Final draft ETSI ES 201 803-2-1 V1.1.1 (2002-08)

ETSI Standard

Dynamic synchronous Transfer Mode (DTM);
Part 2: System characteristics;
Sub-part 1: Data link aspects

Reference

DES/SPAN-130005-1

Keywords

addressing, DTM

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, send your comment to:
editor@etsi.fr

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Services and Protocols for Advanced Networks (SPAN), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 2, sub-part 1 of a multi-part deliverable covering Dynamic synchronous Transfer Mode (DTM), as identified below:

Part 1: "System description";

Part 2: "System characteristics";

Sub-part 1: "Data link aspects";

Sub-part 2: "Network aspects";

Sub-part 3: "Transport network and channel adaptation aspects";

Part 3: "Physical Protocol";

Part 4: "Mapping of DTM frames into SDH containers";

Part 5: "Mapping of PDH over DTM";

Part 6: "Mapping of SDH over DTM";

Part 7: "Ethernet over DTM Mapping";

Part 8: "Mapping of Frame relay over DTM";

Part 9: "Mapping of ATM over DTM";

Part 10: "Routeing and switching of IP flows over DTM";

Part 11: "Mapping of video streams over DTM";

Part 12: "Mapping of MPLS over DTM";

Part 13: "System description of sub-rate DTM";

Part 14: "Network Management".

Introduction

The present document describes the architecture and protocols of signalling and maintenance on the data link layer of the DTM system.

1 Scope

The present document:

- establishes a system for bypass chain management;
- specifies the characteristics of bypass chain management;
- establishes a system for control signalling;
- specifies the characteristics of control signalling;
- establishes a system for bypass chain resource management;
- specifies the characteristics of bypass chain resource management.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

[1] ETSI ES 201 803-1: "Dynamic synchronous Transfer Mode (DTM); Part 1: System description".

[2] ETSI ES 201 803-2-2: "Dynamic synchronous Transfer Mode (DTM); Part 2: System characteristics; Sub-part 2: Network aspects".

[3] ETSI TR 101 287: "Services and Protocols for Advanced Networks (SPAN); Terms and definitions".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

access token: right to use a slot for transmitting data on a physical interface for a certain number of bypass hops

address, DTM address: 64 bit numerical value that uniquely identifies a node in a DTM network

NOTE: See further anycast address and multicast address.

allocation domain: same as a bypass chain where, if the topology is point-to-point or bus, the last node is not counted as member of the AD

Bypass Chain (BC): series of concatenated physical links, where data can be transported end-to-end using bypass switching

bypass switching: space switching of slots from a receiver to a transmitter on the same physical interface on a per slot basis

channel: set of slots allocated from one source Access node to one or more destination Access nodes in a network

NOTE: The source and destination Access nodes can be the same, i.e. the channel is internal to the node.

channel adaptation: channel path termination function that provides an adaptation between the DTM slot based service and some other traffic service (such as word stream, bit stream or asynchronous packets)

Channel Multiplexing Identifier (CMI): identifier by DCAP-1 used for multiplexing packets of different protocols on a single channel

control channel: channel used for DTM control signalling

data channel: channel used for transport data between interworking functions

DTM network address, DTM address: See definition of "address, DTM address".

DTM Service Type (DST): identifying the type of interworking function to be associated with the channel

DTM Service Type Instance (DSTI): identifying which interworking function of a specific type to be addressed by the channel

physical interface address: globally unique identifier of a physical interface that is represented as a 48-bit MAC address

master physical interface: physical interface in the allocation domain having the lowest physical interface address

multicast address: address used to represent all nodes belonging to a multicast group

network address: an address to be used in the network layer of the ISO OSI model

NOTE 1: A network address may take on different forms depending on which protocol is being used in the network layer.

NOTE 2: See also DTM network address.

node: network element containing DTM functions

node control channel: dynamically established control channel from one node to one or more other nodes on the same bypass chain

node identity: identifier that uniquely identifies a node over a global scope

ownership: responsibility to supervise an access token of a slot

physical interface: interface between two equipments

NOTE: Adopted from TR 101 287 [3].

physical link: unidirectional connection between the transmitter of one physical interface and the receiver of another physical interface

quark: smallest resource unit that is one slot wide and one physical link long

NOTE: Used to model dynamic resource management on a bypass chain.

return physical interface: physical interface that receives reply messages from a neighbouring node

switch: node that is capable of switching slots from one physical interface to another

slot: time slot within a frame, capable of transporting 64 bits of data or a number of special codes

topology: specific physical, i.e. real, or logical, i.e. virtual, arrangement of the elements of a network

NOTE: Two networks have the same topology if the connection configuration is the same, although the networks may differ in physical interconnections, distances between nodes, transmission rates, and/or signal types.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AD	Allocation Domain
aTSF	Trail Signal Failure action
BC	Bypass Chain
BCC	Bypass Chain Change
CMI	Channel Multiplexing Identifier
DCAP	DTM Channel Adaptation Protocol
DCC	DTM Control Channel function
DCFF	DTM Control Filtering and Forwarding
DCP	DTM Channel Protocol
DLSP	DTM Link State Protocol
DO	Dynamic Ownership
DPP	Detection Point Processing
DRP	DTM Routing Protocol
DRMP	DTM Resource Management Protocol
DSI	DownStream physical Interface
DST	DTM Service Type
DSTI	DTM Service Type Instance
DSYP	DTM SYnchronization Protocol
DTM	Dynamic synchronous Transfer Mode
FIFO	First In First Out
ID	IDentity
II	physical interface address
PrRpy	Probe Reply
QRequest	Quark Request
QReturn	Quark Return
QT	Quark Transfer
SDL	Standard Description Language
TDM	Time Division Multiplex
USI	Upstream Physical interface

4 Data link management system overview

A DTM network is composed of connected DTM nodes. A DTM node has one instance of the DTM protocol stack, and one or several physical interfaces. The technology support TDM switching and add/drop/bypass switching resulting in that the physical links of the nodes can be connected forming for example rings. The nodes can then be further connected to form a meshed network consisting of for example rings, point-to-point links and buses. The present document describes the management of data links. Throughout the present document, a DTM node is drawn as in figure 1, when topologies are shown. The picture shows a DTM node named 'B', with two physical interfaces, named B:1 and B:2.

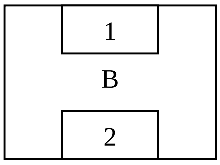


Figure 1: Node representation

4.1 Physical interfaces and physical links

A physical interface has a receive side and a transmit side. One property of the physical interfaces is that time slots can be bypass switched, in which data from a time slot on the receive side, is transparently copied to the same time slot on the transmit side of the physical interface. Bypass switching is default behaviour for time slots unless they are explicitly used for transmitting data on.

Slot 0, which is the first time slot in each DTM frame, is always used for transmitting data on and is thus never bypass switched.

A physical link, the smallest topological unit in a DTM network, is the direct connection from the transmit side of a physical interface to the receive side of a physical interface. In the real world, the physical link is typically synonymous to a fibre.

Since physical links are unidirectional, an arrow is used to depict the direction of data flow. Figure 2 shows DTM nodes A and B, connected with a physical link from A:1 to B:1.

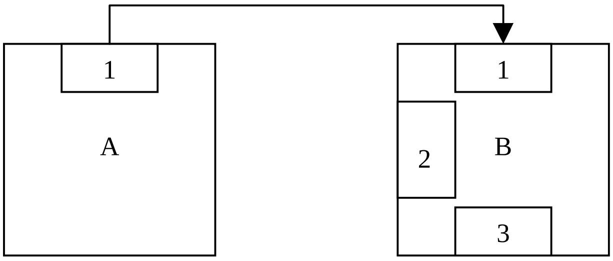


Figure 2: Physical links

Some times physical interface B:1 is said to be downstream of physical interface A:1, and A:1 upstream of B:1.

4.2 Bypass chains

When the physical links are connected such that data can be bypass switched in a circle, the full circle of physical links is referred to as a looped bypass chain. When physical links are connected such that data can be transferred through bypass switching, but not in a circle, the full range from the unconnected receive side to the unconnected transmit side, is called an open-ended bypass chain. In figure 3, a few examples of bypass chains are shown.

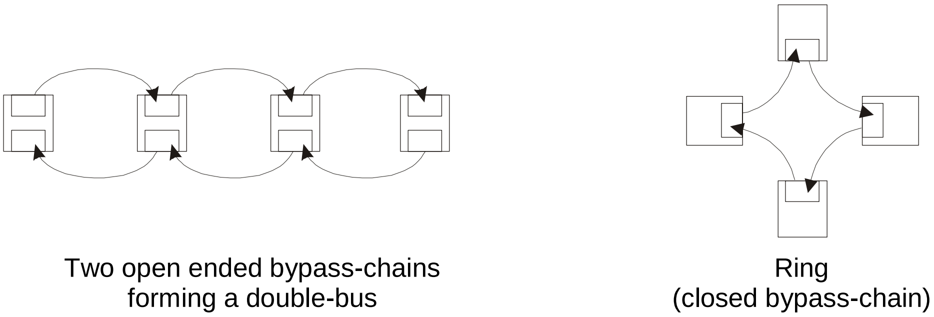
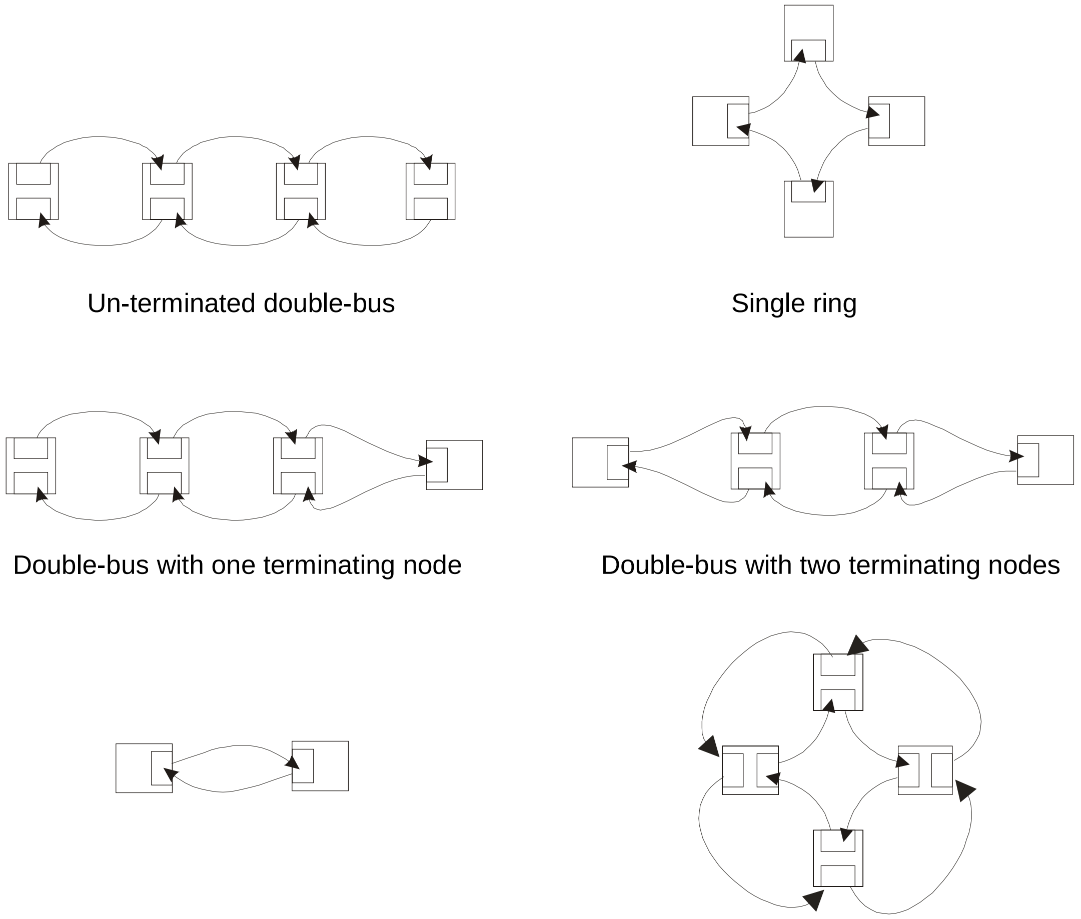


Figure 3: Bypass chains

DTM signalling is bi-directional, which means that single open-ended bypass chains are useless. Bypass chains cannot be discovered or used, unless they together allow bi-directional communication. All looped bypass chains allow bi-directional communication.

A number of ways of combining bypass chains that allow bi-directional communication are common enough to have been given names, see figure 4.



Un-terminated double-bus

Single ring

Double-bus with one terminating node

Double-bus with two terminating nodes

Point-to-point

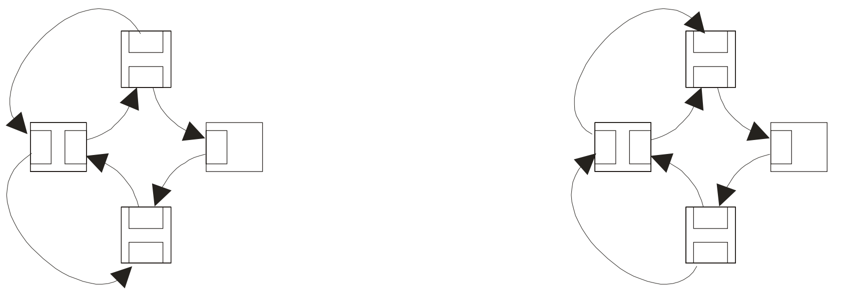
Dual ring

Figure 4: Combination of bypass chains

These are not the only valid combinations, however, and it is perfectly legal to add extra bypass chains to all the above, or to build completely different topologies, as long as the bypass chains follow two simple rules.

- All circular bypass chains are legal.
- For all open-ended bypass chains, there must be at least one physical link in the opposite direction for every physical link that constitutes the open-ended bypass chain.

For invalid combinations, all circular bypass chains are still usable.



Single-bus alongside ring or double-bus in reverse direction (valid combination)

Single-bus alongside ring or double-bus in same direction (invalid combination, the bus is not available)

Figure 5: Example of valid and invalid combinations of bypass chains

The services and protocols described in the present document all operate on bypass chains and deal with physical interfaces rather than DTM nodes.

4.3 Control Services

DTM provides two types of control services that the protocol stack uses for communicating with peering nodes. The Slot-0 service for unidirectional communication over a physical link, and the DTM Control Channel service, DCC, which allows communication over bypass chains. The Slot-0 service is both used for start up communication for some functions and as the normal communication for system critical functions. The DCC service provides a more efficient service with more capacity and shorter delay for communicating over a bypass chain with several physical interfaces connected.

4.4 Topology Management

The DTM system automatically discovers of topologies of the bypass chains, and monitors changes in their topology. This function is handled by the DTM Link State Protocol (DLSP), which discovers and monitors all valid combinations of bypass chains. DLSP also discovers when new nodes are added to the network and when nodes fail. DLSP operates as a continuos monitoring process. One important aspect of topology management is to allow for topology changes with a minimum of disturbance of the network.

4.5 Resource Management

The DTM Resource Management Protocol (DRMP) handles control of transmission resources over bypass chains. Resource Management handles the right to use a time slot for transmission and the right to administer usage rights. The latter function is referred to as slot ownership.

A physical interface owns a set of time slots on the bypass chain. The owner of a time slot has the right to use it for transmission, but can also lend the right to another physical interface if it is in greater need. Ownership can either be statically configured on a bypass chain, or dynamic. In the latter case, the Dynamic Ownership part of DRMP, distributes ownership along the physical interfaces on the bypass chain according to a configured policy. Typically the physical interfaces are configured to own an equal share that exhausts the capacity of the bypass chain.

The right to transmit on a time slot can be limited spatially, so as to cover only as many physical links as is needed on a bypass chain for the intended transmission. On the next physical link downstream of the receiver, and the next physical link upstream of the transmitter, the same time slot can be used again for other data transmissions. This spatial reuse of time slots is referred to as slot scoping.

5 Slot-0 Service

5.1 Functionality

Slot-0 is a simple mechanism to send messages to the next directly connected downstream node.

5.1.1 The Slot-0 data transport

The Slot-0 service can receive and allow transmission of data on Slot-0 for all physical interfaces on a node. The first slot in a DTM frame, slot number zero, must never be bypassed by the physical interfaces. This is to ensure that a message from an upstream node only reaches the first downstream node on the physical link.

5.1.2 Control traffic over Slot-0

Slot-0 does not contain any addressing information. The messages must be formatted according to DCAP-1. There are no other restrictions on the payload of the control message being transported. When a message is received, the received message is forwarded to the Slot-0 client, if there is a client that corresponds to the Channel Multiplexing Identifier (CMI) number in the message header. If the CMI number is not known the message is discarded. The Slot-0 service shall under no circumstances retransmit received messages on any physical interface.

5.2 Characteristics

- The Slot-0 communication is unidirectional and directly from physical interface to physical interface without addressing.
- The message transport is best effort.
- Strict FIFO ordering is guaranteed.
- No bypassing of any physical interface takes place. A message either reaches the next physical interface in the bypass chain, or is lost.

5.3 Addressing

All addressing on the Slot-0 service is implicit. Since Slot-0 is never configured in bypass mode, a message sent on Slot-0 for a physical interface will always reach the nearest downstream physical interface in the bypass chain or no physical interface at all. Likewise, a message received on Slot-0 for a physical interface is always sent from the nearest upstream physical interface on the bypass chain.

5.4 Client Selection

Clients to the Slot-0 service are selected through their CMI field. If a message is received on the Slot-0 service, with a CMI value, which no client has registered, the message will be discarded.

5.5 Slot-0 message and header formats

Slot-0 has no messages or headers of its own, and it does not have any restrictions on the packet format of its clients.

6 Bypass chain topology management

6.1 Bypass chain topology management overview

DTM Link State Protocol (DLSP) discovers the topology of bypass chains automatically and then monitors them to detect topology changes. As soon as DLSP discovers a bypass chain, it provides information to its clients (for example, DRMP and DCC) about the new topology. Notice that DLSP reports bypass chains to its clients, no assumptions are made on the bypass chains that constitute the network. During start up, DLSP provides topology information to DRMP as DLSP discovers a topology. This means that during start up, different nodes in the same bypass chain can have different information about the topology of the bypass chain. The view of the topology of the bypass chain eventually converges and all nodes will have the same view.

The connection of physical links between nodes can result in for example the topologies shown in figure 4. DLSP detects bypass chains in those topologies and reports them to its clients.

6.1.1 Void

6.1.2 Topology discovery algorithm

The goal with the topology discovery is to automatically discover the topology of physical links to avoid manual topology configuration. DLSP discovers the topology by finding bypass chains and distributes information on these bypass chains to all nodes that are part of the respective bypass chain. It also reports the information to its clients in the node.

The algorithm is divided into two steps:

- First the nodes establish a connection with their closest downstream and upstream neighbours and their physical interfaces.

- Then they build up and distribute a complete view of the bypass chains that go through one of their physical interfaces. Notice that only the nodes belonging to a bypass chain knows about it.

During the first phase, the physical link that connects the node to its closest downstream node must fulfil this criterion to be valid:

- The downstream neighbour must either have a direct physical link back to the upstream node, or, the physical link that connects the downstream node and the upstream node must be part of a bypass chain forming a ring.

Notice that both criteria can be satisfied like in figure 6, then DLSP chooses whatever way for the backward communication.

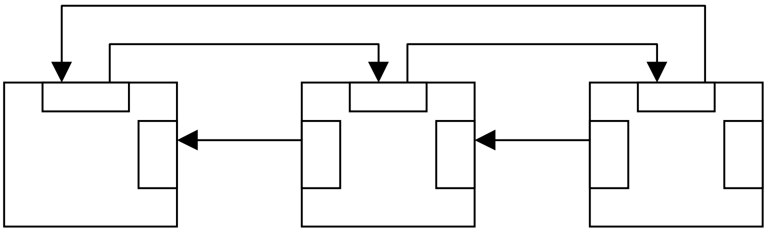


Figure 6: DLSP connectivity

During the second phase of the discovery algorithm, the nodes distribute their knowledge of the bypass chains they know by using the connections established in the first phase.

6.1.3 Bypass chain monitoring

DLSP monitors the detected topology for changes by continuously checking the detected bypass chains. If a bypass chain disappears or changes, the topology has changed and DLSP starts to re-evaluate the topology.

6.2 Service interfaces

6.2.1 Service provided

DLSP provides and maintains the status of a bypass chain topology. The following tasks are performed:

- Discovering the topology of the bypass chains.

- Monitoring the topology to detect changes either by locally detecting changes or by communication using DLSP.

Information is supplied to the DLSP clients using a service primitive:

- BypassChainChange(PHYSICAL INTERFACE, BYPASSCHAINDATA)

The service primitive is issued when a new physical interface is added or there is new information about the bypass chain that the physical interface is connected to. PHYSICAL INTERFACE is a physical interface address for a local physical interface. BYPASSCHAINDATA contains a list of node identities, physical interface addresses and DTM addresses in the order that the physical interfaces appear on the bypass chain. The same node can appear at several positions in the list if the node has more than one physical interface connected to the bypass chain. In that node, BypassChainChange will be issued for each change in the bypass chain, once for each physical interface.

The shortest bypass chain that can be delivered via the BypassChainChange service primitive contains the local physical interface only.

When a physical interface is removed from the node, the following service primitive is used:

- PhysicalInterfaceRemoved(PHYSICAL INTERFACE)

6.2.2 Service required

DLSP requires a DCC Slot-0 service for its transport, see further in clause 5.

DLSP requires that functions notify when a new physical interface is added to or removed from the node.

DLSP requires that a DPP protocol can detect physical link failure (i.e. Trail Signal Failure action, aTSF) for each incoming physical interface and signal that to DLSP.

6.3 Detailed protocol description

The operation of DLSP is divided into two processes, connection establishment and information distribution. Note that the two processes run concurrently. As soon as DLSP has obtained information on the topology, the information is checked for errors and delivered to DLSP's clients. The delivery of new topology information from DLSP to the clients is a continuous process. This means that the clients learns more and more about the topology, but it can not be known if DLSP has detected the entire topology until all detected bypass chains are reported as closed bypass chains, i.e. as rings or point-to-point links. Still new physical interfaces can be added to the bypass chain at any time. If the bypass chain is closed, it must of course be broken before a new physical interface can be added.

6.3.1 Establishing connections

The first phase of the DLSP protocol is to establish a connection between a transmitting physical interface and a receiving physical interface that are directly connected with a physical link. This establishment is handled by a TXprocess in the node with the transmitting physical interface communicating with a RXprocess in the node with the receiving physical interface, see figure 7.

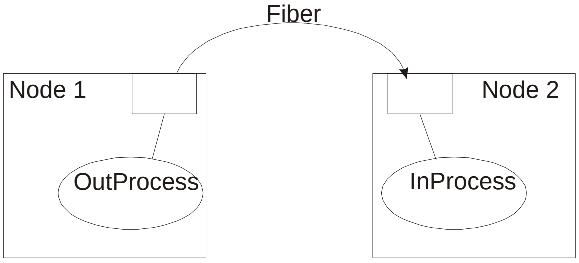


Figure 7: TXprocess and RXprocess connection

Each transmitting physical interface is connected to a TXprocess. The process can only send messages on that transmitting physical interface, but it can receive messages from any receiving physical interface. Incoming messages are sent to the TXprocess if their BLC_ID field matches the physical interface address of the transmitting physical interface. The TXprocess will discard all messages that arrive on a receiving physical interface where it does not expect to receive a message, see clause 6.5.1.6.

Each receiving physical interface is connected to a RXprocess. Messages received on this physical interface are sent to the RXprocess if they are not addressed to a TXprocess as described above. The RXprocess can send out messages on any transmitting physical interface.

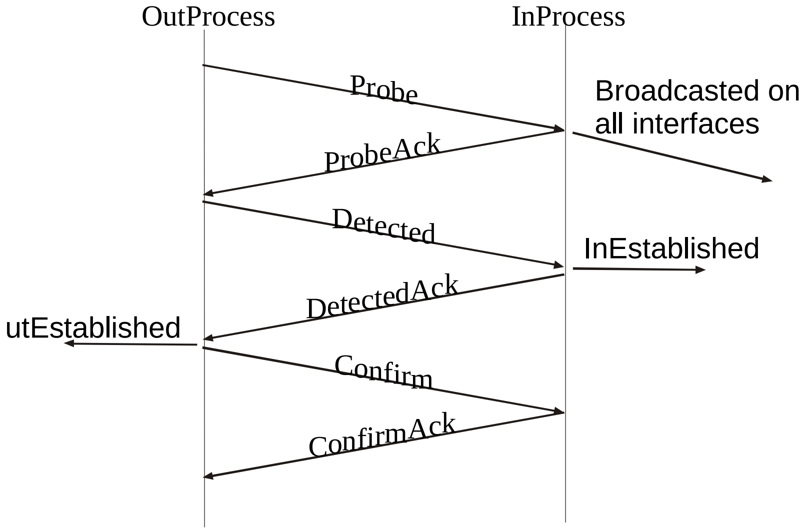


Figure 8: Sequence diagram for the establishment of a connection between an TXprocess and an RXprocess

6.3.1.1 Probing phase

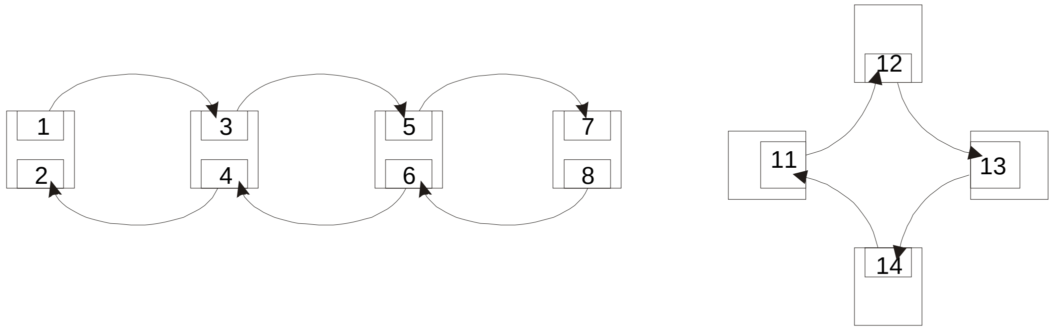
When the TXprocess is started (which happens when the node is started or when a new physical interface is added to a node), the TXprocess starts to send DLSP_PROBE messages on its physical interface periodically. The DLSP_PROBE message contains a BLC_ID field, which identifies a bypass chain connection by using the physical interface address of the transmitting physical interface. If another physical interface is connected to the transmitting physical interface via a direct physical link, the RXprocess connected to that physical interface will receive the DLSP_PROBE message.

When the RXprocess receives the DLSP_PROBE message, it shall respond with a DLSP_PROBE_ACK message. The RXprocess however has no knowledge of which physical interface to send messages back to the TXprocess. Therefore, it must send the DLSP_PROBE_ACK message to all physical interfaces in its node. Each DLSP_PROBE_ACK message contains a REPLY_IF field. In this field, the RXprocess writes a physical interface address for the physical interface that the DLSP_PROBE_ACK message was sent out via. The switched field contains 0 if the message was sent out on the same physical interface as the DLSP_PROBE arrived on and 1 otherwise.

If there is a direct path back from the node with the RXprocess to the node with the TXprocess, the DLSP_PROBE_ACK message reaches the TXprocess directly. If there is no direct way back (e.g. in a single ring configuration), the intermediate nodes in bypass forwarding the DLSP_PROBE_ACK message until it reaches the node with the TXprocess. The details of the message forwarding in DLSP are described in clause 6.3.1.2.

6.3.1.2 Return physical interface

Figure 9 A shows a topology where direct return physical interfaces exist. The RXprocess connected to physical interface 5 can send messages back to the TXprocess for physical interface 3 by sending them on physical interface 6. In figure 9 B, no direct return physical interface exists. If the RXprocess connected to physical interface 11 wants to send a message to the TXprocess in physical interface 14, and then it must send the message out on physical interface 11 and forwarded it via physical interface 12 and physical interface 13 before it reaches physical interface 14. All the RXprocesses in figure 9 B has RETURN_IF equal to zero to indicate that no direct return physical interface exist for them.



A) Topology with direct back-interfaces

B) Topology without direct back-interfaces

Figure 9: Topology examples of direct return physical interfaces

6.3.1.3 Detection phase

As the TXprocess receives the DLSP_PROBE_ACK message, it knows that there is bi-directional connectivity between itself and the RXprocess of the other node. It also knows that the RXprocess can use the physical interface in the REPLY_IF field to send messages back to it. The TXprocess now instructs the RXprocess that it is possible to communicate with the other nodes by sending out a DLSP_DETECTED message. The DLSP_DETECTED message contains a field named RETURN_IF. If the HC (hop count) field in the DLSP_PROBE_ACK message is zero (indicating that the message was sent directly from the node with the RXprocess without being forwarded by any intermediate nodes), the TXprocess includes the physical interface address from the REPLY_IF field in the RETURN_IF field in the DLSP_PROBE_ACK message. This tells the RXprocess that it should use this physical interface to send messages back to the TXprocess. If the HC field was larger than zero, the TXprocess sets the RETURN_IF to 0 to signal to the RXprocess that there is no direct return path, but the TXprocess can be reached by sending messages on the same physical interface that they arrive on.

When the RXprocess receives the DLSP_DETECTED message, it responds with a DLSP_DETECTED_ACK message sent on the physical interface contained in the RETURN_IF field, i.e. the physical interface address contained in the RETURN_IF field indicates where the message should be sent. After this, the RXprocess considers the connection as established and signals this to its DistributeUpstreamProcess.

When the TXprocess receives the DLSP_DETECTED_ACK message, it considers the connection as established and signals this to its DistributeDownstreamProcess.

6.3.1.4 Confirmation phase

The RXprocess and TXprocess now enter a monitoring phase where the TXprocess periodically sends DLSP_CONFIRM messages and the RXprocess responds with a DLSP_CONFIRM_ACK message. The DLSP_CONFIRM messages also serve the purpose of allowing the TXprocess to inform the RXprocess if it must switch to using another return physical interface. If the TXprocess sends the value zero in the RETURN_IF field, it means that there is no direct way back from the node with the RXprocess to the node with the TXprocess. Instead, the RXprocess should respond via the physical interface that it is connected to and the message is forwarded in bypass through a number of other nodes before it reaches the TXprocess.

If the TXprocess does not receive a DLSP_CONFIRM_ACK message, it retransmits the DLSP_CONFIRM message DLSP_MAX_CONFIRM_TRIES before declaring that there is no connection with the peer node. The RXprocess declares that the communication is down upon reception of a PhysicalLinkFailure signal for the receiving physical interface or if it does not receive any more DLSP_CONFIRM messages.

When either the RXprocess or the TXprocess has declared that there is no connection both of them must return to the initial state and restart the connection establishment with the DLSP_PROBE messages before the connection can be established again. This guarantees that both the RXprocess and the TXprocess (and thus also their respective DistributeUpstream and DistributeDownstreamProcesses) knows that the connection has been lost momentarily.

6.3.1.5 Distribution phase

During the phase when connection is established, the RX- and TXprocesses can exchange DLSP_DISTRIBUTE and DLSP_DISTRIBUTE_ACK messages for their DistributeUpstream and DistributeDownstreamProcesses. These messages can be sent from the TXprocess to the RXprocess as well as in the other direction via the return physical interface. An exception is if the RETURN_IF address is set to zero. If so the DistributeUpstreamProcess cannot send DLSP_DISTRIBUTE messages via the RXprocess, since upstream communication is only allowed if a direct path back exists. This is signalled from the RXprocess to the DistributeUpstreamProcess via the BackIfSet and BackIfNull signals.

6.3.1.6 Messages distribution

The messages used when establishing peer relationships are forwarded differently based on the type of message.

DLSP_PROBE, DLSP_DETECTED and DLSP_CONFIRM messages are never forwarded. They are sent out by a TXprocess and received by a RXprocess in a directly connected node.

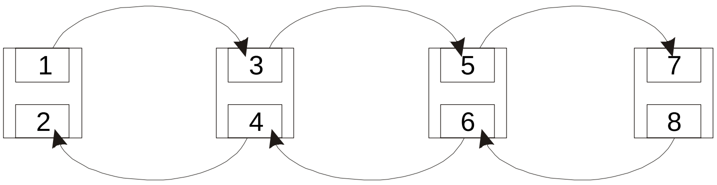
When a DLSP_PROBE_ACK message is received on a physical interface, the node first examines at the BLC_ID field and compares it with the physical interface address of a local physical interface. If they match, this node processes the message. If not, the message is forwarded on the same physical interface as it was received on if switched has the value false and HC (hop count) is less than DLSP_MAX_HOPCOUNT. Before forwarding the message, HC is incremented.

When a DLSP_DETECTED_ACK or DLSP_CONFIRM_ACK message is received on a physical interface, the node first examines the BLC_ID field and compares it with the physical interface address of a local physical interface. If they match, this node processes the message. If not, the message is forwarded on the same physical interface as it was received on if it has HC less than DLSP_MAX_HOPCOUNT. Before forwarding the message, HC is incremented.

6.3.2 Information Distribution Phase

The next phase in the operation of DLSP is the distribution of bypass chain information. Information about a bypass chain is distributed to all nodes that are part of the bypass chain but not to any other nodes. This is achieved by distributing the information along the bypass chain. Information can either be distributed in the downstream or upstream direction. Downstream distribution means that the information is sent via the same bypass chain as it describes. Upstream distribution means that the information is sent along the bypass chain in the opposite direction of the bypass chain. In this case, it is of course not possible to utilize the physical interfaces that are part of the bypass chain since the physical links are uni-directional. Instead, upstream distribution relies on the existence of a physical link between the same two nodes as a physical link of the bypass chain but in the opposite direction. Such a physical link does not exist in all valid topologies (for example single ring topologies). If this is the case, then upstream distribution is disabled and only downstream distribution is used.

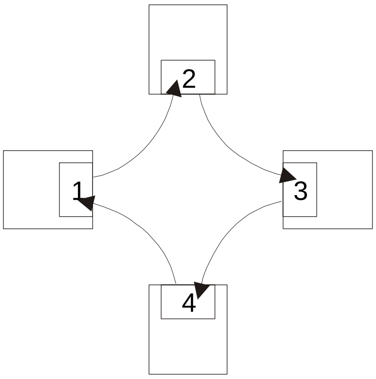
DLSP keeps track of the network topology using two lists of nodes per physical interface. The upstream physical interface list contains all nodes and physical interfaces upstream from a physical interface. The downstream physical interface list contains all nodes and physical interfaces downstream from the physical interface. Each entry in the two lists contains a node identity, a physical interface address and possibly a DTM address for the node. DLSP's clients can use the node identity to find out if there is more than one path to a particular node. The physical interface address can be used to identify each physical interface. DLSP does not use the information about the other nodes contained in the USI and DSI lists. It only checks the lists to see if it can find its own physical interface somewhere in the list to be able to detect loops. To avoid the false detection of loops, DLSP assumes that the physical interface identifiers are globally unique so two different physical interfaces can never have the same physical interface address.



1	usi: (empty) dsi: 3,5,7
3	usi: 1 dsi: 5,7
5	usi: 1,3 dsi: 7
7	usi: 1,3,5 dsi: (empty)

Figure 10: DLSP list example - dual bus

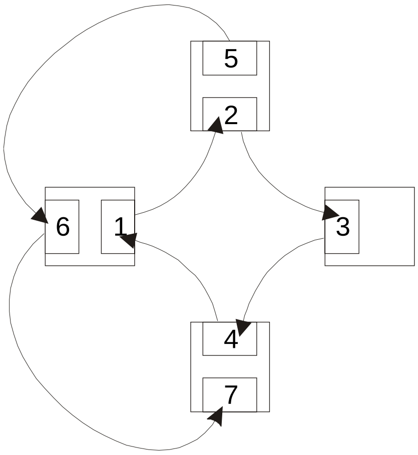
In figure 10, an example network with the contents of the USI and DSI lists for all the four top physical interfaces is shown. Note that in addition to the physical interface address, each position in the USI and DSI lists also contains a node identity and possibly a DTM address.



1	usi: 1,2,3,4 dsi: (empty)
2	usi: 2,3,4,1 dsi: (empty)
3	usi: 3,4,1,2 dsi: (empty)
4	usi: 4,1,2,3 dsi: (empty)

Figure 11: DLSP list example - ring

In figure 11, an example of the USI and DSI lists in a ring is shown. Note that since none of the RXprocesses in the picture have valid return physical interfaces, all DSI lists are empty. The necessary information to describe the ring is provided in the USI lists.



1	usi: 1,2,3,4 dsi: 2
2	usi: 2,3,4,1 dsi: (empty)
3	usi: 3,4,1,2 dsi: (empty)
4	usi: 4,1,2,3 dsi: 1,2
5	usi: (empty) dsi: 6,7
6	usi: 5 dsi: 7
7	usi: 5,6 dsi: (empty)

Figure 12: DLSP list example - complex topology

In figure 12, the USI and DSI lists of a more complicated network are shown. Note that the contents of the DSI lists for the inner ring (1, 2, 3, 4) depends on the existence of physical interfaces in the outer ring. For the inner ring, the USI lists contain enough information to describe the entire ring in all nodes. For the outer single bus (5, 6, 7), each node except the node with physical interface 3 has enough information to find the entire single bus. The node with physical interface 3 does not and should not know about the single bus since it is not part of it.

6.3.2.1 Information propagation

A node shall send out a DLSP_DISTRIBUTE message when the USI or DSI for one of its physical interfaces change. If the USI changes (i.e. the list of upstream nodes change), a DLSP_DISTRIBUTE message shall be sent downstream from the physical interface that the USI belongs to. This means that the node tells the next downstream node about the change, which in its turn notices that its USI list has changed and tells the node downstream from it and so on.

The USI or DSI for a physical interface can change either when a DLSP_DISTRIBUTE message is received from the neighbouring node or when the RXprocess (for USI lists) or TXprocess (for DSI lists) changes state from Established to Unknown. When a RXprocess or TXprocess is not established the USI or DSI should be empty, meaning that there is no knowledge of any physical interfaces in that direction.

When a RXprocess or TXprocess enters the Established state, the node should send out a DLSP_DISTRIBUTE message via the newly established process to update its neighbour with all the information the node has regarding nodes connected to the other part of the physical interface. I.e. if a TXprocess connected to physical interface X enters the state Established, the node should distribute the USI list for physical interface X via the newly established TXprocess.

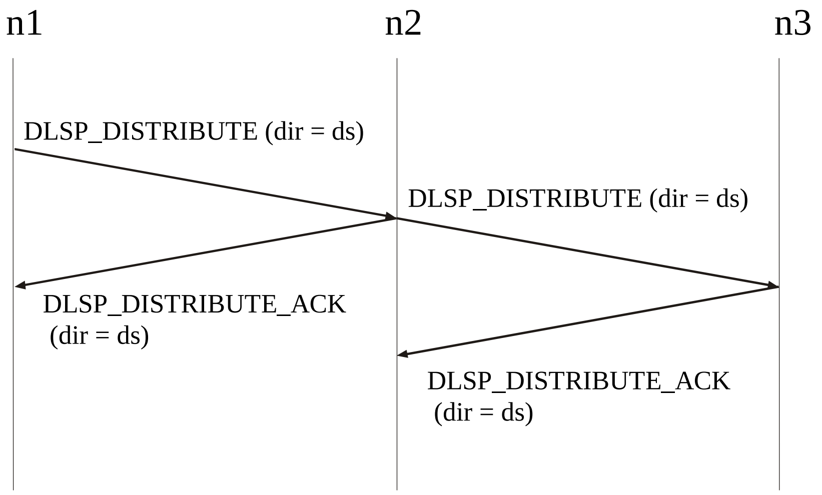

n1
n2
n3
DLSP_DISTRIBUTE (dir = ds)
DLSP_DISTRIBUTE (dir = ds)
DLSP_DISTRIBUTE_ACK (dir = ds)
DLSP_DISTRIBUTE_ACK (dir = ds)

Figure 13: Distribution example

In figure 13, an example of how DLSP_DISTRIBUTE messages propagate from node n1 to node n2 and on to node n3 is shown.

6.3.2.2 Ending the distribution

The distribution of information stops when either:

- The DLSP_DISTRIBUTE message reaches a physical interface where the TXprocess or RXprocess that the message should have been sent out on is not established (i.e. the bypass chain describes a single bus).
- A node receives a DLSP_DISTRIBUTE message with a DSI or USI that exactly matches the USI or DSI that it already has for that physical interface (in rings).

6.3.2.3 Detecting rings

A ring can be detected in DLSP by comparing the received USI and DSI lists. If the first physical interface in a received USI list or the last physical interface in a received DSI list describes the same physical interface as the physical interface list was received via, the physical interface list describes a ring.

6.3.2.4 Sending out node lists

The following actions should be performed on the USI list before sending it in the DLSP_DISTRIBUTE message:

1) Add the physical interface that the DLSP_DISTRIBUTE message should be sent out on after the last physical interface in the list.

2) If the last physical interface in the list is now equal to the first physical interface in the list, remove the first physical interface in the list.

The following actions should be performed on the DSI list before sending it in the DLSP_DISTRIBUTE message:

1) Add the physical interface that the DLSP_DISTRIBUTE message should be sent out on before the first physical interface in the list.

2) If the last physical interface in the list is now equal to the first physical interface in the list, remove the last physical interface in the list.

These rules ensure that information about the local physical interface is added to the DLSP_DISTRIBUTE message and if the list describes a ring, the list will not continue to be forwarded in the ring.

6.3.3 Topology calculation

When a change in the DSI or USI lists are detected for a physical interface, two actions must be performed:

- The change should be reported to DLSP's clients.
- The return physical interfaces used by peering RXprocesses should be evaluated to see if anything has changed.

6.3.3.1 Informing DLSP's clients

DLSP supplies topology information to its clients in the form of BypassChainChange service primitives. Each BypassChainChange contains a physical interface address and a list. The list describes the nodes attached to the bypass chain, to which the physical interface is connected, in the order that they are attached. The physical interface address is always found in the list. All nodes before the physical interface address are upstream from the physical interface and the nodes after the physical interface are downstream from the physical interface.

For the local physical interface where the change was detected, the nodes compare the USI and DSI lists and try to merge them. The two lists are merged according to the following algorithm:

1) Break the USI and DSI into physical links, i.e. if1->if2.

2) Remove any duplicates.

3) If any physical links contradict to each other, i.e. if1->if2 and if1->if3 or if4->if5 and if6->if5, discard all conflicting physical links.

4) Build one bypass chain by following physical links forward and backwards from the local physical interface.

Note that because of the removal of physical links in step 3, all available physical links might not be used in step 4.

6.3.3.2 Informing the RXprocesses about valid return physical interfaces

Each RXprocess has a return physical interface associated with it. The return physical interface is a physical interface that can be used to send messages directly back to the neighbouring node that is connected to the receiving physical interface. For example, in figure 13, physical interface 4 is the return physical interface for the RXprocess associated with physical interface 3. If there is no way to reach the node connected to the receiving physical interface directly (for example in a single ring), the return physical interface should be 0.

The node running the TXprocess connected to an RXprocess informs the RXprocess, which return physical interface it should use. This node is selected to do this since it will learn faster if the return physical interface becomes invalid (it will receive a PhysicalLinkFailure signal on the RXprocess for the return path) and can immediately calculate a new return physical interface and inform the peering RXprocess. A message to the BCManager triggers this calculation. Furthermore, the BCManager informs the TXprocess to send new information in a DLSP_CONFIRM message by sending it a SetPeerReturnIf signal.

The BCManager calculates available return physical interfaces by looking at the USI list for each of its local physical interfaces. The last entry in the USI list contains information about its closest upstream neighbour for that physical interface.

6.4 Messages

6.4.1 Physical interface identifiers field

The physical interface address is always located "to the right" in a DTM frame, with the layout shown in figure 14:

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
0	0	I-face Adr byte 5	I-face Adr byte 4	I-face Adr byte 3	I-face Adr byte 2	I-face Adr byte 1	I-face Adr byte 0

Figure 14: Physical interface address

Table 1: Physical interface address fields

Fields	Slot #	Bits	Size	Description
I-face adr	any	[47:0]	48	This is the field for the 48 bit physical interface MAC address.

6.4.2 General message format

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>VER</td><td>MT</td><td colspan="7">(message parameters)</td></tr>
</table>

Figure 15: Generic format of all DLSP messages

Table 2: Generic DLSP message fields

Name	Size (bits)	Description
VER	4	The version of DLSP. Should be set to zero
MT	4	Message type

6.4.3 DLSP_PROBE

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>0</td><td>reserved</td><td colspan="6">BLC_ID</td></tr>
</table>

Figure 16: Format of the DLSP_PROBE message

Table 3: Fields of a DLSP_PROBE message

Name	Size (bits)	Description
BLC_ID	48	The physical interface address of the TXprocess that sent the message, i.e. the physical interface address of the physical interface that the message was sent on.

6.4.4 DLSP_PROBE_ACK

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th colspan="2">bit 47-40</th><th colspan="2">bit 39-32</th><th colspan="2">bit 31-24</th><th colspan="2">bit 23-16</th><th colspan="2">bit 15-8</th><th colspan="2">bit 7-0</th></tr>
<tr><td>0</td><td>1</td><td>reserved</td><td>SW</td><td colspan="12">BLC_ID</td></tr>
<tr><td colspan="4">HC</td><td colspan="12">SW_IF</td></tr>
</table>

Figure 17: Format of the DLSP_PROBE_ACK message

Table 7: Contents of the fields in a DLSP_PROBE_ACK message

Name	Size (bits)	Description
BLC_ID	48	Copied from the DLSP_PROBE message that this message acknowledges.
HC	16	Hop count for the message. Should be set to zero when the message is sent originally and increased by one each time the message is forwarded.
SW_IF	48	The local physical interface that the DLSP_PROBE_ACK was originally sent on.
SW	1	0 = The DLSP_PROBE_ACK was sent on the physical interface that the DLSP_PROBE was received 1 = The DLSP_PROBE_ACK was sent on another physical interface.

6.4.5 DLSP_DETECTED

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th colspan="2">bit 47-40</th><th colspan="2">bit 39-32</th><th colspan="2">bit 31-24</th><th colspan="2">bit 23-16</th><th colspan="2">bit 15-8</th><th colspan="2">bit 7-0</th></tr>
<tr><td>0</td><td>2</td><td colspan="2">reserved</td><td colspan="12">BLC_ID</td></tr>
<tr><td colspan="4">reserved</td><td colspan="12">BACK_IF</td></tr>
<tr><td colspan="4">reserved</td><td colspan="12">node address</td></tr>
</table>

Figure 18: Format of the DLSP_DETECTED message

Table 8: Contents of the fields in a DLSP_DETECTED message

Name	Size (bits)	Description
BLC_ID	48	The physical interface address of the TXprocess that sent the message, i.e. the physical interface address of the physical interface that the message was sent on.
BACK_IF	48	The physical interface that the receiver should use as return physical interface. 0 if there is no direct way back and the bypass way should be used instead.
Node address	48	The node MAC address of the node that sent the DLSP_DETECTED message.

6.4.6 DLSP_DETECTED_ACK

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th colspan="2">bit 47-40</th><th colspan="2">bit 39-32</th><th colspan="2">bit 31-24</th><th colspan="2">bit 23-16</th><th colspan="2">bit 15-8</th><th colspan="2">bit 7-0</th></tr>
<tr><td>0</td><td>3</td><td colspan="2">reserved</td><td colspan="12">BLC_ID</td></tr>
<tr><td colspan="4">HC</td><td colspan="12">Node address</td></tr>
</table>

Figure 19: Format of the DLSP_DETECTED_ACK message

Table 9: Contents of the fields in a DLSP_DETECTED_ACK message

Name	Size (bits)	Description
BLC_ID	48	Copied from the DLSP_PROBE message that this message acknowledges.
HC	16	Hop count for the message. Should be set to zero when the message is sent originally and increased by one each time the message is forwarded.
Node address	48	The node MAC address of the node that sent the DLSP_DETECTED_ACK message.

6.4.7 DLSP_CONFIRM

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>4</td><td colspan="2">reserved</td><td colspan="6">BLC_ID</td></tr>
<tr><td colspan="4">reserved</td><td colspan="6">BACK_IF</td></tr>
</table>

Figure 20: Format of the DLSP_CONFIRM message

Table 10: Contents of the fields in a DLSP_CONFIRM message

Name	Size (bits)	Description
BLC_ID	48	The physical interface address of the TXprocess that sent the message, i.e. the physical interface address of the physical interface that the message was sent on.
BACK_IF	48	The physical interface that the receiver should use as return physical interface. 0 if there is no direct way back and the bypass way should be used instead.

6.4.8 DLSP_CONFIRM_ACK

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>5</td><td colspan="2">reserved</td><td colspan="6">BLC_ID</td></tr>
<tr><td colspan="4">HC</td><td colspan="6">reserved</td></tr>
</table>

Figure 21: Format of the DLSP_CONFIRM_ACK message

Table 11: Contents of the fields in a DLSP_CONFIRM_ACK message

Name	Size (bits)	Description
BLC_ID	48	Copied from the DLSP_CONFIRM message that this message acknowledges.
HC	16	Hop count for the message. Should be set to zero when the message is sent originally and increased by one each time the message is forwarded.

6.4.9 DLSP_DISTRIBUTE

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>6</td><td>reserved</td><td>D</td><td colspan="6">BLC_ID</td></tr>
<tr><td colspan="4">reserved</td><td colspan="2">ID</td><td colspan="4">reserved</td></tr>
</table>

Figure 22: Format of the DLSP_DISTRIBUTE message

The message is followed by a list of entries until the end of the message.

Table 12: Contents of the fields in a DLSP_DISTRIBUTE message

Name	Size (bits)	Position	Description
D	1	48	0=distribute downstream. 1=distribute upstream.
BLC_ID	48	47-0	If direction=0, SRC_II shall contain the physical interface address of the TXprocess that sent the message. If direction=1, SRC_II shall contain the physical interface address of the TXprocess that is the receiver of the message.
ID	16	47-32	An identifier that identifies the message so that a DLSP_DISTRIBUTE_ACK message can show which message it acknowledges. Shall be increased by one for each message sent by an TXprocess or RXprocess.

The remainder of the DLSP_DISTRIBUTE message is composed of a number of DLSP_DISTRIBUTE list entries as described below until the end of the DCAP-1 packet.

6.4.9.1 DLSP_DISTRIBUTE List Entries

<table>
<tr><th>bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td colspan="2">reserved</td><td>1</td><td colspan="6">Node address</td></tr>
<tr><td colspan="3">reserved</td><td colspan="6">Physical interface Address</td></tr>
<tr><td colspan="9">DTM network address</td></tr>
</table>

Figure 23: Format of a list entry with a DTM address

In the figure above, a DLSP_DISTRIBUTE message with DTM address is shown.

<table>
<tr><th>bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td colspan="2">reserved</td><td>0</td><td colspan="6">Node address</td></tr>
<tr><td colspan="3">reserved</td><td colspan="6">Physical interface Address</td></tr>
</table>

Figure 24: Format of a list entry without a DTM

In the figure above, a DLSP_DISTRIBUTE message without DTM address is shown.

Table 13: Contents of the fields in a DLSP_DISTRIBUTE List entry

Name	Size (bits)	Position	Description
has_dtm_address	1	48	Set to one when the entry contains a DTM address and zero otherwise. Bit 48 in first 64-bit word.
Node address	48	47-0	The node MAC address for the node.
Physical interface Address	48	47-0	The physical interface address for the node.
DTM network address	64	63-0	The DTM network address of the node. Only present if has_dtm_address is one.

6.4.10 DLSP_DISTRIBUTE_ACK

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>7</td><td>reserved</td><td>D</td><td colspan="6">BLC_ID</td></tr>
<tr><td colspan="4">HC</td><td colspan="2">ID</td><td colspan="4">reserved</td></tr>
</table>

Figure 25: Format of a DLSP_DISTRIBUTE_ACK message

Table 14: Contents of the fields in a DLSP_DISTRIBUTE_ACK message

Name	Size (bits)	Position	Description
D	1	48	Contains the same value as the DLSP_DISTRIBUTE message that it acknowledges.
BLC_ID	48	47-0	Contains the same value as the DLSP_DISTRIBUTE message that it acknowledges.
HC	16	63-48	Hop count for the message. Should be set to zero when the message is sent originally and increased by one each time the message is forwarded.
ID	16	47-32	Contains the same value as the DLSP_DISTRIBUTE message that it acknowledges.

6.4.11 DLSP_LINK_ERROR

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>8</td><td>reserved</td><td colspan="6">BLC_ID</td></tr>
</table>

Figure 26: Format of a DLSP_LINK_ERROR message

Table 15: Contents of the fields in a LinkError message

Name	Size (bits)	Description
BLC_ID	48	The physical interface address of the TXprocess connected to the physical link that has failed.

6.5 SDL Descriptions

6.5.1 Processes

The SDL description of DLSP is modelled with a number of processes that communicate with each other by passing messages. The interaction between the different types of processes is shown in figure 27. Note that there can be several processes of each type and the figure does not show which process each message can be sent to, only the type of process it can be sent to.

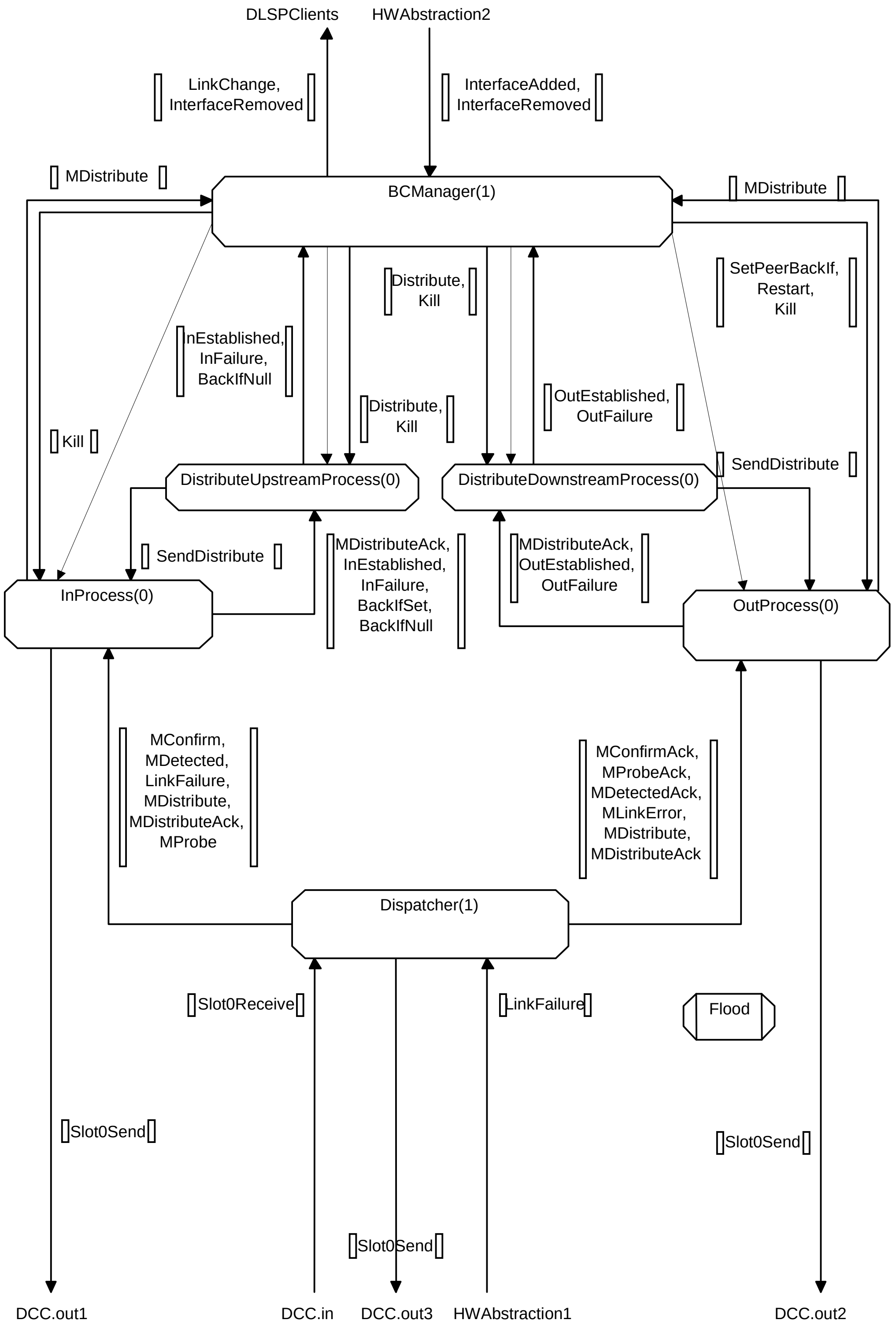

DLSPClients
HWAbstraction2
LinkChange, InterfaceRemoved
InterfaceAdded, InterfaceRemoved
MDistribute
BCManager(1)
MDistribute
Distribute, Kill
SetPeerBackIf, Restart, Kill
nEstablished, InFailure, BackIfNull
Distribute, Kill
OutEstablished, OutFailure
Kill
SendDistribute
DistributeUpstreamProcess(0)
DistributeDownstreamProcess(0)
SendDistribute
MDistributeAck, InEstablished, InFailure, BackIfSet, BackIfNull
MDistributeAck, OutEstablished, OutFailure
InProcess(0)
OutProcess(0)
MConfirm, MDetected, LinkFailure, MDistribute, MDistributeAck, MProbe
MConfirmAck, MProbeAck, MDetectedAck, MLinkError, MDistribute, MDistributeAck
Dispatcher(1)
Slot0Receive
LinkFailure
Flood
Slot0Send
Slot0Send
Slot0Send
DCC.out1
DCC.in
DCC.out3
HWAbstraction1
DCC.out2

Figure 27: Process interaction in DLSP

6.5.1.1 Relationships between processes in a node

Within a node, there is one Dispatcher process and one BCManager process. For each physical interface in the node, there is one RXprocess and an associated DistributeUpstreamProcess as well as an TXprocess and an associated DistributeDownstreamProcess. The BCManager creates these processes when it is informed about new physical interfaces and terminates the processes as physical interfaces are removed.

The Dispatcher and BCManager processes can send signals to any RXprocess or TXprocess. The BCManager can also send signals to any DistributeUpstream or DistributeDownstreamProcess.

An RXprocess only sends signals to its associated DistributeUpstreamProcess, not to any other DistributeUpstreamProcess and vice versa. The same is true for TXprocesses and DistributeDownstreamProcesses.

6.5.1.2 Relationships between processes in neighbouring nodes

As shown in figure 7, there is a relationship between an TXprocess and an RXprocess in neighbouring nodes. An TXprocess and its peering RXprocess keep track of each other. Together, they guarantee that if one of them moves into or out of the **established** state, the other one will eventually make the same transition. This means that if an TXprocess should for some reason be in state **established** and transition to **unknown**, the peering RXprocess leaves the state **established** and transition to **unknown** before the TXprocess can transition to state **established** again. These transitions are important for DLSP, since it guarantees that if A detects that it has momentarily lost contact with B, B consequently detects that it has momentarily lost contact with A.

6.5.1.3 DistributeDownstreamProcess

6.5.1.3.1 Purpose

Each transmitting physical interface has a DistributeDownstreamProcess associated with it. The DistributeDownstreamProcess is responsible for forwarding topology information downstream from its transmitting physical interface to the receiving physical interface connected to the transmitting physical interface via a physical link. This is only possible to do if the TXprocess is in **established** state.

6.5.1.3.2 Operation

The BCManager sends DLSP_DISTRIBUTE signals to a downstream process when there is new topology information that needs to be sent to any downstream node. The DistributeDownstreamProcess sends a Send Distribute signal to the TXprocess and the TXprocess sends out a DLSP_DISTRIBUTE message on its transmitting physical interface. The DistributeDownstreamProcess is responsible for making sure that the downstream node acknowledges the DLSP_DISTRIBUTE message with a DLSP_DISTRIBUTE_ACK message. If a DLSP_DISTRIBUTE_ACK message is not received within the time specified by DLSP_TIMEOUT_DISTRIBUTE, the DistributeDownstreamProcess resends the DLSP_DISTRIBUTE message. This continues until a DLSP_DISTRIBUTE_ACK message is received or the TXprocess signals OutFailure (communication with peer lost).

For consistent behaviour of the system, it is very important that all nodes on the same bypass chain see the same sequence of events. Therefore, the DistributeDownstreamProcess makes sure that the neighbouring node acknowledges all DLSP_DISTRIBUTE messages that the BCManager sends for distribution downstream. This means that the DistributeDownstreamProcess keeps a queue of DLSP_DISTRIBUTE messages that should be sent to its downstream neighbour. It waits for an acknowledgement for the previous DLSP_DISTRIBUTE message before the next message is sent.

There are two cases when the DistributeDownstreamProcess can discard DLSP_DISTRIBUTE messages that it has received from the BCManager:

- If the TXprocess is not in state **established,** all DLSP_DISTRIBUTE messages can be safely discarded since the BCManager sends a new DLSP_DISTRIBUTE message as soon as the TXprocess is established.

- If the DistributeDownstreamProcess receives a DLSP_DISTRIBUTE message from the BCManager that contains the exact same physical interfaces as a DLSP_DISTRIBUTE message that is in the queue of messages waiting to be sent, the DistributeDownstreamProcess can discard the old DLSP_DISTRIBUTE message. This is because a transition from topology A to B to C to B is equivalent to a transition from A to C to B for all the clients of DLSP.

The DistributeDownstreamProcess can be in two different states, **closed** and **open**. The DistributeDownstreamProcess is in **closed** state when its TXprocess is not established and in **open** state when the TXprocess is established. The DistributeDownstreamProcess changes between these states upon receiving the relevant signals from its TXprocess.

6.5.1.3.3 Provided services

The DistributeDownstreamProcess provides a reliable transport service for DLSP_DISTRIBUTE messages in the downstream direction.

It also sends OutEstablished and OutFailure signals to the BCManager when it receives them from the TXprocess. This is done to ensure that the DistributeDownstreamProcess always learns about physical interface changes before the BCManager.

6.5.1.3.4 Timers

A timer called DLSP_TIMEOUT_DISTRIBUTE is used when waiting for a DLSP_DISTRIBUTE_ACK message from the downstream node.

6.5.1.4 DistributeUpstreamProcess

6.5.1.4.1 Purpose

Each receiving physical interface has a DistributeUpstreamProcess associated with it. The DistributeUpstreamProcess is responsible for forwarding topology information upstream from its receiving physical interface to the transmitting physical interface connected to the receiving physical interface with a physical link. This is only possible to do in the case the RXprocess for the receiving physical interface has knowledge of a return physical interface that can be used to communicate with the upstream node. Therefore, the RXprocess informs its DistributeUpstreamProcess about the availability of a return physical interface.

6.5.1.4.2 Operation

The BCManager sends Distribute signals to a DistributeUpstreamProcess when there is new topology information that needs to be sent to any upstream nodes. The DistributeUpstreamProcess sends a SendDistribute signal to the RXprocess and the RXprocess sends a DLSP_DISTRIBUTE message on its return physical interface. The DistributeUpstreamProcess is responsible for making sure that the upstream node acknowledges the DLSP_DISTRIBUTE message with a DLSP_DISTRIBUTE_ACK message. If a DLSP_DISTRIBUTE_ACK message is not received within the time specified by DLSP_TIMEOUT_DISTRIBUTE, the DistributeUpstreamProcess shall resend the DLSP_DISTRIBUTE message. This process continues until a DLSP_DISTRIBUTE_ACK message is received or the RXprocess signals ReturnIfNull (no return physical interface exists) or InFailure (connection to peer lost).

For consistent behaviour of the system, it is important for all nodes on the same bypass chain to experience the same sequence of events. Therefore, the DistributeUpstreamProcess ensures that the neighbouring node acknowledges all DLSP_DISTRIBUTE messages that the BCManager sends for distribution upstream. This means that the DistributeUpstreamProcess keeps a queue of DLSP_DISTRIBUTE messages that should be sent to its upstream neighbour. It waits for an acknowledgement for the previous DLSP_DISTRIBUTE message before sending the next.

There are three cases when the DistributeUpstreamProcess can discard DLSP_DISTRIBUTE messages it has received from the BCManager:

- If the RXprocess is not **established** then all DLSP_DISTRIBUTE messages can be discarded since the BCManager will send a new DLSP_DISTRIBUTE message as soon as the RXprocess is established. If the RXprocess does not have a return physical interface, then all Distribute messages can be safely discarded since the BCManager will send a new Distribute message as soon as it receives a ReturnIfSet message from the DistributeUpstreamProcess.

- If the DistributeUpstreamProcess receives a DLSP_DISTRIBUTE message from the BCManager that contains the exact same nodes as a DLSP_DISTRIBUTE message that is in the queue of messages waiting to be sent, then the DistributeUpstreamProcess can discard the old DLSP_DISTRIBUTE message. This is because a transition from topology A to B to C to B is equivalent to a transition from A to C to B for all the clients of DLSP.

The DistributeUpstreamProcess can be in three different states, **closed**, **pending** and **open**. The DistributeUpstreamProcess is in **closed** state when its RXprocess is not established, in **pending** state when its RXprocess is established but there is no return physical interface and in **open** state when the RXprocess is established and there is a return physical interface. The DistributeUpstreamProcess changes between these states upon receiving the relevant signals from its RXprocess.

6.5.1.4.3 Provided services

The DistributeUpstreamProcess provides a reliable transport service for DLSP_DISTRIBUTE messages in the upstream direction.

It also sends InEstablished and InFailure signals to the BCManager when it receives them from the RXprocess. This is done to ensure that the DistributeUpstreamProcess always learns about physical interface changes before the BCManager.

6.5.1.4.4 Timers

A timer called DLSP_TIMEOUT_DISTRIBUTE is used when waiting for a DLSP_DISTRIBUTE_ACK message from the upstream node.

6.5.1.5 Dispatcher

The Dispatcher process receives DLSP messages from all physical interfaces and distributes them to the correct RX/TX process. The dispatcher process does the following:

- DLSP_DISTRIBUTE messages in the downstream direction, DLSP_DISTRIBUTE_ACK messages with direction equal upstream, PROBE, DLSP_DETECTED and DLSP_CONFIRM and messages are always sent to the RXprocess attached to the physical interface that the message arrived on.

- DLSP_DISTRIBUTE messages in the upstream direction, DLSP_DISTRIBUTE_ACK messages in the downstream direction, DLSP_PROBE_ACK, DetectedAck, DLSP_CONFIRM_ACK and PhysicalLinkError are sent to the TXprocess connected to the physical interface found in the physical interface address field of the message.

If the physical interface address is not a local physical interface, the Dispatcher forwards the message according to the forwarding rules described in clause 7.3.1.2. The Dispatcher also forwards the LinkFailure signal to the RXprocess that it belongs to.

6.5.1.6 TXprocess

6.5.1.6.1 Purpose

The TXprocess establishes and monitors the connection with a neighbouring RXprocess. There is one TXprocess connected to each physical interface. Each TXprocess has a relation to the DistributeDownstreamProcess for the same physical interface.

6.5.1.6.2 Operation

The TXprocess has three possible states, **unknown**, **detected** and **established**. The process starts in **unknown** state, transitions to **detected** when it receives a DLSP_PROBE_ACK message and then to **established** when it receives a DLSP_DETECTED_ACK message.

In the **unknown** state, the process performs the following tasks:

- Send a DLSP_PROBE message out on the physical interface periodically.

- Listen for DLSP_PROBE_ACK messages and transition to **detected** state upon reception of such a message.

When the process transitions to **detected** state, it sends out a DLSP_DETECTED message to its peering receiving physical interface.

In the **detected** state, the process waits for a DLSP_DETECTED_ACK message from its peering RXprocess. If no such message is received within DLSP_TIMEOUT_DETECTED milliseconds, it transitions back to **unknown** state. If another DLSP_PROBE_ACK message is received, it is ignored. A single DLSP_PROBE message can generate several DLSP_PROBE_ACK messages if several paths exist from the peering node back to this node.

When a DLSP_DETECTED_ACK message is received, the process signals OutEstablished to its DistributeDownstreamProcess, sends a DLSP_CONFIRM message to its peering RXprocess and transitions to **established** state.

In the **established** state, the TXprocess sends DLSP_CONFIRM messages to its peering RXprocess periodically. Two timers control the behaviour of the process in this state. The ConfirmPeriod controls the periodic re-transmission of DLSP_CONFIRM messages. The ConfirmTimeout determines for how long the TXprocess should wait for a DLSP_CONFIRM_ACK from its peer before re-sending the DLSP_CONFIRM message. The TXprocess should transmit the DLSP_CONFIRM message DLSP_MAX_CONFIRM_TRIES before giving up and transitioning back to **unknown** state.

DLSP_PERIOD_CONFIRM_OUT + DLSP_TIMEOUT_CONFIRM_OUT * DLSP_MAX_CONFIRM_TRIES milliseconds

The formula above gives the maximum time is takes before a TXprocess knows a physical link is down, which is a rather slow failure detection mechanism. However, this is not the primary method of detecting a physical link fault. The first that happens when a physical link fails is the receiving RXprocess's detection via the PhysicalLinkFailure signal. As a result of detecting the fault, the RXprocess floods a PhysicalLinkError message to all its physical interfaces. The PhysicalLinkError message contains the physical interface address of the transmitting physical interface for the physical link that appears to have failed. If there is a direct path from the node that runs the RXprocess to the node that runs the TXprocess, then the TXprocess receives the PhysicalLinkError message and transition to **unknown** state. The timeout mechanism described above is thus only used in the cases where either there is no direct path back or the PhysicalLinkError is dropped due to for example a bit error.

In the **established** state, the TXprocess also performs the following tasks:

- Responding to DLSP_DISTRIBUTE messages with DLSP_DISTRIBUTE_ACK messages.
- Forwarding DLSP_DISTRIBUTE messages to the BCManager.
- Forwarding DLSP_DISTRIBUTE_ACK messages to the DistributeDownstreamProcess.

The TXprocess always checks that messages from the peering RXprocess are received on the correct receiving physical interface. The RXprocess should always respond to messages via the return physical interface as instructed by the TXprocess. This means that the TXprocess should always know which physical interface the responses should arrive on and all responses received via other physical interfaces are discarded.

6.5.1.6.3 Provided services

The TXprocess informs its DistributeDownstreamProcess about its state by sending OutEstablished and OutFailure signals.

The TXprocess can send DLSP_DISTRIBUTE messages in the downstream direction for its DistributeDownstreamProcess. This is done via the SendDistribute signal.

6.5.1.6.4 Timers

This clause describes all timers used in the process.

6.5.1.6.4.1 ProbeTimer

Used in the **unknown** state when waiting for a DLSP_PROBE_ACK message from the peering RXprocess. It is set to DLSP_TIMEOUT_PROBE when a DLSP_PROBE message is sent.

6.5.1.6.4.2 DetectedTimer

Used in the **detected** state when waiting for a DLSP_DETECTED_ACK message from the peering RXprocess. It is set to DLSP_TIMEOUT_DETECTED when a DLSP_DETECTED message is sent.

6.5.1.6.4.3 ConfirmPeriod

Used in the **established** state to send DLSP_CONFIRM messages periodically. It is reset to DLSP_PERIOD_CONFIRM_OUT when it fires.

6.5.1.6.4.4 ConfirmTimeout

Used in the **established** state when waiting for a DLSP_CONFIRM_ACK message from the peering RXprocess. It is set to DLSP_TIMEOUT_CONFIRM_OUT when a DLSP_CONFIRM message is sent.

6.5.1.7 RXprocess

6.5.1.7.1 Purpose

The RXprocess establishes and monitors the communication with a neighbouring TXprocess. There shall be one RXprocess connected to each physical interface. Each RXprocess shall have a relation to the DistributeUpstreamProcess for the same physical interface.

6.5.1.7.2 Operation

An RXprocess has two possible states, **unknown** and **established**. The process starts in the **unknown** state and transitions to the **established** state when it receives a Detected-message.

In the **unknown** state, the RXprocess performs the following tasks:

- Responding to DLSP_PROBE messages with DLSP_PROBE_ACK messages on ALL transmitting physical interfaces in the node.
- Responding to a DLSP_DETECTED messages with a DLSP_DETECTED_ACK message followed by a transition into the **established** state.

Note that the RXprocess only sends a DLSP_PROBE_ACK message after receiving DLSP_PROBE messages.

In the **established** state, the RXprocess performs the following tasks:

- Responding to DLSP_CONFIRM messages with DLSP_CONFIRM_ACK messages.
- Responding to DLSP_DISTRIBUTE messages with DLSP_DISTRIBUTE_ACK messages.
- Forwarding DLSP_DISTRIBUTE messages to the BCManager.
- Forwarding DLSP_DISTRIBUTE_ACK messages to the DistributeUpstreamProcess.

In addition to this, the RXprocess stores the node MAC address and physical interface address of its peering TXprocess (or rather its physical interface) as well as the return physical interface that can be used to send messages to the peering TXprocess. The physical interface address is contained in each received DLSP_CONFIRM message. The RXprocess checks each incoming DLSP_CONFIRM message to make sure that the physical interface address of the peer does not change. If the physical interface address should change (due to a very quick change in network topology because of e.g. a protection switching in a WDM system), the RXprocess sends out a PhysicalLinkError message on all physical interfaces and transitions into the **unknown** state, thus forcing a re-establishment of the peering relationship.

The RXprocess shall transition from **established** to **unknown** state upon the following events:

- Reception of a DLSP_CONFIRM message with the wrong physical interface address.
- Reception of a PhysicalLinkFailure signal from the driver for the receving physical interface.
- If the time since the reception of the last DLSP_CONFIRM message is longer than DLSP_TIMEOUT_CONFIRM_IN.
- Reception of a DLSP_PROBE message. This indicates that the peering TXprocess is in **unknown** state, so the RXprocess should also transition to **unknown** state.

6.5.1.7.3 Provided services

The RXprocess informs its DistributeUpstream process about its state by sending InEstablished and InFailure signals. The RXprocess also informs the DistributeUpstream process about the existence of a direct way back to the peering TXprocess by sending the signals ReturnIfSet and ReturnIfNull. The RXprocess receives the information concerning the existence of a direct connection from the DLSP_DETECTED and DLSP_CONFIRM messages received from the TXprocess. The node running the peering TXprocess thus decides whether a direct path back exists from this RXprocess. This method is preferable since physical link failures are detected fast in the receiving node (via a PhysicalLinkFailure signal from the physical interface) and the node with the peering TXprocess will thus receive a notification quicker when the selected return physical interface no longer works due to a physical link failure.

The RXprocess can send DLSP_DISTRIBUTE messages in the upstream direction for its DistributeUpstreamProcess. This is done via the SendDistribute signal.

6.5.1.7.4 Timers

This clause describes all timers used in the process.

6.5.1.7.4.1 ConfirmTimer

Used in the **established** state when waiting for a DLSP_CONFIRM message from the peering TXprocess. It is set to DLSP_TIMEOUT_CONFIRM_IN when a DLSP_CONFIRM message is received. The timeout is long enough to take into account that DLSP_CONFIRM messages sometimes can be lost and the TXprocess must have time to retransmit them DLSP_MAXCONFIRM_TRIES times.

6.5.1.8 BCManager

6.5.1.8.1 Purpose

The BCManager is the main process of DLSP. It performs the following tasks:

- It detects when new physical interfaces are discovered and when physical interfaces are removed to create the appropriate processes for each physical interface.
- It allows clients to register with DLSP.
- It receives DLSP_DISTRIBUTE messages from the RXprocesses and TXprocesses and sends new DLSP_DISTRIBUTE messages via the DistributeUpstream and DistributeDownstreamProcesses.
- It informs all clients about the topology of the different physical links.

6.5.1.8.2 Operation

The BCManager has only a single state, **idle** and no timers. All the actions of the BCManager are event driven. The BCManager keeps track of all the available physical interfaces, their corresponding processes and USI and DSI lists.

The BCManager handles the information distribution algorithm as described in clause 7.3.1. It also monitors the return physical interfaces for its peering nodes as described in clause 7.3.3.2. This is done by receiving InFailure messages from the RXprocesses and if a RXprocess that is used for a back path fails, a new return path is calculated and a SetPeerReturnIf message is sent to the TXprocess peering with the RXprocess that needs a new return physical interface. If the topology calculations show that there is no direct return physical interface, but the topology is a ring, the RXprocess can send acknowledge messages via some other nodes. The BCManager sends a SetPeerBackIf signal with the argument 0. If no indirect return physical interface exists (i.e. there is no known path for the peering RXprocess to send a message back to the TXprocess), a Restart signal is sent to the TXprocess forcing it into the **unknown** state. This is done to more rapidly detect failures.

Note that a TXprocess is only restarted if the RXprocess that the return path was using fails. It is allowed for a return path to use an RXprocess that has not yet been established, since otherwise we would have a dead lock situation in a double-bus topology; both connections would depend on the other being established first.

NOTE: This note refers to the ProcessDistDownstream procedure. There is a check made to see if the received physical interface list is a smaller subset (i.e. it contains a subset of the physical interfaces in the list but not all) of the current USI list. If this is the case, an error has occurred somewhere and the node has had lost connectivity. In that case, the node checks the downstream node list to see if it describes the same physical link (e.g.) in the case of a double ring. If it does, the node knows that the downstream list is probably is false but the node has not yet been updated. The downstream list is truncated before calculating the topology and informing the clients of the new result to solve this problem. To avoid disturbing the Distribute process, the information that is distributed to the neighbour nodes are not updated, i.e. the DSI list after the topology calculation is restored. The mechanism described above is implemented to speed up the failure detection. The RXprocess always detects an error before the TXprocess (the RXprocess receives a LinkFailure signal from the hardware) and thus the DistributeDownstream mechanism will deliver notifications sooner than the DistributeUpstream mechanism.

6.5.1.8.3 Timers

None.

6.5.2 Data types

6.5.3 Functions

This clause describes all functions that are referenced by the SDL diagrams but not defined there.

6.5.3.1 IsBypassOk

Check if the bypass path from transmitting physical interface ever returns to this node. If it does, make sure that the incoming physical interface is still established. The following algorithm checks the bypass path:

1) Check if the transmitting physical interface address is equal to first physical interface in the Upstream physical interface list.

2) Loop through the upstream physical interface list to find the first local physical interface and determine if it is established. If the physical interface is found but is not established, it is not a ring topology.

3) If the end of the list is reached without finding a local physical interface, the entry after the last one is implicitly the local physical interface. Since the USI exists, this physical interface must be established, which means that there is a complete ring.

6.5.3.2 RecalcTopology

RecalcTopology takes one parameter, a physical interface address; this is the local physical interface for which the topology should be re-calculated. RecalcTopology should evaluate the algorithm, as described in clause 7.3.3.1 for the specified local physical interface and report the new topology for the local physical interface to all clients via a BypassChainChange signal. RecalcTopology should keep track of the result the last time the topology was calculated for this particular physical interface and only report a new topology to the clients if the topology has actually changed.

6.5.4 Constants

This clause describes the constants used in the protocol descriptions.

6.5.4.1 DLSP_MAX_HOPCOUNT

The maximum number of bypass hops that a message can be sent. This limits the size of the rings that can be built to DLSP_MAX_HC + 1 nodes.

6.5.4.2 DLSP_MAX_CONFIRM_TRIES

The number of times an TXprocess should try to send confirm messages without getting a response back, before declaring the TXprocess as **unknown**.

6.5.4.3 DLSP_MIN_PROBE

The minimum number of probes that an RXprocess must receive before sending a DLSP_PROBE_ACK message in response.

6.5.4.4 DLSP_TIMEOUT_DISTRIBUTE

The time a DistributeUpstream or DistributeDownstreamProcess should wait for a DLSP_DISTRIBUTE_ACK after sending a DLSP_DISTRIBUTE message.

6.5.4.5 DLSP_TIMEOUT_CONFIRM_IN

An RXprocess expects to receive Confirm messages periodically when it is in the Established State. If more than DLSP_TIMEOUT_CONFIRM_IN milliseconds have passed since it last received a Confirm message, the RXprocess shall transition to the **unknown** state.

6.5.4.6 DLSP_TIMEOUT_CONFIRM_OUT'

The time a TXprocess shall wait for a ConfirmAck message in response to its Confirm message. If a ConfirmAck message is not received within the time specified by DLSP_TIMEOUT_CONFIRM_OUT, the TXprocess shall resend the Confirm message or transition to the **unknown** state if it has already sent DLSP_MAX_CONFIRM_TRIES messages.

6.5.4.7 DLSP_TIMEOUT_FIRSTCONFIRM_IN

The time a RXprocess shall wait for a Confirm message from its peer after receiving a Detected message before giving up and transitioning back to the **unknown** state.

6.5.4.8 DLSP_TIMEOUT_DETECTED

The time a TXprocess shall wait for a DetectedAck message after sending out a Detected message.

6.5.4.9 DLSP_PERIOD_PROBE

The time a TXprocess shall wait between its periodic Probe messages.

6.5.4.10 DLSP_PERIOD_CONFIRM_OUT

The TXprocess should send a DLSP_CONFIRM message every DLSP_PERIOD_CONFIRM_OUT milliseconds.

6.5.4.11 Timer values

The following relationship between the timers must be respected:

Max round-trip time
< DLSP_TIMEOUT_CONFIRM_OUT
= DLSP_TIMEOUT_FIRST_CONFIRM_IN
< DLSP_MAX_CONFIRM_TRIES * DLSP_TIMEOUT_CONFIRM_OUT
< DLSP_PERIOD_CONFIRM_OUT
< DLSP_TIMEOUT_CONFIRM_IN

7 Control channel management

7.1 Overview

The DTM Control Channel system (DCC) is used for control signalling. DCC provides the higher level DTM protocols, such as DRMP, DRP [2] and DCP [2], with an interface to the control signalling system.

The DCC system operates in two modes:

- In the normal case a two-layered mode with dedicated control channels is used, see figure 28.

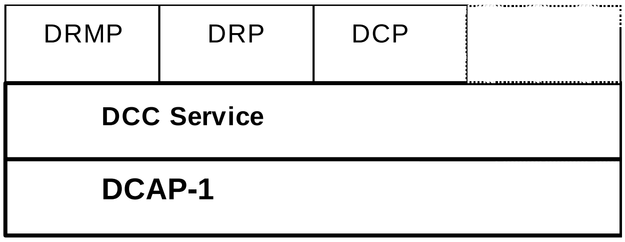


Figure 28: DCC established control channel mode

- In special circumstances, such as start up, dedicated control channels are not yet established. Instead a three-layer mode is used, see figure 29, which provides hop-by-hop forwarding. All signalling is here utilizing the Slot-0 service. At a slightly higher level is the DTM Control-Forwarding and Filtering, (DCFF) service, enabling addressing and downstream signalling. To the higher level protocols, the DCC system provides the same service when using the 3-layer and 2-layer mode, the only difference is the smaller capacity in the 3-layer mode, due to the one slot size of the Slot-0 service.

Common to both modes, the service provided by DCC offers addressed communication between physical interfaces along bypass chains. Dedicated control channels can be either point-to-point or point-to-multipoint.

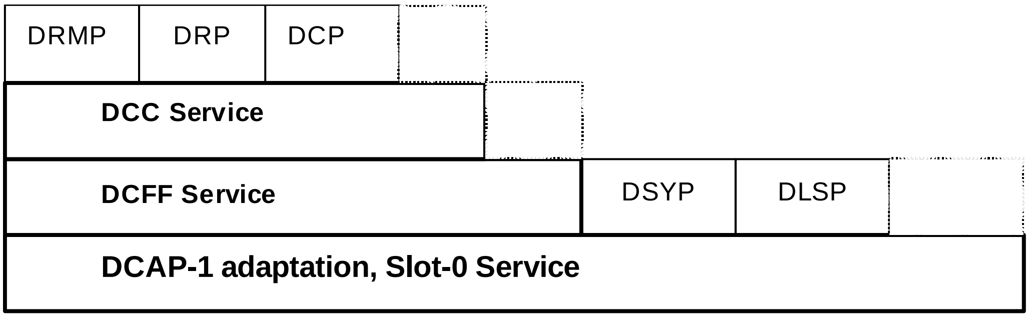


Figure 29: DCC Service 3-layer mode

The DCC service has the following characteristics:

- All messages are encapsulated using DCAP-1. DCAP-1 provides a Channel Multiplexing Identifier (CMI) that identifies the higher layer protocol information carried over the service. For example, DCP has a specific CMI number. DCC uses the CMI number to deliver the incoming control messages to the correct client.

- DCC does not guarantee delivery of sent messages.

- DCC relies on DLSP for information concerning which physical interfaces are present on the bypass chains.

7.2 Void

7.3 DTM Control Forwarding & Filtering Service

7.3.1 Functionality

The DTM Control Forwarding and Filtering (DCFF) Service is a Slot-0 client and is bound to local physical interfaces.

7.3.1.1 Handling of messages

All messages sent through DCFF are prepended with a DCFF Tunnel Header that contains addressing and a maximum message life time mechanism, see clause 7.4.1.

The addressing of DCFF enables messages to be addressed directly to a specific physical interface on a specific node or broadcast.

When a message is sent from a physical interface it is transported over the Slot-0 service to the nearest downstream physical interface. Depending on the contained address, DCFF will further forward the message to the next downstream physical interface in the bypass chain and/or deliver the message to the corresponding DCFF client. To avoid circulating messages a Hop Limit field is decremented for all forwarded messages.

The DCFF Tunnel Header is removed from the messages before the messages are delivered to the DCFF clients. The following procedure shows the decisions made for messages received by DCFF.

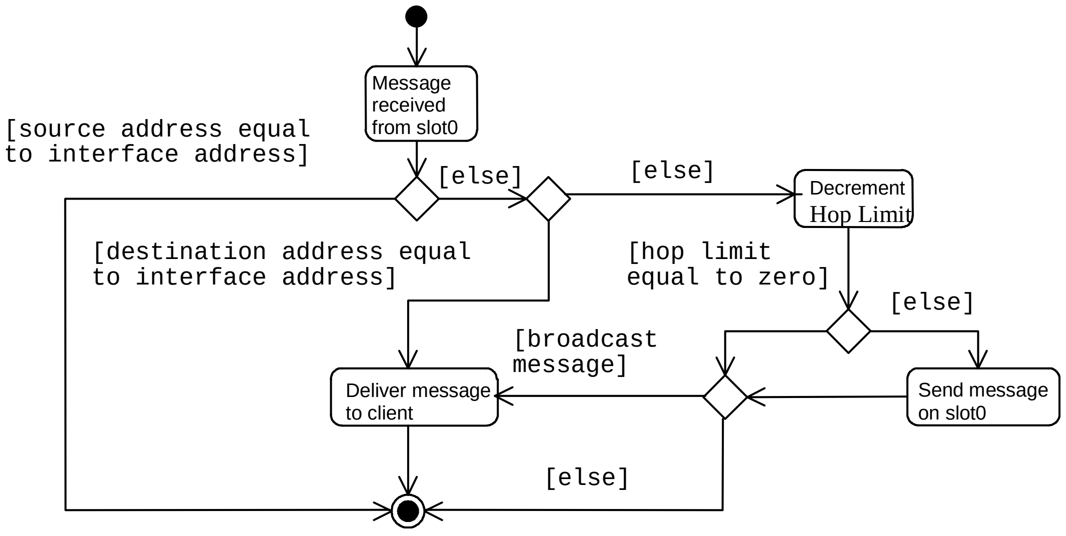

Message received from slot0
[source address equal to interface address]
[else]
[else]
Decrement Hop Limit
[destination address equal to interface address]
[hop limit equal to zero]
[else]
[broadcast message]
Deliver message to client
Send message on slot0
[else]

Figure 30: Decisions made for messages received by DCFF

7.3.2 Characteristics

DCFF has the following characteristics:

- Messages are either unicast or broadcast addressed.
- Messages are forwarded from physical interface to physical interface using best effort transport.
- DCFF guarantees strict FIFO ordering and no duplication for unicast messages.
- Broadcast messages might be duplicated.

7.3.3 Addressing

All messages sent over the DCFF service contain the destination physical interface address. The messages also contain the originating physical interface address to avoid message-passing loops. The destination address can be the broadcast physical interface address FF:FF:FF:FF:FF:FF, in which case all physical interfaces downstream the originating physical interface pick up the message and deliver it to local clients before forwarding the message further. Upstream physical interfaces cannot be reached.

7.3.4 Client Selection

Clients to the DCFF service are selected through the CMI field in the DCFF Tunnel Header. If the CMI field in the header of a received message does not correspond to that of a DCFF client, the message is discarded.

7.4 Messages and Headers

The messages must conform to the following payload format:

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
		Destination physical interface address					

Figure 31: Payload of DCFF clients

The lowest 48 bits in the first slot of all client payloads *must* be a physical interface address in little endian format. The address may be the broadcast address FF:FF:FF:FF:FF:FF. Apart from this, DCFF puts no restrictions on the contents of a client payload.

7.4.1 The DCFF Tunnel Header

DCFF adds a Tunnel Header described in clause 6.5.7.1 to all messages sent through it.

The DCFF Tunnel Header contains a Hop Limit counter, which ensures that messages do not circulate forever in the network in case the address is malformed or that, for some reason, the node containing the addressed physical interface can not receive the message. The Hop Limit counter is decremented by one at every physical interface where the message is forwarded. If the counter is equal to zero the message is discarded, see figure 4.

The source physical interface address is also included in the DCFF Tunnel Header. A received message is removed if the receiving physical interface address equals the originating physical interface address.

All DCFF messages contain a destination physical interface address that can be either the address of a physical interface or a broadcast address.

7.5 DCC Service

7.5.1 Functionality

DCC relies on bypass chain information given by DLSP. The bypass chain information is used by DCC to deduce which physical interfaces are upstream and downstream relative to local physical interfaces. Control signalling is normally done through control channels, except for example during the boot-up sequence, when DCFF is used. DCC will typically try to establish outgoing control channels that reach all downstream nodes. Signalling over dedicated control channels is illustrated in figure 32.

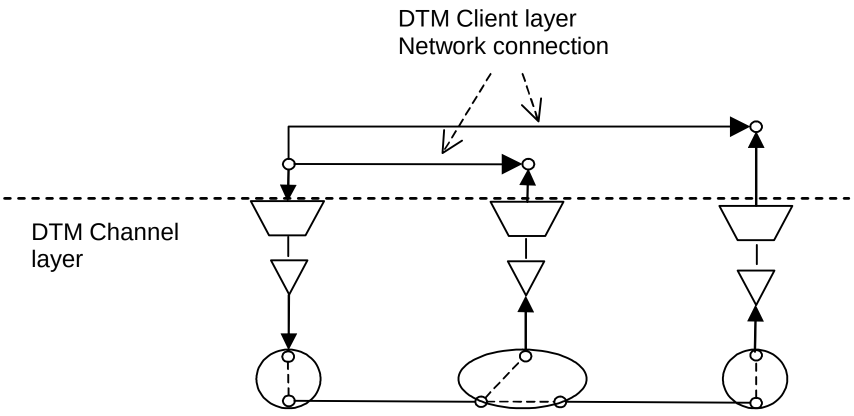


Figure 32: Signalling over dedicated control channels

Establishment of outgoing control channels is not mandatory, and if no control channel is established, the control signalling will continue over DCFF. This is shown in figure 33.

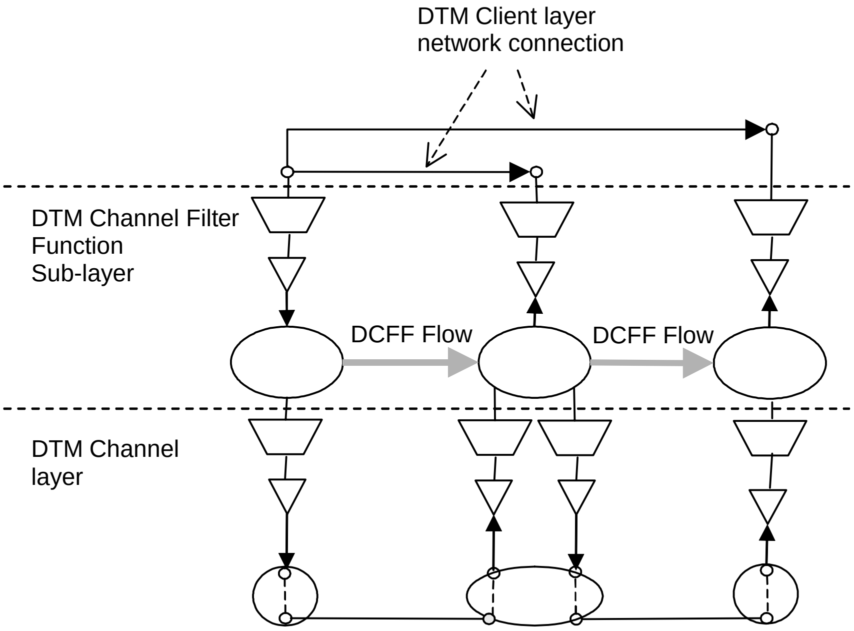


Figure 33: Signalling over DCFF

When a destination physical interface is removed from the topology, DCC removes the control channel to that physical interface.

7.5.1.1 Signalling path

The physical medium used for transmission connects the transmit side of one physical interface, with the receive side of another physical interface. A bypass chain is the concatenation of such connections, in a way that permits messages to be transmitted in bypass from an upstream physical interface to a downstream physical interface. Nodes can have more than one physical interface on the same bypass chain. For each node a bypass chain must conceptually be split such that the local physical interfaces become the boundaries for bypass chain segments. A downstream signalling path is the shortest path from one local physical interface to another node, in the downstream direction within the bypass chain segment that begins at the local physical interface. An upstream signalling path is the shortest path from a physical interface in another node to a local physical interface along the bypass chain segment that ends at the local physical interface.

EXAMPLE: If node B, in figure 34, wants to send a message to node C, there are many possible routes available. Since both physical interface 1 and 2 are attached to the same bypass chain one of them must be chosen. Either physical interface 1 or 2 in node C must also be picked. To determine the signal path, first split the bypass chain in segments as seen from node B. Segment 1 starts at C:3 and goes to B:1, segment 2 starts at B:1 and terminates at B:2. Segment 3 starts at B:2 and terminates at C:2. The downstream signalling path from B to C will, as described above, go from B:2 to C:1. There is accordingly no downstream signalling path to C:2. In the same way, there is an upstream signalling path from node C to node B, that goes from C:3 to B:1. There is no upstream signalling path from C:3 terminating in B:2.

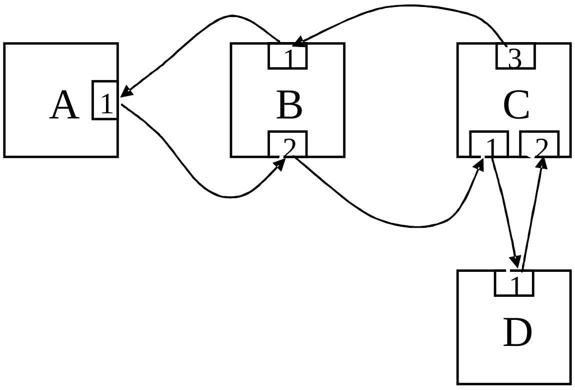


Figure 34: Control channel example

7.5.1.2 Route path finding

Messages can be sent from one physical interface to another physical interface, if both the receiving and sending node has knowledge about both physical interfaces.

1) If the destination physical interface is a receiver physical interface in the signal path downstream the transmit physical interface, that path is chosen to the receiver physical interface.

2) For all the physical interfaces, I_i, along the upstream bypass chain containing the destination physical interface. I_i is initially set to the destination physical interface and then in succession to the other physical interfaces along the bypass chain. If there exists a downstream signalling path from a local transmit physical interface to a receiver physical interface in the node containing I_i, the first matching physical interface found this way is chosen as the receiver physical interface.

3) If no path is found using rules 1 or 2 above, the message has to be discarded.

The receiver physical interface is always specified in a mandatory route header, necessary for message filtering.

If the destination physical interface differs from the receiver physical interface, a route header with the destination physical interface is added after the mandatory route header.

If the transmit physical interface found, differs from the source physical interface for a message, a source address header is added.

To address a node, a physical interface in the destination node is chosen. A route path is found according to the above rules. It is preferred to choose a destination physical interface in such a way that rule 1, above, applies.

To send from a node, the transmit physical interface found, according to the above rules, is chosen as the source physical interface.

A chosen path to a node should be kept conservatively. If the topology changes in such a way that it is possible to find a route path to a destination physical interface along a signalling path. I.e. rule 1 applies, the receiver physical interface is the same as the destination physical interface. In that case, an existing route path conforming to rule 2, shall be abandoned.

EXAMPLE 1: Send a message from node A to physical interface C:1 in figure 35. C:1 is the destination physical interface. C:1 is in the downstream signalling path from A:1. According to rule 1 A:1 is therefore chosen as the transmit physical interface and C:1 is chosen as the receiver physical interface. Now, by setting the source physical interface to the same as the transmit physical interface the source address header is not needed. Since the receiver physical interface is the same as the destination physical interface, no extra route header is needed.

EXAMPLE 2: Send a message from node C to node A in figure 35. The only physical interface in node A that node C has any knowledge about is physical interface A:1, which is part of the bypass chain {A:1->B:1->C:1}. A:1 becomes the destination physical interface. Rule 1 is not applicable, since the only signalling path from C is C:2 to B:2. Applying rule 2 gives that the transmitting physical interface is C:2 and that the receiving physical interface becomes B:2. The source physical interface is chosen as C:2, which eliminates the need for a source address header. The address of physical interface B:2 is put in the mandatory route header. The address of physical interface A:1 is put in an extra route header added after the mandatory one. At arrival in node B, the source physical interface is assumed to be C:2, since there was no extra source address header in the message. The destination is set to A:1, the address found in the extra route header. The route headers are stripped from the message.

B must now find a path to A:1. Rule 1 does not work so rule 2 is applied once more. Physical interface A:3 is chosen as receive physical interface and B:3 as transmit physical interface. The address of A:3 is put in the mandatory route header and an extra route header is added with the destination address A:1. The address of C:2 is put in a source address header, since the transmit and source physical interfaces are not the same. Upon reception in node A, the extra route header now gives the destination as A:1 and the source address header gives the source physical interface as C:2.

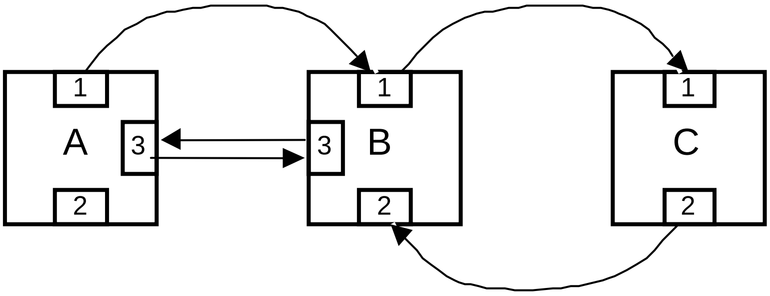


Figure 35: Upstream signalling

7.5.1.3 Establishment and Removal of control channels

DCC can establish control channels from each of the local physical interfaces to all receiver physical interfaces in the downstream signalling path belonging to the local physical interface. The control channels can be point-to-point or point-to-multipoint.

DCC shall only accept requests for control channel establishment coming from physical interfaces along the upstream signalling path of the channel's destination physical interface.

DCC establishes the control channels using DCP by creating a point-to-multipoint channel for DCC.

After the control channel has been created, DCC stops using DCFF and starts using the established control channel. Since DCC guarantees that the traffic is not reordered at the destination, the destination needs to be notified when the source is switching from DCFF to the established channel. DCC uses the DCC_FLUSH message and the control channel's source DSTI for this.

As soon as the destination has acknowledged the creation of the control channel, the source sends a FLUSH message over DCFF to all the control channels destinations, to indicate the change from DCFF to the established control channel, see figure 36.

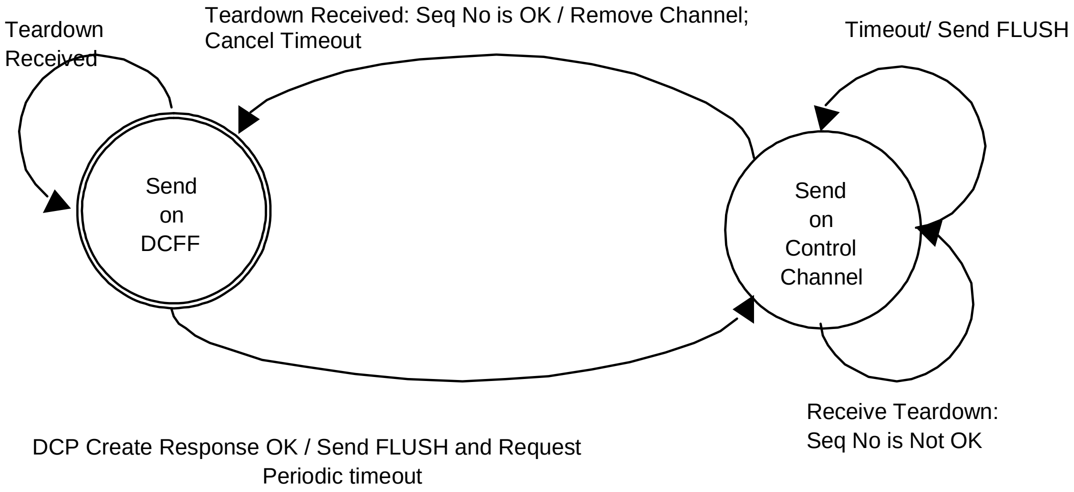

Teardown Received
Teardown Received: Seq No is OK / Remove Channel; Cancel Timeout
Timeout/ Send FLUSH
Send on DCFF
Send on Control Channel
Receive Teardown: Seq No is Not OK
DCP Create Response OK / Send FLUSH and Request Periodic timeout

Figure 36: Sending on DCFF and Control Channel

DCC in the destination starts to buffer incoming messages on the control channel immediately after the acknowledgement, see figure 37, and continues to receive messages over DCFF. When a FLUSH message is received over DCFF, with a sequence number matching the DSTI number (DTM Service Type Instance) of the control channel, DCC stops using DCFF and all signalling continues over the established channel.

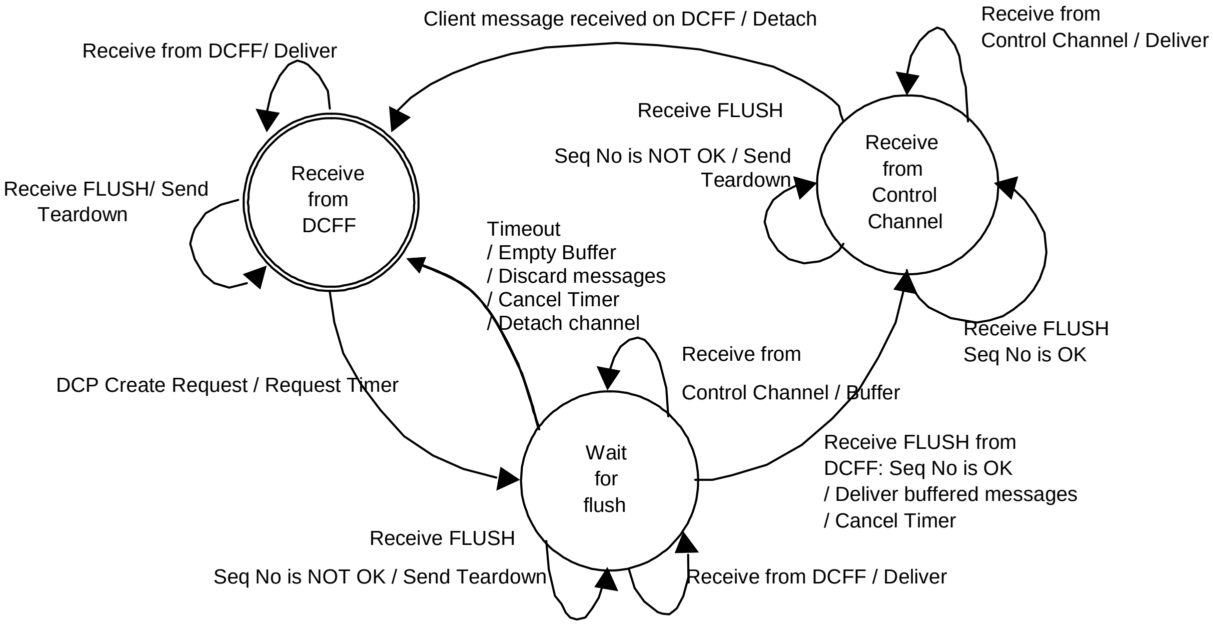

Client message received on DCFF / Detach
Receive from Control Channel / Deliver
Receive from DCFF/ Deliver
Receive FLUSH
Seq No is NOT OK / Send Teardown
Receive from Control Channel
Receive FLUSH/ Send Teardown
Receive from DCFF
Timeout
/ Empty Buffer
/ Discard messages
/ Cancel Timer
/ Detach channel
Receive FLUSH Seq No is OK
Receive from
Control Channel / Buffer
DCP Create Request / Request Timer
Receive FLUSH from DCFF: Seq No is OK
/ Deliver buffered messages
/ Cancel Timer
Wait for flush
Receive FLUSH
Seq No is NOT OK / Send Teardown
Receive from DCFF / Deliver

Figure 37: Receiving on DCFF and Control Channel

The DSTI for the control channel is used as a sequence number, i.e. it must be incremented at each channel connection attempt.

If a FLUSH message has not been received after a time has passed since the channel was acknowledged, the destination node orders DCP to remove the control channel. When a channel is removed, all the buffered control messages are discarded.

When a channel is established, the originator periodically sends the DCC_FLUSH message over DCFF, to all receivers for a control channel. DCC always monitors DCFF for internal control messages even when a control channel is available.

The source of a control channel is responsible for creating and removing control channels. To remove an established control channel the source first starts to use DCFF to maintain control channel functionality and orders DCP to remove the control channel.

A control channel is removed by the source of the channel when:

- The topology over which the control channel is established has been changed.
- The source physical interface can no longer be used.
- A destination has requested a removal of the control channel.

A DCC destination will request control channel removal locally when:

- The topology has changed and the control channel source is no longer an upstream physical interface as seen from the destination.
- The destination physical interface can no longer be used.
- The destination physical interface receives a message from DCFF not intended for DCC itself, but for a DCC client. The destination will then switch to use DCFF.

A DCC destination will request the sender to remove the control channel if:

- The destination has switched to DCFF but receives the DCC_FLUSH message from the sender over DCFF, see figure 37.

When the source of a control channel receives a Teardown message, the correct control channel is identified by the checking the local physical interface against the destination address in the Teardown message and the channels receivers against the source address in the Teardown message. If a match is found, the message is passed to the state machine for the matching control channel, see figure 38.

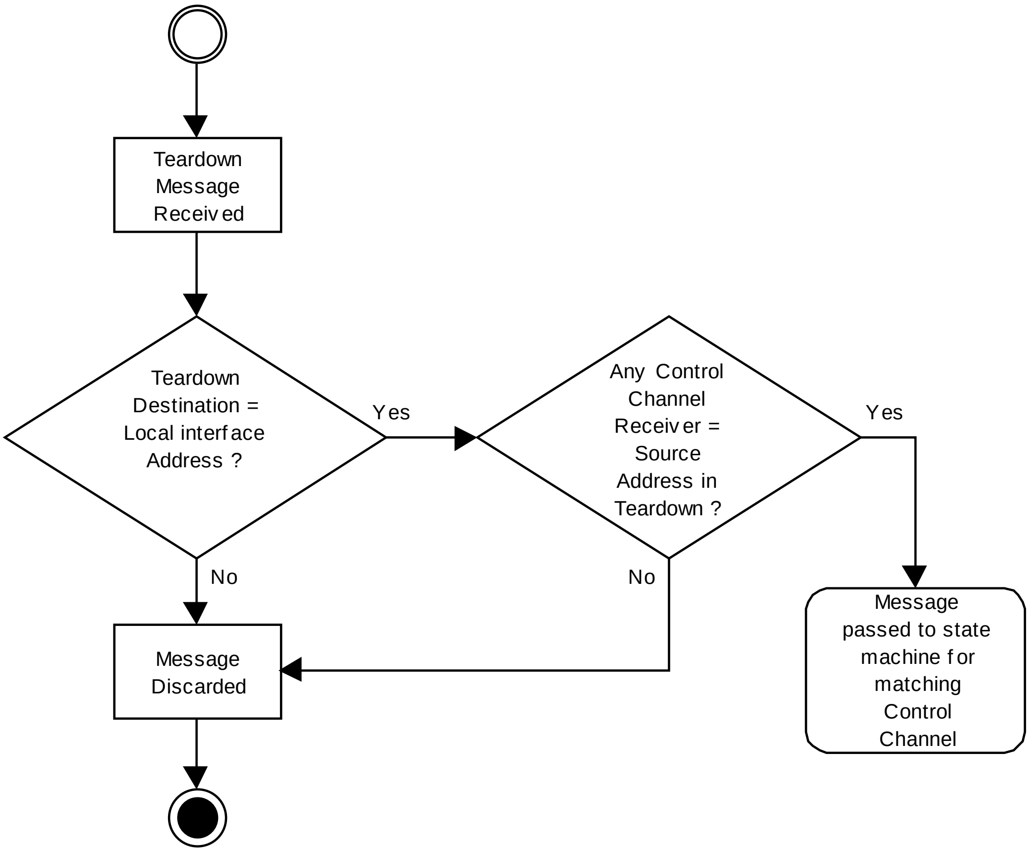

Teardown Message Received
Teardown Destination = Local interface Address ?
Yes
Any Control Channel Receiver = Source Address in Teardown ?
Yes
No
No
Message Discarded
Message passed to state machine for matching Control Channel

Figure 38: A control channels source node receives a Teardown message

7.5.1.4 Message filtering

When a message is received by DCC, the destination physical interface address in the message is compared to the local physical interface address and the broadcast address. If any of them match, the message is delivered to the receiver of the message. Otherwise the message is discarded. The lowest 48 bits in the first slot of all control messages contains the destination physical interface address.

7.5.2 Characteristics

DCC provides two different services: Sending messages by following a bypass chain and sending messages by back tracking a bypass chain. The DCC client selects which service to use.

7.5.2.1 Sending messages in the direction of the bypass chain

All messages sent following a bypass chain in the downstream direction are guaranteed to arrive in strict FIFO order without risk for duplication of broadcast messages.

- Messages are sent using a unicast or broadcast address.
- There is no guarantee that sent messages will arrive at the destination.

7.5.2.2 Sending messages in opposite direction of a bypass chain

Messages that are sent upstream a bypass chain can be reordered. The mechanism is intended for sending acknowledgement messages to upstream physical interfaces that cannot be directly reached. They must be reached by routing through an intermediate node.

- Messages are sent using a unicast address. Broadcast addressing is not allowed.
- There is no guarantee that sent messages will arrive at the destination.
- Messages are sent without strict FIFO guarantees. Messages can arrive in any order.

7.5.3 Addressing

7.5.3.1 Sending messages along a bypass chain

The message can be sent using unicast or broadcast physical interface addresses. A unicast message must contain the address of the destination physical interface. A broadcast message contains the broadcast physical interface address, FF:FF:FF:FF:FF:FF, and is forwarded to all physical interfaces downstream the originating physical interface. Note that in a ring topology, all physical interfaces can be viewed as being downstream.

7.5.3.2 Sending messages in opposite direction of a bypass chain

In this case only unicast addressing is possible. The messages contain the address of an upstream physical interface. If the upstream physical interface can be reached directly, for example in a ring topology, the message is passed directly to that physical interface; otherwise DCC must route the message towards the destination.

7.5.4 Client Selection

All clients to the DCC service have a unique CMI value. A received message is delivered to the DCC client whose CMI value matches the CMI field in the message. Messages with a CMI field that does not match any client's CMI value are discarded.

7.5.5 DCC message and header format

In this clause, the format of all message and header formats needed for the control signalling is defined. All fields in all messages are in 64-bit little endian format.

7.5.5.1 Payload of DCC clients

There are no restrictions on the payload of DCC clients.

7.5.5.2 DCC FLUSH message

This message is used when control channels are established. It is *always transported over DCFF*.

bit 63-56	**bit 55-48**	**bit 47-40**	**bit 39-32**	**bit 31-24**	**bit 23-16**	**bit 15-8**	**bit 7-0**
Type = 0x09	Reserved-A	Destination physical interface address					
Reserved-B						Sequence number	

Figure 39: DCC FLUSH message format

Table 17: DCC FLUSH messages fields

Fields	Size	Description
Type	8 bits	Message type. Set to 0x09.
Reserved-A	8 bits	Must be all zeros.
Destination physical interface address	48 bits	Destination interface address. It is always a unicast address.
Reserved-B	48 bits	Must be all zeros.
Sequence number	16 bits	FLUSH Sequence Number.

7.5.5.3 DCC Source Address Header

bit 63-56	**bit 55-48**	**bit 47-40**	**bit 39-32**	**bit 31-24**	**bit 23-16**	**bit 15-8**	**bit 7-0**
Type = 0x07	CMI	Source Physical interface address					

Figure 40: DCC Source Address Header format

Table 18: DCC Source Address Header fields

Fields	Size	Description
Type	8 bits	Message type. Set to 0x07.
CMI	8 bits	CMI of DCC client identifies the DCC client, since the message will be sent with DCC's CMI for the channel.
Source Physical interface address	48 bits	Address of the source physical interface.

7.5.5.4 DCC Route header

A message to a physical interface not located downstream on any bypass chain of the sending node can be sent by adding a DCC Route header. The message following this header must conform to the format of the DCC client payload described in clause 6.4, and the contained physical interface address must be the final destination physical interface.

bit 63-56	**bit 55-48**	**bit 47-40**	**bit 39-32**	**bit 31-24**	**bit 23-16**	**bit 15-8**	**bit 7-0**
Type = 0x06	CMI	Route Physical interface address					

Figure 41: DCC Route headers format

Table 19: DCC Route header fields

Fields	Size	Description
Type	8 bits	Message type. Set to 0x06.
CMI	8 bits	CMI of DCC client identifies the DCC client, since the message will be sent with DCC's CMI for the channel.
Route Physical interface address	48 bits	Address of intermediate physical interface. The address of a physical interface closer to the final destination.

7.5.5.5 DCC Tear Down

A message to the originator of a dedicated control channel. Upon reception the channel is removed if the destination in the message header matches a local physical interface address and if any receiver of the control channel matches the source address header and if the sequence number matches the channel.

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
Type = 0x0c						Flush Sequence Number	

Figure 42: DCC Tear Down format

Table 20: DCC Tear Down fields

Fields	Size	Description
Type	8 bits	Message type. Set to 0x0c.
Reserved	40 bits	Reserved for future use.
Flush Sequence Number	16 bits	

7.5.6 DCFF message and header formats

7.5.6.1 DCFF Tunnel Header

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
HOP LIMIT	CMI	Source physical interface address					

Figure 43: DCFF Tunnel Headers format

Table 21: Fields of the DCFF Tunnel Header

Fields	Size	Description
HOP LIMIT	8 bits	Max number of hops
CMI	8 bits	CMI number of client.
Source physical interface address	48 bits	Address of source physical interface

8 Resource management

8.1 System overview

8.1.1 Functional overview

DRMP efficiently distributes write access rights to the time slots in a DTM system. This is done with the following mechanisms:

- Ownership and Announcement of resources.
- Borrowing and Probing of resource tokens.

8.1.1.1 Resource ownership

The resource ownership is used to distribute the responsibility to handle the access right to use a slot, lend a slot to a remote node and monitor the slot.

In static operation mode, all slots on the bypass chain have their ownership defined at start up. It is possible to change the ownership distribution in a running system via the network management system.

In dynamic operation mode, a negotiation concerning the ownership of slots is started by all nodes on the bypass chain initially requesting how many slots a node wants to allocate. Dynamic operation mode avoids manual configuration of specific slot ranges. A master physical interface determines the allocation for the bypass chain and distributes information of the allocated ownership to all participating physical interfaces, including the master itself. Overlapping may occur transiently but the Dynamic Ownership (DO) ensures that this state does not persist and does not jeopardize consistent allocation of slots on the bypass chain.

8.1.1.2 Borrowing

If a unit finds that it has not got enough access rights for sufficiently many slots (called access tokens), it will attempt to borrow access rights to resources owned by other units. An owner can lend slots to other units, which is done by passing access tokens between the owner and borrower using Dynamic Resource Management Protocol (DRMP). When the access tokens for slots are no longer used for a channel, the access tokens are returned to the owner.

8.1.1.3 Resource announcement

A node continuously announce how many access tokens is can lend to other nodes on the bypass chain by doing resource announcements. The unit that receives an announcement stores the information to be used to decide from which node(s) it should issue requests for access tokens.

8.1.1.4 Probe mechanism

The Probe mechanism has three main functions:

- Initially create access tokens.
- Recreate lost access tokens.
- Resolve situations when two units simultaneously claim to have the access right to the same slot.

The probing is carried out in the same way for both cases. In order to do a successful probe of a set of tokens, the owner of the resource must do one query to **all** other units sharing this token and get a reply from all of them. Even if the owner of the slot use it locally, it must probe periodically to detect if some other node falsely has gained access of the slot. There are three results of a successful probe. The token may be:

- non-allocated;
- allocated once; or
- allocated by several nodes.

In point-to-point topologies, probing is not necessary since there is only one physical interface connected to the allocation domain and that physical interface owns all slots on the bypass chain.

8.1.2 Fault management

This clause describes the fault situations and how DRMP resolve them.

8.1.2.1 Borrowing failed

It can always be an inconsistency between a node's view of the amount of resources the other nodes has available and the correct situation, which results in that borrowing of access tokens might fail. If DRMP fails to allocate the minimum number of slots requested by its client, it signals the failure to the client, which is responsible for handling this situation.

8.1.2.2 Ownership overlapping

With statically allocated ownership, there can potentially be an overlap in the allocation of ownership resulting in that more than one node owns the slot, which can for example be the result of configuration errors. Such errors are reported to the network management system.

Dynamic ownership protocol ensures that either the ownership is not overlapping or the probe is not active.

8.1.2.3 Loss of access tokens

The access tokens of a message can be lost as a result of for example message loss. Lost access tokens are handled by the probing mechanism. At start up all tokens are considered as lost and the probe mechanism creates access tokens in the same way as if they were lost. Loss of access tokens results in performance degradation since not all capacity can be used for establishing channels.

8.1.2.4 Tokens allocated by several nodes

When to or more nodes have access right to the same slot at the same time is the most serious fault in DRMP. This can result in that the integrity of the data transported in the channels using these slots is violated. DRMP is designed to avoid this to happen. This problem is however very hard to avoid and can in rare occasions and under special circumstances occur. Therefore, the kill mechanism recovers from those unusual situations.

8.2 Concepts

8.2.1 Terminology

8.2.1.1 Allocation domain

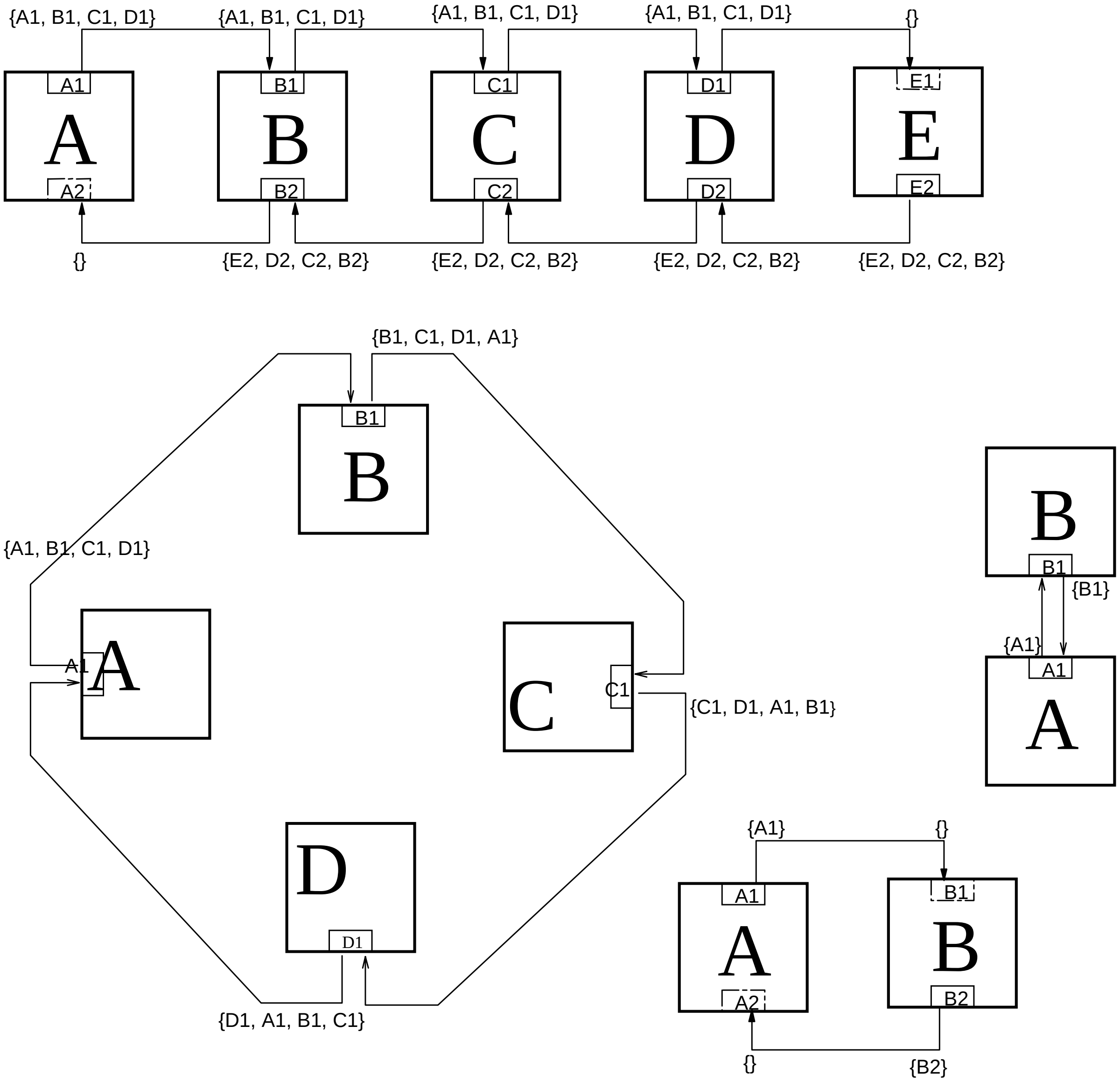


Figure 44: Examples of allocation domains

An allocation domain is a set of physical interfaces connected in a sequence, i.e., transmitting physical interface to next node receiving physical interface and from transmitting physical interface on the same physical interface to the next node receiving physical interface and so on. The allocation domain type can be point-to-point, bus or ring. In the bus case, the last physical interface is defined not to be part of the allocation domain. Its transmitter is not used and hence it needs no outgoing resource. The figure below shows: five-node bus, ring, two-node bus and point-to-point topologies respectively.

8.2.1.2 Short circuit

Two or several physical interfaces can belong to the same node. This is sometimes referred to as "short-circuited" physical interfaces. Figure 35 shows an example of this. This is handled the same way as if C:1 and C:2 would have been located in different nodes, with the exception that C:1 and C:2 communicate internally in the node and that channels originating in C:1 may not pass beyond C:2 and vice versa.

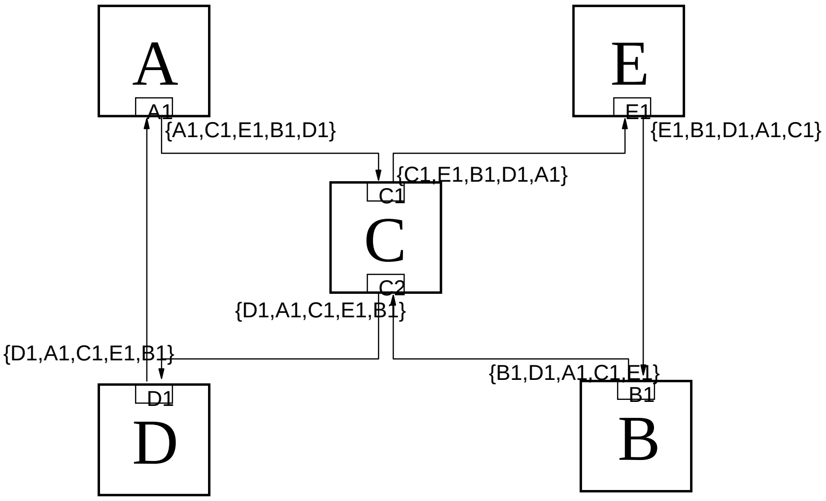


Figure 45: A short circuit

A short circuit is two physical interfaces that both belong to the same node and to the same allocation domain. In figure 45, the ports C:1 and C:2 communicate with each other using the ordinary DRMP messages, via a node-local control channel.

8.2.1.3 Probe

The probe mechanism is responsible for detecting that there is an access token missing and if so, recreate an access token for the specific slot. The node that owns a specific slot is responsible for probing the slot. The DRMP_PROBE is issued to all nodes in the allocation domain asking them if they currently use the slot. If the node that owns the slot expects that a node uses a probed slot but finds out that no one does so, it creates an access token for the slot and adds it to its free pool. The probe is the only mechanism responsible to recreate tokens. To avoid the risk of ownership overlaps and thus the risk of that several nodes has access to the same slot, the probe mechanism is turned off during ownership changes.

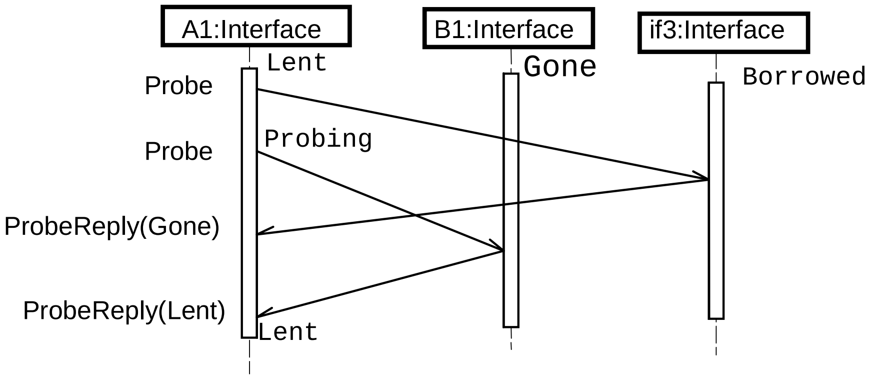


Figure 46: Typical probe

Figure 46 shows a typical probe session. The probe simply checks that a borrower still has the slot. Probing is done because of two reasons:

- At start up to get the resources initially, all nodes must request other nodes for access tokens for its slots since they can be lend by a previous owner (or by this node before a reboot).
- Access tokens can get lost when transmitted from one node to another.

The node has just started. It always starts with no resources but with an ownership range assigned. This means that it also has a responsibility to probe its resource tokens at a regular interval. At start up, the probe is intensified, which is called fast probe mechanism. The fast probe procedure is to make at least one successful probing for each slot and then continue probing at lower frequency. The fast probe is also re-triggered whenever the topology of the specific allocation domain changes (due to physical link or node failures).

The fast probe is the mechanism of sending probes at a faster rate, which can potentially overload resources in the network. To avoid overload, a simple flow control mechanism is used where the last DRMP_PROBE_REPLY message, for a given slot or block of slots, triggers the transmission of the next DRMP_PROBE. This flow control mechanism is only used during fast probe phase.

8.2.1.4 Borrowing and lending of token(s)

Tokens can be borrowed from other nodes in the allocation domain.

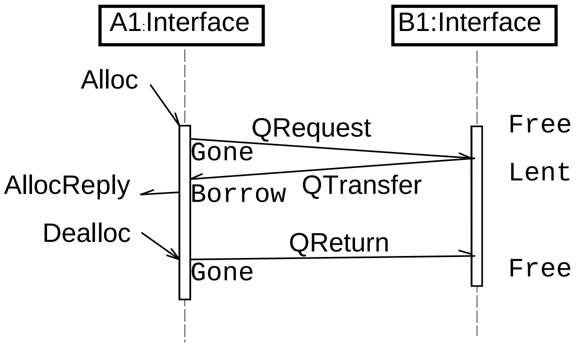


Figure 47: Example of borrow and return of resources

In figure 47, an example of a typical borrow session is shown. Physical interface A:1 borrows from B:1. After the channel is removed, the borrowed access token is returned to the owner.

8.2.1.5 Ownership of tokens

The ownership of tokens is independent of the access to tokens, which means that changing ownership does not necessarily change what node currently has the token for slot. The intention of the distributed ownership is to try to have node local communication resources where they are needed, since borrowing and lending process with remote nodes result in takes more time and communication resources than local allocation.

8.2.1.6 Access to token(s)

The access to tokens means that the node either is already using the slot or has the right to do so. For example, the node keeps the access tokens in the local free list or uses it for an active channel.

8.2.1.7 DRMP_GATHER

This message sent out from a physical interface that requests a new ownership distribution as a result of for example need for more capacity to set up channels or as a result of management actions.

8.2.1.8 DRMP_SYNC

The master node (or a node that has the impression that it is the master node) sends the DRMP_SYNC message to inform all nodes in the allocation domains which slots are owned by whom.

8.2.1.9 Start up

The process taking place when a node is started or restarted.

8.2.1.10 Master physical interface

The master physical interface is defined as the node connected to the allocation domain having the physical interface address with lowest value. The arbitration of master physical interface can change as a result of failing nodes, new nodes, etc. during operation.

8.2.1.11 Transitional master physical interface

This is the name of a physical interface, which becomes the master for a short period, since it has inconsistent view of the topology of the allocation domain. Other physical interfaces can have different information concerning the topology and which node currently is the master physical interface. Since the full allocation domain information is contained in the DRMP_SYNC message, it is easy for a physical interface to determine if a DRMP_SYNC message should be ignored.

8.2.1.12 Probe

The probe mechanism checks the consistency of access tokens in the allocation domain. It is also responsible for resolving resource conflicts and re-creating tokens that have been lost and at start up initially create of the access tokens.

8.2.1.13 Topology

A topology is a set of physical interfaces given to us as an ordered list from DLSP. A topology is either a ring or a bus and the last terminating physical interface is included in the topology.

8.2.1.14 Fairness algorithm

This is the algorithm that is used by the master of the dynamic ownership system to calculate the total amount of slots that each node should own. Please note that it is the policing parameters that are sent from each node which are distributed to each node and the calculation of the fairness takes place at each node, not only at the master node. Therefore, it is important that the fairness algorithm is standardized.

8.2.1.15 Range change

This is an incoming event to the Master physical interface telling it how many slots a specific physical interface would like to own. If the requests for the total amount of slots exceed the total available range, a fairness algorithm is used for this calculation.

8.2.1.16 Double booking

Double booking is said to have occurred when two or more physical interfaces believe they have access right to the same slot (or set of slots).

8.2.1.17 Quark

This is the smallest resource unit available. It is defined to be one slot "high" and one physical link "wide". The quark machine defines resource management with the assertion that any implementation will be an aggregation of several of these Quarks. The Quark is only used to simplify the description of DRMP in the present document.

8.2.1.18 Got ownership

An event telling the quark state machine it now owns its quark.

8.2.1.19 Lost ownership

An event telling the quark state machine it does no longer own its quark.

8.2.1.20 Alloc

Telling the quark state machine its resource is used for a channel.

8.2.1.21 Dealloc

Telling the quark state machine its resource is no longer used for a channel.

8.2.1.22 Fragmentation

As the pool of tokens for slots has been accessed several give. The repetitive allocations and deallocations results in fragmentation of the pool meaning that the consecutive block of free slots will be smaller and thus will the number of blocks for a given number of slots be larger. This behaviour is not desirable since more computation and memory is required and the slot lists of the DRMP messages increases. To reduce this effect the following is performed:

- The ownership of slots are kept in consecutive ranges resulting in that it is more likely that a node has free slots in a consecutive range.
- When tokens for slots are returned to the free pool, the resource management function tries to merge blocks to larger blocks.
- An implementation can also use an algorithm to allocate slots from the pool, which reduces the fragmentation.

8.3 Detailed protocol description

8.3.1 The Quark machine

The Quark machine is a model where it is assumed that only the system has only **one** resource unit, i.e., one slot. This is only an illustration to simplify the description of the system. The generalization to a complete system is straightforward since the slots can be modelled as blocks of quarks.

8.3.1.1 Simplified diagram

Below is a simplified graphical description the state machine of quarks. Please note that only the "normal" transitions are shown in this diagram. For complete description of the state machine see the tables 22 to 27.

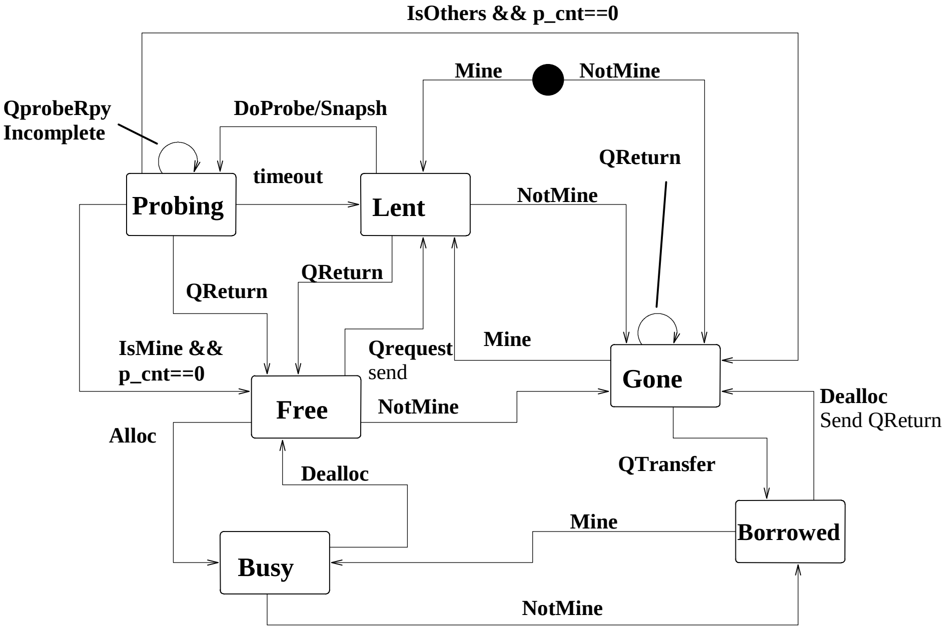


Figure 48: Most common transitions of the Quark machine

8.3.1.2 Events

- QRequest (Quark Request) - This message is a request for an access token for a Quark
- QTransfer (Quark Transfer) - This message is sent when a token is being transferred from one physical interface to another in the allocation domain. This message belongs to specific transfer session, that is, it is targeted to a specific channel in the borrowing node. There are two possibilities: The resource message reaches the destination and the two state-machines changes state or the resource message gets lost for some reason. Only the state machine where the message leaves changes places.
- QReturn (Quark Return) - Similar to QTransfer, but this is just returned to another physical interface in the allocation domain and left in the free resource pool at the other node.
- Got ownership - This represents a change of ownership such that the local physical interface that gets the message now owns the resource.
- Lost ownership - This represents a change of ownership such that the physical interface that gets the message is no longer responsible for managing the resource.
- Alloc - This is a local request for slots from the local pool of free resources. A slot allocated is moved to state Busy.
- Dealloc - A local slot is returned to the pool of free resources.
- Probe - This message is used to ask other physical interface in the allocation domain if they are using a certain slot.
- ProbeReply - This message is the reply for the Probe telling the node the state of the slot at that specific physical interface.
- Timeout - Currently, this only happens in the Probe state. (Remember that sending a QRequest does not change the state of the Quark machine at the node that initiates the message).

8.3.1.3 States

This clause describes the states in the quark machine:

- Free – Resource is free to use.
- Lent – Resource is used by someone else.
- Gone – In this state the node neither own or has access token to the slot.
- Borrowed – The node has a token for the slot but does not own the slot.
- Busy – The node has a token for the slot but does not own the slot.
- Probing – The slot is undergoing an examination of whether someone is using it or not.

8.3.1.4 Void

8.3.1.5 State transition tables

This is all the states and events possible for the quark machine. The side effect "WARN" shows something has happened that "should not" meaning that it is illegal.

8.3.1.5.1 Free

The free state represents the state that the resource is known to be available for use (either remotely or locally).

Table 22: Transition table for the Free State

Event	Condition	NextState	Action
QRequest		Lent	send QTransfer
QTransfer		Free	WARN
QReturn		Free	WARN
Got ownership		Free	
Lost ownership		Gone	
Alloc		Busy	
Dealloc		Free	WARN
Probe		Free	
PrRpy		Free	
Kill		Lent	
time-out		--	

8.3.1.5.2 Busy

This state represents that the slot is used locally.

Table 23: Transition table for the Busy State

Event	Condition	NextState	Action
QRequest		Busy	
QTransfer		Busy	WARN
QReturn		Busy	WARN
Got ownership		Busy	
Lost ownership		Borrowed	
Alloc		Busy	
Dealloc		Free	WARN
Probe		Busy	
PrRpy		Busy	
Kill		Busy	send DcpRemove
time-out		--	

8.3.1.5.3 Lent

This state represents that it is assumed that the slot is used by another node in the allocation domain. Since it is a distributed system it might not be the case that it is so. This state is also the starting state for resources owned by the local node. At start it is assumed that someone is using the slot until all nodes are queried.

Table 24: Transition table for the Lent State

Event	Condition	NextState	Action
QRequest		Lent	
QTransfer		Lent	WARN
QRet		Free	
Got ownership		Lent	
Lost ownership		Gone	
Alloc		Lent	WARN
Dealloc		Lent	WARN
Probe		Lent	
PrRpy		Lent	
Kill		Lent	
time-out		Probing	Send Probe's

8.3.1.5.4 Gone

In this state, the node does not have the impression that it owns the slot or has the access right to the slot.

Table 25: Transition table for the Gone State

Event	Condition	NextState	Action
QRequest		Gone	
QT		Borrowed	
QRet		Gone	
Got ownership		Lent	
Lost ownership		Gone	
Alloc		Gone	WARN
Dealloc		Gone	WARN
Probe		Gone	send PrRpy(Gone)
PrRpy		Gone	
Kill		Gone	
time-out		--	

8.3.1.5.5 Borrowed

In this state, the node has borrowed the slot from a remote node and uses it. The slot is owned by another node.

Table 26: Transition table for the Borrowed State

Event	Condition	NextState	Action
QRequest		Borrowed	
QTransfer		Borrowed	WARN
QRet		Borrowed	WARN
Got ownership		Busy	
Lost ownership		Borrowed	
Alloc		Borrowed	WARN
Dealloc		Gone	send QRet
Probe		Borrowed	send PrRpy(Borrowed)
PrRpy		Borrowed	
Kill		Borrowed	send DcpRemove
time-out		--	

8.3.1.5.6 Probing

In this state, there are outstanding probe messages to other nodes. Replies may have been received from some but not all nodes.

Table 27: Transition table for the Probing State

Event	Condition	NextState	Action
QRequest		Probing	
QT		Probing	WARN
QRet		Free	
Got ownership		Probing	
Lost ownership		Gone	
Alloc		Probing	WARN
Dealloc		Probing	WARN
Probe		Probing	
PrRpy	Not Last PrRpy	Probing	p[i]=state
Kill		Probing	
time-out		Lent	
PrRpy	Last && InUse(p, state)	Lent	p[i]=state
PrRpy	Last && UnUsed(p, state)	Free	p[i]=state

The functions inUse and unUsed is described below. These functions depend on the condition that the last probe reply has just arrived and that p contains all but the last reply. The condition is checked *before* actions are taken.

- inUse – This function is defined as true if *any* of its arguments are true. If more than one of its arguments is true a protocol error has occurred. This should trigger the kill function. The unUsed function implies that *all* of p is false (i.e. unused).
- unUsed – This function is true if none of its arguments are true. This tells us that a recreation of the resource should be done.

8.3.2 The Dynamic Ownership state machine

The Dynamic Ownership (DO) machine handles the ownership negotiations. The protocol negotiates ownership in such a way that there is never an overlap in ownership (i.e., no slot is owned by more than one node) before starting the probe process. The distribution of ownership ranges requires co-ordinated procedures.

8.3.2.1 Typical scenarios

The DO machine is optimized for the common cases, such as minimizing the time for the start up phase. See further in figures 49 to 51.

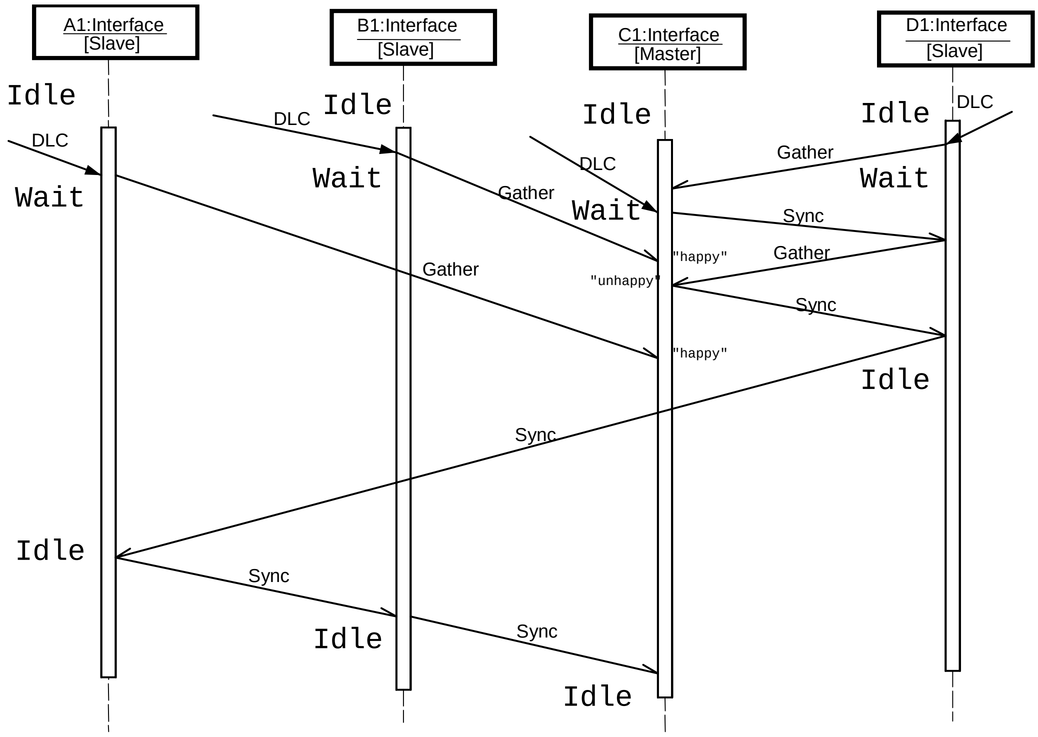


Figure 49: A node is added to the bus

In figure 49, node D1 is added to a bus. The other nodes attached to bus are informed by BCC that a new node is added. All nodes initiate a DRMP_GATHER message to obtain new ownership.

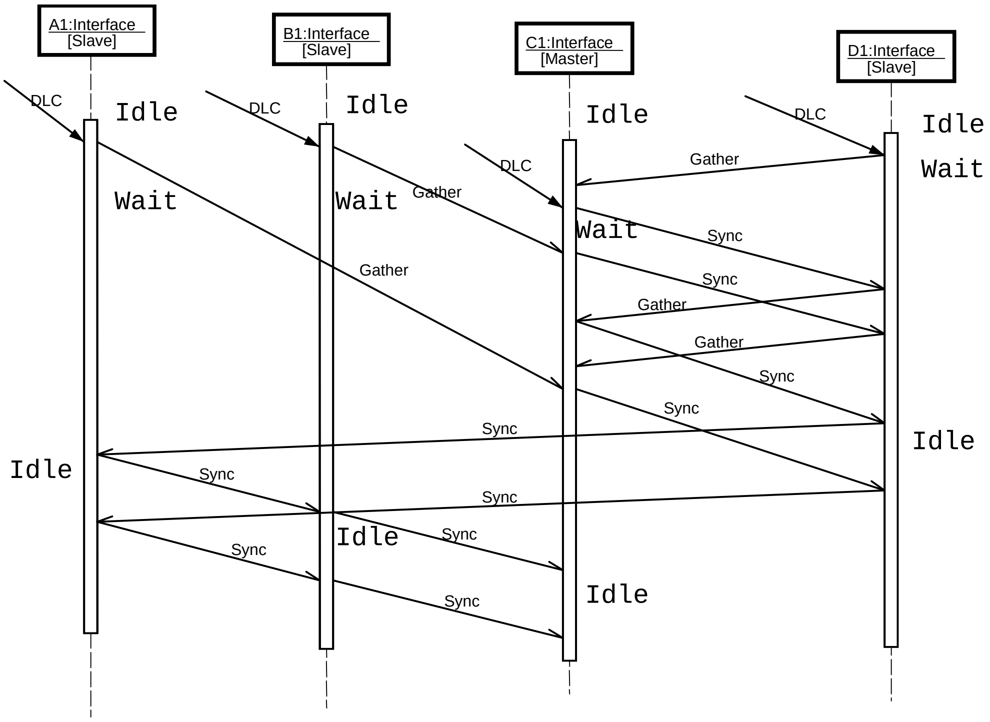


Figure 50: Ring gets closed

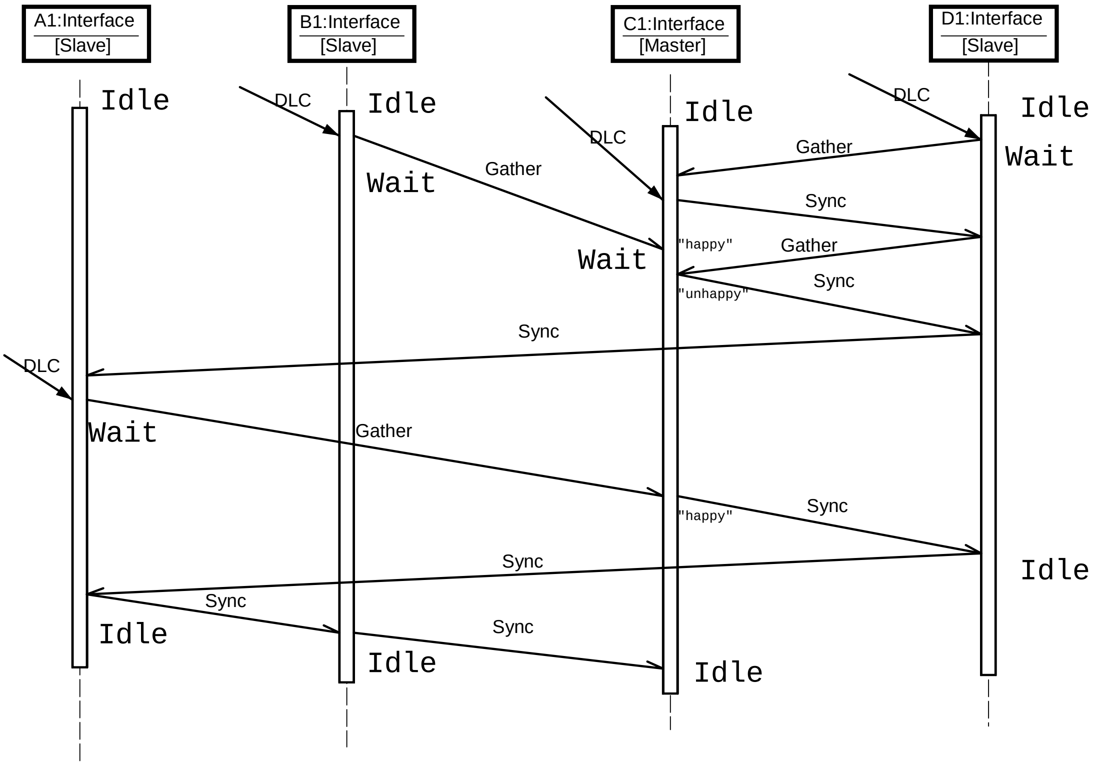


Figure 51: A ring is closed and A1 slave is late

In figure 51, node A:1 is late to detect that the ring has been closed thereby with requesting for ownership and the Master has already started to set the ownership for nodes. When node A:1 gets the DRMP_SYNC message from the master node, it sends a DRMP_GATHER message to the master requesting for ownership. The Master restarts the sync process to all nodes in the allocation domain.

8.3.2.2 Transition diagrams

This clause describes the transition diagram for the dynamic ownership distribution.

8.3.2.3 State types

The following types are used for state variables in the DO machine:

Table 28: Types used in the DO machine

Type	Description
AD	A representation of an ordered set of physical interface addresses.
Boolean	see logical values.
Integer	An unsigned number at least 32 bits large.
SyncStates	A variable, which can take two values: Idle or Wait.
Physical interface	This is just a 48 bit physical interface iaddress.
List<T>	Some kind of ordered template which can hold several of these variables.
Limit	The Limit is a structure with two attributes: An ownership limit and a starvation flag, Integer and Boolean respectively.
StarvFlag	A flag that indicates if a physical interface is set to starvation mode or not (i.e. it does not accept any new channels to/from or over it).

8.3.2.4 State variables

The following state variables are used for each local physical interface on the local node:

Table 29: The state variables on a per local physical interface basis

Name	Type	Description
pending	AD	Holds the topology info pending before synchronization is done. Initial value should be the own physical interface if a range_change triggers or the BCC topology if BCC triggers.
syncState	SyncStates	This holds the state the node is currently in, Idle or Wait. Idle means the node is ready to issue and reply to DRMP_PROBE messages. However, all DRMP links should be Idle before the global isMaster call yields true. Wait means that the node is waiting for a synchronization and thus does not probe on any of the node's physical interfaces. This is a security measure since a physical interface might suddenly become part of another link when the topology changes.
me	Physical interface	This is the Physical interface address of the local physical interface.
limits	List<Limits>	This variable holds the pending ownership ranges. The value is compared with incoming DRMP_GATHER messages and it is determined if a new sync procedure should be done. DRMP_GATHER
MY_LIM	Limit	This variable holds the ownership limit. The own limit that is set only by RangeChange message. Initially one element of zero for the local Unit, or, if this State machine instance is booted with range_change an initial element with the given limit.
ST_BIT	StarvFlag	This is a flag that is configured via management interfaces. It is sent in the DRMP_GATHER message and is used to avoid starvation.
ST_BITS	List<StarvFlag>	This variable contains flags for all physical interfaces in the allocation domain that have requested that any free resources crossing their physical link must be dropped, this is used in the DRMP_SYNC message.

8.3.2.5 Queries

8.3.2.5.1 Probe conditions

This message is a query from DRMP if the node currently is allowed to probe. The criteria is as follows:

- For each known local physical interface, the state is currently set to idle. This means it is allowed to probe right now.
- One, several or all of the local physical interface states are set to Wait. This means it currently is not allowed to probe for *any* of the physical interfaces.

Implementations might gain from caching the value of this variable and only recalculate it on any event that alters the state of any physical interface and changes to/from Wait/Idle.

8.3.2.6 Incoming messages and signals

Asynchronous events going into the Dynamic Ownership machine.

8.3.2.6.1 Bypass Chain Change (BCC)

Bypass Chain Change comes originally from DLSP. When stored, it is to be regarded with the probeUnit view (one less physical interface is bus or point-to-point).

Table 30: Reception of BCC

master®	state	Action	New State	Send
no	Idle	pending: = r, clear(oldlimits), limits[me]: =MY_LIM	Wait	DRMP_GATHER(me, MY_LIM)
no	Wait	pending: = r, clear(oldlimits), limits[me]: =MY_LIM	Wait	
yes	Idle	pending: = r, clear(oldlimits), limits[me]: =MY_LIM	Wait	DRMP_SYNC(me, r, limits)
yes	Wait	pending: = r, clear(oldlimits), limits[me]: =MY_LIM	Wait	DRMP_SYNC(me, r, limits)

Clear(oldlimits) means that the entries in the nodes limits-container, which contains identifiers of physical interfaces that has possibly been removed from the topology, shall be removed.

8.3.2.6.2 RangeChange

This is done from the event is initiated from the management system. In the case of the master getting a range change, the DRMP_GATHER message is only sent locally.

Table 31: Reception of RangeChange

Action	Send
MY_LIM: = l	DRMP_GATHER(me, MY_LIM)

If this message gets lost on the way to the master a periodic retransmission of DRMP_GATHER will ensure that the message will eventually get through. If this physical interface is the master, the DRMP_GATHER is sent locally.

8.3.2.7 Signals and messages

8.3.2.7.1 DRMP_SYNC message

The DLSP_SYNC message is sent hop-by-hop between the physical interfaces in the allocation domain (from Master towards the most downstream node, from most downstream node towards most upstream node and finally upstream back towards the Master). Different physical interfaces may have different information of the topology and who is currently the Master. Since information concerning the nodes in the allocation domain is included in the DLSP_SYNC message, a physical interface can determine if a DLSP_SYNC message should be ignored.

Table 32: Reception of DRMP_SYNC(AD r, List<Limit> l)

master (pending)	pending = r	MY_LIM = l [src]	State	Action	New State	Send
X	no	X	Idle		Idle	
X	no	X	Wait		Wait	
no	yes	yes	Idle	limits:= l	Idle	DRMP_DIST_OWN(l)DRMP_SYNC(me, r, l)
no	yes	yes	Wait	limits:= l	Idle	DRMP_DIST_OWN(l)DRMP_SYNC(me, r, l)
no	yes	no	Idle		Idle	DRMP_GATHER(MY_LIM)
no	yes	no	Wait		Wait	DRMP_GATHER(MY_LIM)
yes	yes	yes	Idle	limits:= l	Idle	
yes	yes	yes	Wait	limits:= l	Idle	DRMP_DIST_OWN(l)
yes	yes	no	Idle		Idle	DRMP_GATHER(MY_LIM)
yes	yes	no	Wait		Wait	

Note that the Limits parameter in the DRMP_SYNC message should be the set of limits given from the physical interfaces. The actual ownership distribution should be calculated at each node, which implies that the algorithm for doing this calculation must be the same in every node.

For each physical interface the start and range is calculated and the result is passed to DRMP via the RangeChange event. The calculated limits, stored in my_lim, are compared to the limits obtained the sync process. If the values do not match, the slave sends a DRMP_GATHER with the proper value. The reason for having this procedure is that several desired limit vectors maps to a certain set of ownership distributions. For example, if there are 5 physical interfaces and 10 slots total, setting the policing parameters of all nodes to the same value results in the same results (2,2,2,2,2) for the ownership limits. Thus, it is unclear for a node if a DRMP_GATHER should be sent.

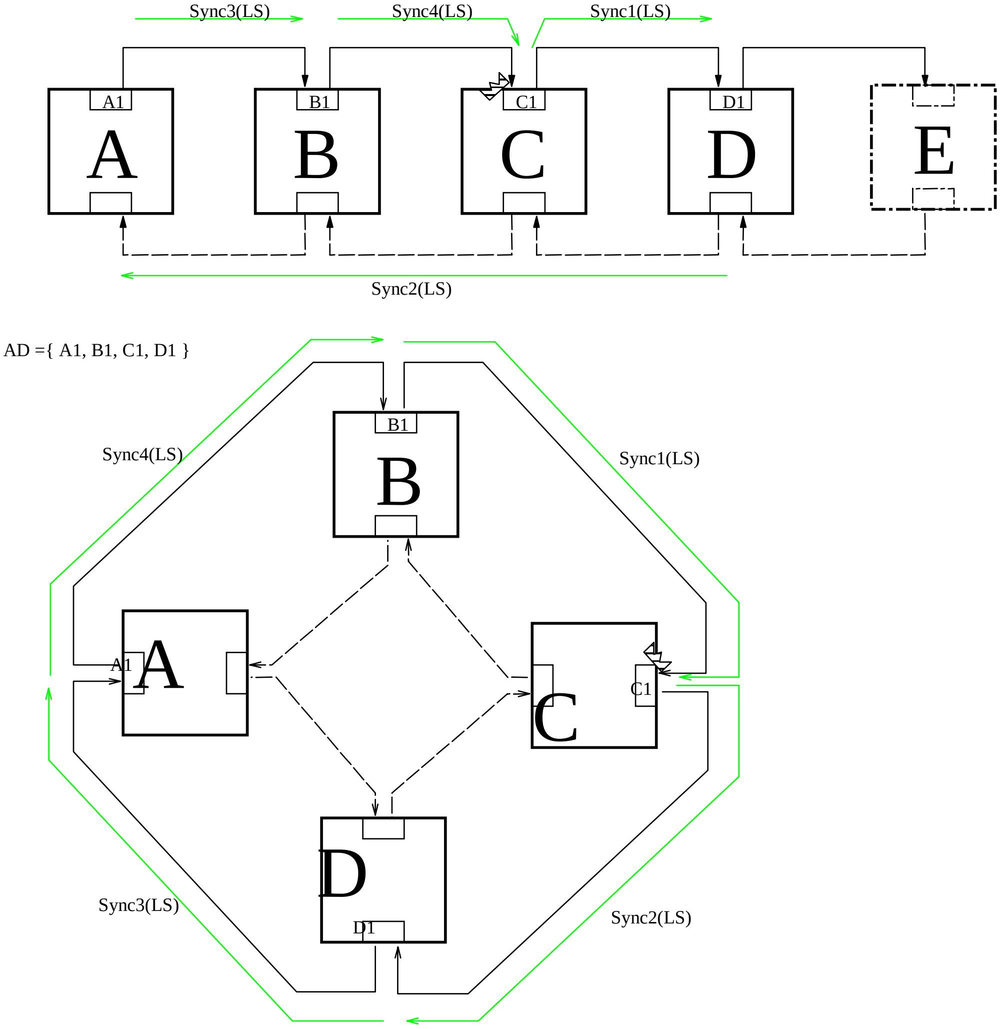


Figure 51a: Signal flow in two important DO cases

In figure 51a, the signal paths from the master physical interface (marked with a crown) C1 to the other nodes is shown. In the bus case, the DRMP_SYNC messages are sent C1-D1-A1-B1-C1, while in the ring case, it is sent C1-D1-A1-B1-C1. Please also note that even though physical interface E1 is present, it is not part of the ownership negotiation.

8.3.2.7.2 DRMP_GATHER

This is periodically sent to the master of a pending link. A 'yes' in the second column, implies that the range for this unit has already been transmitted.

Table 33: Reception of DRMP_GATHER

master (pending)	limits[src] = l	State	Action	New State	Send
no	X	Wait		Wait	
no	X	Idle		Idle	
yes	no	Idle	limits[src]:=l	Wait	DRMP_SYNC(me, pending, limits)
yes	no	Wait	limits[src]:=l	Wait	DRMP_SYNC(me, pending, limits)
yes	yes	Wait		Wait	
yes	yes	Idle		Idle	

8.3.2.8 Outgoing Messages

8.3.2.8.1 DRMP_DIST_OWN

DRMP_DIST_OWN can be issued several times, with the own-limits (l) changing for each l. The probe procedure must be turned off when sending messages. Within the scope of a physical interface's physical link, a node must drop the free resources that it has access tokens to if any StarvFlag bits are set.

8.3.2.9 Timeouts

8.3.2.9.1 (Re) send DRMP_GATHER

The DRMP_GATHER is sent periodically so that even if a message is lost it will eventually get through to the master. This time-out should be in the order of several seconds, since it is transmitted on a regular basis.

Table 34: Timeouts

master(pending)	Action
no	send DRMP_GATHER(me, MY_LIM)
yes	

8.3.2.10 Distribution and calculation of ownership ranges

The DRMP_DIST_OWN message of clause 8.3.2.8.1 defines a list of limits. The order of the limits is the same as in the DRMP_SYNC message, which in turn is defined by the signal flow on the DRMP_SYNC message, clause 8.3.2.7.1.

The rules for calculating the ranges starts with assigning the master starting from slot 1 (Slot-0 is not available for dynamic resource management) and continue with the other physical interfaces in the order of which DRMP_SYNC is forwarded according to the limits list.

There are two cases:

- In the first case, the limit list is less than or equal to the number of available slots. In that case, the mapping from limits to actual ranges is straightforward.
- In the second case, the sum of the limit list exceeds the number of available slots. In this case, a formula is used for a proportional distribution, in which the remainder (if any) is allocated to the master. The remainder can never be greater than the number of physical interfaces in the allocation domain, and so the error is relatively small for most cases of operation.

All variables described in the calculations below must be performed with integers with at least 32-bit precision and the intermediate results inside any calculation must be at least width 64 bits unsigned integer precision.

The variables used in the calculations are:

N Number of physical interfaces in the allocation domain.

h_n A helper function, $h_n \leq C$ in all cases.

r_n An actual ownership range of a certain physical interface $r_n \leq C$ in all cases

p_n The policy parameter assigned to a certain physical interface (32 bit unsigned).

C This variable indicates number of slots available on the bypass chain (24 bits unsigned). Note that C does not include slot zero, which is not resource handled. Therefore, C is defined to be one less than the total number of physical slots on a specific bypass chain.

Then the actual ownership ranges, as a function of the policing ranges are:

If $C \geq \sum_{k=1}^{N} (p_k)$ then simply, $r_n = p_n$ for all n

If $C < \sum_{k=1}^{N} (p_k)$ then we have two cases:

Helping function:

$$h_n = \left\| \frac{C\, p_n}{\sum_{k=1}^{N} (p_k)} \right\| \text{(Integer maths)}$$

- For $n = 1, r_1 = C - \sum_{k=2}^{N} (h_k)$ Master gets the remainder of the slots, the remainder is always $< N$
- For $n > 1, r_n = h_n$

8.4 Message formats

This clause describes the formats of the DRMP messages. All fields are to be 64-bit little endian.

In some messages, the division sign "/" is used to be indicate integer division, i.e. any fraction is immediately truncated by the operation itself. Modulus calculation is indicated with "%" which results in the remainder of an integer division.

8.4.1 Generic fields

This is repetitive parts of messages that are frequent in the actual messages below. They are not themselves to be regarded as complete messages.

8.4.1.1 Physical interface identifiers field

The physical interface address is always located "to the right" in a DTM frame, with the layout shown in figure 52:

bit 63-56	**bit 55-48**	**bit 47-40**	**bit 39-32**	**bit 31-24**	**bit 23-16**	**bit 15-8**	**bit 7-0**
0	0	I-face Id byte 5	I-face Id byte 4	I-face Id byte 3	I-face Id byte 2	I-face Id byte 1	I-face Id byte 0

Figure 52: Physical interface address

Table 35: Physical physical interface address fields

Fields	Slot #	Bits	Size	Description
Physical interface address	any	[47:0]	48	This is the field for the 48 bit physical physical interface address

8.4.1.2 Slot fragment

This field is generic for the description of DRMP. It is used to represent tokens in the messages.

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
AMOUNT n	start n			AMOUNT n+1	start n+1		

Figure 53: Slot fragment

Table 36: Slot fragment fields

Fields	Slot #	Bits	Size	Description
AMOUNT n	any	[63:56]	8	Amount of these slots
start n	any	[55:32]	24	Start of this slot fragment
AMOUNT n+1	any	[31:24]	8	Amount of next slots
start n+1	any	[23:0]	24	Start of next slot fragment

8.4.1.3 Long slot fragment

This field is used to represent a block of slots in a DRMP message. If the amount field is set to zero, the field has a different meaning, which is described below.

bit 63-56	bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
0	start			0	amount		

Figure 54: Long slot fragment

Table 37: Long slot fragment fields

Fields	Slot #	Bits	Size	Description
start	any	[55:32]	24	Start of this slot fragment
amount	any	[23:0]	24	Amount of slots

8.4.1.4 Generic header

This clause describes the generic DRMP header. Note that the destination physical interface address represents the physical interface of which the operation concerns, which is not necessarily the same as the physical interface that received the message.

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>VER</td><td>CMD</td><td>0</td><td>Dst. I-face Id byte 5</td><td>Dst. I-face Id byte 4</td><td>Dst. I-face Id byte 3</td><td>Dst. I-face Id byte 2</td><td>Dst. I-face Id byte 1</td><td>Dst. I-face Id byte 0</td></tr>
</table>

Figure 55: Generic header

Table 38: Generic header fields

Fields	Slot #	Bits	Size	Description
VER	0	[63:61]	3	The version of the protocol (currently 0)
CMD	0	[60:56]	5	The command of the protocol
Dst. I-face Id	0	[47:0]	48	Destination physical physical interface address that we are talking about

8.4.2 Statistical information

In DRMP, some messages that do not change the state of the Quark machine. These messages are used as information.

If dynamic ownership is used, the message with CMD = 2 is used. If not, messages with CMD = 3 is used. All implementations should handle both message types and if an announce with code 2 is received when employing static ownership, the fields Own Start and Own Range should be interpreted as being zero although they are not present in the actual message.

8.4.2.1 DRMP_RESOURCE_ANNOUNCE (Dynamic ownership)

This message is used if the physical interface ownership distribution is dynamic.

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>2</td><td>N</td><td colspan="6">Broadcast physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="6">Source physical interface address</td></tr>
<tr><td colspan="2">0</td><td>SLOT HW 1</td><td colspan="6">Upstream physical interface address 1</td></tr>
<tr><td colspan="3">SLOT LW 1</td><td colspan="6">Downstream physical interface address 1</td></tr>
<tr><td colspan="2">0</td><td>slot HW n</td><td colspan="6">Upstream physical interface address n</td></tr>
<tr><td colspan="3">SLOT LW n</td><td colspan="6">Downstream physical interface address n</td></tr>
<tr><td colspan="2">0</td><td>slot HW N</td><td colspan="6">Upstream physical interface address N</td></tr>
<tr><td colspan="3">SLOT LW N</td><td colspan="6">Downstream physical interface address N</td></tr>
</table>

Figure 56: DRMP_RESOURCE_ANNOUNCE

Table 39: DRMP_RESOURCE_ANNOUNCE fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=2
N	0	[55:48]	8	This is the number of fragments in this announce message
Source physical interface address	1	[47:0]	48	This is the physical interface address of the announcing physical interface
SLOT n	2n, 2n+1	2n[55:48]|| (2n+1)[63:49]	24	The number of slots announced for this scope. In the figure, HW means higher part of word and LW lower part of word.
Upstream physical interface address n	2n	2n[47:0]	48	The address to the physical interface which finishes off the scope of this announce fragment
Downstream physical interface address n	2n+1	(2n+1)[47:0]	48	The address to the physical interface which starts of the scope of this announce fragment

8.4.2.2 DRMP_RESOURCE_ANNOUNCE (Static ownership)

This message is used if the physical interface ownership distribution is static.

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>3</td><td>N</td><td colspan="6">Broadcast physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="6">Source physical interface address</td></tr>
<tr><td colspan="2">0</td><td colspan="3">Own start</td><td>0</td><td colspan="3">Own range</td></tr>
<tr><td colspan="2">0</td><td>SLOT HW 1</td><td colspan="6">Upstream physical interface address 1</td></tr>
<tr><td colspan="3">SLOT LW 1</td><td colspan="6">Downstream physical interface address 1</td></tr>
<tr><td colspan="2">0</td><td>SLOT HW n</td><td colspan="6">Upstream physical interface address n</td></tr>
<tr><td colspan="3">SLOT LW n</td><td colspan="6">Downstream physical interface address n</td></tr>
<tr><td colspan="2">0</td><td>SLOT HW N</td><td colspan="6">Upstream physical interface address N</td></tr>
<tr><td colspan="3">SLOT LW N</td><td colspan="6">Downstream physical interface address N</td></tr>
</table>

Figure 57: DRMP_RESOURCE_ANNOUNCE

Table 40: DRMP_RESOURCE_ANNOUNCE fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=3
N	0	[55:48]	8	This is the number of fragments in this announce message
Source Physical interface address	1	[47:0]	48	This is the physical interface address of the announcing physical interface
Own Start	2	[47:32]	24	The start of the ownership of the announcer
Own Range	2	[23:0]	24	The range of the ownership of the announcer
SLOT n	2n+1, 2n+2	(2n+1)[55:48]| (2n+2)[63:49]	24	The amount of slots announced for this scope. In the figure, HW means higher part of word and LW lower part of word
Upstream physical interface address n	2n+1	(2n+1)[47:0]	48	The address to the physical interface which finishes off the scope of this announce fragment
Downstream physical interface address n	2n+2	(2n+2)[47:0]	48	The address to the physical interface which starts of the scope of this announce fragment

8.4.3 Access token passing

These messages are used to request or change access rights for tokens.

8.4.3.1 DRMP_TOKEN_REQUEST

This message is used to request a set of resource from one physical interface to another.

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th colspan="2">bit 47-40</th><th colspan="2">bit 39-32</th><th colspan="2">bit 31-24</th><th colspan="2">bit 23-16</th><th colspan="2">bit 15-8</th><th colspan="2">bit 7-0</th></tr>
<tr><td>0</td><td>4</td><td colspan="2">N</td><td colspan="12">Destination physical interface address</td></tr>
<tr><td colspan="2">0</td><td colspan="2">SLOT HW</td><td colspan="12">Source physical interface address</td></tr>
<tr><td colspan="4">SLOT LW</td><td colspan="12">Downstream physical interface address</td></tr>
<tr><td colspan="8">0</td><td>B</td><td colspan="7">Session identifier</td></tr>
</table>

Figure 58: DRMP_TOKEN_REQUEST

Table 41: DRMP_TOKEN_REQUEST fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=4
Source physical interface address	1	[47:0]	48	The address of the physical interface which requests to lend resources
SLOT	1,2	1[55:48]|2[63:49]	24	The amount of slots requested
Downstream physical interface address	2	[47:0]	48	The scope for the request, note that the start scope is implicit from the source physical interface address, since no node will need to borrow resources which does not start at its own scope
B	3	[31:31]	1	If set to one, we indicate that this node is capable of accepting Resource transfers that are multi part
Session identifier	3	[30:0]	31	An identifier which is to be sent back in the reply to the borrowing request, it is used by the receiver to distinguish between possibly many outstanding borrowing sessions, it needs only be unique for each allocation domain

8.4.3.2 DRMP_TOKEN_TRANSFER

This is used to transfer the access right from one physical interface to another. The source of the sent resource is not sent since it is not needed until we return the resources, but by then the owner may well have changed for all or part of the slots. A session identifier is used to map together the request with the reply in case more than one channel is being requested for borrowing at the same time. The session id is also what holds the information on what scope the resource transfer is valid for. In the figure 51 we pad the last slot fragment to the right with 32 bits of zeros.

bit 63-56		bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
0	5	N	Destination physical interface address					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT n		Start slot n			AMOUNT n+1	Start slot n+1		
AMOUNT N-1		Start slot N-1			AMOUNT N	Start slot N		
0					B	Session identifier		

Figure 59: DRMP_TOKEN_TRANSFER

Table 42: DRMP_TOKEN_TRANSFER fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, Cmd=5
Number of fragments	0	[55:48]	8	The number of fragments in this resource transfer
AMOUNT n	n/2+1	(n/2+1) [32*(n%2)+31: 32*(n%2)+24]	8	The slot amount for this fragment
Start slot n	n/2+1	(n/2+1) [32*(n%2)+23:32*(n%2)]	24	The start slot for this fragment
B	N/2+1	[31:31]	1	More to come Bit, this indicates that there will be more slots in a coming message
Session identifier	N/2+1	[30:0]	31	The session identifier provided by the borrower

8.4.3.3 DRMP_TOKEN_RETURN

This message is used to return tokens. The source address is not included in the message since the owner does not have any knowledge of which node borrowed the slot since the ownership can have changed between the borrowing and the return of the slot. To support spatial reuse of slots, the scope fields part of this message. If in figure 52, the number of slot fragments is odd the message is padded with the last (rightmost) 32 bits with zeros.

bit 63-56		bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
0	7	N	Destination physical interface address					
0			Upstream physical interface address					
0			Downstream physical interface address					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT n		Start slot n			AMOUNT n+1	Start slot n+1		
AMOUNT N-1		Start slot N-1			AMOUNT N	Start slot N		

Figure 60: DRMP_TOKEN_RETURN

Table 43: DRMP_TOKEN_RETURN fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=6
Number of fragments	0	[55:48]	8	Number of fragments in Resource Transfer Return
Upstream physical interface address	1	[47:0]	48	The start scope for the return of this resource (applies to all slot fragments)
Downstream physical interface address	2	[47:0]	48	The end scope for the return of this resource (applies to all slot fragments)
AMOUNT	(n-1)/2+3	((n-1)/2+3) [32*(n%2)+31: 32*(n%2)+24]	8	Slot amount of current fragment
Start slot	(n-1)/2+3	((n-1)/2+3) [32*(n%2)+23: 32*(n%2)]	24	Slot start of current fragment

8.4.3.4 DRMP_PROBE

This message is used to query one or more of the owned slots to check the status of the slot.

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>7</td><td>0</td><td colspan="6">Destination physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="6">Source physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="6">Upstream physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="6">Downstream physical interface address</td></tr>
<tr><td colspan="3">0</td><td colspan="2">Probe session id</td><td>AMOUNT</td><td colspan="3">Start slot</td></tr>
</table>

Figure 61: DRMP_PROBE

Table 44: DRMP_PROBE fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=7
Source physical interface address	1	[47:0]	48	Address of the physical interface probing for this set of resources
Upstream physical interface address	2	[47:0]	48	The start of the scope for which the probe is valid
Downstream physical interface address	3	[47:0]	48	The end of the scope for which the probe is valid
Probe session id	4	[47:32]	16	An identifier mostly used for robustness against protocol faults such as multiple outstanding probes for same resource, needs to be unique per allocation domain
AMOUNT	4	[31:24]	8	Slot amount of current probed fragment
Start slot	4	[23:0]	24	Start of current probed fragment

8.4.3.5 DRMP_PROBE_REPLY

This message is used to reply to a query for the status of a slot. Only the downstream scope of the resources is used in the fields below, the upstream scope is implicit from the source of this physical interface. Also, note that it is only the resources that are in use (borrowed) that are listed in this reply. During ownership transitions, it is possible to get probes for slots that a physical interface owns itself. The node shall not send a reply to probes for slots that it currently does not own.

If the number of slot fragments, M, is odd for one or any of the slot fragments, 32 bits of zeros must be padded in the right hand 32 bits of that slot.

bit 63-56		bit 55-48	bit 47-40	bit 39-32	bit 31-24	bit 23-16	bit 15-8	bit 7-0
0	8	N	Destination physical interface address					
probe session identifier			Source physical interface address					
0		FRAGS. 1	Downstream physical interface address 1					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT M-1		Start slot M-1			AMOUNT M	Start slot M		
0		FRAGS. 2	Downstream physical interface address 2					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT M-1		Start slot M-1			AMOUNT M	Start slot M		
0		FRAGS. n	Downstream physical interface address n					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT M-1		Start slot M-1			AMOUNT M	Start slot M		
0		FRAGS. N	Downstream physical interface address N					
AMOUNT 1		Start slot 1			AMOUNT 2	Start slot 2		
AMOUNT M-1		Start slot M-1			AMOUNT M	Start slot M		

Figure 62: DRMP_PROBE_REPLY

Table 45: DRMP_PROBE_REPLY fields

Fields	Slot #	Bits	S ize	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=8
N	0	[55:48]	8	Number of different scopes in probe reply
Probe session identifier	1	[63:48]	16	The id of this probe session, the receiver of this message should check so that this is not old
Source physical interface address	1	[47:0]	48	The source of the physical interface answering to this probe(remember that probe is broadcast so we need to distinguish the answers, which are unicast
FRAGS. n	f(n,1)	f(n,1)[55:48]	8	Number of fragments for a certain scope in a probe reply
Downstream physical interface address n	f(n,1)	f(n,1)[47:0]	48	Down stream address of this probe (upstream is implicit, since no node will use borrowed slots upstream of its own location)
AMOUNT n, m		f(n,m)[(m%2)*32+31:(m%2)*32+24]	8	The amount of slots in the fragment
Start slot n,m		f(n,m)[(m%2)*32+23:(m%2)*32]	24	The start of this slot fragment

The function f(n) in table 39 is defined by the formula below. It gives us the slot offset as a function of n and m. This is the function f(n,m) that defines the slot index for various n´s and m´s. Note the variable M_n, which is defined by the number of slot fragments for each scope. A sum formula is used since all the M_n´s may have different values. Also, if $n \equiv 1$, the sum evaluates to zero.

$$f(n.m) = 2 + \tfrac{m+1}{2} + \sum_{1}^{n-1}\left(\tfrac{M_{n+1}}{2} + 1\right)$$

8.4.4 Ownership passing

These messages are used for negotiating ownership. The starvation bit is a "vertical" ownership information. When receiving the starvation bit is set in a DRMP_SYNC message, the physical interface receiving the valid DRMP_SYNC message must stop using the slot and discard the access token for all slots on that physical link. The physical interface must not probe nor respond to probes when it is put in starvation mode via the management system.

8.4.4.1 DRMP_SYNC

This message is originated from the master of an allocation domain. The master waits a certain time for the message to return. If it does not return, the master re-transmits it.

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>9</td><td colspan="2">N</td><td colspan="6">Destination physical interface address</td></tr>
<tr><td colspan="3">0</td><td>S 1</td><td colspan="6">Physical interface address 1</td></tr>
<tr><td colspan="3">0</td><td>S n</td><td colspan="6">Physical interface address n</td></tr>
<tr><td colspan="3">0</td><td>S N</td><td colspan="6">Physical interface address N</td></tr>
<tr><td colspan="2">0</td><td colspan="4">Request own 1</td><td>0</td><td colspan="3">Request own 2</td></tr>
<tr><td colspan="2">0</td><td colspan="4">Request own n-1</td><td>0</td><td colspan="3">Request own n</td></tr>
<tr><td colspan="2">0</td><td colspan="4">Request own N-1</td><td>0</td><td colspan="3">Request own N</td></tr>
</table>

Figure 63: DRMP_SYNC

If the number of physical interfaces in the allocation domain is odd, the message is padded with zeros in the lower leftmost 24 bits.

Table 46: DRMP_SYNC fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=9
Number of physical interfaces in this allocation domain	0	[55:48]	8	The number of physical interfaces in this allocation domain, known at the time of transmission of this message.
S	n	48	1	This indicates that the specific physical interface does not accept channels in, out or through it
Physical interface address n	n	n[47:0]	48	An physical interface address in the list
request own n	(n-1)/2+3	(n-1)/2+3 [32*(n%2)+23:32*(n%2)]	24	The requested ownership limit that this physical interface wanted last time the master got that information

8.4.4.2 DRMP_GATHER

The DRMP_GATHER is sent by any physical interface (including the master that then talks to itself) whenever a need to change the policing parameter arises.

<table>
<tr><th colspan="2">bit 63-56</th><th colspan="2">bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>10</td><td colspan="2">0</td><td colspan="6">Destination physical interface address</td></tr>
<tr><td colspan="3">0</td><td>S</td><td colspan="6">Physical interface address</td></tr>
<tr><td colspan="7">0</td><td colspan="3">Request own</td></tr>
</table>

Figure 64: DRMP_GATHER

Table 47: DRMP_GATHER fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]|[47:0]	56	Header, Cmd=10
S	1	48	1	Starvation bit, this informs the master that we do not want any more new channels to/from or over this physical interface
Physical interface address	1	[47:0]	48	The address of the physical interface doing the request for a certain ownership range
Request own	2	[23:0]	24	The ownership range requested by this physical interface

8.4.4.3 DRMP_KILL

The owner of a resource can send this message after doing a probe when the owner detects that more than one node has an access token for the slot.

<table>
<tr><th colspan="2">bit 63-56</th><th>bit 55-48</th><th>bit 47-40</th><th>bit 39-32</th><th>bit 31-24</th><th>bit 23-16</th><th>bit 15-8</th><th>bit 7-0</th></tr>
<tr><td>0</td><td>11</td><td>0</td><td colspan="6">Destination physical interface address</td></tr>
<tr><td colspan="6">0</td><td colspan="3">Kill slot</td></tr>
</table>

Figure 65: DRMP_KILL

Table 48:DRMP_KILL message fields

Fields	Slot #	Bits	Size	Description
DRMP generic Header	0	[63:56]||[47:0]	56	Header, CMD = 11
Kill slot	1	[23:0]	24	The slot requested to be killed due to double booking

History

Document history		
V1.1.1	August 2002	Membership Approval Procedure MV 20021018: 2002-08-20 to 2002-10-18