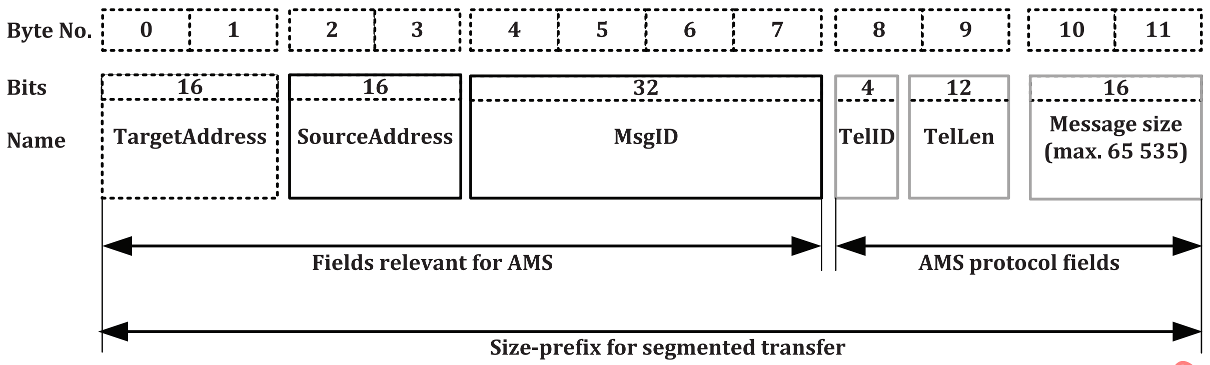


Figure 5 — Size-prefix for segmented transfer

7.2.2.8.1.2 TL — Message size

The message size field (2 byte) allows the receiver to allocate the exact amount of buffer space before the first payload chunk arrives.

The remaining available bytes of the size-prefix are reserved for future use.

REQ	4.65 TL – Service for control data – Size-prefix with TelLen less than 2
If a node receives a telegram with `TelID 4` that has a `TelLen` of less than `2`, it shall discard the telegram.	

REQ	4.66 TL – Service for control data – Segmented transfer with missing size-prefix
If a node receives a telegram with `TelID 1` without previously having received a telegram with `TelID 4`, the node shall not, as a consequence, reject the following segmented transfer and shall not send an error message to the originator. NOTE If `TelID 4` is not received, the available buffer size might not be sufficient.	

REQ	4.67 TL – Service for control data – Size-prefix with TelLen 2 or greater
If a node receives a telegram with `TelID 4` that has a `TelLen` of `2` or greater, the node shall decode the message size only.	

REQ	4.68 TL – Service for control data – Size-prefix decoding
The receiver of a telegram with `TelID 4` shall not attempt to decode beyond the message size field.	

REQ	4.69 TL – Service for control data – Size-prefix with TelLen less than 2
If a node receives a telegram with `TelID 4` that has a `TelLen` of less than `2`, it shall discard the telegram.	

7.2.2.8.2 TL — Transfer of payload

Within segmented transfers, the segments with `TelID 1` or `2` should use the entire available payload, that is, `TelLen` = L_{AMSmax}.

The final segment has `TelID 3`. Compared to single transfer, the maximum payload is reduced by one byte that is used by the segment counter (`SegCnt`).

REQ	4.70 TL – Service for control data – Segmented transfer start
The first telegram of a segmented transfer shall have `TelID 1`.	
NOTE 1 After a telegram with `TelID 1`, 0 or more `TelID 2` segments follow.	

REQ	4.71 TL – Service for control data – No empty payload for TelID 3
For segments with `TelID 3`, the available payload shall not be entirely unused, that is, `TelLen` shall be greater than 1.	

REQ	4.72 TL – Service for control data – Accept TelID 3 and TelLen=1
The receiving node shall accept telegrams with `TelID 3` and `TelLen=1` and close the message transfer.	

REQ	4.73 TL – Service for control data – Discard TelID 3 and TelLen 0
The receiving node shall discard telegrams with `TelID 3` and `TelLen=0`.	
NOTE 2 `TelID 3` with `TelLen=0` is discarded because the `SegCnt` field cannot be verified.	

Figure 6 shows one telegram of a segmented transfer.

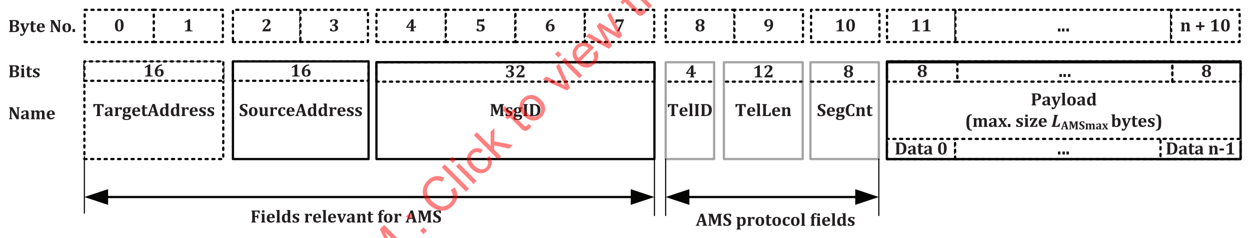


Figure 6 — One telegram (n bytes payload) of a segmented transfer

REQ	4.74 TL – Service for control data – Accept partially filled telegrams
The receiver of a segmented transfer shall accept telegrams with `TelID 1` or `2` that do not use the entire available payload.	

Figure 7 is an example of a segmented transfer where the entire available payload is used in the first telegram (`TelID = 1`, `SegCnt = 0`) and the second telegram (`TelID = 2`, `SegCnt = 1`). The maximum available payload (L_{AMSmax}) is 45; with one byte of the payload being used by the segment counter, 44 data bytes are available.

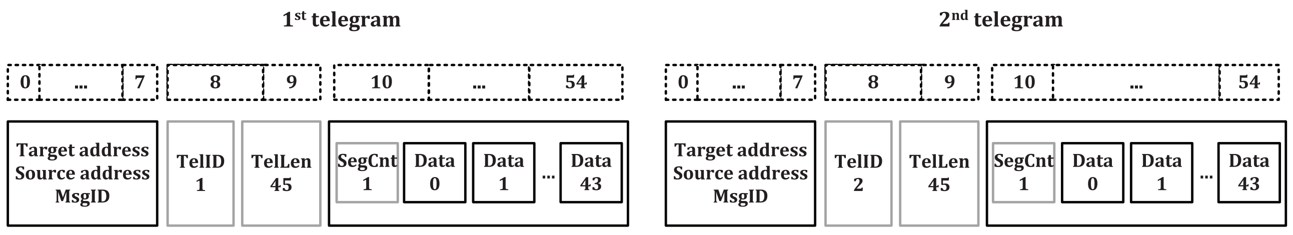


Figure 7 — Segmented transfer example with entire available payload used

Figure 8 depicts a segmented transfer where the available payload for the first and second telegram is not used entirely. Of the available 44 data bytes for the first telegram only 2 bytes are used. The second telegram only uses 19 data bytes where 44 are available.

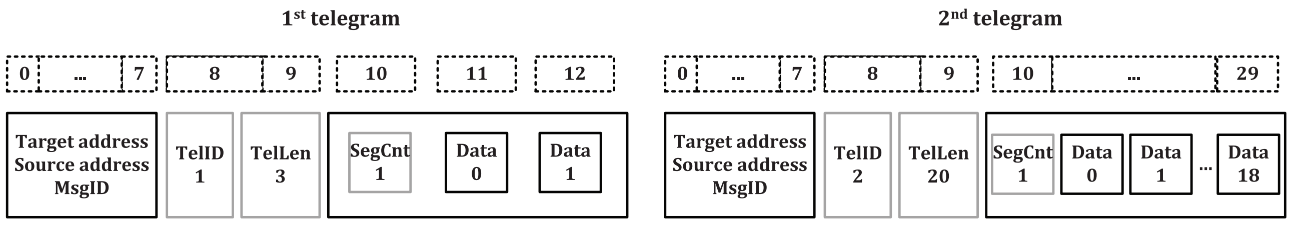


Figure 8 — Segmented transfer example with available payload not entirely used

REQ	4.75 TL – Service for control data – SegCnt counting rules
`SegCnt` shall start at 0 for the first telegram and increase by `1` with each following telegram.	

REQ	4.76 TL – Service for control data – SegCnt wraparound
When `SegCnt` reaches `255`, counting shall restart at `0`.	

REQ	4.77 TL – Service for control data – $t_{\text{WaitForNextSegment}}$
Individual telegrams of a segmented transfer shall not be more than $t_{\text{WaitForNextSegment}}$ apart.	

REQ	4.78 TL – Service for control data – No interleaving
Segmented transfers from one node to the same target with identical MsgID shall not be interleaved. NOTE 3 This means that during an ongoing segmented transfer any attempt by the sending node to start a segmented transfer with an identical signature results in an error.	

REQ	4.79 TL – Service for control data – Reporting segmented transfer errors
If an error occurs during a segmented transfer, the application message service shall provide the error cause to the application.	

7.2.3 TL — Service for source data

7.2.3.1 TL — General

“Source data” refers to packet data and streaming data.

7.2.3.2 TL — Packet data

For the transmission of large data packets, the MOST high protocol[5] may be used. The MOST high protocol is connection oriented and provides flow control and traffic shaping.

7.2.3.3 TL — Streaming data

Streaming data is accepted from lower layers and passed through to upper layers without modification.

8 NL — Network layer

8.1 NL — Overview

This clause comprises the NL service specification and the NL protocol specification.

The NL service specification specifies:

— NL states and state transitions,

— network error detection and management, and

— diagnosis.

The NL protocol specification specifies:

— the MOST network service,

— control data,

— packet data, and

— streaming data.

8.2 NL — Services

8.2.1 NL — States and state transitions

8.2.1.1 NL — Overview

This subclause comprises the NetInterface specification which consists of the following states:

— `s_NetInterface_Sleep`;

— `s_NetInterface_Off`;

— `s_NetInterface_Init`;

— `s_NetInterface_Normal_Operation`;

— `s_NetInterface_Diagnosis`.

8.2.1.2 NL — Dynamic behaviour of a node

This subclause describes the NetInterface states and reset of the TimingMaster, as well as active and passive TimingSlaves and network wake-up, startup, and shutdown behaviour.

An active TimingSlave initiates a network startup when it receives a `cmd_Network_Startup` request. A passive TimingSlave participates in a network startup but does not initiate it. The roles of active and passive TimingSlaves are not permanent but may change with every network startup.

8.2.1.3 NL — NetInterface states

8.2.1.3.1 NL — Overview

Figure 9 shows the states of the NetInterface and the events that lead to state transitions. Dashed lines indicate that a state or transition is optional.

8.2.1.3.2 to 8.2.1.3.6 specify the individual states.

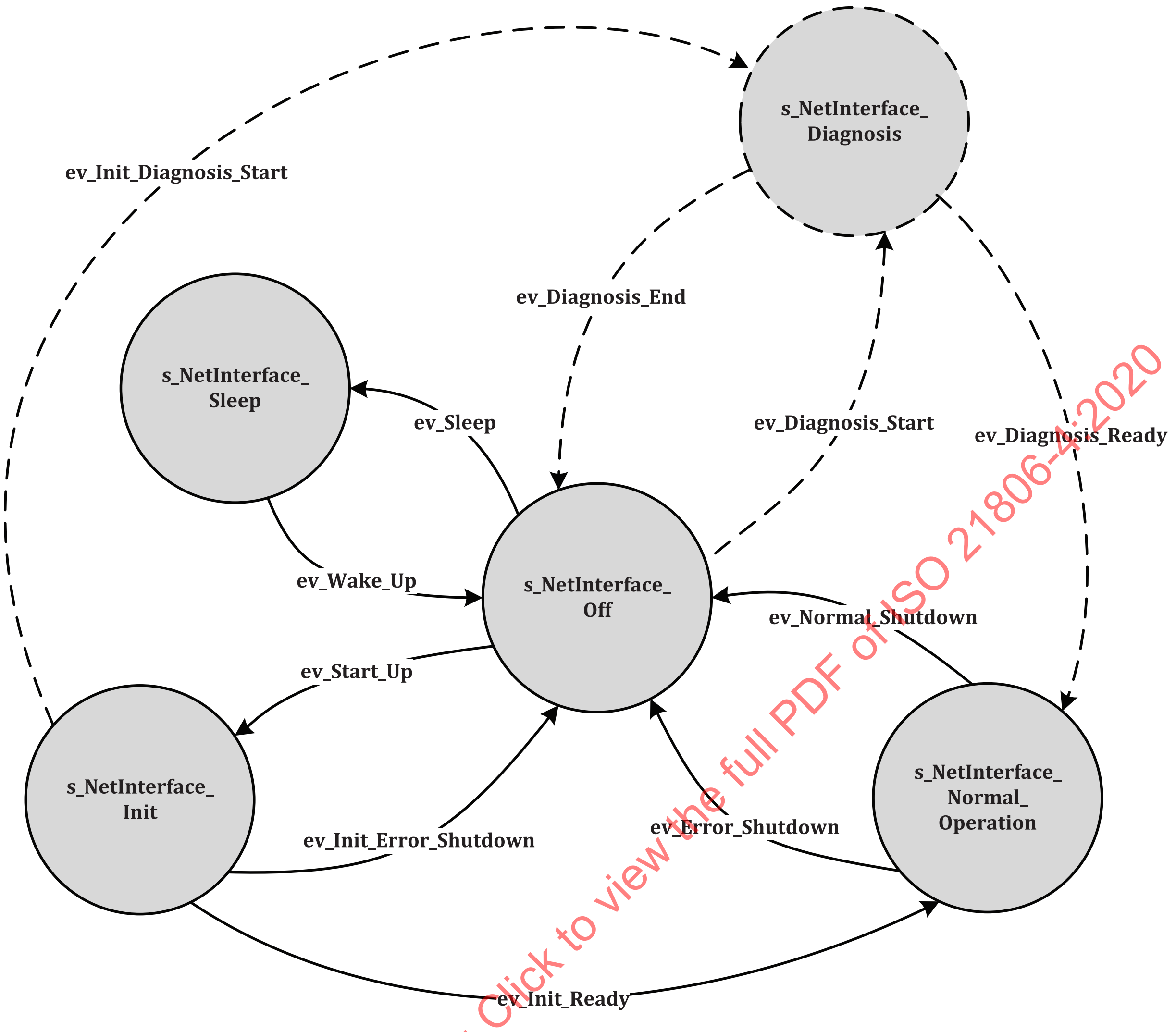


Figure 9 — Overview of the states in the NetInterface

`ev_Init_Error_Shutdown` and `ev_Error_Shutdown` can be caused by a `cmd_Emergency_Shutdown` request, which is used to initiate a shutdown due to low voltage (see 8.2.2.3) or an over-temperature condition (see ISO 21806-2).

8.2.1.3.2 NL — NetInterface state s_NetInterface_Sleep

In this state, the node consumes a minimum of power.

NOTE Typically, only the circuitry to wake-up the node remains powered.

The application may be awakened, for example, by a timer; if no qualified local wake-up event exists, the node remains in NetInterface state `s_NetInterface_Sleep`.

A local wake-up event is a local trigger (one that does not affect all devices) that caused this node to exit `s_NetInterface_Sleep`. A qualified local wake-up event exists if the local wake-up event is found to be valid (not a glitch) and results in waking up the rest of the MOST devices (network wake-up).

EXAMPLE Qualified local wake-up events include an ON switch being pressed, a non-MOST network message or reception of a wireless call.

If a qualified local wake-up event exists, the node starts up the NetInterface or asserts the electrical wake-up line to wake-up the entire MOST network. The node changes to NetInterface state `s_NetInterface_Off` with the purpose of reaching NetInterface state `s_NetInterface_Normal_Operation`.

An electrical wake-up line is any line (except the MOST electrical network) connected to every device in the MOST network which can initiate a network wake-up event, e.g. the MOST electrical control line (ECL).

As detailed in Table 13, the node may be awakened by a network wake-up event `ev_Wake_Up` (network activity or electrical wake-up line) and then start the application as part of the node initialisation. The node changes to NetInterface state `s_NetInterface_Off` with the purpose of reaching NetInterface state `s_NetInterface_Normal_Operation`.

Table 13 — Events in state s_NetInterface_Sleep

Event	Transition to	Cause
`ev_Wake_Up`	`s_NetInterface_Off`	The node changes to `s_NetInterface_Off` due to a qualified local wake-up event or a network wake-up event. A network wake-up event affects every device in the MOST network.

8.2.1.3.3 NL — NetInterface state s_NetInterface_Off

In NetInterface state `s_NetInterface_Off`, the NetInterface is switched off from the view of the MOST network.

That means that the MOST output is switched off or the bypass is closed (activated).

The MOST network controller does not necessarily need to be switched off since the application may still use function groups of it (e.g. local clock generation).

State NetInterface `s_NetInterface_Off` is left when one of the events in Table 14 occurs.

Table 14 — Events in state s_NetInterface_Off

Event	Transition to	Cause
`ev_Start_Up`	`s_NetInterface_Init`	The NetInterface changes to NetInterface state `s_NetInterface_Init` by network activity or by the application attempting to start up the MOST network (`cmd_Network_Startup`).
`ev_Diagnosis_Start`	`s_NetInterface_Diagnosis`	The NetInterface changes to the NetInterface state `s_NetInterface_Diagnosis` by connecting to power or initiated by the application.
`ev_Sleep`	`s_NetInterface_Sleep`	The node changes to NetInterface state `s_NetInterface_Sleep`, for example, because the MOST output is off for $t_{PwrSwitchOffDelay}$.

8.2.1.3.4 NL — NetInterface state s_NetInterface_Init

In this state, the NetInterface is initialising to reach `s_NetInterface_Normal_Operation`.

When entering state `s_NetInterface_Init`, the TimingMaster clears the lock flag on data link layer. This value is transferred to all MOST network controllers. As soon as the TimingMaster reaches stable lock, it sets the lock flag to true in the administrative area of the network frame.

This state is left when one of the events in Table 15 occurs.

Table 15 — Events in state s_NetInterface_Init

Event	Transition to	Cause
`ev_Init_Ready`	`s_NetInterface_Normal_Operation`	NetInterface is ready for communication (see below).
`ev_Init_Error_Shutdown`	`s_NetInterface_Off`	Error occurred during initialisation (see below). `cmd_Set_Shutdown_Flag` is not requested and $t_{\text{SSO\_Shutdown}}$ does not apply.
`ev_Init_Diagnosis_Start`	`s_NetInterface_Diagnosis`	A "Diagnosis Start" trigger is received.

When the NetInterface detects lock, it determines within t_{Lock} that stable lock is reached.

Causes for event `ev_Init_Ready`:

— in the TimingMaster: stable lock is reached;

— in a TimingSlave: stable lock is reached and the lock flag is detected.

Causes for event `ev_Init_Error_Shutdown`:

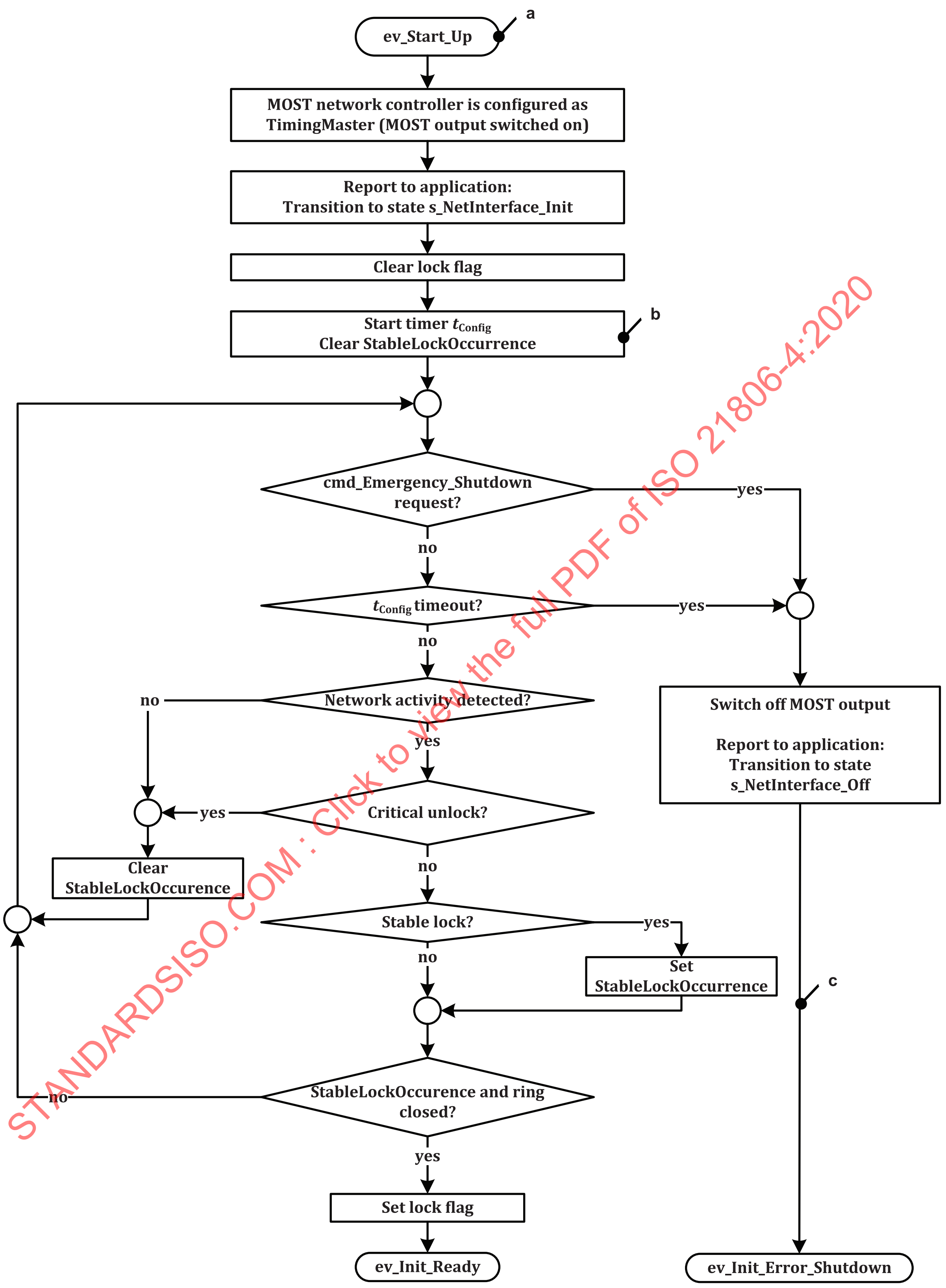
— in the TimingMaster: t_{Config} expires before stable lock is reached;

— in a passive TimingSlave: network activity disappears.

In a passive TimingSlave, the bypass of the MOST network controller is opened (deactivated) as soon as lock is recognized.

In an active TimingSlave and TimingMaster, the bypass is opened immediately after having entered this state (MOST output switched on).

Figure 10 shows the behaviour of the TimingMaster in state `s_NetInterface_Init`.


ev_Start_Up
a
MOST network controller is configured as TimingMaster (MOST output switched on)
Report to application: Transition to state s_NetInterface_Init
Clear lock flag
Start timer t_{Config}
Clear StableLockOccurrence
b
cmd_Emergency_Shutdown request?
yes
no
t_{Config} timeout?
yes
no
Network activity detected?
no
yes
Critical unlock?
yes
no
Clear StableLockOccurence
Stable lock?
yes
no
Set StableLockOccurrence
StableLockOccurence and ring closed?
no
yes
Set lock flag
ev_Init_Ready
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
c
ev_Init_Error_Shutdown

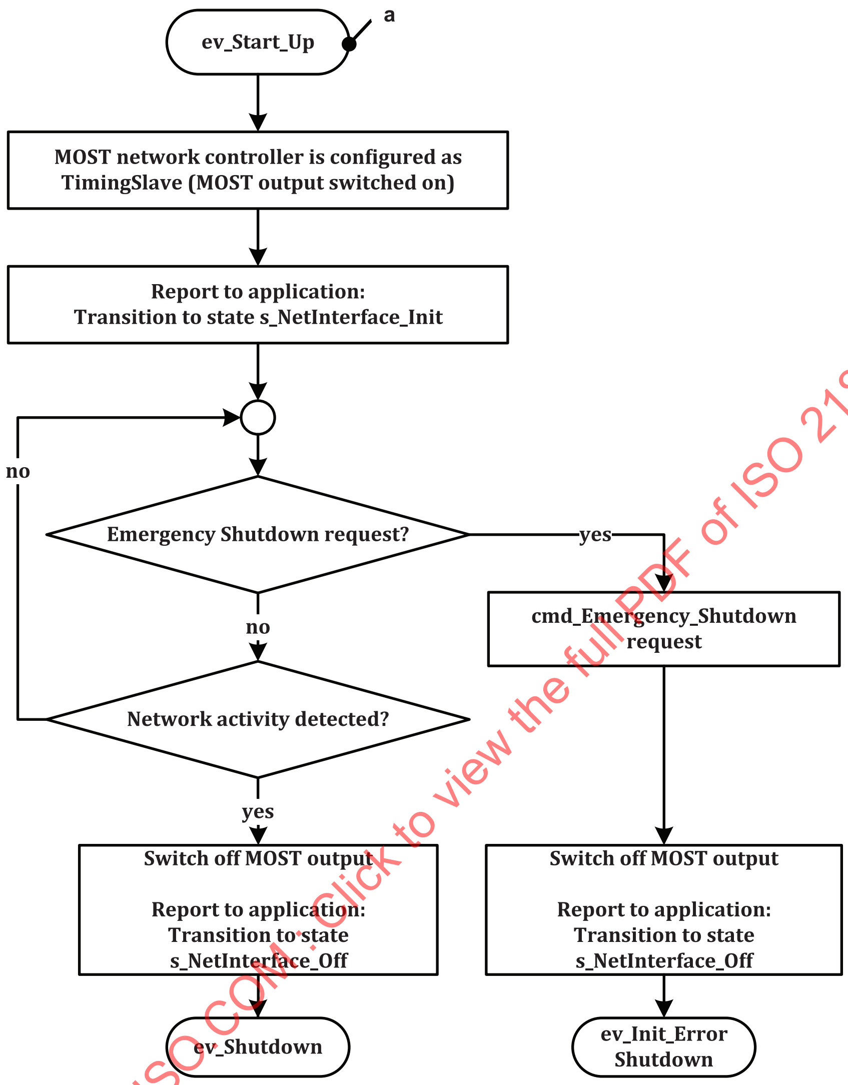
[a] `ev_Start_Up` is caused by a qualified local wake-up event, or a network wake-up event initiated by an active TimingSlave.

[b] `StableLockOccurrence` indicates that stable lock occurred at least once.

[c] The ring is closed when the TimingMaster has verified that the received MOST frame is identical with the transmitted MOST frame.

Figure 10 — TimingMaster attempting startup

Figure 11 shows the behaviour of an active TimingSlave in state `s_NetInterface_Init`.



a
ev_Start_Up
MOST network controller is configured as TimingSlave (MOST output switched on)
Report to application: Transition to state s_NetInterface_Init
no
Emergency Shutdown request?
yes
cmd_Emergency_Shutdown request
no
Network activity detected?
yes
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
ev_Shutdown
ev_Init_Error Shutdown

[a] Network startup is initiated by a qualified local wake-up event.

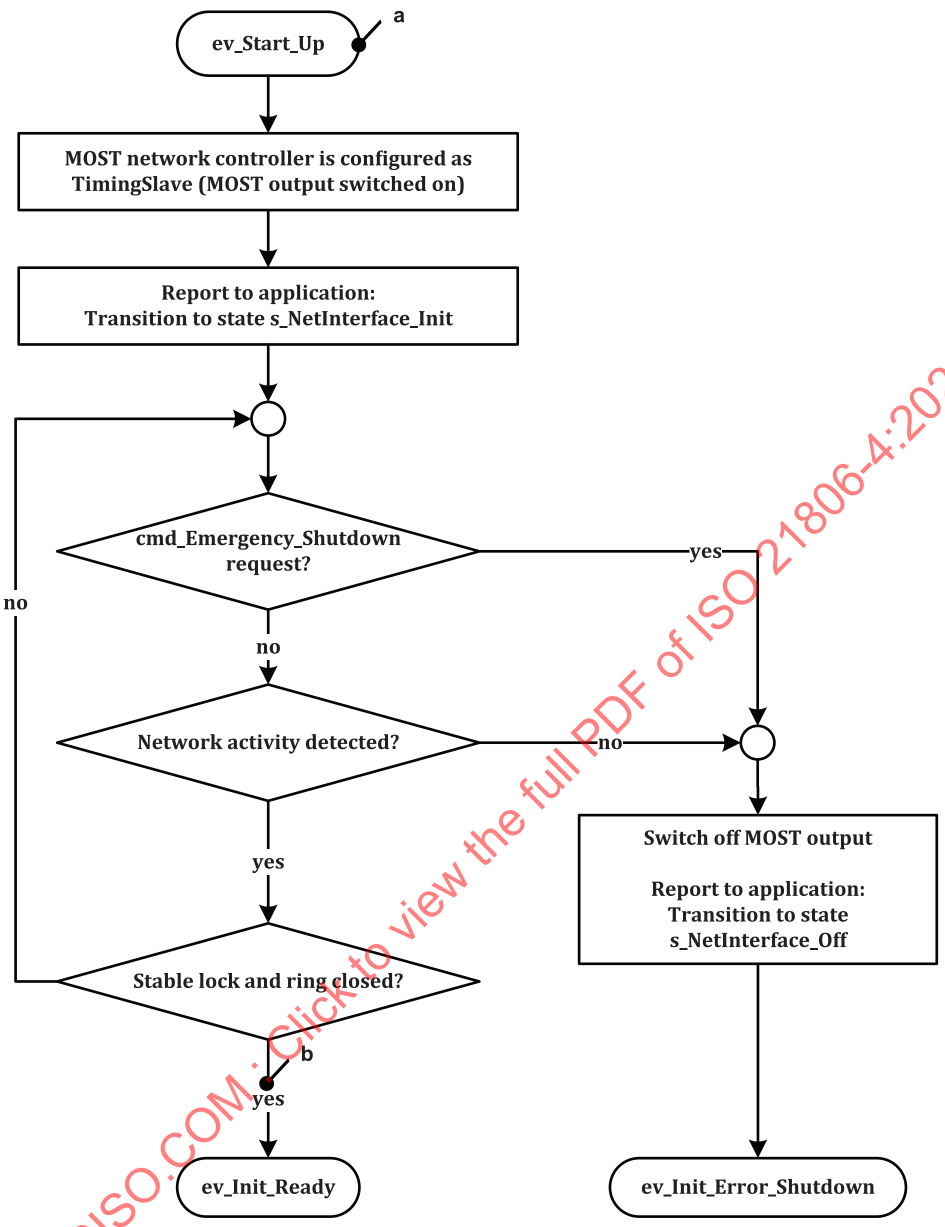
Figure 11 — Network startup by active TimingSlave[2)]

The active TimingSlave proceeds to state `s_NetInterface_Off` when it detects network activity. It then starts up as a passive TimingSlave.

Alternatively, the TimingSlave asserts the electrical wake-up line to wake-up all MOST devices.

Figure 12 shows the behaviour of a passive TimingSlave in state `s_NetInterface_Init`.

2) The `ev_Shutdown` transition is not contained in Figure 11. It leads to state `s_NetInterface_Off`.



ev_Start_Up
a
MOST network controller is configured as TimingSlave (MOST output switched on)
Report to application: Transition to state s_NetInterface_Init
cmd_Emergency_Shutdown request?
yes
no
no
Network activity detected?
no
yes
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
Stable lock and ring closed?
b
yes
ev_Init_Ready
ev_Init_Error_Shutdown

[a] `ev_Start_Up` is caused by a network wake-up event.

[b] The ring is closed as soon as the `Lock_Flag` event occurs.

Figure 12 — Passive TimingSlave starting up due to network activity

8.2.1.3.5 NL — NetInterface state s_NetInterface_Normal_Operation

This state is entered as soon as the MOST network controller starts to communicate with other nodes in the MOST network. When entering this state, the part of the application that is connected to the communication section is initialised.

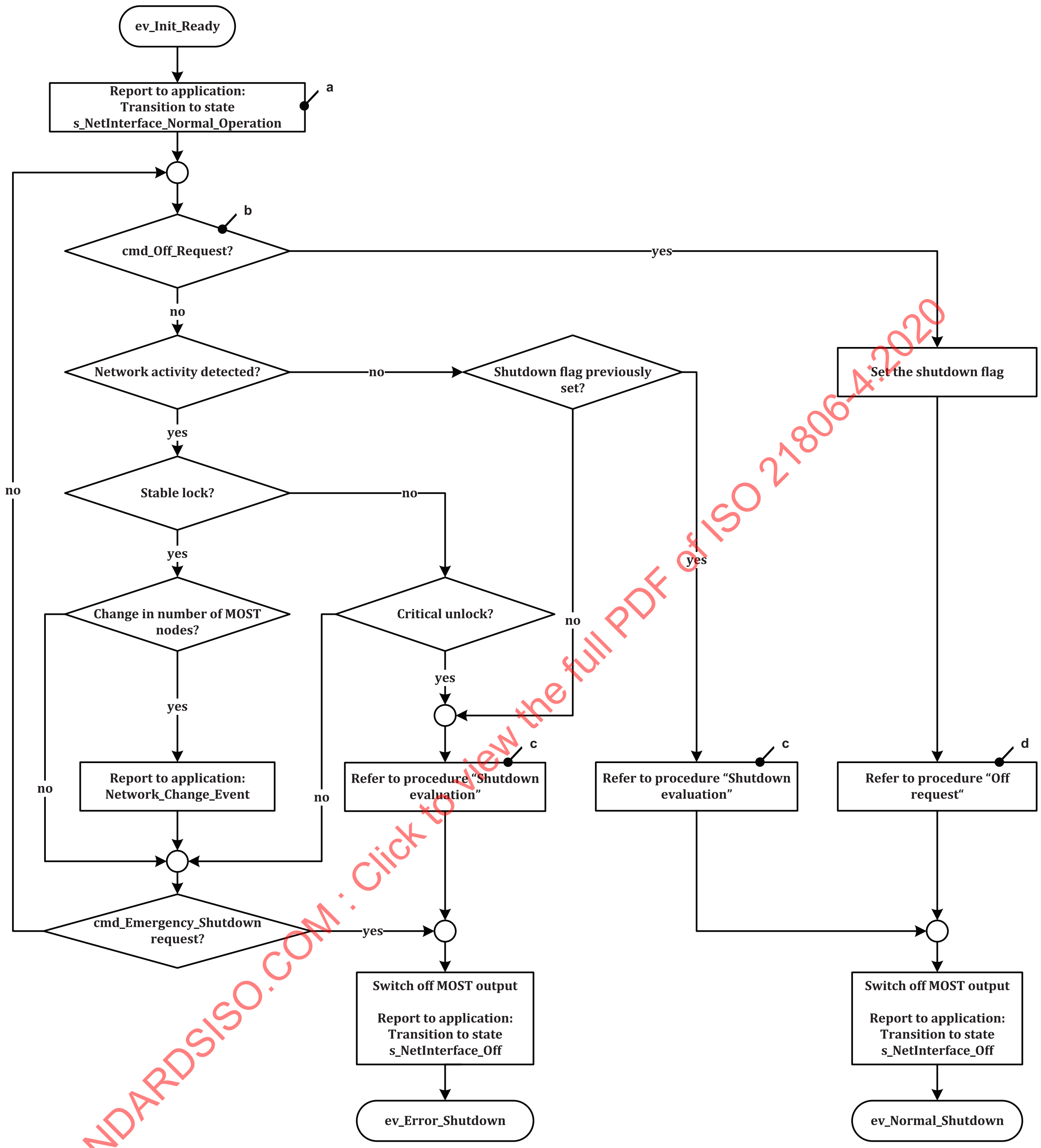
Table 16 lists the events in state `s_NetInterface_Normal_Operation`.

Table 16 — Events in state s_NetInterface_Normal_Operation

Event	Transition to	Cause
`ev_Normal_Shutdown`	`s_NetInterface_Off`	One of these causes applies: — After a `cmd_MOST_Output_Off` request, network activity ceases. — The shutdown flag is detected and network activity ceases.
`ev_Error_Shutdown`	`s_NetInterface_Off`	Critical unlock or sudden signal off.

If the shutdown flag is detected, the event `ev_Normal_Shutdown` is generated as soon as network activity ceases.

Figure 13 shows the behaviour in NetInterface state `s_NetInterface_Normal_Operation`.


ev_Init_Ready
Report to application: Transition to state s_NetInterface_Normal_Operation
a
b
cmd_Off_Request?
yes
no
Network activity detected?
no
Shutdown flag previously set?
Set the shutdown flag
yes
Stable lock?
no
yes
yes
Change in number of MOST nodes?
Critical unlock?
no
no
yes
yes
c
c
d
Report to application: Network_Change_Event
Refer to procedure "Shutdown evaluation"
Refer to procedure "Shutdown evaluation"
Refer to procedure "Off request"
no
no
cmd_Emergency_Shutdown request?
yes
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
Switch off MOST output
Report to application: Transition to state s_NetInterface_Off
ev_Error_Shutdown
ev_Normal_Shutdown

a Initialisation of the next upper layer, e.g. setting address, verifying FBlock configuration, etc.

b `cmd_Off_Request` is from the application.

c The procedure "shutdown evaluation" is shown in Figure 15.

d The procedure "off request" that is contained is shown in Figure 15.

Figure 13 — Behaviour in NetInterface state s_NetInterface_Normal_Operation

8.2.1.3.6 NL — NetInterface state s_NetInterface_Diagnosis

The diagnosis process can be started by various triggers, which are determined by the network designer.

If there is no fatal error, the NetInterface changes to NetInterface state s_NetInterface_Normal_Operation or NetInterface s_NetInterface_Off depending on the implemented diagnosis behaviour.

Table 17 lists the events in state NetInterface s_NetInterface_Diagnosis.

Table 17 — Events in state NetInterface s_NetInterface_Diagnosis

Event	Transition to	Cause
ev_Diagnosis_Ready	s_NetInterface_Normal_Operation	No fatal error and diagnosis behaviour chosen that results in NetInterface state s_NetInterface_Normal_Operation.
ev_Diagnosis_End	s_NetInterface_Off	Fatal error (defective network link or device) No fatal error and diagnosis behaviour chosen that results in NetInterface state s_NetInterface_Off.

8.2.1.4 NL — Network wake-up and startup

Waking the MOST network is done by a network wake-up event, which can be switching on the MOST output by requesting cmd_MOST_Output_On or by triggering an electrical wake-up line.

If a network-startup attempt fails and network activity does not propagate through the entire MOST network, the NetInterface of the TimingMaster changes to state s_NetInterface_Off after t_{Config}.

When a node requests cmd_MOST_Output_Off to switch off the MOST output, it starts t_{Restart}.

A node does not perform another network-startup attempt before t_{Restart} expires.

This applies even if the node recognizes network activity or any other network wake-up trigger.

If timer t_{Restart} is running, the NetInterface ignores network activity until t_{Restart} expires.

8.2.1.5 NL — Network shutdown

As a consequence of a cmd_Off_Request, the NetInterface requests cmd_MOST_Output_Off to switch off the MOST output when $t_{\text{SSO\_Shutdown}}$ (see Figure 15) expires.

REQ	3.1 NL – NetInterface states – Shutdown flag and end of network activity
If network activity ends after a node detects the presence of the shutdown flag, the node shall request cmd_MOST_Output_Off to switch off its MOST output.	

REQ	3.2 NL – NetInterface states – Network activity ends unexpectedly
If in state NetInterface s_NetInterface_Normal_Operation a node does not detect the shutdown flag and network activity ends, the node shall request cmd_Set_Shutdown_Flag.	

If a node keeps its MOST output off for the duration of $t_{\text{PwrSwitchOffDelay}}$, the node changes to s_NetInterface_Sleep.

8.2.2 NL — Network error detection and management

8.2.2.1 NL — Network failure

The unavailability of the MOST network is referred to as network failure (see 8.2.2.3).

Typical faults that result in a network failure are low voltage, ring breaks, or defective MOST physical interface units.

Apart from that, the failure of a device may be divided into two scenarios: MOST network controller failure and application failure, which is described in ISO 21806-2.

If the MOST network controller detects its own failure based on its internal watchdog timer, it performs a reset.

If the MOST network service detects the failure of the MOST network controller of a MOST node, the MOST network service requests a reset of the MOST network controller from the application.

8.2.2.2 NL — Unlock

An unlock occurs when a TimingSlave loses lock. An unlock leads to transmission errors.

Unlocks may be caused if, for example, more than one node is configured as TimingMaster, signal quality is insufficient, or a node opens or closes its bypass.

Downstream between the first node that detects an unlock and the TimingMaster, every node detects an unlock. Downstream between the TimingMaster and the first node that detects an unlock, no node detects an unlock.

REQ	**3.3 NL – Network error detection and management – Monitor unlock duration**
The transport/network layer implementation shall monitor the duration of an unlock or the occurrence of a series of unlocks.	

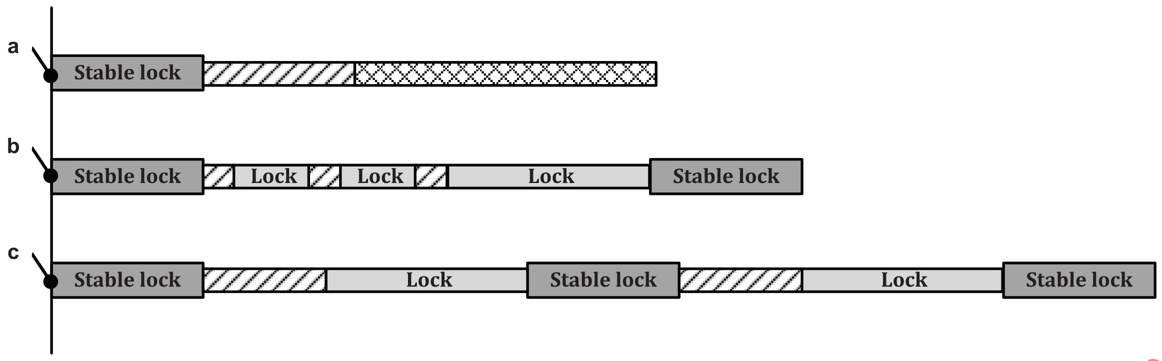
When a series of unlocks occurs, the time of the different unlocks is accumulated.

The accumulated unlock time is reset when stable lock is reached.

If the duration of a single unlock or a series of different unlocks exceeds the time t_{Unlock}, this is referred to as critical unlock.

If a critical unlock is detected, the NetInterface generates an `ev_Error_Shutdown` event.

Figure 14 provides examples (see 8.3 for details).



Key

▩	stable lock
☐	lock
▨	unlock
▩	critical unlock

[a] The first example shows an unlock that persists longer than t_{Unlock}. This results in a critical unlock.

[b] A series of short unlocks with an accumulated time less than t_{Unlock} does not lead to a critical unlock.

[c] The duration of unlocks is close to t_{Unlock}. The example shows that the MOST network can withstand a series of unlocks, provided that a lock time of at least t_{Lock} is interspersed.

Figure 14 — Examples of the behaviour when unlocks occur

8.2.2.3 NL – Handling sudden signal off and critical unlock

REQ	**3.4 NL – Network error detection and management – SSO or CU in s_NetInterface_Normal_Operation**
When a node detects a sudden signal off (SSO) or critical unlock (CU) in NetInterface state `s_NetInterface_Normal_Operation`, it shall configure itself as TimingMaster.	

REQ	**3.5 NL – Network error detection and management – Detecting node sets shutdown flag**
After configuring itself as TimingMaster, the detecting node shall request `cmd_Set_Shutdown_Flag`.	

The shutdown flag is contained in the administrative area of the network frame.

The detecting node starts $t_{\mathrm{SSO\_Shutdown}}$ when it requests `cmd_Set_Shutdown_Flag`. When $t_{\mathrm{SSO\_Shutdown}}$ expires, the node requests `cmd_MOST_Output_Off`, see 8.3.5.3.

The delay $t_{\mathrm{SSO\_Shutdown}}$ ensures that the following nodes are able to detect the shutdown flag.

REQ	**3.6 NL – Network error detection and management – Do not close bypass after SSO or CU**
The detecting node shall not close the bypass after requesting `cmd_MOST_Output_Off`.	

REQ	**3.7 NL – Network error detection and management – Store shutdown reason after SSO or CU**
The NetInterface of the detecting node shall store the cause of the fault (SSO or CU according to Figure 14) as shutdown reason.	

REQ	**3.8 NL – Network error detection and management – “No fault saved” due to shutdown flag**
The NetInterface of a node that detects the shutdown flag shall store “No fault saved” as shutdown reason when network activity ceases if no error is stored already.	

A node that is configured as TimingMaster does not have the ability to detect the shutdown flag.

REQ	**3.9 NL – Network error detection and management – Report shutdown reason**
The NetInterface shall report the shutdown reason to the application.	

REQ	**3.10 NL – Network error detection and management – Report shutdown reason due to cmd_ Shutdown_Reason**
The NetInterface shall report the shutdown reason to the application when it receives a `cmd_Shutdown_Reason` request.	

The initial value of the shutdown reason is “No result available”.

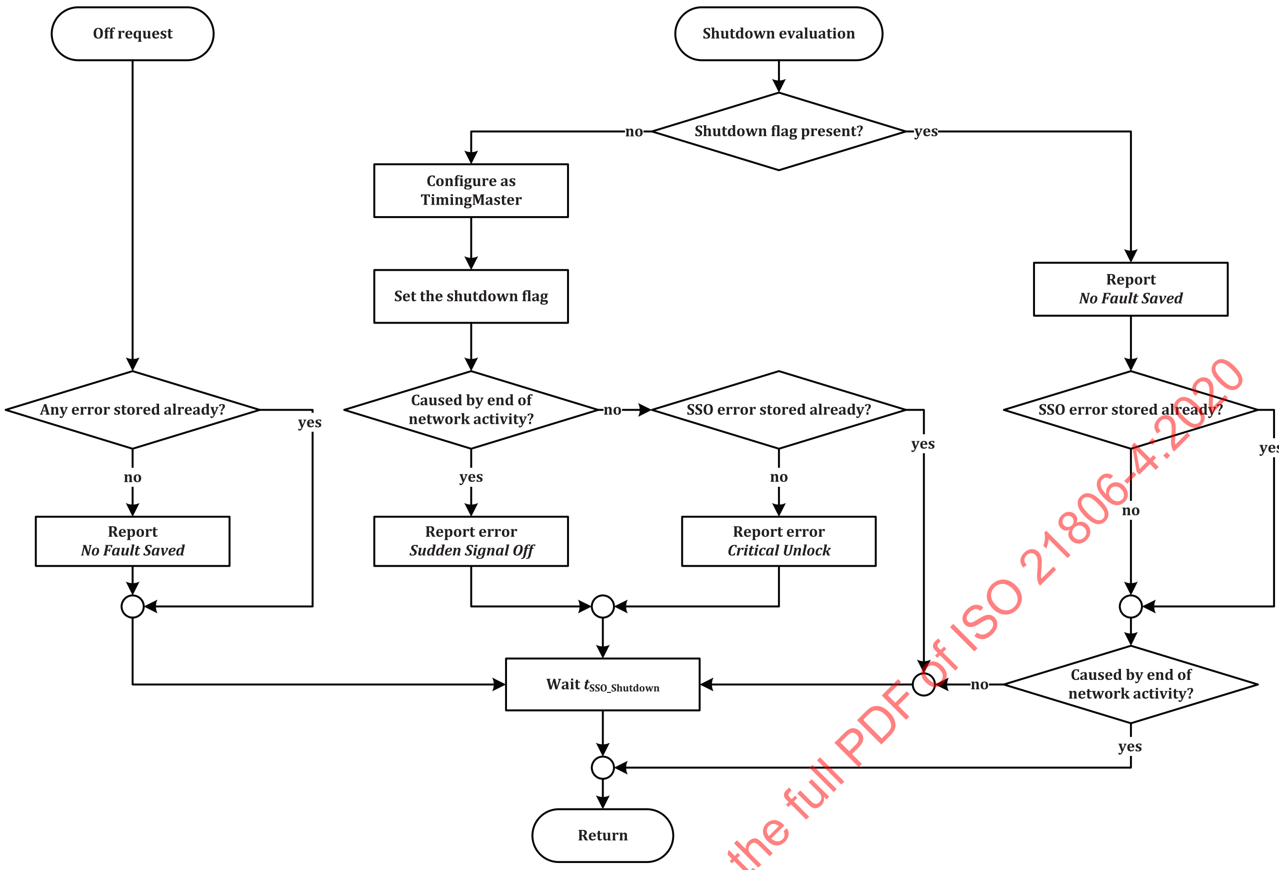
REQ	**3.11 NL – Network error detection and management – “No result available” due to reset**
After a reset, the NetInterface shall set the shutdown reason to “No result available”.	

SSO and CU detection applies only to NetInterface state s_NetInterface_Normal_Operation.

REQ	**3.12 NL – Network error detection and management – No shutdown flag outside s_NetInterface_Normal_Operation**
In any state other than NetInterface `s_NetInterface_Normal_Operation`, the detecting node shall not set the shutdown flag.	

REQ	**3.13 NL – Network error detection and management – No shutdown reason outside s_NetInterface_Normal_Operation**
In any state other than NetInterface `s_NetInterface_Normal_Operation`, the NetInterface shall not report a shutdown reason.	

Figure 15 shows this behaviour and is a subroutine of Figure 13.



Off request
Shutdown evaluation
Shutdown flag present?
no
yes
Configure as TimingMaster
Set the shutdown flag
Report No Fault Saved
Any error stored already?
Caused by end of network activity?
SSO error stored already?
SSO error stored already?
yes
no
yes
no
Report No Fault Saved
Report error Sudden Signal Off
Report error Critical Unlock
Wait $t_{SSO\_Shutdown}$
Caused by end of network activity?
Return

Figure 15 — Procedure “Off request” and procedure “Shutdown evaluation”

8.2.2.4 NL — Undervoltage management

8.2.2.5 NL — General

Undervoltage conditions rarely occur in every node at the same time and in the same severity. There are two limits regarding the supply voltage of a node: active voltage and sleep voltage.

8.2.2.6 NL — Active voltage U_{Active}

Above this voltage level, a node may exit NetInterface state `s_NetInterface_Sleep` if a startup trigger is detected.

U_{Active} is specified by the network designer.

8.2.2.7 NL — Sleep voltage U_{Sleep}

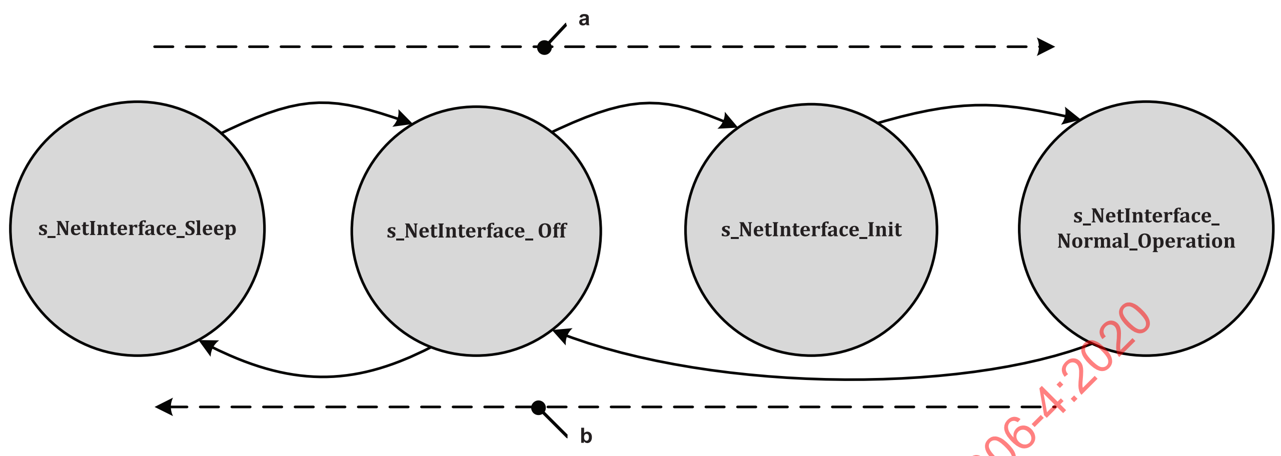
The sleep voltage U_{Sleep} is a node specific limit. Below this voltage level the NetInterface no longer works reliably and communication cannot be maintained.

U_{Sleep} is specified by the network designer.

If U_{Sleep} is reached, the node requests `cmd_MOST_Output_Off` to switch off the MOST output and changes to state `s_NetInterface_Sleep`. The node stays in state `s_NetInterface_Sleep`, even if the supply voltage recovers.

The node is awakened either by a network wake-up event (network activity or electrical wake-up line), or by a `cmd_Network_Startup` request due to a qualified local wake-up event. It changes to NetInterface state `s_NetInterface_Normal_Operation`, executing the standard initialisation process.

Figure 16 shows the NetInterface states and the transitions to switch from one state to the next state.



a ((network wake-up event) OR (qualified local wake-up event)) AND ($U > U_{\text{Active}}$).

b $U < U_{\text{Sleep}}$.

Figure 16 — Behaviour of a node depending on supply voltage

8.3 NL — Timing

8.3.1 NL — General

Timers can be interrupted and as a potential result the corresponding timing restrictions are not met when a node executes a `cmd_Emergency_Shutdown` request, or transitions through reset or `s_NetInterface_Sleep`.

8.3.2 NL — Timers

Timers are initialised by setting the current timer value to 0. After a timer is started, it runs until it is paused, stopped, or it expires. When a timer is paused, the current timer value is retained. When a timer is stopped or expires, the current timer value is set to 0. When a timer is started, it runs from the current timer value.

When a timer expires, the specified action (e.g. error handling) is performed.

Whenever a timer definition contains the character "-" instead of a value, that particular value is "not defined".

8.3.3 NL — Timing constraints

All timing constraints related to the application message service are delimited by the presence of the corresponding telegrams on the MOST network.

Timing constraints represent an upper limit for a particular action.

8.3.4 NL — Network startup and changes

8.3.4.1 NL — General

Table 18 and Table 19 specify constraints and timers, respectively, which are related to network startup and changes.

REQ	3.14 NL – Network startup and changes – Timing constraints
The values specified in Table 18 shall apply to the corresponding timing constraints.	

Table 18 — Network startup and changes timing constraints

Name	Value	Unit	Purpose
t_{WakeUp}	Network design specific	ms	Limit for requesting `cmd_MOST_Output_On`
$t_{\mathrm{WaitNodes}}$	100	ms	Limit for requesting `cmd_Open_Bypass`
t_{Lock}	110	ms	Limit for reaching stable lock

REQ	3.15 NL – Network startup and changes – Timers
The values specified in Table 19 shall apply to the corresponding timers.	

Table 19 — Network startup and changes timers

Name	Minimum value	Typical value	Maximum value	Unit	Purpose
t_{Config}	Network design specific	2 000	Network design specific	ms	Time before `ev_Init_Error_Shutdown`
t_{Restart}	300	300	310	ms	Time that the MOST output remains off

8.3.4.2 NL — Constraint t_{WakeUp}

8.3.4.2.1 NL — General

For the NetInterface, t_{WakeUp} is the permissible duration between the detection of network activity and requesting `cmd_MOST_Output_On` to switch on the MOST output.

8.3.4.2.2 NL — Validity conditions

REQ	3.16 NL – Timing – NL – Network startup and changes – t_{WakeUp} becomes valid
t_{WakeUp} shall become valid when the NetInterface detects network activity.	

If timer t_{Restart} is running, the NetInterface ignores network activity until t_{Restart} expires.

REQ	3.17 NL – Timing – NL – Network startup and changes – t_{WakeUp} no longer valid
t_{WakeUp} shall no longer be valid when the NetInterface requests `cmd_MOST_Output_On`.	

8.3.4.2.3 NL — Violation consequences

If the MOST output is not switched on after t_{WakeUp}, network startup might fail.

8.3.4.3 NL — Constraint $t_{\text{WaitNodes}}$

8.3.4.3.1 NL — General

For the NetInterface, $t_{\text{WaitNodes}}$ is the permissible duration between the detection of network activity and requesting `cmd_Open_Bypass` to open the bypass. This constraint is valid only when starting up the network.

8.3.4.3.2 NL — Validity conditions

REQ	3.18 NL – Timing – NL – Network startup and changes – $t_{\text{WaitNodes}}$ becomes valid
$t_{\text{WaitNodes}}$ shall become valid when the NetInterface detects network activity during `s_NetInterface_Off` or `s_NetInterface_Init`.	

REQ	3.19 NL – Timing – NL – Network startup and changes – $t_{\text{WaitNodes}}$ no longer valid
$t_{\text{WaitNodes}}$ shall no longer be valid when the NetInterface requests `cmd_Open_Bypass`.	

8.3.4.3.3 NL — Violation consequences

If the bypass is not open after $t_{\text{WaitNodes}}$, the node might not be available for communication in the MOST network.

8.3.4.4 NL — Constraint t_{Lock}

8.3.4.4.1 NL — General

For the NetInterface, t_{Lock} is the permissible duration between the detection of a lock and reaching stable lock.

8.3.4.4.2 NL — Validity conditions

REQ	3.20 NL – Timing – NL – Network startup and changes – t_{Lock} becomes valid
t_{Lock} shall become valid when the NetInterface detects lock.	

REQ	3.21 NL – Timing – NL – Network startup and changes – t_{Lock} no longer valid due to unlock
t_{Lock} shall no longer be valid when the NetInterface detects an unlock.	

REQ	3.22 NL – Timing – NL – Network startup and changes – t_{Lock} no longer valid due to stable lock
t_{Lock} shall no longer be valid when the NetInterface reaches stable lock.	

8.3.4.4.3 NL — Violation consequences

If stable lock is not reached after t_{Lock}, the node might not be available for communication in the MOST network.

8.3.4.5 NL — Timer t_{Config}

8.3.4.5.1 NL — General

For the NetInterface of the TimingMaster, t_{Config} controls when the `ev_Init_Error_Shutdown` event is triggered.

$$(\text{N} \times t_{\text{WakeUp}}) + t_{\text{WaitNodes}} + t_{\text{Lock}} < t_{\text{Config}}$$

where N is the maximum node count.

8.3.4.5.2 NL — Start and stop conditions

REQ	3.23 NL – Timing – NL – Network startup and changes – Start t_{Config}
In the TimingMaster, the NetInterface shall start t_{Config} when `s_NetInterface_Init` is entered.	

REQ	3.24 NL – Timing – NL – Network startup and changes – Stop t_{Config} in s_NetInterface_Off
In the TimingMaster, the NetInterface shall stop t_{Config} when `s_NetInterface_Off` is entered.	

REQ	3.25 NL – Timing – NL – Network startup and changes – Stop t_{Config} in s_NetInterface_Normal_Operation
In the TimingMaster, the NetInterface shall stop t_{Config} when `s_NetInterface_Normal_Operation` is entered.	

8.3.4.5.3 NL — Timer expiration

REQ	3.26 NL – Timing – NL – Network startup and changes – t_{Config} expires
In the TimingMaster, if t_{Config} expires in `s_NetInterface_Init`, the NetInterface shall change to state `s_NetInterface_Off`.	

The expiration of t_{Config} in `s_NetInterface_Init` causes the event `ev_Init_Error_Shutdown`. For details, see Figure 9 in 8.2.1.3.4.

8.3.4.6 NL — Timer t_{Restart}

8.3.4.6.1 NL — General

For the NetInterface, $t_{Restart}$ controls how long a MOST output remains off after it is switched off.

REQ	3.27 NL – Timing – NL – Network startup and changes – MOST output remains off during t_{Restart}
While t_{Restart} is running, the NetInterface shall not request `cmd_MOST_Output_On`.	

This applies even if the node recognizes network activity or any other network wake-up trigger.

REQ	3.28 NL – Timing – NL – Network startup and changes – Ignore network activity during t_{Restart}
If timer t_{Restart} is running, the NetInterface shall ignore network activity until t_{Restart} expires.	

8.3.4.6.2 NL — Start and stop conditions

REQ	3.29 NL – Timing – NL – Network startup and changes – Start t_{Restart}
The NetInterface shall start t_{Restart} when it requests `cmd_MOST_Output_Off`.	

REQ	3.30 NL – Timing – NL – Network startup and changes – Do not stop t_{Restart}
The NetInterface shall not stop t_{Restart}.	

8.3.4.6.3 NL — Timer expiration

After $t_{Restart}$ expires, the NetInterface may request `cmd_MOST_Output_On` to switch on the MOST output anytime a wake-up condition exists.

8.3.5 NL — Network shutdown

8.3.5.1 NL — General

Table 20 and Table 21 specify constraints and timers, respectively, which are related to network shutdown.

REQ	3.31 NL – Network shutdown – Timing constraints
The values specified in Table 20 shall apply to the corresponding timing constraints.	

Table 20 — Network shutdown timing constraints

Name	Value	Unit	Purpose
t_{Shutdown}	6 (per node)	ms	Limit for multiple network shutdown scenarios

REQ	3.32 NL – Network shutdown – Timers
The values specified in Table 21 shall apply to the corresponding timers.	

Table 21 — Network shutdown timers

Name	Minimum value	Typical value	Maximum value	Unit	Purpose
$t_{\text{SSO\_Shutdown}}$	100	100	110	ms	Time after which `cmd_MOST_Output_Off` is requested due to `cmd_Off_Request`, SSO, or CU.
t_{Unlock}	60	70	100	ms	Time after which a critical unlock is determined.
$t_{\text{PwrSwitchOffDelay}}$	Network design specific	---	Device specific	s	Time after which the `s_NetInterface_Sleep` is entered.

8.3.5.2 NL — Constraint t_{Shutdown}

8.3.5.2.1 NL — General

t_{Shutdown} is relevant for multiple different shutdown scenarios. Depending on the scenario, t_{Shutdown} is the permissible duration for a distinct stage in the final phase of the shutdown.

The network designer ensures that the accumulated shutdown delay (t_{Shutdown} of all nodes) is below t_{Restart}.

8.3.5.2.2 NL — Validity conditions

REQ	3.33 NL – Timing – NL – Network shutdown – t_{Shutdown} becomes valid due to end of network activity
t_{Shutdown} shall become valid when a node detects the end of network activity.	

REQ	3.34 NL – Timing – NL – Network shutdown – t_{Shutdown} becomes valid due to a critical unlock
In `s_NetInterface_Normal_Operation`, t_{Shutdown} shall become valid when a node detects a critical unlock.	

Depending on the network designer preference, for an active TimingSlave, t_{Shutdown} may become valid when the TimingSlave detects network activity.

REQ	3.35 NL – Timing – NL – Network shutdown – t_{Shutdown} no longer valid due to cmd_MOST_Output_Off
t_{Shutdown} shall no longer be valid when a node requests `cmd_MOST_Output_Off`.	

REQ	3.36 NL – Timing – NL – Network shutdown – t_{Shutdown} no longer valid due to shutdown flag
In `s_NetInterface_Normal_Operation`, t_{Shutdown} shall no longer be valid when a node that does not detect the shutdown flag requests `cmd_Set_Shutdown_Flag`.	

8.3.5.2.3 NL — Violation consequences

Potentially, the shutdown is not executed.

8.3.5.3 NL — Timer $t_{\text{SSO\_Shutdown}}$

8.3.5.3.1 NL — General

$t_{\text{SSO\_Shutdown}}$ controls when `cmd_MOST_Output_Off` is requested to switch off the MOST output in multiple shutdown scenarios.

The delay $t_{\text{SSO\_Shutdown}}$ ensures that the following nodes can detect the shutdown flag.

8.3.5.3.2 NL — Start and stop conditions

8.3.5.3.3 NL — General

REQ	3.37 NL – Timing – NL – Network shutdown – Start $t_{\text{SSO\_Shutdown}}$ due to cmd_Off_Request
A NetInterface shall start $t_{\text{SSO\_Shutdown}}$ when it begins executing a `cmd_Off_Request`.	

REQ	3.38 NL – Timing – NL – Network shutdown – Start $t_{\text{SSO\_Shutdown}}$ due to SSO
If it does not detect the shutdown flag, a NetInterface that detects an SSO shall start $t_{\text{SSO\_Shutdown}}$ when it requests `cmd_Set_Shutdown_Flag`.	

REQ	3.39 NL – Timing – NL – Network shutdown – Start $t_{\text{SSO\_Shutdown}}$ due to critical unlock
If it does not detect the shutdown flag, a NetInterface that detects a critical unlock shall start $t_{\text{SSO\_Shutdown}}$ when it requests `cmd_Set_Shutdown_Flag`.	

REQ	3.40 NL – Timing – NL – Network shutdown – Start $t_{\text{SSO\_Shutdown}}$ due to shutdown flag
If it detects the shutdown flag, a NetInterface shall start $t_{\text{SSO\_Shutdown}}$ when a critical unlock occurs.	

REQ	3.41 NL – Timing – NL – Network shutdown – Do not stop $t_{\text{SSO\_Shutdown}}$
A NetInterface shall not stop $t_{\text{SSO\_Shutdown}}$.	