Network Working Group
Internet Draft
Expiration Date: January 2001

Jonathan P. Lang (Calient Networks)
Krishna Mitra (Calient Networks)
John Drake (Calient Networks)
Kireeti Kompella (Juniper Networks)
Yakov Rekhter (Cisco Systems)
Lou Berger (LabN Consulting, LLC)
Debanjan Saha (Tellium)
Debashis Basak (Marconi)
Hal Sandick (Nortel Networks)

Link Management Protocol (LMP)

draft-ietf-mpls-lmp-00.txt

1. Status of this Memo

2. Abstract

Future networks will consist of photonic switches, optical crossconnects, and routers that may be configured with bundled links consisting of a number of user component links and an associated control channel. This draft specifies a link management protocol (LMP) that runs between neighboring nodes and will be used for both link provisioning and fault isolation. A unique feature of LMP is that it is able to isolate faults in both opaque and transparent networks, independent of the encoding scheme used for the component links. LMP will be used to maintain control channel connectivity,

verify component link connectivity, and isolate link, fiber, or channel failures within the network.

3. Introduction

Future networks will consist of photonic switches (PXCs), optical crossconnects (OXCs), routers, switches, DWDM systems, and add-drop multiplexors (ADMs) that use the MPLS control plane to dynamically provision resources and to provide network survivability using protection and restoration techniques. A pair of nodes (e.g., two PXCs) may be connected by thousands of fibers, and each fiber may be used to transmit multiple wavelengths if DWDM is used. Furthermore, multiple fibers and/or multiple wavelengths may be combined into a single bundled link, where we define such a link as a logical relationship associating a bi-directional control channel with zero or more unidirectional component links (see also [KRB00]).

For the purposes of this document, the granularity of a component link is a wavelength, a waveband, or a fiber depending on the ports that are exposed to the adjacent nodes. For example, if a cross-connect is connected to a DWDM device, then the ports of the cross-connect (and hence the component links) correspond to wavelengths. If, however, the cross-connect multiplexes the wavelengths internally before connecting them to another node, then the ports of the cross-connect (and hence the component links) correspond to a fiber. In general, a bundled link between two nodes will be identified by a local and remote 32-bit interface (possibly a virtual interface) address, and the control channel will be identified by a local and remote 32-bit control channel Id (CCId). Similarly, each node will identify each component link with a local LinkId. LMP gives the association between the endpoints of the bundled link, control channel, and component links.

Within the framework of the Generalized Label [ABD00], the LinkId is required whenever there is ambiguity as to where the labels are to be allocated. In the following, we describe how the LinkId is used for the various multiplexing capabilities of a node. For the PSC case, the Generalized Label includes a LinkId (corresponding to a fiber) and the normal MPLS label. For the TDM case, the Generalized Label includes a LinkId (corresponding to a fiber) and the Mannie encoded label. For the LSC case there are two options for the Generalized Label depending on if the wavelengths are exposed to the adjacent nodes or not; it could include the LinkId (corresponding to

a fiber) and a wavelength label (corresponding to a wavelength within the fiber), or it could include just the LinkId (corresponding to a wavelength), where a label is present but ignored. Finally, for the FSC case, the Generalized Label includes only a linkId (corresponding to a fiber) and a label is present but ignored.

A unique feature of a link as defined above is that the control channel and the associated component links are not required to be transmitted along the same physical medium. For example, the control channel could be transmitted along a separate wavelength or fiber, or on an Ethernet link between the two neighbors. A consequence of allowing the control channel of a link to be physically diverse from the associated component links is that the

---

health of a control channel on a link does not necessarily correlate to the health of the component links, and vice-versa. Therefore, new mechanisms must be developed to manage links, both in terms of link provisioning and fault isolation.

This draft specifies a link management protocol (LMP) that runs between neighboring nodes and will be used for both link provisioning and fault isolation. All of the LMP messages transmitted over the control channel are IP encoded, so that the link level encoding becomes an implementation agreement and is not part of LMP specifications. The LMP messages that are transmitted over a component link will be a new protocol type. For example, if it is over Ethernet, it will be a new Ethertype, and if it is over SONET/SDH, the HDLC framing defined for PPP over SONET will be used with the MPLSCP defined in Section 4 of [RNT99].

In this draft, we will follow the naming convention of [ARD99] and use OXC to refer to all categories of optical crossconnects, irrespective of the internal switching fabric. We distinguish between crossconnects that require opto-electronic conversion, called digital crossconnects (DXCs), and those that are all-optical, called photonic switches or photonic crossconnects (PXCs) - referred to as pure crossconnects in [ARD99], because the transparent nature of PXCs introduces new restrictions for monitoring and managing the data channels (see [CBD00] for proposed extensions to MPLS for performance monitoring in photonic networks). LMP, however, can be used for any type of node, enhancing the functionality of traditional DXCs, DWDMs, and routers, while enabling PXCs to intelligently interoperate in heterogeneous optical networks.

Due to the transparent nature of PXCs, traditional methods can no longer be used to monitor and manage links and LMP has been designed to address these issues in optical networks. In addition, since LMP does not dictate the actual transport mechanism, it can be implemented in a heterogeneous network. A requirement for LMP is that each link has an associated bi-directional control channel and that a free (unallocated) component link must be opaque (i.e., able to be terminated); however, once a component link is allocated, it may become transparent. Note that there is no requirement that all of the component links must be terminated simultaneously, but at a minimum, they must be able to be terminated one at a time. There is no requirement that the control channel and component links share the same medium; however, the control channel must terminate on the same two nodes that the component links span.

LMP is a protocol that runs between adjacent nodes and is designed to provide four basic functions for the node pair: control channel management, link connectivity verification, link property correlation, and fault isolation. Control channel management is used to establish and maintain link connectivity between neighboring nodes. This is done using lightweight Hello messages that act as a fast keep-alive mechanism between the nodes. Link connectivity verification is used to verify the physical connectivity of the component links as well as exchange the LinkIds that are used in the

---

Generalized Label [ABD00] for MPLS. Link property correlation consists of LinkSummary messages that exchange the local/remote CCId mappings that were learned when establishing control channel connectivity; the local/remote LinkId mappings that were discovered as a result of the link connectivity verification process; the protection control channels for maintaining link connectivity; and the protection component links that are used for span protection. The fault isolation mechanism can localize failures in both opaque and transparent networks, independent of the encoding scheme used for the component links, and as a result, both local span and end-to-end path protection/restoration procedures can be initiated.

The organization of the remainder of this document is as follows. In Section 4, we discuss the role of the control channel and the messages used to establish and maintain link connectivity. The link verification procedure is discussed in Section 5. In Section 6, we show how LMP will be used to isolate link and channel failures within the optical network.

4. Control channel management

To establish a bundled link between two nodes, a bi-directional primary control channel must first be configured. The control channel can be used to exchange MPLS control-plane information such as link provisioning and fault isolation information (implemented using a messaging protocol such as LMP, proposed in this draft), path management and label distribution information (implemented using a signaling protocol such as RSVP-TE [ABG99]or CR-LDP [Jam99]), and topology and state distribution information (implemented using traffic engineering extended protocols such as OSPF [KaY99]and IS-IS [SmL99]). Each bundled link is identified by a 32-bit IP interface (possibly virtual interface) and each bundled link MUST have an associated control channel; however, we do not specify the exact implementation of the control channel. Rather, we assign a 32-bit integer control channel identifier (CCId) to each direction of the control channel and we define the control channel messages to be IP encoded. This allows the control channel implementation to encompass both in-band and out-of-band mechanisms, including the case where the control channel is transmitted separately from the associated component link(s), either on a separate wavelength or on a separate fiber. Furthermore, since the messages are IP encoded, the link level encoding is not part of LMP.

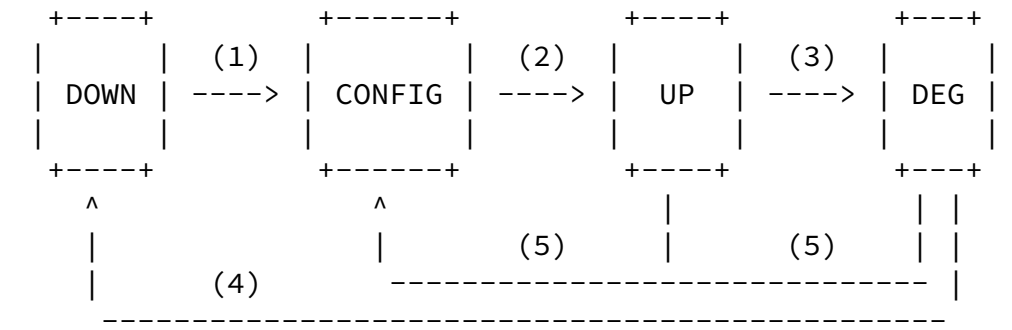
The control channel of a link can be either explicitly configured or automatically selected, however, for the purpose of this document we will assume the control channel is explicitly configured. Note that for in-band signaling, a control channel could be allocated to a component link; however, this is not true when the control channel is transmitted separately from the component links. In addition to a primary control channel, an ordered list of backup control channels can also be specified. Depending on the control channel implementation, the list of backup control channels may include component links, provided control channels have preemptive priority over the user data traffic.

For LMP, it is essential that a control channel is always available for a link, and in the event of a control channel failure, an alternate (or backup) control channel must be made available to reestablish communication with the neighboring node. Since control channels are electrically terminated at each node, the failure of a control channel can be detected by lower layers (e.g., SONET/SDH). If the primary control channel cannot be established, then a backup control channel SHOULD be tried. Of course, alternate control channels SHOULD be pre-configured, however, coordinating the

switchover of the control channel to an alternate channel is still an important issue. Specifically, if the control channel fails but the node is still operational (i.e., the component links are still passing user data), then both the local and remote nodes should switch to an alternate control channel. If the bi-directional control channel is implemented using two separate unidirectional channels, and only one direction of the control channel has failed, both the local and remote nodes need to understand that the channel has failed so that they can coordinate a switchover.

4.1. LMP Link States

A link can be in any of four well-defined states: DOWN, Configure (CONFIG), UP, and Degraded (DEG). These states apply to the link as a whole, including both control channel and component links. Many of these states have multiple sub-states that are described later in this document.



4.1.1 Link States

DOWN: Link Down. Communication not established, Hello configuration not initiated, and component links not in use.

CONFIG: Hello configuration parameters are negotiated and the control channel is brought up, but link properties are not yet agreed upon.

UP: Link Up. Control channel is UP and Hello messages are being exchanged.

DEG: Control channel and backup control channel(s) are not available, but component links are still in use.

4.1.2 State transition events

(1) The parameter negotiation phase is initiated.

(2) The control channel is UP and Hello messages are being exchanged.

(3) The control channel and backup control channel(s) are not available, and the component links are still in use.

(4) The control channel and backup control channel(s) are not available, and the component links are failed or not in use. This includes the case where a control channel is brought down administratively.

(5) The parameter negotiation phase is initiated.

4.2. Hello protocol

Once a control channel is configured between two neighboring nodes, a Hello protocol will be used to establish and maintain connectivity between the nodes and to detect link and channel failures. The Hello protocol of LMP is intended to be a lightweight keep-alive mechanism that will react to control channel failures rapidly so that IGP Hellos are not lost and the associated link-state adjacencies are not removed unnecessarily. Furthermore, the RSVP Hello of [ABG99] is not needed since the LMP Hellos will detect link layer failures.

The Hello protocol consists of two phases: a negotiation phase and a keep-alive phase. Negotiation MUST only be done when the link is in the CONFIG state, and is used to exchange the CCIds and agree upon the parameters used in the keep-alive phase. The keep-alive phase consists of a fast lightweight Hello message exchange.

4.2.1. Parameter Negotiation

Before initiating the Hello protocol of the keep-alive phase, the local and remote CCId must be exchanged and the HelloInterval and HelloDeadInterval parameters must be agreed upon. The HelloInterval indicates how frequently LMP Hello messages will be sent, and is measured in milliseconds (ms). For example, if the value were 5, then the transmitting node would send the Hello message at least every 5ms. The HelloDeadInterval indicates how long a device should wait to receive a Hello message before declaring a control channel dead, and is measured in milliseconds (ms). The HelloDeadInterval MUST be greater than the HelloInterval, and SHOULD be at least 3 times the value of HelloInterval.

The parameter negotiation consists of three messages: a HelloConfig message, a HelloConfigAck message, and a HelloConfigNack message.

The HelloConfigAck and HelloConfigNack messages are used to acknowledge receipt of the HelloConfig message. The HelloConfigNack message is also used to suggest alternate values for the

HelloInterval and HelloDeadInterval parameters. To initiate the negotiation process, a node sends a HelloConfig message containing the CCId for the control channel, the IP address of the bundled link, and the proposed HelloInterval and HelloDeadInterval. The node also starts a single-shot timer that is used for retransmissions in the event of message loss.

When a HelloConfig message is received at a node, a HelloConfigAck message SHOULD be transmitted if the received HelloInterval and HelloDeadInterval values are acceptable. Otherwise, the node MUST reject the parameters by sending a HelloConfigNack message. The HelloConfigNack message MUST include acceptable values for the HelloInterval and HelloDeadInterval.

When a node has either sent or received a HelloConfigAck message, it may begin sending Hello messages. Once it has both sent and received a Hello message, the link is UP. If, however, a node receives a HelloConfigNack message instead of a HelloConfigAck message, the node MUST not begin sending Hello messages. However, if the HelloInterval and HelloDeadInterval included in the received HelloConfigNack message are locally acceptable, the node SHOULD send a new HelloConfig message with these values to the adjacent node.

4.2.2. Fast keep-alive

Once the parameters have been agreed upon and a node has sent and received a HelloConfigAck message, it may begin sending Hello messages. Each Hello message will contain two sequence numbers: the first sequence number (TxSeqNum) is the sequence number for this Hello message and the second sequence number (RcvSeqNum) is the sequence number of the last Hello message received from the adjacent node. Each node increments its sequence number when it sees its current sequence number reflected in Hellos received from its peer. The sequence numbers will be 32-bit lollipop sequence numbers that start at 1 and wrap around back to 2; 0 is used in the RcvSeqNum to indicate that a Hello has not yet been seen and 1 is used to indicate a node boot/reboot.

Under normal operation, the difference between the RecSeqNum and local SendSeqNum will be at most 1. There are only two cases where this difference can be more than 1: when a node reboots and when

switching over to a backup control channel.

Having sequence numbers in the Hello messages allows each node to verify that its peer is receiving its Hello messages. This provides a two-fold service. First, the remote node will detect that a node has rebooted if TxSeqNum=1. If this occurs, the remote node will indicate its knowledge of the reboot by setting RcvSeqNum=1 in the Hello messages that it sends and SHOULD wait to receive a Hello message with TxSeqNum=2 before transmitting any messages other than Hello messages. Second, by including the RcvSeqNum in Hello packets, the local node will know which Hello packets the remote node has received. This is important because it helps coordinate control-channel switchover in case of a control channel failure.

4.2.3. Control channel switchover

As mentioned above, LMP requires that a control channel always be available for a link, and multiple mechanisms are used within LMP to ensure that the switchover of a control channel is both smooth and proper. Control channels may need to be switched as a result of a control channel failure or for administration purposes (e.g., routine fiber maintenance, reverting back to a primary control channel, etc.), and peer connectivity must be maintained to ensure that unnecessary rerouting of user traffic is avoided and false failures are not reported.

To ensure that a smooth transition occurs when switching to a backup control channel, a ControlChannelSwitchover flag is available in the Common Header of LMP packets. The receipt of a Hello message with ControlChannelSwitchover = 1 indicates that the remote node is switching to the backup control channel, and the local node MUST begin listening to the backup control channel in addition to the primary control channel.

To ensure that both nodes switch to the backup control channel successfully, both the local and remote nodes MUST transmit messages over both the primary and backup control channels until the switchover is successful. Messages on the primary control channel MUST have the ControlChannelSwitchover flag set to 1 and MUST not increment the TxSeqNum (even upon the receipt of a Hello message with the current TxSeqNum reflected in the RcvSeqNum field). Messages on the backup control channel MUST set the ControlChannelSwitchover flag to 0 and MUST increment the TxSeqNum by 1 to distinguish messages on the two channels. If the TxSeqNum

of the Hello messages on the backup control channel are reflected in the RcvSeqNum of Hello messages being received, then the TxSeqNum MUST be incremented (as per normal operation); this indicates that the backup control channel is operational in the transmit direction and the local node may now stop transmitting Hello messages over the primary control channel. Once a Hello message is received over the backup control channel indicating that the remote node is receiving confirmation of Hello message receipt (this is indicated by an incrementing TxSeqNum), then the local node may stop listening on the primary control channel. When both nodes are only transmitting/receiving Hello packets over the backup control channel, the switchover is successful.

4.2.4. Taking a link down administratively

As mentioned above, a link is DOWN when the control channel and backup control channel(s) are not available and none of the component links are in use. A link may be DOWN, for example, when a link is reconfigured for administrative purposes. A link SHOULD only be administratively taken down if the component links are not in use. To ensure that bringing a link DOWN is done gracefully for administration purposes, a LinkDown flag is available in the Common Header of LMP packets.

When a node receives LMP packets with LinkDown = 1, it must first verify that it is able to bring the link down on its end. Once the verification is done, it must set the LinkDown flag to 1 on all of the LMP packets that it sends. When the node that initiated the LinkDown procedure receives LMP packets with LinkDown = 1, it may then bring the link DOWN.

4.2.5. Degraded (DEG) state

A consequence of allowing the control channels and component links to be transmitted along a separate medium is that the link may be in a state where a control channel and backup control channel(s) are not available, but the component links are still in use. For many applications, it is unacceptable to drop traffic that is in use simply because the control channel is no longer available; however, the traffic that is using the component links may no longer be guaranteed the same level of service. Hence the link is in a Degraded (DEG) state.

When a link is in the DEG state, the routing protocol should be

notified so that new connections are not accepted and resources are no longer advertised for the link. To bring a link back UP out of a degraded state, a node may begin transmitting HelloConfig messages over the primary control channel.

5. Verifying link connectivity

In this section, we describe the mechanism used to verify the physical connectivity of the component links. This will be done initially when a link is established, and subsequently, on a periodic basis for all free component links of a bundled link. A unique characteristic of all-optical PXCs is that the data being transmitted over a component link is not terminated at the PXC, but instead passes through transparently. This characteristic of PXCs poses a challenge for validating the connectivity of the component links since shining unmodulated light through a component link may not result in received light at the next PXC. This is because there may be terminating (or opaque) elements, such as DWDM equipment, in between the PXCs. Therefore, to ensure proper verification of component link connectivity, we require that until the component links are allocated, they must be opaque. There is no requirement that all component links be terminated simultaneously, but at a minimum, the component links must be able to be terminated one at a time. Furthermore, we assume that the nodal architecture is designed so that messages can be sent and received over any component link. Note that this requirement is trivial for DXCs (and OEO nodes in general) since each component link is received electronically before being forwarded to the next DXC, but that in PXCs this is an additional requirement.

To interconnect two nodes, a link must be added between them, and at a minimum, the link must contain a control channel spanning the two nodes. Optionally, the attributes of a link may include the

protection mechanism for the control channel (possibly including an ordered list of backup control channels), a list of component links, and the protection mechanism for each component link.

As part of the link verification protocol, the primary control channel is first verified, and connectivity maintained, using the Hello protocol discussed in Section 4. Once the control channel has been established between the two nodes, component link connectivity is verified by exchanging Ping-type Test messages over each of the component links specified in the bundled link. It should be noted that all LMP messages except for the Test message are exchanged over

the control channel and that Hello messages continue to be exchanged over the control channel during the component link verification process. The Test message is sent over the component link that is being verified. Component links are tested in the transmit direction as they are uni-directional, and as such, it may be possible for both nodes to exchange the Test messages simultaneously.

To initiate the link verification process, the local node first sends a BeginVerify message over the control channel to indicate that the node will begin sending Test messages across the component links of a particular bundled link. The BeginVerify message contains the number of component links that are to be verified; the interval (called VerifyInterval) at which the Test messages will be sent; the encoding scheme and data rate for Test messages; and, in the case where the component links correspond to fibers, the wavelength over which the Test messages will be transmitted. When a node generates a BeginVerify message, it waits either to receive a BeginVerifyAck or BeginVerifyNack message from the adjacent node to accept or reject the verify process.

If the remote node receives a BeginVerify message and it is ready to process Test messages, it MUST send a BeginVerifyAck message back to the local node. When the local node receives a BeginVerifyAck message from the remote node, it will begin testing the component links by transmitting periodic Test messages over each component link. The Test message will include the local LinkId for the associated component link. The remote node will return a TestStatusSuccess or TestStatusFail message in response for each component link and will expect a TestStatusAck message from the local node to confirm receipt of these messages.

The local (transmitting) node will send a given Test message periodically (at least every VerifyInterval ms) on the corresponding component link until it receives a correlating TestStatusSuccess or TestStatusFailure message on the control channel from the remote (receiving) node. The remote node will send a given TestStatus message periodically over the control channel until it receives either a correlating TestStatusAck message or an EndVerify message on the control channel. It is also permissible for the sender to terminate Test messages over a component link without receiving a TestStatusSuccess or TestStatusFailure message. Message correlation is done using the local node's LinkId and message identifiers.

When the Test message is detected at a node, the received LinkId is recorded and mapped to the local LinkId for that channel. The receipt of a TestStatusSuccess message indicates that the Test message was detected and the physical connectivity of the component link has been verified. The TestStatusSuccess message includes both the local LinkId and remote nodeÆs LinkId. When the TestStatusSuccess message is received, the local node SHOULD mark the component link as UP, send a TestStatusAck message to the remote node, and begin testing the next component link. If, however, the Test message is not detected at the remote node within an observation period (specified by the VerifyDeadInterval), the remote node will send a TestStatusFailure message over the control channel indicating that the verification of the physical connectivity of the component link has failed. When the local node receives a TestStatusFailure message, it will mark the component link as FAILED, send a TestStatusAck message to the remote node, and begin testing the next component link. When all the component links on the list have been tested, the local node will send an EndVerify message to indicate that testing has been completed on this link. Upon the receipt of an EndVerify message, an EndVerifyAck message MUST be sent.

Both the local and remote nodes will maintain the complete list of LinkId mappings for correlation purposes.

5.1. Example of link verification

Figure 1 shows an example of the link verification scenario that is executed when a link between PXC A and PXC B is added. In this example, the bundled link will consist of a bi-directional control channel (indicated by a "c") and three free component links (each transmitted along a separate fiber). The verification process is as follows: PXC A sends a BeginVerify message over the control channel to PXC B indicating it will begin verifying the component links. PXC B receives the BeginVerify message and returns the BeginVerifyAck message over the control channel to PXC A. When PXC A receives the BeginVerifyAck message, it begins transmitting periodic Test messages over the first component link (LinkId=1). When PXC B receives the Test messages, it maps the received LinkId to its own local LinkId = 10 and transmits a TestStatusSuccess message over the control channel back to PXC A. The TestStatusSuccess message will include both the local and received LinkIds for the component link. PXC A will send a TestStatusAck message over the control channel back to PXC B indicating it received the TestStatusSuccess message. The process is repeated until all of the component links are verified. At this point, PXC A will send an EndVerify message over the control channel to PXC B to indicate that testing is complete and PXC B will respond by sending an EndVerifyAck message over the control channel back to PXC A.

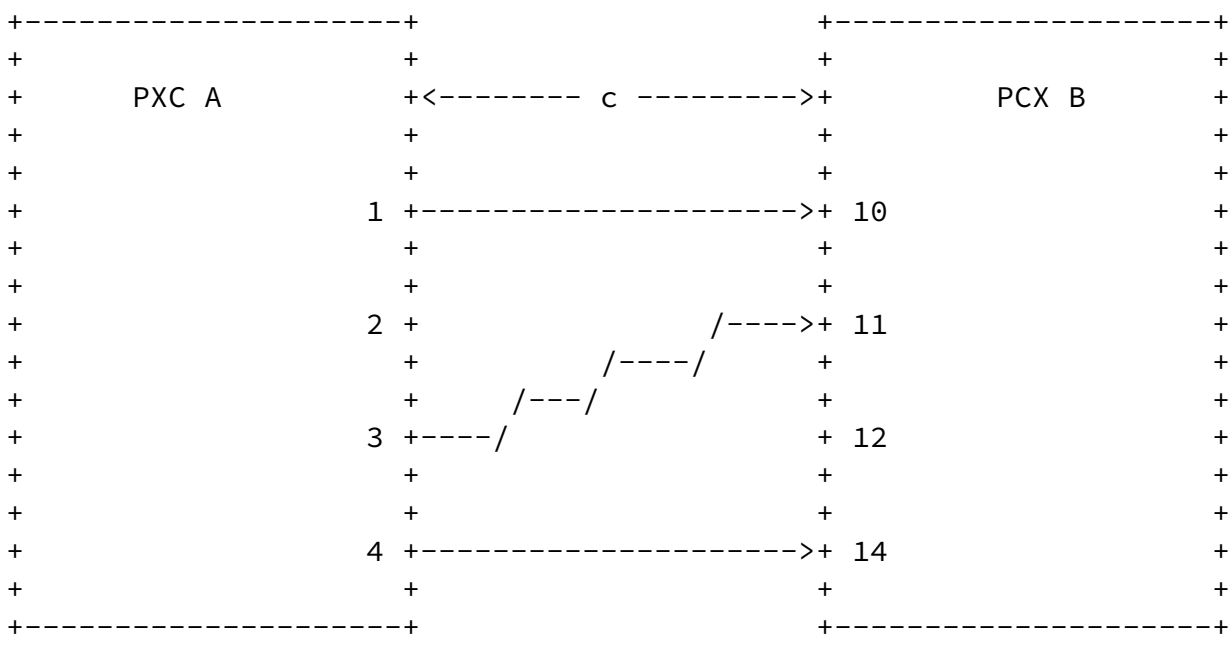


Figure 1: Example of link connectivity between PXC A and PXC B.

6. LinkSummary message

As part of LMP, a LinkSummary message must be transmitted in order to add component links to a bundled link, change LinkIds, or change a link's protection mechanism. In addition, the LinkSummary message can be exchanged at any time a link is UP and not in the Verification process. The LinkSummary message contains the primary and backup CCIds, the IP address for the link (binding the CCIds to the link IP addresses), and the local and remote LinkIds for each component link and their associated priorities. In addition, each component link may have one or more associated protection component links defined for local (span) protection (e.g., M:N, 1+1). If the LinkSummary message is received from a remote node and the LinkId mappings match those that are stored locally, then the two nodes have agreement on the Verify process. Furthermore, any protection definitions that are included in the LinkSummary message must be accepted or rejected by the local node. To signal agreement on the LinkId mappings and protection definitions, a LinkSummaryAck message is transmitted. Otherwise, a LinkSummaryNack message will be transmitted, indicating which channels are not correct and/or which protection definitions are not accepted. If a LinkSummaryNack message indicates that the LinkId mappings are not correct, the link

verification process should be repeated for all mismatched free component links; if an allocated component link has a mapping mismatch, it should be flagged and verified when it becomes free. If, however, a LinkSummaryNack message indicates that a component link's protection mechanism is not accepted, then that component link's protection mechanism cannot be changed; in other words, both local and remote nodes must agree on the protection mechanism for each component link.

7. Fault localization

In this section, we describe a mechanism in LMP that is used to rapidly isolate link failures. As before, we assume each link has a bi-directional control channel that is always available for inter-node communication and that the control channel spans a single hop between two neighboring nodes. The case where a control channel is no longer available between two nodes is beyond the scope of this draft. The mechanism used to rapidly isolate link failures is designed to work for unidirectional LSPs, and can be easily extended to work for bi-directional LSPs; however, for the purposes of this document, we only discuss the operation when the LSPs are unidirectional.

Recall that a bundled link connecting two nodes consists of a control channel and a number of component links. If one or more component links fail between two nodes, a mechanism must be used to rapidly locate the failure so that appropriate protection/restoration mechanisms can be initiated. An important implication of using PXCs is that traditional methods that are used to monitor the health of allocated component links in OEO nodes (e.g., DXCs) may no longer be appropriate, since PXCs are transparent to the bit-rate, format, and wavelength. Instead, fault detection is delegated to the physical layer (i.e., loss of light or optical monitoring of the data) instead of layer 2 or layer 3.

7.1. Fault detection

As mentioned earlier, fault detection must be handled at the layer closest to the failure; for optical networks, this is the physical (optical) layer. One measure of fault detection at the physical layer is simply detecting loss of light (LOL). Other techniques for

monitoring optical signals are still being developed and will not be further considered in this document. However, it should be clear that the mechanism used to locate the failure is independent of the mechanism used to detect the failure, but simply relies on the fact that a failure is detected.

7.2. Fault localization mechanism

If component links fail between two PXCs, the power monitoring system in all of the downstream nodes will detect LOL and indicate a failure. To correlate multiple failures between a pair of nodes, a monitoring window can be used in each node to determine if a single component link has failed or if multiple component links have failed.

As part of the fault localization, a downstream node that detects component link failures will send a ChannelFail message to its upstream neighbor (bundling together the notification of all of the failed component links) and the ports associated with the failed component links will be put into the standby state. An upstream node that receives the ChannelFail message will correlate the failure to see if there is a failure on the corresponding input and

---

output ports for the LSP(s). If there is also a failure on the input port(s) of the upstream node, the node will return a ChannelFailAck message to the downstream node (bundling together the notification of all the component links), indicating that it too has detected a failure. If, however, the fault is CLEAR in the upstream node (e.g., there is no LOL on the corresponding input channels), then the upstream node will have localized the failure and will return a ChannelFailNack message to the downstream node. Once the failure has been localized, the signaling protocols can be used to initiate span or path protection/restoration procedures.

7.3. Examples of fault localization

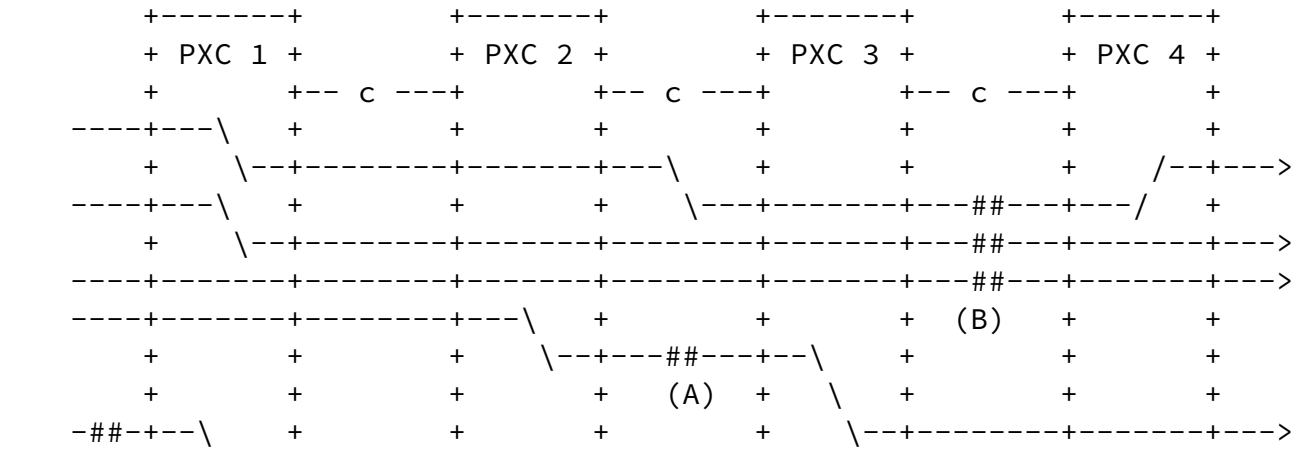
In Fig. 2, a sample network is shown where four PXCs are connected in a linear array configuration. The control channels are bi-directional and are labeled with a "c". All LSPs are uni-directional going left to right.

In the first example [see Fig. 2(A)], there is a failure on a single component link between PXC2 and PXC3. Both PXC3 and PXC4 will detect the failure and each node will send a ChannelFail message to the corresponding upstream node (PXC3 will send a message to PXC2

and PXC4 will send a message to PXC3). When PXC3 receives the ChannelFail message from PXC4, it will correlate the failure and return a ChannelFailAck message back to PXC4. Upon receipt of the ChannelFailAck message, PXC4 will move the associated ports into a standby state. When PXC2 receives the ChannelFail message from PXC3, it will correlate the failure, verify that it is CLEAR, localize the failure to the component link between PXC2 and PXC3, and send a ChannelFailNack message back to PXC3.

In the second example [see Fig. 2(B)], there is a failure on three component links between PXC3 and PXC4. In this example, PXC4 has correlated the failures and will send a bundled ChannelFail message for the three failures to PXC3. PXC3 will correlate the failures, localize them to the channels between PXC3 and PXC4, and return a bundled ChannelFailNack message back to PXC4.

In the last example [see Fig. 2(C)], there is a failure on the tributary link of the ingress node (PXC1) to the network. Each downstream node will detect the failure on the corresponding input ports and send a ChannelFail message to the upstream neighboring node. When PXC2 receives the message from PXC3, it will correlate the ChannelFail message and return a ChannelFailAck message to PXC3 (PXC3 and PXC4 will also act accordingly). Since PXC1 is the ingress node to the optical network, it will correlate the failure and localize the failure to the component link between itself and the network element outside the optical network.



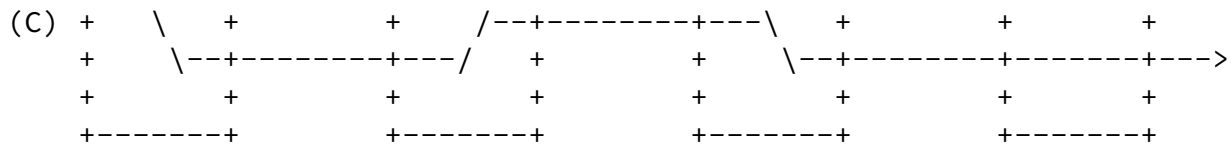


Figure 2: We show three types of component link failures (indicated by ## in the figure): (A) a single component link fails between two PXCs, (B) three component links fail between two PXCs, and (C) a single component link fails on the tributary input of PXC 1. The control channel connecting two PXCs is indicated with a "c".

8. Finite State Machine

8.1. Bringing a link UP

	0	1	2
External event starts process	1a	-	-
Receive HelloConfig with agreable parameters	2b,d	2b,d	2b,d
Receive HelloConfig with unacceptable parameters	0c	1c	0c
Receive HelloConfigAck	-	2d	2d
Receive HelloConfigNack	-	0	2b,d
Receive Hello	-	1a	3

States
0 DOWN
1 CONFIG - Wait for HelloConfig response
2 CONFIG - Wait for Hello message
3 UP

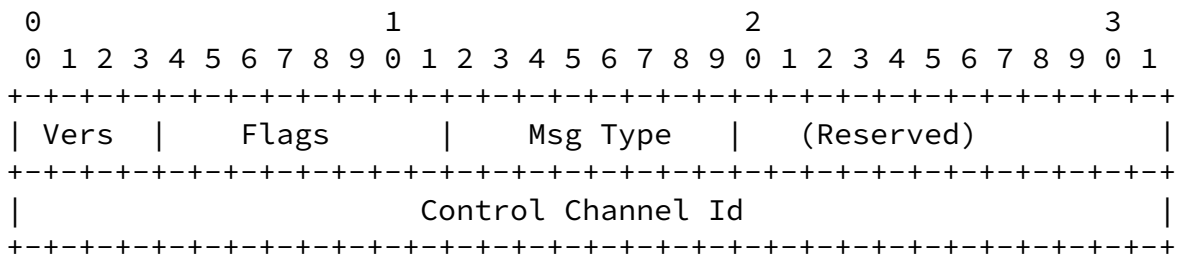
Actions
a send HelloConfig

```
 b  send HelloConfigAck
 c  send HelloConfigNack
 d  send Hello
```

9. LMP Message Formats

9.1. Common Header

In addition to the standard IP header, all LMP control-channel messages have the following common header:



Vers: 4 bits

Protocol version number. This is version 1.

Flags: 8 bits

1 = LinkDown

2 = ControlChannelSwitchover

The remaining flag bits are not yet defined.

Msg Type: 8 bits

1 = HelloConfig

2 = HelloConfigAck

3 = HelloConfigNack

4 = Hello

5 = BeginVerify

6 = BeginVerifyAck

7 = BeginVerifyNack

8 = EndVerify

9 = EndVerifyAck

10 = Test

11 = TestStatusSuccess

12 = TestStatusFailure

13 = TestStatusAck

14 = LinkSummary

15 = LinkSummaryAck

16 = LinkSummaryNack

17 = ChannelFail

18 = ChannelFailAck

19 = ChannelFailNack

All of the messages are sent over the control channel EXCEPT the Test message (Msg Type = 10) which is sent over the component link that is being tested.

Control Channel Id: 32 bits

The Control Channel Id (CCId) identifies the control channel of the sender associated with the message. For the Test message, which is sent over a component link, this is the control channel associated with the Verify procedure.

9.2 Parameter Negotiation

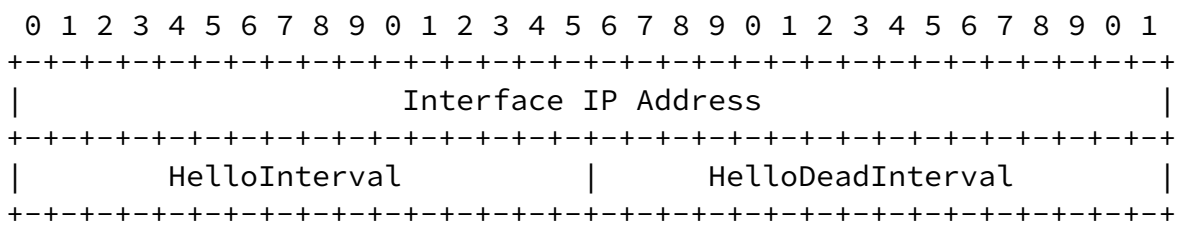
9.2.1 HelloConfig Message (MsgType = 1)

The HelloConfig message is used to negotiate parameters for the Hello phase of LMP. The format of the HelloConfig message is as follows:

<HelloConfig Message> ::= <Common Header> <HelloConfig>

The HelloConfig Object has the following format:

0 1 2 3



Interface IP Address: 32 bits.

This is the address of the interface (or possibly virtual interface) for the bundled link. If the bundled link is unnumbered, the address is the Router ID of the node.

HelloInterval: 16 bits.

Indicates how frequently the Hello packets will be sent and is measured in milliseconds (ms).

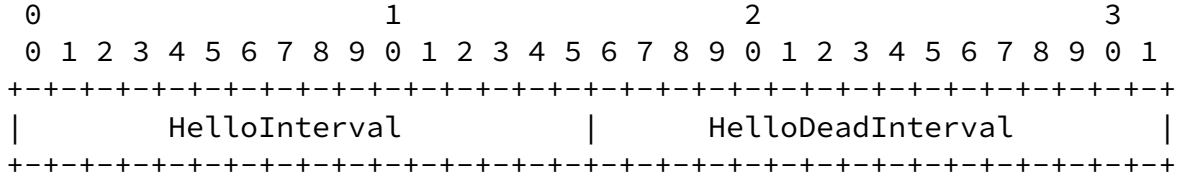
HelloDeadInterval: 16 bits.

If no Hello packets are received within the HelloDeadInterval, the control channel is assumed to have failed and is measured in milliseconds (ms).

9.2.2 HelloConfigAck Message (MsgType = 2)

<HelloConfigAck Message> ::= <Common Header> <HelloConfigAck>

The HelloConfigAck Object has the following format:



HelloInterval: 16 bits.

Indicates how frequently the Hello packets will be sent and is measured in milliseconds (ms).

HelloDeadInterval: 16 bits.

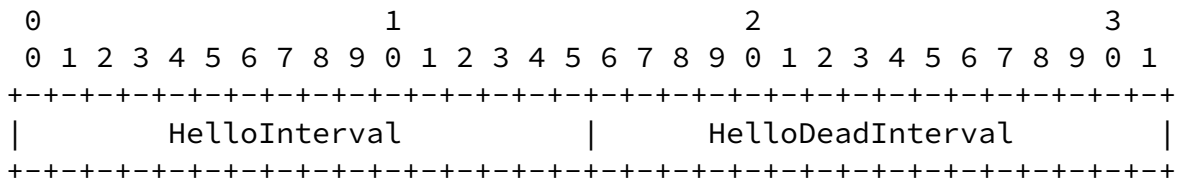
If no Hello packets are received within the HelloDeadInterval, the control channel is assumed to be dead and is measured in milliseconds (ms).

The values of the HelloInterval and HelloDeadInterval are copied from the HelloConfig message that is being acknowledged.

9.2.3 HelloConfigNack Message (MsgType = 3)

<HelloConfigNack Message> ::= <Common Header> <HelloConfigNack>

The HelloConfigNack Object has the following format:



HelloInterval: 16 bits.

Indicates how frequently the Hello packets will be sent and is measured in milliseconds (ms).

HelloDeadInterval: 16 bits.

If no Hello packets are received within the HelloDeadInterval, the control channel is assumed to be dead and is measured in milliseconds (ms).

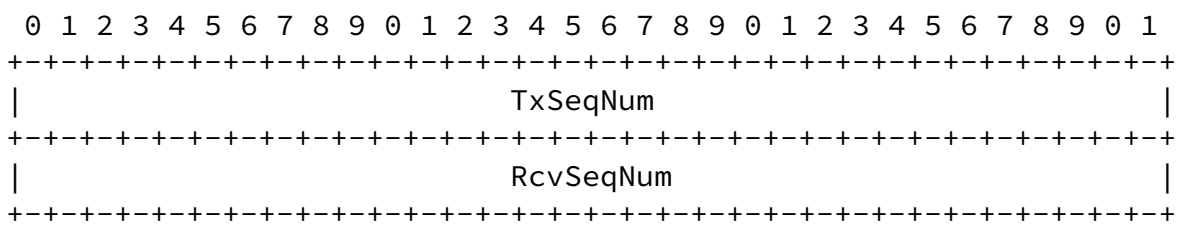
The values of the HelloInterval and HelloDeadInterval MUST be equal to the locally accepted values.

9.3 Hello Message (MsgType = 4)

The format of the Hello message is as follows:

<Hello Message> ::= <Common Header> <Hello>.

The Hello object format is shown below:



TxSeqNum: 32 bits

This is the current sequence number for this Hello message. This sequence number will be incremented when either (a) the sequence number is reflected in the RcvSeqNum of a Hello packet that is received over the control channel, or (b) the Hello packet is transmitted over a backup control channel.

TxSeqNum=0 is not allowed.

TxSeqNum=1 is reserved to indicate that a node has booted or rebooted.

RcvSeqNum: 32 bits

This is the sequence number of the last Hello message received over the control channel.

RcvSeqNum=0 is reserved to indicate that a Hello message has not yet been received.

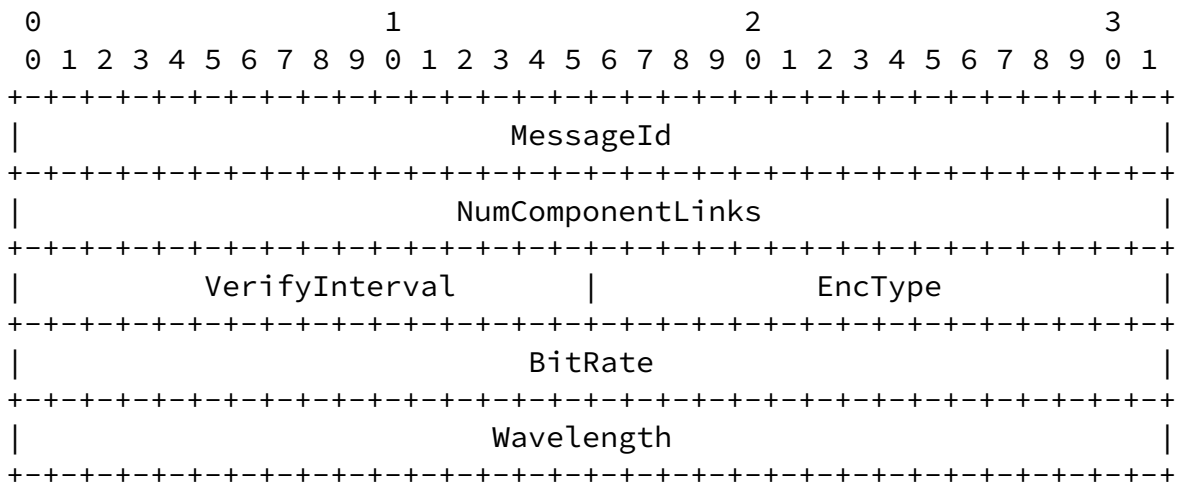
9.4 Link Verification

9.4.1 BeginVerify Message (MsgType = 5)

The BeginVerify message is sent over the control channel and is used to initiate the link verification process. The format is as follows:

<BeginVerify Message> ::= <Common Header> <BeginVerify>

The BeginVerify object has the following format:



MessageId: 32 bits

When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgment in the BeginVerifyAck and BeginVerifyNack messages.

NumComponentLinks: 32 bits

This is the number of component links that will be verified.

VerifyInterval: 16 bits

This is the interval between successive Test messages.

EncType: 16 bits

This is required for the purpose of testing where the component links are not required to be the same encoding type as the control channel. The EncType values are consistent with the Link Encoding Type values of [KRB00a] and [KRB00b]and are taken from the following list:

1	Standard SONET
2	Arbitrary SONET
3	Standard SDH
4	Arbitrary SDH

5	Clear (not used)

6	GigE
7	10GigE

BitRate: 32 bits

This is the bit rate at which the Test messages will be transmitted and is expressed in bytes.

Wavelength: 32 bits

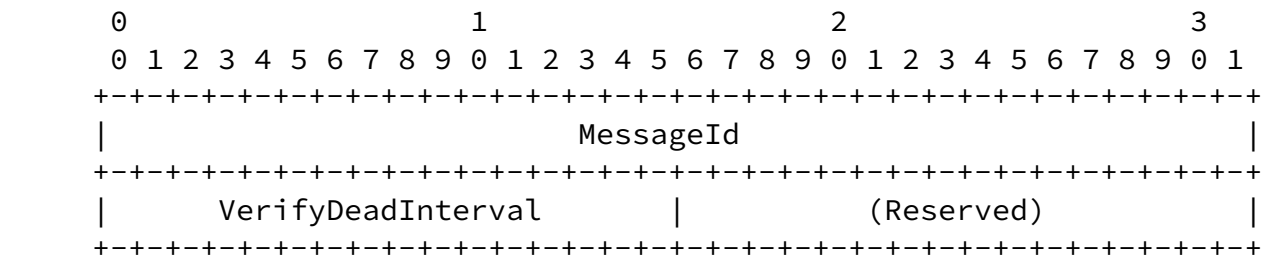
When a component link is assigned to a fiber, it is essential to know which wavelength the test messages will be transmitted over. This value corresponds to the wavelength at which the Test messages will be transmitted over and is measured in nanometers (nm). If each component link corresponds to a separate wavelength, than this value SHOULD be set to 0.

9.4.2 BeginVerifyAck Message (MsgType = 6)

When a BeginVerify message is received and Test messages are ready to be processed, a BeginVerifyAck message MUST be transmitted.

<BeginVerifyAck Message> ::= <Common Header> <BeginVerifyAck>

The BeginVerifyAck object has the following format:



MessageId: 32 bits

This is copied from the BeginVerify message being acknowledged.

VerifyDeadInterval: 16 bits

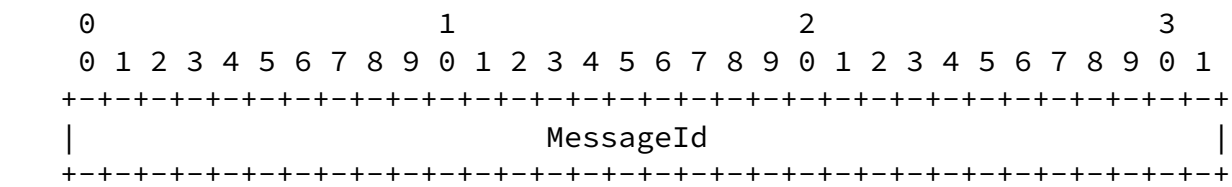
If a Test message is not detected within the VerifyDeadInterval, then a node will send the TestStatusFailure message for that component link.

9.4.3 BeginVerifyNack Message (MsgType = 7)

If a BeginVerify message is received and a node is unwilling or unable to begin the Verification procedure, a BeginVerifyNack message MUST be transmitted.

<BeginVerifyNack Message> ::= <Common Header> <BeginVerifyNack>

The BeginVerifyNack object has the following format:



MessageId: 32 bits

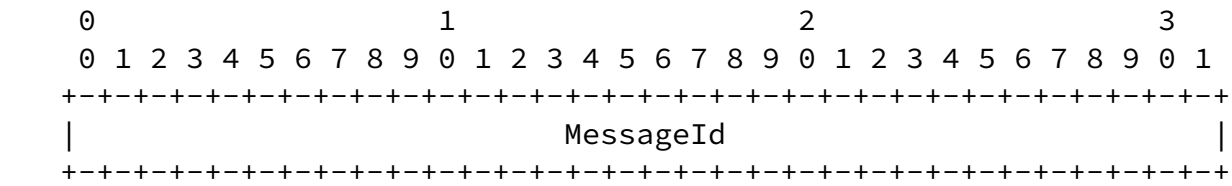
This is copied from the BeginVerify message being negatively acknowledged.

9.4.4 EndVerify Message (MsgType = 8)

The EndVerify message is sent over the control channel and is used to terminate the link verification process. The format is as follows:

<EndVerify Message> ::= <Common Header> <EndVerify>

The EndVerify object has the following format:



MessageId: 32 bits

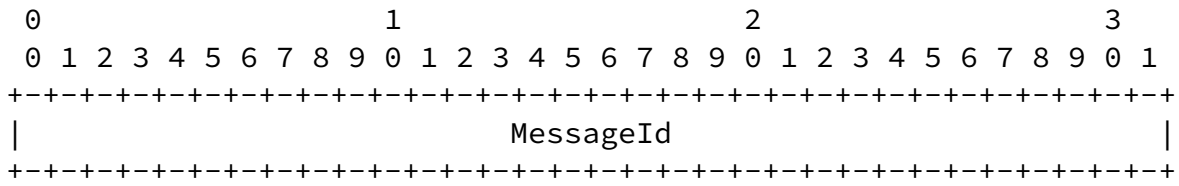
When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgement in the EndVerifyAck message.

9.4.5 EndVerifyAck Message (MsgType =9)

The EndVerifyAck message is sent over the control channel and is used to acknowledge the termination of the link verification process. The format is as follows:

<EndVerifyAck Message> ::= <Common Header> <EndVerifyAck>

The EndVerifyNack object has the following format:



MessageId: 32 bits

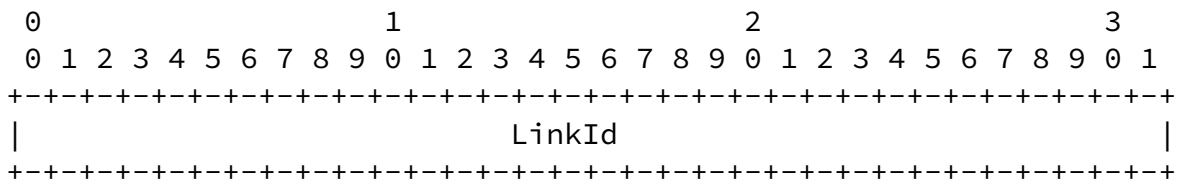
This is copied from the EndVerify message being acknowledged.

9.4.6 Test Message (MsgType = 10)

The Test message is transmitted over the component link and is used to verify the component link connectivity. The format is as follows:

<Test Message> ::= <Common Header> <Test>

The Test object has the following format:



LinkId: 32 bits

The LinkId identifies the component link over which this message is sent. A valid LinkId MUST be nonzero.

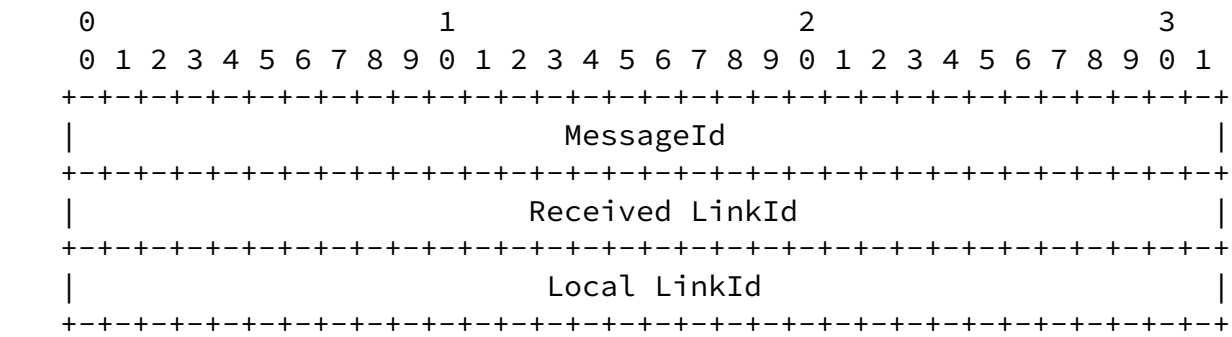
Note that this message is sent over a component link and NOT over the control channel.

9.4.7 TestStatusSuccess Message (MsgType = 11)

The TestStatusSuccess message is transmitted over the control channel and is used to transmit the mapping between the local LinkId and the LinkId that was received in the Test message.

<TestStatus Message> ::= <Common Header> <TestStatusSuccess>

The TestStatusSuccess object has the following format:



MessageId: 32 bits

When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgement in the TestStatusAck message.

Received LinkId: 32 bits

This is the value of the LinkId that was received in the Test

message. A valid LinkId MUST be nonzero, therefore, a value of 0 in the Received LinkId indicates that the Test message was not detected.

Local LinkId: 32 bits

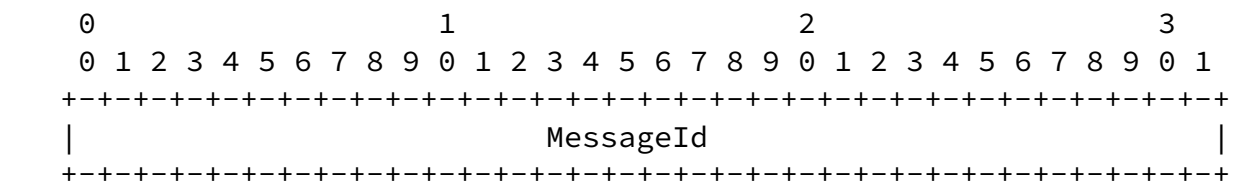
This is the local value of the LinkId. A valid LinkId MUST be nonzero.

9.4.8 TestStatusFailure Message (MsgType = 12)

The TestStatusFailure message is transmitted over the control channel and is used to indicate that the Test message was not received.

<TestStatus Message> ::= <Common Header> <TestStatusFailure>

The TestStatusFailure object has the following format:



MessageId: 32 bits

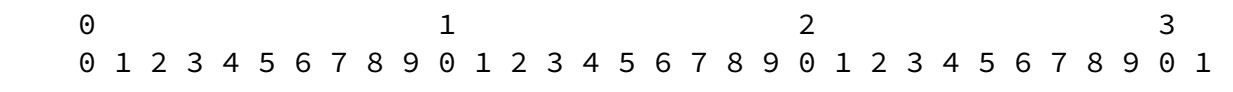
When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgement in the TestStatusAck message.

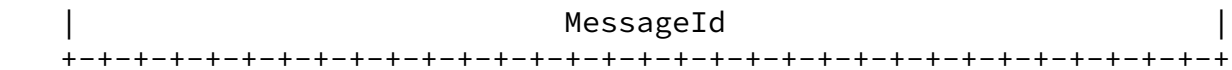
9.4.9 TestStatusAck Message (MsgType = 13)

The TestStatusAck message is used to acknowledge receipt of the TestStatusSuccess or TestStatusFailure messages.

<TestStatusAck Message> ::= <Common Header> <TestStatusAck>

The TestStatusAck object has the following format:





MessageId: 32 bits

This is copied from the TestStatusSuccess or TestStatusFailure message being acknowledged.

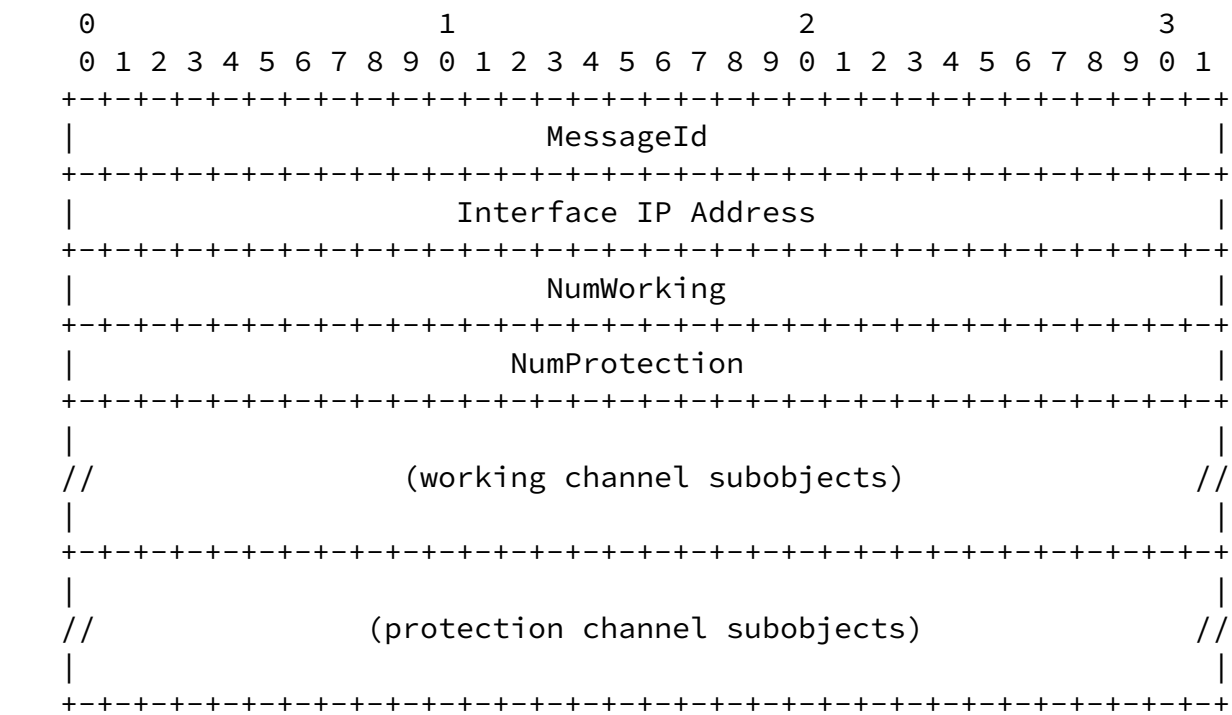
9.5 Link Summary Messages

9.5.1 LinkSummary Message (MsgType = 14)

The LinkSummary message is used to synchronize the LinkIds and correlate the properties of the link. The format of the LinkSummary message is as follows:

<LinkSummary Message> ::= <Common Header> <LinkSummary>

The LinkSummary Object has the following format:



MessageId: 32 bits

When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgement in the LinkSummaryAck and LinkSummaryNack messages.

Interface IP Address: 32 bits

This is the local IP address of the interface (or possibly virtual interface) for the bundled link. If the bundled link is unnumbered, the address is the Router ID of the node.

NumWorking: 32 bits

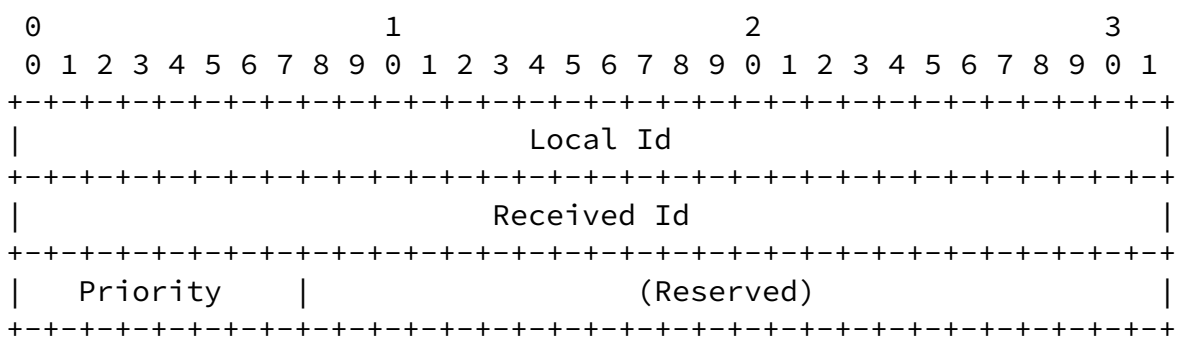
This value is the number of working channels in the link. This also indicates how many working channel subobjects are in the LinkSummary message.

NumProtection: 32 bits

This value is the number of protection channels in the link. This also indicates how many protection channel subobjects are in the LinkSummary message.

The LinkSummary message contains a list of working channel subobjects and protection channel subobjects. The list of working channels MUST include the control channel. Any backup control channels that are defined MUST be included in the list of protection channel subobjects. Note that a backup control channel can also be a working component link (provided it has preemptive priority over the working component link), so it is possible to appear in both the working channel subobject as well as the protection channel subobject.

The Working Channel Subobject has the following format:



Local Id: 32 bits

This is the local value of the LinkId (for component link) or CCId (for control channel).

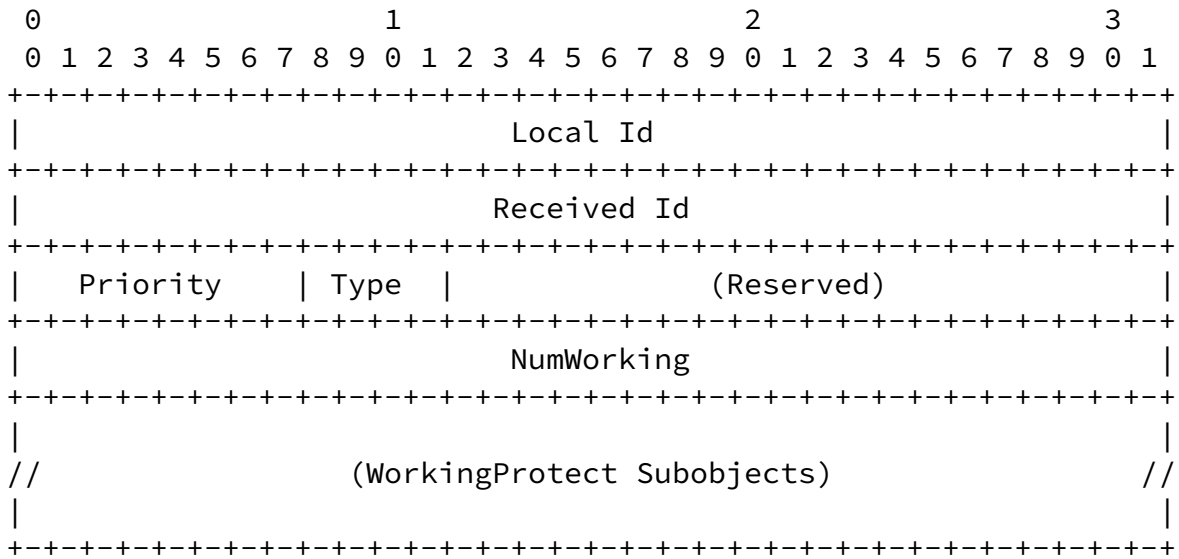
Received Id: 32 bits

This is the value of the corresponding Id. If this is a component link, then this is the value that was received in the Test message. If this is the primary control channel, then this is the value that is received in all of the Verify messages.

Priority: 8 bits

This is the channel priority and is in the range of 0 to 7. The value 0 is the highest priority. The control channel MUST have a priority of 0.

The Protection Channel Subobject has the following format:



Local Id: 32 bits

This is the local value of the LinkId. This could be a protection component link and/or a protection control channel. In addition, a protection control channel could also be a working component link (so it could appear in both the working channel subobject as well as the protection channel subobject).

Received Id: 32 bits

This is the value of the corresponding LinkId that was received

in the Test message.

Priority: 8 bits.

The priority of the resources, in the range of 0 to 7. The value 0 is the highest priority.

Type: 4 bits.

This is the protection type.

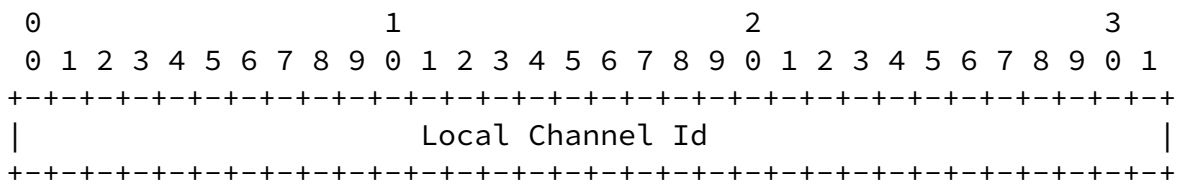
1 = 1+1 protection

2 = M:N protection

NumWorking: 32 bits

This is the number of working channels that this channel is protecting. This defines the number of WorkingProtect subjects.

The WorkingProtect Subobject has the following format:



Local Channel Id: 32 bits

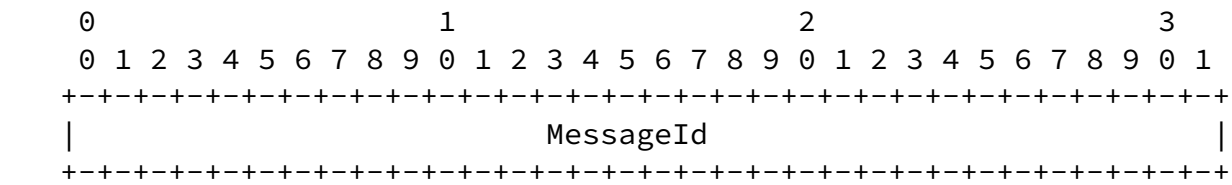
This is the local channel Id of the working channel that is being protected. This channel could be a control channel or a component link.

9.5.2 LinkSummaryAck Message (MsgType = 15)

The LinkSummaryAck message is used to indicate agreement on the LinkId synchronization and acceptance/agreement on all the link parameters.

<LinkSummaryAck Message> ::= <Common Header> <LinkSummaryAck>

The LinkSummaryAck object has the following format:



MessageId: 32 bits

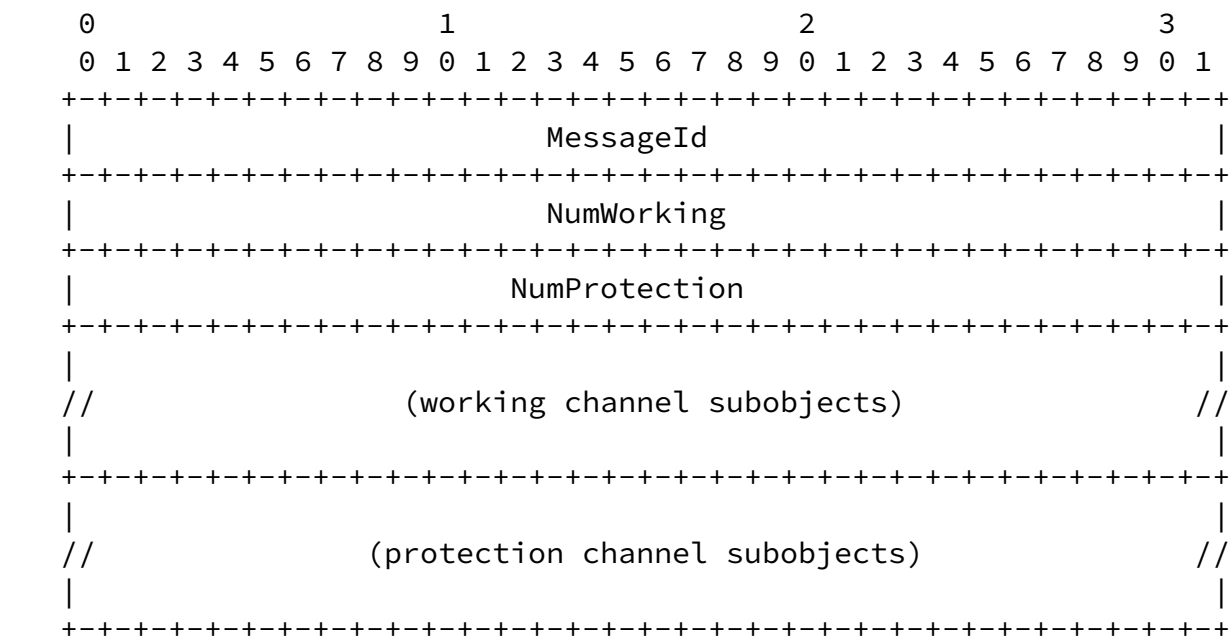
This is copied from the LinkSummary message being acknowledged.

9.5.3 LinkSummaryNack Message (MsgType = 16)

The LinkSummaryNack message is used to indicate disagreement on LinkId synchronization and/or the link parameters.

<LinkSummaryNack Message> ::= <Common Header> <LinkSummaryNack>

The LinkSummaryNack object has the following format:



MessageId: 32 bits

This is copied from the LinkSummary message being negatively acknowledged.

NumWorking: 32 bits

This value is the number of working channels in the LinkSummary message that are being negatively acknowledged. This also indicates how many working channel subobjects are in the LinkSummaryNack message.

NumProtection: 32 bits

This value is the number of protection channels in the LinkSummary message that are being negatively acknowledged. This also indicates how many protection channel subobjects are in the LinkSummaryNack message.

The Working Channel and Protection Channel Subobjects are copied from the LinkSummary message being negatively acknowledged. These represent the Subobjects that were not accepted.

As an optimization, the entire LinkSummary message can be rejected by setting NumWorking = NumProtection = 0. If this is done, the working and protection channel subobjects are not required in the LinkSummaryNack message.

9.6 Failure Messages

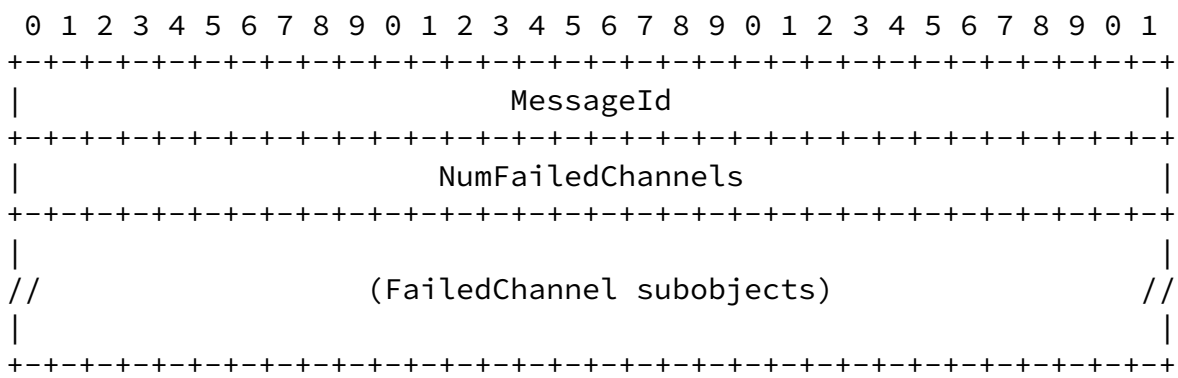
9.6.1 ChannelFail Message (MsgType = 17)

The ChannelFail message is sent over the control channel and is used to query a neighboring node when a link or channel failure is detected. The format is as follows:

<ChannelFail Message> ::= <Common Header> <ChannelFail>

The format of the ChannelFail object is as follows:

0 1 2 3



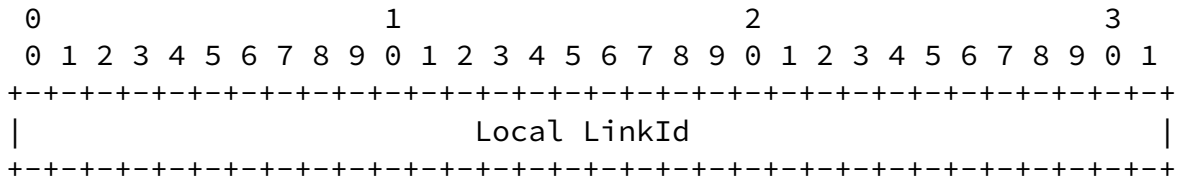
MessageId: 32 bits

When combined with the CCId and MsgType, the MessageId field uniquely identifies a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgement in the ChannelFailAck and ChannelFailNack messages.

NumFailedChannels: 32 bits

This value indicates how many channels have failed. This also defines the number of FailedChannel subobjects.

The FailedChannel Subobjects is a list of the failed channels and has the following format:



Local LinkId: 32 bits

This is the local LinkId of the component link that has failed.

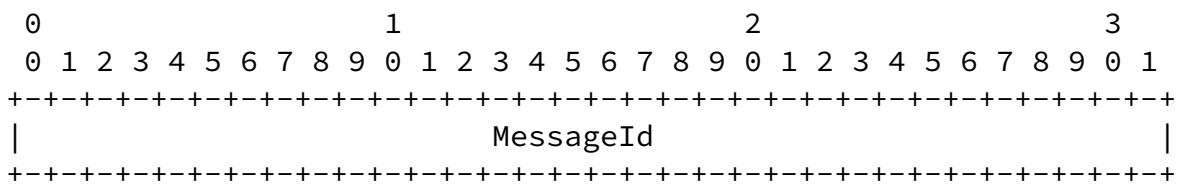
9.6.2 ChannelFailAck Message (MsgType = 18)

The ChannelFailAck message is used to indicate that all of the failed channels reported in the ChannelFail message also have

failures on the corresponding input channels. The format is as follows:

<ChannelFailureAck Message> ::= <Common Header> <ChannelFailureAck>

The ChannelFailureAck object has the following format:



MessageId: 32 bits

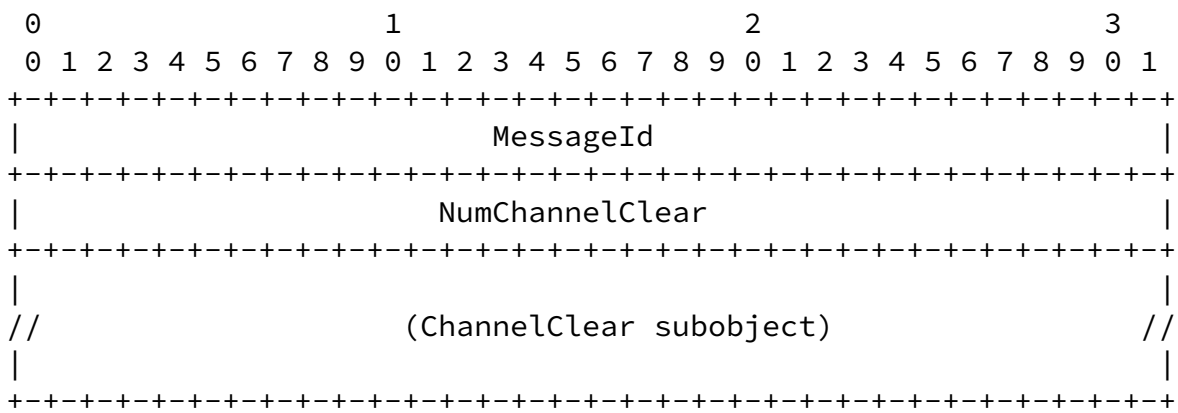
This is copied from the ChannelFail message being acknowledged.

9.6.3 ChannelFailNack Message (MsgType = 19)

The ChannelFailNack message is used to indicate that the failed component link(s) reported in the ChannelFail message are CLEAR in the upstream node, and hence, the failure has been isolated between the two nodes.

<ChannelFailNack Message> ::= <Common Header> <ChannelFailNack>

The ChannelFailNack object has the following format:



MessageId: 32 bits

This is copied from the ChannelFail message being negatively acknowledged.

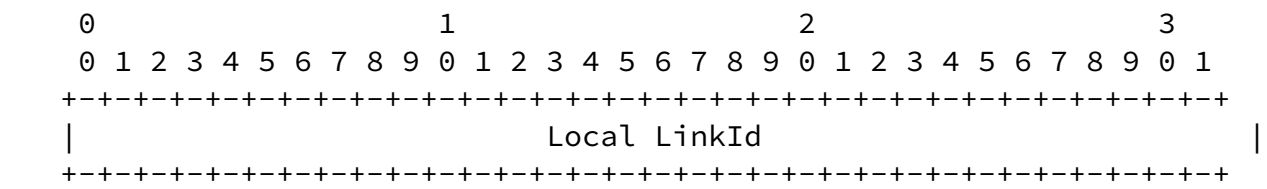
NumChannelFail: 32 bits

This value is the number of failed component links reported in the ChannelFail message that also have failures on the corresponding inputs.

NumChannelClear: 32 bits

This is the number of failed component links reported in the ChannelFail message that are CLEAR in the upstream node. This also indicates how many ChannelClear subobjects are in the ChannelFailNack message.

The ChannelClear subobject is used to indicate which failed component links have been isolated and has the following format:



Local LinkId: 32 bits

This is the local LinkId of the component link where the failure has been isolated.

10. Security Considerations

Security considerations are for future study, however, LMP is a point-to-point protocol so security is largely derived from the physical security of the optical network.

11. References

[Bra96] Bradner, S., "The Internet Standards Process -- Revision 3," BCP 9, RFC 2026, October 1996.

[KRB00] Kompella, K., Rekhter, Y., Berger, L., "Link Bundling in MPLS Traffic Engineering," Internet Draft, draft-kompella-mpls-bundle-02.txt, (work in progress), July 2000.

[ABD00] Ashwood-Smith, P., Berger, L., et al, "Generalized MPLS - Signaling Functional Description," Internet Draft, draft-generalized-signaling-00.txt, (work in progress), July 2000.

[RNT99] Rosen, E. C., Rekhter, Y., et al, "MPLS Label Stack Encoding," Internet Draft, draft-ietf-mpls-label-encaps-07.txt, (work in progress), September 1999.

[ARD99] Awduche, D. O., Rekhter, Y., Drake, J., Coltun, R., "Multi-

Protocol Lambda Switching: Combining MPLS Traffic Engineering Control with Optical Crossconnects," Internet Draft, draft-awduche-mpls-te-optical-02.txt, (work in progress), July 2000.

[CBD00] Ceuppens, L., Blumenthal, D., Drake, J., Chrostowski, J., Edwards, W. L., "Performance Monitoring in Photonic Networks," Internet Draft, draft-ceuppens-mpls-optical-00.txt, (work in progress), March 2000.

[ABG99] Awduche, D. O., Berger, L., Gan, D.-H., Li, T., Swallow, G., Srinivasan, V., "Extensions to RSVP for LSP Tunnels," Internet Draft, draft-ietf-mpls-rsvp-lsp-tunnel-06.txt, (work in progress), July 2000.

[Jam99] Jamoussi, B., et al, "Constraint-Based LSP Setup using LDP," Internet Draft, draft-ietf-mpls-cr-ldp-03.txt, (work in progress), September 1999.

[KaY99] Katz, D., Yeung, D., "Traffic Engineering Extensions to OSPF," Internet Draft, draft-katz-yeung-ospf-traffic-02.txt, (work in progress), August 2000.

[SmL99] Smit, H. and Li, T., "IS-IS extensions for Traffic Engineering," Internet Draft,draft-ietf-isis-traffic.txt, (work in progress), September 2000.

[KRB00a] Kompella, K., Rekhter, Y., Banerjee, A., et al, "OSPF Extensions in Support of Generalized MPLS," Internet Draft, draft-kompella-ospf-extensions-00.txt, (work in progress), July 2000.

[KRB00b] Kompella, K., Rekhter, Y., Banerjee, A., et al, "IS-IS Extensions in Support of Generalized MPLS," Internet Draft, draft-kompella-isis-extensions-00.txt, (work in progress), July 2000.

12. Acknowledgments

The authors would like to thank Adrian Farrel for his comments on the draft.

13. Author's Addresses

Jonathan P. Lang
Calient Networks
25 Castilian Drive
Goleta, CA 93117
Email: jplang@calient.net

Krishna Mitra
Calient Networks
5853 Rue Ferrari
San Jose, CA 95138
email: krishna@calient.net

John Drake
Calient Networks
5853 Rue Ferrari
San Jose, CA 95138
email: jdrake@calient.net

Kireeti Kompella
Juniper Networks, Inc.
385 Ravendale Drive
Mountain View, CA 94043
email: kireeti@juniper.net

Yakov Rekhter
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
email: yakov@cisco.com

Lou Berger
LabN Consulting, LLC
email: lberger@labn.net

Debanjan Saha
Tellium Optical Systems
2 Crescent Place
Oceanport, NJ 07757-0901
email: dsaha@tellium.com

Debashis Basak
Marconi
1000 Fore Drive
Warrendale, PA 15086-7502
email: dbasak@fore.com

Hal Sandick
Nortel Networks
email: hsandick@nortelnetworks.com