LINX Protocols

Abstract

*This document describes the protocols used in Enea **LINX version 2.1**. See the Revision History section (and document header line) for the version of this document.*

LINX is a distributed communication protocol stack for transparent inter node and inter process communication for a heterogeneous mix of systems.

Disclaimer: The information in this document is subject to change without notice and should not be construed as a commitment by Enea Software AB.

Table of Contents

1. Introduction

1.1 Purpose

This document describes the Enea LINX protocol.

1.2 Revision History

Current version is also seen in document header when printed (HTML title line).

Revision	Author	Date	Status and Description of purpose for new revision
16	wivo	2008-07-10	Updated TCP CM protocol with OOB.
15	wivo	2008-05-13	Updated Ethernet protocol with OOB.
14	debu	2007-11-05	Document updates in RLNH and Ethernet protocol.
13	debu	2007-09-25	Updated Linxdisc protocol descrition.
12	mwal	2007-09-24	Added feature negotiation. Increased rlnh version to 2 and ethcm version to 3.
11	lejo,wivo	2007-09-20	Added TCP CM ver.2
10	zalpa,lejo	2007-08-29	**Approved**. Added PTP CM ver.1
9	lejo	2006-10-25	Added copyright on front page. Cleaned out prerelease entries of document history.
8	lejo	2006-10-13	Converted document from Word to XHTML.
7	jonj	2006-09-13	Added reserved field in UDATA Header, Fixed bug in MAIN header.
6	jonj	2006-09-11	Fixed a few bugs, improved RLNH protocol description.
5	jonj	2006-08-05	**Approved**. Updated after review (internal reference ida 010209).

1.3 References

1.4 Definitions and Acronyms

Definitions

A
: The active (initiating) side in a protocol exchange.

B
: The passive (responding) side in a protocol exchange.

CM
: Connection Manager, the entity implementing the transport layer of the Enea LINX protocol.

Endpoint
: A (part of) an application that uses the Enea LINX messaging services.

Connection
: An association between Connection Managers.

Connection ID
: A key used by Ethernet Connection Managers to quickly lookup the destination of an incoming packet. Connection ID based lookup is much efficient than MAC-address based lookup.

Connection Manager
: A Connection Manager provides reliable communication for message passing. The connection layer roughly corresponds to layer four, the transport layer, in the OSI model.

Connection Supervision
: A function within the Connection Manager responsible for detection of connection failure.

Header
: Protocol Data filled in by the Connection Manager. A PDU consists of several headers.

Link
: An association between link handlers using the Enea LINX protocol.

Link Handler
: A concept from the OSE world. A Link Handler makes the OSE messaging IPC mechanism available in a distributed context.

MAC-address
: Link layer address on Ethernet.

Message
: A unit of information transported over LINX. Messages are transformed by Connection Managers in order to fit the media.

Packet
: Protocol Data Unit sent over a connection.

RLNH
: Rapid Link Handler, the link handler of Enea LINX.

Feature
: A functional extension to the default characteristic.

Abbreviation

IPC
: Inter Process Communication.

PDU
: Protocol Data Unit.

2. Overview

Enea LINX is an open technology for distributed system IPC which is platform and interconnect independent, scales well to large systems with any topology, but that still has the performance needed for high traffic bearing components of the system. It is based on a transparent message passing method.

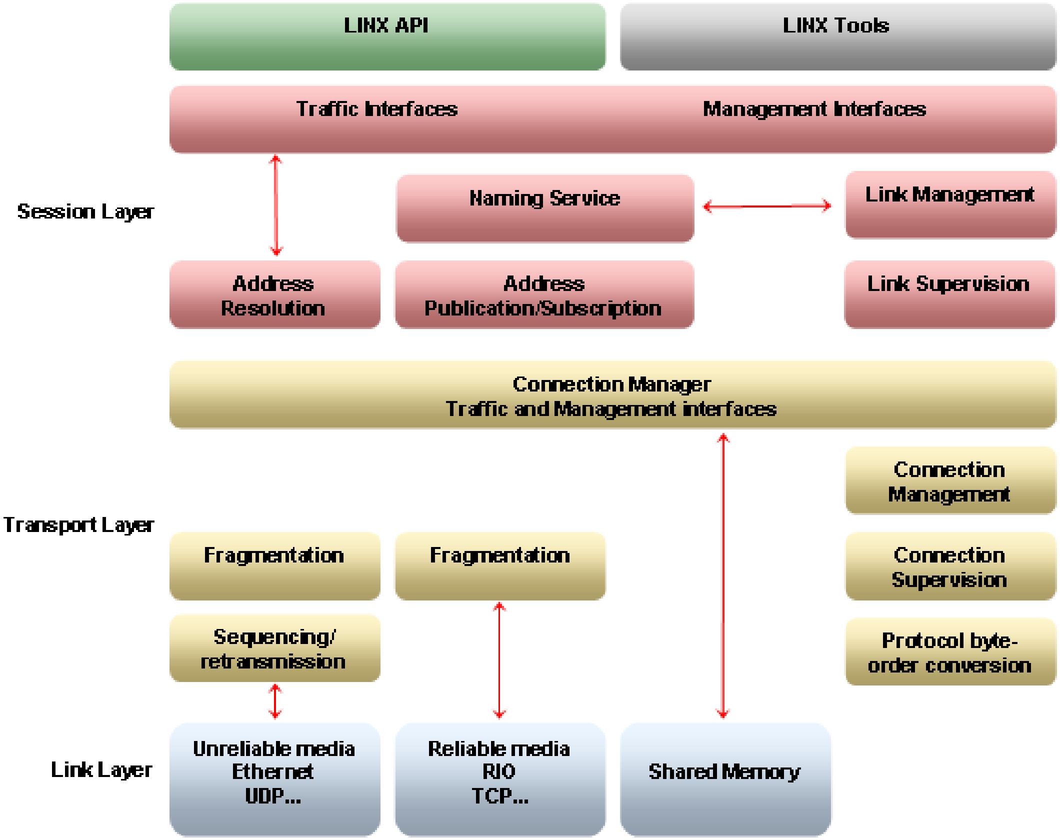


Figure 1 LINX Architecture

Enea LINX provides a solution for inter process communication for the growing class of heterogeneous systems using a mixture of operating systems, CPUs, microcontrollers DSPs and media interconnects such as shared memory, RapidIO, Gigabit Ethernet or network stacks. Architectures like this poses obvious problems, endpoints on one CPU typically uses the IPC mechanism native to that particular platform and they are seldom usable on platform running other OSes. For distributed IPC other methods, such as TCP/IP, must be used but that comes with rather high overhead and TCP/IP stacks may not be available on small systems like DSPs. Enea LINX solves the problem since it can be used as the sole IPC mechanism for local and remote communication in the entire heterogeneous distributed system.

The Enea LINX protocol stack has two layers - the RLNH and the Connection Manager, or CM, layers. RLNH corresponds to the session layer in the OSI model and implements IPC functions including methods to look up endpoints by name and to supervise to get asynchronous notifications if they die. The Connection Manager layer corresponds to the transport layer in the OSI model and implements reliable in order transmission of arbitrarily sized messages over any media.

RLNH is responsible for resolving remote endpoint names, and for setting up and removing local representations of remote endpoints. The RLNH layer provides translation of endpoint OS IDs from the source to the destination system. It also handles the interaction with the local OS and applications that use the Enea LINX messaging services. RLNH consists of a common, OS independent protocol module and an OS adaptation layer that handles OS specific interactions.

The Connection Manager provides a reliable transport mechanism for RLNH. When the media is unreliable, such as Ethernet, or have other quirks the Connection Manager must implement means like flow control, retransmission of dropped packets, peer supervision, and becomes much more complex. On reliable media, such as shared memory or RapidIO, Connection Managers can be quite simple.

The rest of this document contains a detailed description of the protocols used by the RLNH and the CM layers of Enea LINX. Chapter 3 describes the RLNH protocol. Chapter 4 describes features shared by all Connection Managers, i.e. what RLNH requires from a Connection Manager. The description is intentionally kept on a high level since the functionality actually implemented in any instance of a CM depends very much on the properties of the underlying media. It is the intention that when Connection Managers for new media are implemented the protocols will be described in the following chapters. Finally Chapter 5 describes version two of the Ethernet Connection Manager protocol.

The table below shows how protocol headers are described. PDUs are sent in network byte order (e.g. big endian). Bits and bytes are numbered in the order they are transmitted on the media. For example:

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="8">One</td><td colspan="4">Two</td><td colspan="4">Three</td><td colspan="16">Reserved</td></tr>
<tr><td colspan="32">Four</td></tr>
</table>

Table 1. Protocol Message legend

The first two rows show bit numbers, the third row byte numbers for those more comfortable with that view, and the last two rows example protocol fields. In the header above, the fields in the header are sent in order: one, two, three, reserved, four.

3. The RLNH Protocol

The RLNH protocol is designed to be light-weight and efficient. The RLNH-to-RLNH control PDUs are described in detail below. The required overhead associated with user signal transmission is not carried in any dedicated RLNH PDU. Instead it is passed as arguments along with the signal data to the Connection Manager layer, where an optimized transmission scheme and message layout can be implemented based on knowledge of the underlying media and/or protocols. In particular the source and destination link addresses are sent this way, the addresses identifies the sending and receiving endpoints respectively. RLNH protocol messages are sent with source and destination link addresses set to zero (0).

3.1 RLNH Link Creation and Initialization

The Connection Manager is initialized by RLNH. It is responsible for providing reliable, in-order delivery of messages and for notifying RLNH when the connection to the peer becomes available / unavailable. An RLNH link is created maps a unique name to a Connection Manager object.

As soon as the Connection Manager has indicated that the connection is up, RLNH transmits an RLNH_INIT message to the peer carrying its protocol version number. Upon receiving this message, the peer responds with an RLNH_INIT_REPLY message indicating whether it supports the given protocol version or not. When remote and local RLNH versions differ the lowest of the two versions are used by both peers. If the message exchange has been successfully completed, RLNH is ready to provide messaging services for the link name. The RLNH protocol is stateless after this point.

3.2 RLNH Feature Negotiation

The RLNH Feat_neg_string is sent once by both peers to negotiate what features enable. It is contained in the INIT_REPLY message. The RLNH Feat_neg_string contain a string with all feature names and corresponding arguments. Only the features used by both peers (the intersection) are enabled. All the features supported by none or one peer (the complement) are disabled by both peers. An optional argument can be specified and It is up up to each feature how the argument is used and how the negotiation of the argument is performed.

3.3 Publication of Names

RLNH assigns each endpoint a link address identifier. The association between the name of an endpoint and the link address that shall be used to refer to it is published to the peer in an RLNH_PUBLISH message. RLNH_PUBLISH is always the first RLNH message transmitted when a new endpoint starts using the link.

Upon receiving an RLNH_PUBLISH message, RLNH creates a local representation of the remote name. Local applications can communicate transparently with such a representation, but the signals are in reality forwarded by RLNH to the peer system where the real destination endpoint resides (and the destination will receive the signals from a representation of the true sender).

The link addresses of source and destination for each signal are given to the Connection Manager for transmission along with the signal data and delivered to the peer RLNH. The link addressing scheme is designed to enable O(1) translation between link addresses and their corresponding local OS IDs.

3.4 Remote Name Lookup

When a hunt call is performed for the name of a remote endpoint, the request is passed on by RLNH to the peer as a RLNH_QUERY_NAME message. If the hunter has not used the link before, an RLNH_PUBLISH message is sent first.

Upon receiving a RLNH_QUERY_NAME message, RLNH resolves the requested endpoint name locally and returns an RLNH_PUBLISH message when it has been found and assigned a link address. As described above, this triggers the creation of a representation at the remote node. This in turn resolves the hunt call - the hunter is provided with the OS ID of the representation.

3.5 Link Supervision

RLNH supervises all endpoints that use the link. When a published name is terminated, a RLNH_UNPUBLISH message with its link address is sent to the peer.

When a RLNH_UNPUBLISH message is received, RLNH removes its local representation of the endpoint referred to by the given link address. When RLNH no longer associates the link address with any resources, it returns a RLNH_UNPUBLISH_ACK message to the peer.

The link address can be reused after a RLNH_UNPUBLISH_ACK message.

3.6 Protocol Messages

3.6.1 RLNH_INIT

The RLNH_INIT message initiates link establishment between two peers. It is sent when the Connection Manager indicates to RLNH that the connection is up.

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="24">Reserved</td><td colspan="8">Type</td></tr>
<tr><td colspan="32">Version</td></tr>
</table>

Table 2. RLNH Init Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_INIT is 5.
Version	The RLNH protocol version. The current version is 1.

Table 3. RLNH Init Header Description

3.6.2 RLNH_INIT_REPLY

During link set up, RLNH responds to the RLNH_INIT message by sending an RLNH_INIT_REPLY message. The status field indicates whether the protocol is supported or not. The Feat_neg_string tells the remote RLNH what features the local RLNH supports.

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="24">Reserved</td><td colspan="8">Type</td></tr>
<tr><td colspan="32">Status</td></tr>
<tr><td colspan="32">Feat_neg_string (variable length, null-terminated string)</td></tr>
</table>

Table 4. RLNH Init Reply Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_INIT_REPLY is 6.
Status	Status code indicating whether the protocol version received in the RLNH_INIT message is supported (0) or not (1).
Feat_neg_string	String containing feature name and argument pairs. Example: "feature1:arg1,feature2:arg2\0".

Table 5. RLNH Init Reply Header Description

3.6.3 RLNH_PUBLISH

The RLNH_PUBLISH message publishes an association between an endpoint name and the link address that shall be used to refer to it in subsequent messaging. Upon receiving an RLNH_PUBLISH message, RLNH creates a local representation of the remote name. This resolves any pending hunt calls for link_name/remote_name .

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="24">Reserved</td><td colspan="8">Type</td></tr>
<tr><td colspan="32">Linkaddr</td></tr>
<tr><td colspan="32">Name (variable length, null-terminated string)</td></tr>
</table>

Table 6. RLNH Publish Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_PUBLISH is 2.
Linkaddr	The link address being published.
Name	The name being published.

Table 7. RLNH Publish Header Description

3.6.4 RLNH_QUERY_NAME

The RLNH_QUERY_NAME message is sent in order to resolve a remote name. An RLNH_PUBLISH message will be sent in response when the name has been found and assigned a link address by the peer.

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="24">Reserved</td><td colspan="8">Type</td></tr>
<tr><td colspan="32">src_linkaddr</td></tr>
<tr><td colspan="32">name (variable length, nul-terminated string)</td></tr>
</table>

Table 8. RLNH Query Name Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_QUERY_NAME is 1.
src_linkaddr	Link address of the endpoint that issued the query.
Name	The name to be looked up.

Table 9. RLNH Query Name Header Description

3.6.5 RLNH_UNPUBLISH

The RLNH_UNPUBLISH message tells the remote RLNH that the endpoint previously assigned the given link-address has been closed.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Reserved																								Type							
Linkaddr																															

Table 10. RLNH Unpublish Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_UNPUBLISH is 3.
Linkaddr	The address of the closed endpoint.

Table 11. RLNH Unpublish Header Description

3.6.6 RLNH_UNPUBLISH_ACK

The RLNH_UNPUBLISH_ACK message tells the remote RLNH that all associations regarding an unpublished link address have been terminated. This indicates to the peer that it is ok to reuse the link address.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Reserved																								Type							
Linkaddr																															

Table 12. RLNH Unpublish Ack Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_UNPUBLISH_ACK is 4.
Linkaddr	The link address of the unpublished endpoint.

Table 13. RLNH Unpublish Ack Header Description

3.6.7 RLNH_PUBLISH_PEER

When a remote LINX endpoint is used as the sender in a send_w_sender() function call and the receiver exists on the same node as the local LINX endpoint, the remote LINX endpoint is published as a remote sender.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Reserved																								Type							
Linkaddr																															
Peer linkaddr																															

Table 14. RLNH Publish Peer Header

Field	Definition
Reserved	Reserved for future use, must be 0.
Type	Message type. The value of RLNH_PUBLISH_PEER is 5.

Linkaddr	The link address being published.
Peer linkaddr	The link address of the endpoint that previously published it self on the current link.

Table 15. RLNH Init Reply Header Description

4. Enea LINX Connection ManagerProtocols

This chapter describes in general terms the functionality a Connection Manager must provide.

A Connection Manager shall hide details of the underlying media, e.g. addressing, general media properties, how to go about to establish connections, media aggregation, media redundancy and so on. A Connection Manager shall support creation of associations, known as connections that are suitable for reliable message passing of arbitrarily sized messages. A Connection Manager must not rely on implicit connection supervision since this can cause long delay between peer failure and detection if the link is idle. A message accepted by a Connection Manager must be delivered to its destination.

If, for any reason, a Connection Manager is unable to deliver a message RLNH must be notified and the connection restarted since its state at this point is inconsistent.

Since Enea LINX runs on systems with very differing size, the Connection Manager protocol must be scalable. A particular implementation may ignore this requirement either because the underlying media doesn't support scalability or that a non-scalable implementation has been chosen. However, interfaces to the Connection Manager and protocol design when the media supports big configuration shall not make design decisions that prevents the system to grow to big configurations should the need arise.

Enea LINX must be able to coexist with other protocols when sharing media.

4.1 Connection Establishment

Creation of connections always requires some sort of hand shake protocol to ensure that the Connection Manager at the endpoints agrees on communication parameters and the properties of the media.

4.2 Reliable Message Passing

A channel suitable for reliable message passing must appear to have the following properties:

- Messages are delivered in the order they are sent.
- Messages can be of arbitrary size.
- Messages are never lost.
- The channel has infinite bandwidth.

Although no medium has all these properties some come rather close and others simulate them by implementing functions like fragmentation of messages larger than the media can handle, retransmission of lost messages, and flow control i.e. don't send more than the other side can consume. If a Connection Manager is unable to continue deliver messages according to these rules the connection must be reset and a notification sent to RLNH.

4.3 Connection Supervision

A distributed IPC mechanism should support means for applications to be informed when a remote endpoint fails. Connection Managers are responsible for detecting when the media fails to deliver messages and when the host where a remote endpoint runs has crashed. There are of course other reasons an endpoint might fail to respond but in those situations the communication link continues to function and higher layers in the LINX architecture manages supervision.

5. Enea LINX Ethernet Connection Manager

This chapter describes version 3 of the Enea LINX Ethernet Connection Manager Protocol.

5.1 Protocol Descriptions

Enea LINX PDUs are stacked in front of, possible, user data to form an Enea LINX Ethernet packet. All PDUs contain a field next header which contain the protocol number of following headers or the value -1 (1111b) if this is the last PDU. All headers except Enea LINX main header are optional. Everything from the transmit() down call, including possible control plane signaling, from the RLNH layer is sent reliably as user data. The first field in all headers is the next header field, having this field in the same place simplifies implementation and speeds up processing.

If a malformed packet is received the Ethernet Connection Manager resets the connection and informs RLNH.

When the Ethernet Connection Manager encounters problems which prevents delivery of a message or part of a message it must reset the connection. Notification of RLNH is implicit in the Ethernet CM, when the peer replies with RESET or, if the peer has crashed, the Connection Supervision timer fires.

5.1.1 Enea LINX Ethernet Connection Manager Headers

Version 3 of the Enea LINX Ethernet Connection Manager protocol defines these headers.

Protocol number	Definition
ETHCM_MAIN	Main header sent first in all Enea LINX packets.
ETHCM_CONN	Connect header. Used to establish and tear down connections.
ETHCM_UDATA	User data header. All messages generated outside the Connection Manager are sent as UDATA.
ETHCM_FRAG	Fragment header. Messages bigger than the MTU are sent fragmented and PDUs following the first carries the ETHCM_FRAG header instead of ETHCM_UDATA.
ETHCM_ACK	Reliability header. Carries seqno and ackno. ACK doubles as empty acknowledge PDU and ACK-request PDU, in sliding window management and connection supervision.
ETHCM_NACK	Request retransmission of one or more packets.
ETHCM_NONE	Indicates that the current header is the last in the PDU.

Table 16. Ethernet Connection Manager Protocol Headers

5.1.1.1 ETHCM_MAIN Header

The ETHCM_MAIN header is sent first in all Enea LINX PDU's. It carries protocol version number, connection id, and packet size. Connection ID is negotiated when a connection is establish and is used to lookup the destination for incoming packets.

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="4">Next</td><td colspan="3">Ver</td><td colspan="2">Res</td><td colspan="8">Conn_ID</td><td colspan="1">R</td><td colspan="14">Packet size</td></tr>
</table>

Table 17. Main header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
Ver	Enea LINX Ethernet Connection Manager protocol version. Version 3 decimal is currently used, 0 is illegal.
Res	Reserved for future use, must be 0.
Conn ID	A key representing the connection, used for fast identification of destination of incoming packets.
R	Reserved for future use, must be 0.
Packet size	Total packet size in bytes including this and following headers.

Table 18. Main Header Description

5.1.2 Enea LINX Connect Protocol

The Enea LINX Connect Protocol is used to establish a connection on the Connection Manager level between two peers A and B. A Connection Manager will only try to establish a connection or accept connection attempt from a peer if it has been explicitly configured to do. After configuration a CM will maintain connection with the peer until explicitly told to destroy the connection or an un-recoverable error occurs.

If a Connect-message is received, with a version number different from 2, the Ethernet CM refuses to connect.

The Connect Protocol determines Connection IDs to be used for this connection. Connection IDs are small keys used by receivers to quickly lookup a packets destination. Each side selects the Connection ID to be used by the peer when sending packets over the connection. The peer saves the ID and will use it for all future communication. If a node has a lot of connections it may run out of available Connection IDs. In this case the node sends Connection ID 0, which means no Connection ID and reverts to the slower MAC-addressing mode to determine destination for incoming packets.

A window size options may be sent in the CONNECT-header to indicate how the sender configuration deviates from the default. Deviations from default values are configured per connection in the create_conn()-call.

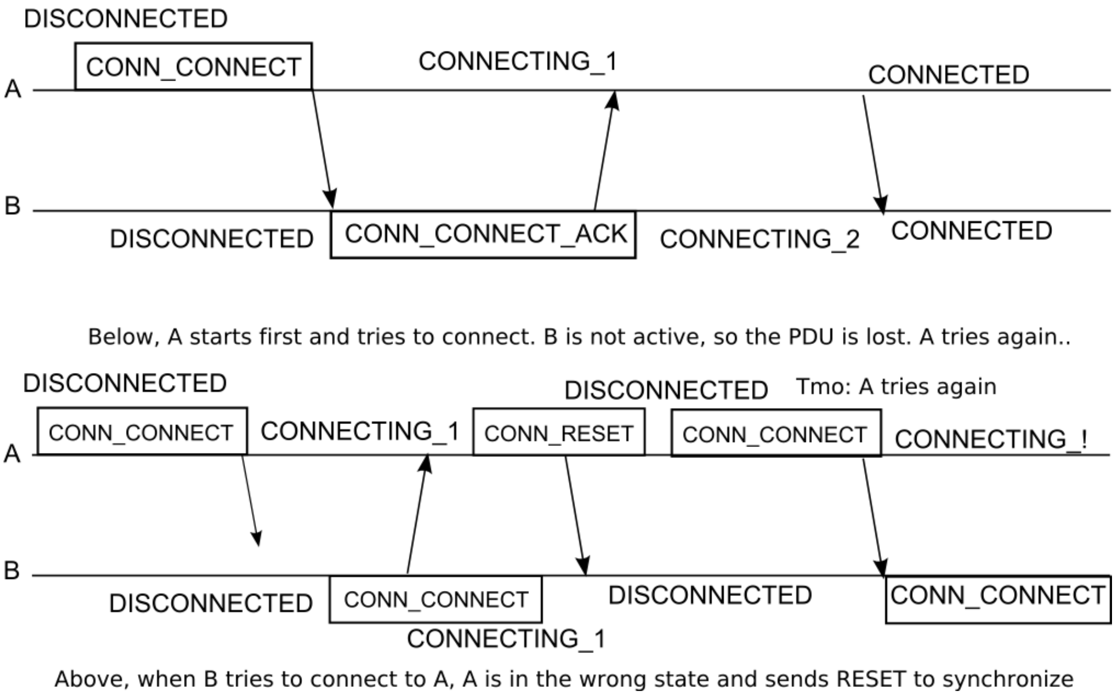


Figure 2 Successful Connect.

5.1.2.1 Connect Protocol

In the following protocol description the side initiating the connection is called A and the side responding to the request is called B. All protocol transitions are supervised by a timer, if the timer fires before the next step in the protocol have been completed the state machine reverts to state disconnected.

The Connect Protocol is symmetrical, there is no master and both sides try to initiate the connection. Collisions, i.e. when the initial CONNECT PDU is sent simultaneously from both sides, are handled by the protocol and the connection restarted after a randomized back of timeout.

5.1.2.2 Connect Protocol Description

Unless a Connection Manager has been configured to create a connection to a peer no messages are sent and the Connection Manager doesn't respond to connection attempts.

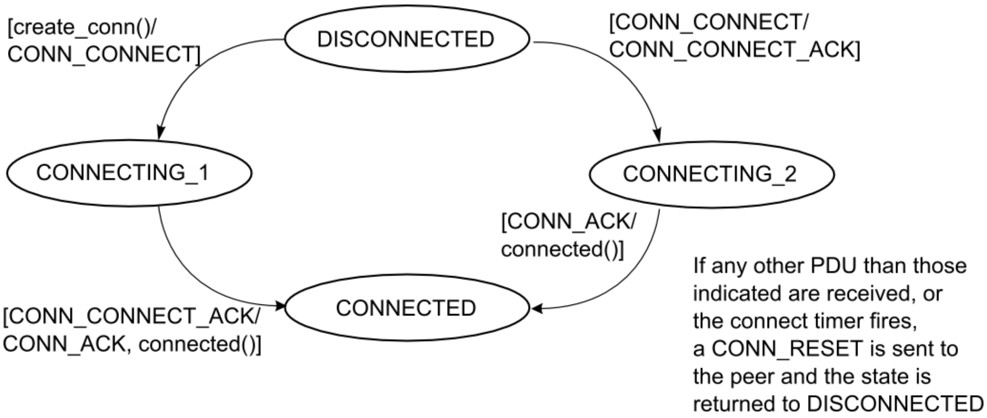
<u>A-side</u>

1. When a Connection Object is created at A by calling create_conn() A starts from DISCONNECTED state, sends a CONNECT PDU to B and enters state CONNECTING_1.
    a. If tmo, go back to step 1 and try again after a grace period.
    b. If B replies with a CONNECT_ACK PDU, move to CONNECTED state, send ACK PDU to B, notify RLNH that a connection have been established, and start the Connection Supervision function.
    c. If any other PDU is received from B send RESET and go back to step 1 and try again after a short grace period.

<u>B-side</u>

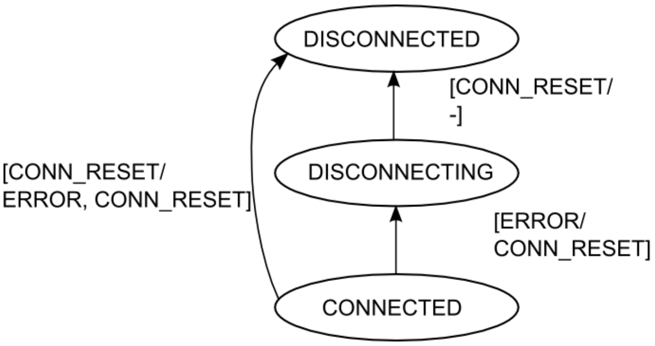
1. After configuration, B waits in state DISCONNECTED for a CONNECT PDU from A.
    a. If a CONNECT PDU arrives, send a CONNECT_ACK PDU and go to state CONNECTING_2
    b. If some other PDU arrives, send RESET to A and go back to step 2.
2. In state CONNECTING_2, B waits for an ACK PDU from A.
    a. When CONN_ACK arrives, the connect protocol is complete, the Connection Manager notifies higher layers that a connection have been established, and start the Connection Supervision function.
    b. If some other PDU is received or the timer fires, send RESET and go to state DISCONNECTED

Allowed messages and state changes are summarized in this state diagram. The notation [xxx/yyy] means: event xxx causes action yyy.



If any other PDU than those indicated are received, or the connect timer fires, a CONN_RESET is sent to the peer and the state is returned to DISCONNECTED

Figure 3 Connection protocol state diagram.



During an active disconnect, the CM paus in disconnected state until CONN_RESET have been received, which indicates that the peer have disconnected or until supervision timer expires, indicating that the peer have crashed.

Figure 4 Disconnect state diagram.

5.1.2.3 Feature Negotiation

A Feature Negotiation string is sent during connection establishment. The string is sent in the CONNECT ACK and the ACK type of ETHCM_CONN messages; the other messages contain an empty string '\0'). The string contains all feature names and corresponding argument. Only the features used by both peers (the intersection) are enabled. All the features supported by none or one peer (the complement) are disabled by both peers. An optional argument can be specified and It is up up to each feature how the argument is used and how the negotiation of the argument is performed.

5.1.2.4 ETHCM_CONN Header

The Connection header varies in size depending on the size of the address on the media, on Ethernet it is 16 bytes.

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="4">Next</td><td colspan="4">Type</td><td colspan="3">Size</td><td colspan="4">Window</td><td colspan="9">Reserved</td><td colspan="8">Conn ID</td></tr>
<tr><td colspan="32">Dst media address followed by src media address</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Feature Negotiation string (variable length, null-terminated string)</td></tr>
</table>

Table 19. Connect Header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
Type	CONNECT. Start a connect transaction.
	CONNECT ACK. Reply from passive side.
	ACK. Confirm that the connection have been created.
	RESET. Sent if any error occurs. Also sent if the next step in connect protocol fails to complete within allowed time.
Size	Media address size.
Window	Window size. Power of 2, thus Window == 5 means a window of 2^5 == 32 packets.
Reserved	Reserved must be 0.
Conn ID	Use this Connection ID. Informs peer which connection ID to use when sending packets over this connection. Conn ID == 0, means don't use Connection ID.
Dst and src media addresses	Dst media address immediately followed by src media address.
Feature Negotiation string	String containing feature name and argument pairs. Example: "feature1:arg1,feature2:arg2\0".

Table 20. Connect Header Description

5.1.3 Enea LINX User Data Protocol

All messages originating outside the Connection Manger are sent as USER_DATA. There are two types of USER_DATA header. The first type is used for messages not requiring fragmentation and for the first fragment of fragmented messages. The second type is used for all remaining fragments.

5.1.3.1 User data and Fragmentation Protocol

A-side

1. Accept a new message from RLNH. Calculate how many fragments are required to send this message.
2. Frag_cnt = 0.
3. For each fragment in the message:
   a. If first fragment send as USER_DATA else send as FRAG.
   b. If only fragment set fragno to -1 else set fragno to frag_cnt and increment frag_cnt.
   c. If last fragment set MORE to 0 else set MORE to 1.
   d. Forward the packet to lower layer for transmission.
   e. If last fragment go back to step 1.

B-side

1. When a USER_DATA or a FRAG packet arrives:
2. If fragno = -1 (0x7fff) deliver() to RLNH since this is a complete message and wait for next packet.
3. If fragno ≠ -1 find the reassembly queue for this message and add the packet to the tail of the queue (lower layers doesn't emit packets out-of-order). If the packet is the last fragment deliver the complete message to RLNH else wait for next packet.

5.1.3.2 ETHCM_UDATA Header

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next				O	Reserved											M	Frag no														
Dst addr																															
Src addr																															

Table 21. User Data Header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
Reserved	Reserved for future use, must be 0.
O	OOB bit, the UDATA is out of band.
M	More fragment follows.
Frag no	Number of this fragment. Fragments are numbered 0 to (number of fragments - 1). Un-fragmented messages have fragment number -1 (0x7fff).
Dst addr	Opaque address (to CM) identifying the receiver.
Src addr	Opaque address identifying the sender.

Table 22. User Data Header Description

5.1.3.3 ETHCM_FRAG Header

<table>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">0</td><td colspan="8">1</td><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="4">Next</td><td colspan="12">Reserved</td><td>M</td><td colspan="15">Frag no</td></tr>
</table>

Table 23. Fragment Header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
Reserved	Reserved for future use, must be 0.
M	More fragment follows.
Frag no	Number of this fragment. Fragments are numbered 0 to (number of fragments - 1). Un-fragmented messages have fragment number -1 (0x7fff).

Table 24. Fragment Header Description

5.1.4 Enea LINX Reliability Protocol

The Connection Manager uses a selective repeat sliding window protocol with modulo m sequence numbers. The ack header carries sequence and request numbers for all reliable messages sent over the connection. Sequence and Request numbers are 12 bits wide and m is thus 4096.

In the Selective Retransmit Sliding Window algorithm the B-side explicitly request retransmission of dropped packets and remembers packets received out of order. If the sliding window is full, the A-side queues outgoing packets in a deferred queue. When space becomes available in the window (as sent packets are acknowledged by B) packets are from the front of the deferred queue and sent as usual. A strict ordering is maintained, as long as there are packets waiting in the deferred queue new packets from RLNH are deferred, even if there is space available in the window.

5.1.4.1 Reliability Protocol Description

For sliding window operation the sender A have the variables SN_{min} and SN_{max}, SN_{min} points to the first unacknowledged packet in the sliding window and SN_{max} points at the next packet to be sent. In addition the sliding window size is denoted **n** and size of sequence numbers are modulo **m**. The receiver side B maintains a variable **RN** which denotes the next expected sequence number. In the protocol description, the sequence number and the request number of a packet is called **sn** and **rn** respectively. The module number **m** must be ≥ **2n**. In version two of the Enea LINX Ethernet protocol **m** is 4096 and **n** is a power of 2 ≤128. Drawing sequence number from a much larger space than the size of the window allows the reliability protocol to detect random packets with bad sequence numbers.

At A, the algorithm works as follows:

1. Set modulo variables SN_{min} and SN_{max} to 0.
2. In A if a message is sent from higher layer or there are packets in the defer queue, and (SM_{max} - SN_{min}) mod **m** < **n**, accept a packet into the sliding window set **sn** to SN_{max} and increment SN_{max} to (SN_{max} + 1) mod **m**. If (SN_{max} - SN_{min}) mod **m** ≥ **n** defer sending the packet, i.e. queue the packet until there is room in the sliding window.
3. If an error free frame is received from B containing **rn**, and (**RN** - SN_{min}) mod **m** ≤ (**RN** - SN_{max}) mod **m**, set SN_{min} to **RN** and remove packets with **sn** < SN_{min} mod **m** from send queue.
4. If a NACK frame is received, retransmit NACKed packets in-order with **rn** set to **RN**.
5. At arbitrary times but within bounded delay after receiving a reliable packet from B and if there are unacknowledged packets in the sliding window, send the first un-acked packet with **rn RN** and the request bit set.

The selective repeat algorithm at B:

1. Set the modulo m variable **RN** to 0.
2. When an error free frame is received from A containing **sn** equal to **RN**, release the packet as well as following queued packets with consecutive **sn** to higher layer and increment **RN** to (last released **sn** + 1) mod **m**.
3. When an error free frame is received from A containing sn in the interval **RN** < **sn** < RN + **n**, put the packet in sequence number order in the receive queue and send NACK requesting retransmission of sequentially dropped packets to A. The seqno field in the NACK frame shall contain RN and the count field shall contain the number of missing packets.
4. At arbitrary times but within a bounded delay after receiving an error free frame from A transmit a frame containing **RN** to A. If there are frames in the receive queue send a NACK indicating missing frames.

Note that only user data is sent reliable, i.e. consume sequence numbers. ACKR, NACK and empty ACK are unreliable messages sequence numbers are not incremented as they are sent.

5.1.4.2 ETHCM_ACK header

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next				R	Res			Ackno												Seqno											

Table 25. Ack Header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
R	ACK-request, peer shall respond with an ACK of its own as soon as possible. (Used during connection supervision.)
Res	Reserved for future use, must be 0.
Ackno	Shall be set to last consecutive received seqno from peer.
Seqno	Incremented for every sent packet. Note that seqnos are set based on sent packets rather than sent bytes as in TCP.

Table 26. Ack Header Description

5.1.4.3 ETHCM_NACK header

NACK is sent when a hole in the stream of received reliable packets is detected. If a sequence of packets is missing all are NACKed by the same NACK packet. A timer ensures that NACKs are sent as long as there are out-of-order packets in the receive queue.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next				Reserved				Count								Res				Seqno											

Table 27. Nack Header

Field	Definition
Next	Next header, the protocol number of the following Enea LINX header or 1111b if last header.
Reserved	Reserved, must be 0.
Count	Number of NACKed seqnos.
Res	Reserved, must be 0.
Seqno	First NACKed seqno, the rest are assumed to follow consecutively seqno + 1, seqno + 2, ... , seqno + count - 1.

Table 28. nack Header Description

5.1.5 Enea LINX Connection Supervision Protocol

The purpose of the Connection Supervision function is to detect crashes on the B-side. If B has been silent for some time A sends a few rapid pings and, if B doesn't respond, A decides that B has crashed, notifies higher layers, and initiates teardown of the connection. There is no separate PDU for the Connection Supervision function, the ACKR header from the reliability protocol doubles as ping PDU.

5.1.5.1 Connection Supervision protocol description

A-side

When a connection has been established, the Ethernet Connection Manager initializes the Connection Supervision function to state PASSIVE and starts a timer which will fire at regular interval as long as the link is up.

1. When the timer fires, the Ethernet Connection Manager checks if packets have been received from B since the last time the timer was active.
    a. If yes, clear packets counter, set Connection Supervision state to PASSIVE and go to step 2.
    b. If no, check if connection supervision is already in active state.
        i. If no, enter state Active and send an ACKR to B.
        ii. If yes, check if the ping limit has been reached.
            1. If yes, the connection is down, notify higher layers and go to state Disconnected.
            2. If no, resend ACKR and increment the ping counter.
2. Restart the timer.
3. If the connection is closed, stop the send timer.

B-side

1. When an ACKR is received, reply to the sender with an empty ACK [RN,SN$_{max}$-1].

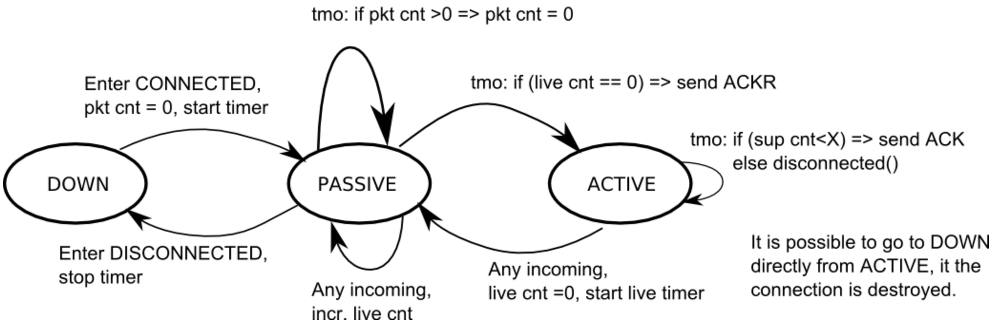

tmo: if pkt cnt >0 => pkt cnt = 0
Enter CONNECTED,
pkt cnt = 0, start timer
tmo: if (live cnt == 0) => send ACKR
tmo: if (sup cnt<X) => send ACK
else disconnected()
DOWN
PASSIVE
ACTIVE
Enter DISCONNECTED,
stop timer
Any incoming,
incr. live cnt
Any incoming,
live cnt =0, start live timer
It is possible to go to DOWN
directly from ACTIVE, it the
connection is destroyed.

Figure 5 State diagram for Connection Supervision

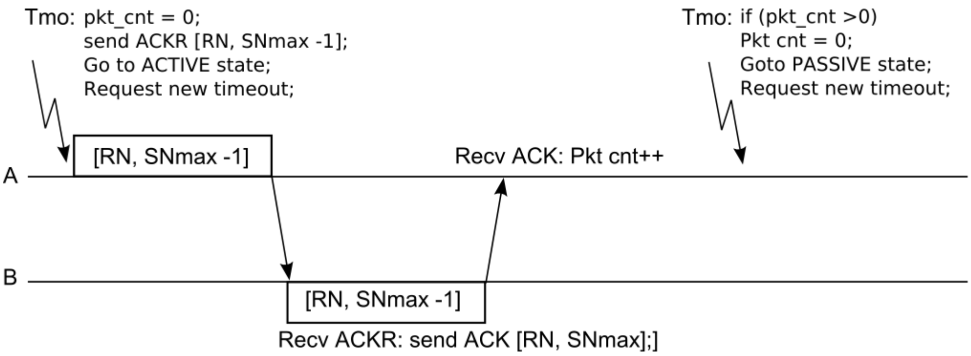


Figure 6 Connection supervision: the peer replies.

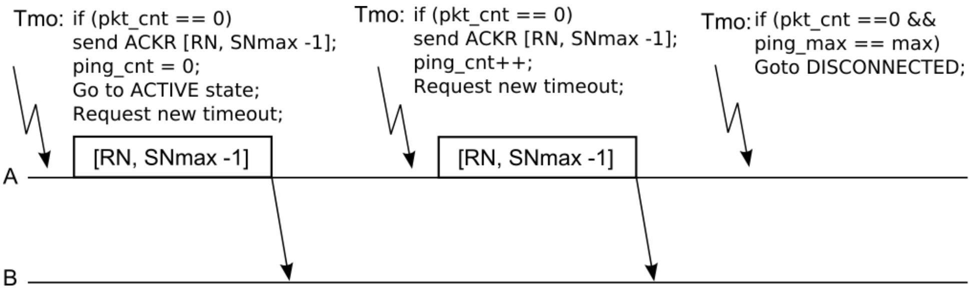


Figure 7 Connection supervision: the peer is down.

5.2 LINX Discovery Daemon

The LINX Discovery Daemon, **linxdisc**, automatically discovers LINX network topology and creates links to other LINX hosts. Linxdisc finds LINX by periodically broadcasting and listen for LINX advertisement messages on available Ethernet interfaces. When advertisements from other linxdiscs' arrive linxdisc examines the messages and if it passes a number of controls a LINX connection is created to the sending host. When a collision occurs, i.e. two or more nodes advertising the same name, collision resolution messages are exchanged to determine which node gets to remain in the cluster. The node that wins is the one that has been up for the longest time, if the nodes can't agree which one has been up longest, the node with the highest MAC-address wins. All Linxdisc messages from this version and forward carries a version number. For this implementaition of Linxdisc all messages containg a higher version number are discarded, allowing newer versions of Linxdisc to revert to version 1 of the Linxdisc protocol.

5.2.1 Linxdisc protocol

Connection procedure:

A-side

At startup read configuration file, build a list of all active interfaces, extract hostname and cluster name.

1. Periodically broadcast advertisement messages on interfaces allowed by the configuration.

B-side

1. When an advertisement message arrives, linxdisc performs the following steps to determine whether to connect to the sender:
    a. Version number matches.
    b. Not my own message, i.e. the sender address is the same as one of my interfaces.
    c. Network cluster name is same as configuration.
    d. Name doesn't match mine, if the name is identical to my name send a collision resolution message back to sender.
    e. Name doesn't match any already established connections.
    f. Check that the configuration doesn't forbid connections to this host.
2. If all tests pass create a connection to the peer using the advertised name and store information about the connection for later.
3. When terminated, linxdisc closes all connections it has created before exiting.
4. If the configuration is changed, closes connection not allowed by the new configuration, and updates advertisements messages to reflect the new configuration.

When a collision occurs:

A-side

1. Send a collision resolution message containing uptime and preferred decision.

B-side

1. When receiving a collision resolution message
    a. Check uptime and preferred decision if in agreement with sender refrain from sending advertisements if not in agreement, break tie based on MAC-address

5.2.1.1 Linxdisc Advertisement Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Last 2 bytes Ethernet header																Version															
Type																Reserved															
Uptime sec																															
Uptime usec																															
Linklen																															
Netlen																															
Strings																															

Table 29. Linxdisc Advertisement Message

Field	Description
Version	Linxdisc version type (1).
Type	Linxdisc message type, for advertisement (2) is used.
Reserved	Reserved for future use.
Uptime sec	The seconds part of the uptime.
Uptime usec	This microsecond part of the uptime.
Linklen	Length of linxname, c.f. below.
Netlen	Length of netname, c.f. below.
Strings	Carries two null-terminated strings, Linkname and Netname (network cluster name). Link name is a unique identifier for the sending host ment to be used as the name of the link created by the receiving linxdisc. Network cluster name is a unique cluster id string identifying a group of hosts that will form a **LINX cluster**.

Table 30. Linxdisc Advertisment Message Definition

5.2.1.2 Linxdisc Collision Resolution Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Last 2 bytes Ethernet header																Version															
Type																Reserved															
Uptime sec																															
Uptime usec																															
Preferred decision																															

Table 31. Linxdisc Collision Resolution Message

Field	Description
Version	Linxdisc version type (1)
Type	Linxdisc message type, for collision resolution (3) is used.
Reserved	Reserved for future use.
Uptime sec	The seconds part of the uptime.
Uptime usec	This microsecond part of the uptime.
Preferred decision	Preferred decision on whether the receiver should exit from the network.

Table 32. Linxdisc Collision Resolution Message Definition

6. Enea LINX Point-To-Point (PTP) Connection Manager

This chapter describes version 1 of the Enea LINX Point-To-Point Connection Manager Protocol.

The PTP Connection Manager is designed to meet the following requirements:

- Generic design - The design should allow several shared memory concepts. It should be easy to port the solution to other hardware. The generic parts should be separated for reuse with interfaces supporting as many different underlying layers as possible.
- Performance - The throughput should be as high as possible and the latency as low as possible. This includes minimizing the number of data copying and use of locks.
- Robust - The data exchange should be performed as simple as possible. Minimizing the complexity of the solution will minimize the possibility of making errors.

This CM is intended to communicate over reliable media, but in case of one side failure, the peer side is able to detect and react, as described in chapter 6.2 PTP Connection Supervision Protocol

6.1 Protocol Descriptions

Enea LINX PDUs are stacked in front of, possible, user data to form an Enea LINX packet. All PDUs contain a "next header" field which indicates the type of the next PDU to be sent. Everything from the transmit() down call, including possible control plane signaling, from the RLNH layer is sent reliably as user data. The first field in all headers is the current type, followed by "next" field.

If a malformed packet is received the PTP Connection Manager resets the connection and informs RLNH.

When the PTP Connection Manager encounters problems which prevents delivery of a message or part of a message it must reset the connection. If the peer replies with RESET, or if the peer has crashed and the Connection Supervision timer fires, the PTP CM will notify the RLNH.

6.1.1 Enea LINX Point-To-Point Connection Manager Headers

The Enea LINX Point-To-Point Connection Manager protocol defines these headers.

Protocol number	Value	Definition
PTP_CM_MAIN	0x01	Main header. It is always sent first.
PTP_CM_CONN_RESET	0x02	Reset header. Used to notify the remote side that the connection will be torn down and reinitialized.
PTP_CM_CONN_CONNECT	0x03	Connect header. Used to establish connections.
PTP_CM_CONN_CONNECT_ACK	0x04	Notify remote side that connection was established.
PTP_CM_UDATA	0x05	All messages generated outside the Connection Manager are send as UDATA.
PTP_CM_FRAG	0x06	Fragment header. Messages larger than MTU are fragmented into several packages. The first fragmented package has a PTP_CM_UDATA header followed by PTP_CM_FRAG header, the following packages has only PDU PTP_CM_FRAG.
PTP_CM_HEARTBEAT	0x07	Header for heart beat control messages. These messages are used to discover a connection failure.
PTP_CM_HEARTBEAT_ACK	0x08	Acknowledgement header for heartbeat messages. Sent as a reply to PTP_CM_HEARTBEAT messages.
PTP_CM_NONE	0xFF	Indicates that the current header is the last in the PDU.

Table 33. PTP Connection Manager Protocol Headers

6.1.1.1 PTP_CM_MAIN Header

All messages originating outside the Connection Manger begin with MAIN header. This header specify the type of the next header to come.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next Header Type																															

Table 34. PTP_CM Main Header

Field	Definition
Next Header Type	Specify what type of PDU will be next.

Table 35. PTP_CM Main Header Description

6.1.1.2 PTP_CM_UDATA Header

All messages originating outside the Connection Manger are sent as USER_DATA. There are two types of USER_DATA header. The first type is used for messages not requiring fragmentation and for the first fragment of fragmented messages. The second type - PTP_CM_FRAG is used for all remaining fragments.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next Header Type								Total Size																							
Total Size								Upper layer data 1																							
Upper layer data 1								Upper layer data 2																							
Upper layer data 2																															

Table 36. PTP_CM User Data Header

Field	Definition
Header type	Specify what type the current packet is (control, data, fragment).
Next Header Type	Specify what type of packet will be next.
Total size	Size of user data to be sent. If the size exceeds MTU, there message will be fragmented.
Upper layer data 1	Source address. CM does not modify this field.
Upper layer data 2	Destination address. CM does not modify this field.

Table 37. PTP_CM User Data Header Description

6.1.1.3 PTP_CM_FRAG Header

At destination, fragments will be expected to arrive in order, but may be interrupted by other packets, and the message will be re-composed based on total size information.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next Header Type								Size																							
Size								ID																							

Table 38. PTP_CM_FRAG Header

Field	Definition
Next Header type	Specify what header type will follow in the current PDU.
Size	The size of the fragment.
ID	Fragment ID.

Table 39. PTP_CM Fragment Header Description

6.1.1.4 PTP Connection Manager Control Headers

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Next Header Type								CM Version																							

Table 40. PTP CM Control Header

Field	Definition
Next Header Type	Type for next packet in PDU. Currently it is PTP_CM_NONE only.
CM Version	Connection Manager version.

Table 41. PTP CM Control Header Description

6.1.1.5 PTP Connection Protocol Description

Unless a Connection Manager has been configured to create a connection to a peer, no messages are sent and the Connection Manager does not respond to connection attempts. Below is described one case when a connection is created and B side initiates the connection later then A side.

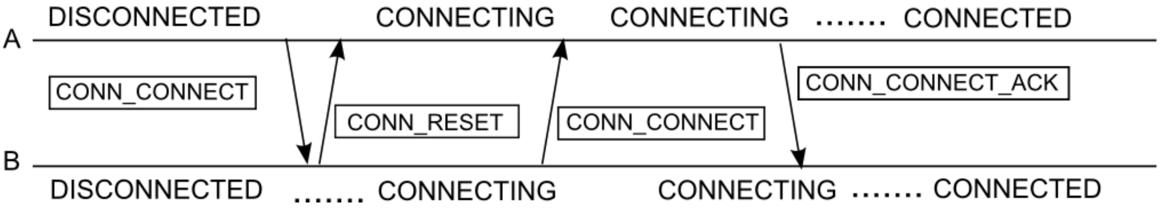

A
DISCONNECTED
CONNECTING
CONNECTING .... CONNECTED
CONN_CONNECT
CONN_RESET
CONN_CONNECT
CONN_CONNECT_ACK
B
DISCONNECTED .... CONNECTING
CONNECTING .... CONNECTED

Figure 8 Connect Sequence diagram for PTP Connection Manager

A side

1. When a Connection Object is created, A moves from DISCONNECTED state to CONNECTING, then sends a PTP_CM_CONN_CONNECT to B.
   a. If B replies with a PTP_CM_CONN_RESET PDU, A remains in CONNECTING state.
   b. If B sends a PTP_CM_CONN_CONNECT PDU, A moves to CONNECTED state, sends PTP_CM_CONN_CONNECT_ACK PDU to B and notifies RLNH that a connection have been established.
   c. If PTP_CM_CONN_RESET or PTP_CM_CONN_CONNECT PDUs are received, while in CONNECTED state, A side returns to disconnected state.

B side

1. In DISCONNECTED state, B does not evolve from this state unless a local attempt to create a connection is made.
2. When a Connection Object is created, B moves from DISCONNECTED state to CONNECTING and sends a PTP_CM_CONN_CONNECT to A.
   a. When PTP_CM_CONN_CONNECT arrives, A is waiting already in CONNECTING state so it enters CONNECTED state and responds to B with PTP_CM_CONN_CONNECT_ACK.
   b. When receiving PTP_CM_CONN_CONNECT_ACK, B side enters in CONNECTED state too.
   c. If PTP_CM_CONN_RESET or PTP_CM_CONN_CONNECT PDUs are received while in CONNECTED state, B side returns to disconnected state

If either sides receive PTP_CM_CONN_CONNECT message while in CONNECTED state, it moves to DISCONNECTED and sends a PTP_CM_CONN_RESET message.

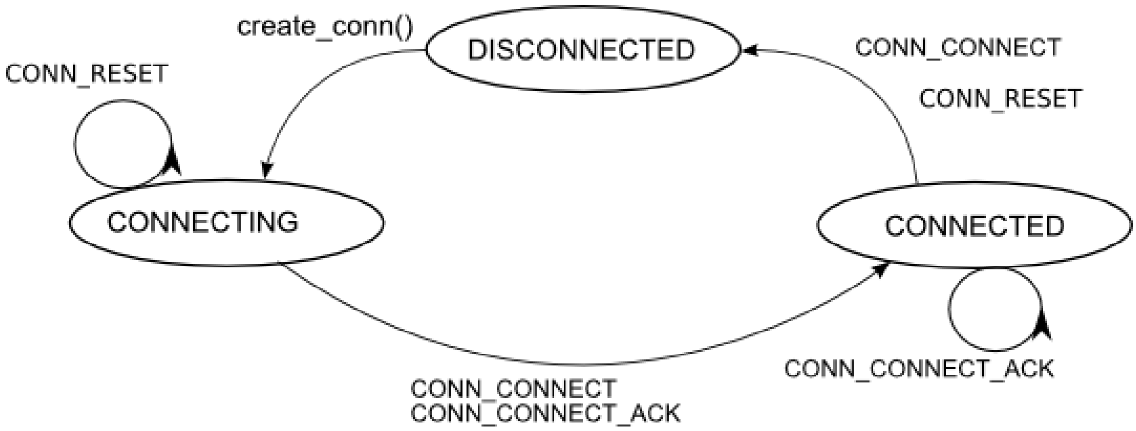


Figure 9 State diagram for PTP Connection Manager

6.2 PTP Connection Supervision Protocol

The purpose of Supervision function is to detect crashed on peer side. In order to do that, PTP_CM_HEARTBEAT PDU is sent periodically to the other side. When a PTP_CM_HEARTBEAT_ACK is received, an internal counter is re-set. If no acknowledgement is received within a certain time interval, the heartbeat counter is decremented. After a number of ACK missed, the connection is considered crashed and will be disconnected. The timeout and the number of acknowledgments are configurable.

The supervision activity is independent of user data flow over the connection. Each side is responsible to detect peer's activity, and each side will request acknowledgment to own heartbeat messages.

7. Enea LINX TCP Connection Manager

This chapter describes version 3 of the Enea LINX TCP Connection Manager Protocol.

The TCP Connection Manager uses a TCP socket (SOCK_STREAM) as a connection between two LINX endpoints. As the TCP protocol is reliable, the CM itself has no mechanism for reliability of its own. This CM is suitable to use across the internet.

7.1 TCP CM Protocol Descriptions

With the Enea LINX TCP Connection Manager Protocol, a connection is established in the following manner.

- The TCP CM listens on port 19790 by default. Node A wants to connect to node B. A creates a TCP socket and connects it to B, sends a TCP_CONN message and then waits for a randomly amount of time for an acknowledgement. If an acknowledgment is not received, A will restart the connection procedure.
- B accepts the socket and when B wants to connect to A, it will lookup the previously accepted socket and read the TCP_CONN header. Then, it will send an acknowledgement TCP_CONN header to A.
- B considers the connection established if the send was successful and then notifies the upper layer of the established connection.
- A receives the TCP_CONN header and notifies the upper layer of the connection.

If both nodes try to connect to each other at the same time, neither of the nodes will receive an acknowledgment since the headers are sent on different sockets. This will lead to retries of the connection procedure. The timeouts for the retries are random.

7.1.1 TCP Connection Manager Headers

The Enea LINX TCP Connection Manager protocol defines the following header and package types.

Protocol number	Value	Definition
TCP_CONN	0x43	Connect type. Used for connection acknowledgement .
TCP_UDATA	0x55	User data type.
TCP_PING	0x50	Keep-alive header type.
TCP_PONG	0x51	Keep-alive response header type.

Table 42. TCP Connection Manager Protocol Header Types

7.1.1.1 TCP CM Generic Header

All messages in the TCP CM protocol have the following header. Only if **Type** indicates TCP_UDATA the fields source and destination are used - otherwise they must be set to zero. The size field is always used in the TCP_UDATA header and it may also be used in the TCP_CONN header.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0								1								2								3							
Reserved															O	Version								Type							
Source																															
Destination																															
Size																															

Table 43. TCPCM Generic Header

Field	Definition
Reserved	Reserved for future use, must be 0.
O	OOB bit, the TCP_UDATA is out of band.
Version	Version of the TCP CM protocol.
Type	Type of the current packet.
Source	Source link id. Used in type TCP_UDATA, otherwise 0.
Destination	Destination link id. Used in type TCP_UDATA, otherwise 0.
Size	Size of user data in bytes, followed by the header. Used in type TCP_UDATA and TCP_CONN, otherwise 0.

Table 44. TCPCM Header Description

7.1.1.2 TCP CM TCP_UDATA Header

All messages that don't originate from the Connection Manager are sent as TCP_UDATA. The user data is preceded by the TCP CM header with type TCP_UDATA.

7.2 TCP CM Connection Supervision Protocol

The two endpoints of a connection send TCP_PING headers to one another every configurable amount of milliseconds (default is 1000). When an endpoint receives a TCP_PING header, it will respond by sending a TCP_PONG header to the peer. If the connection goes down in any way, this will be detected and the CM will report this to the upper layer.