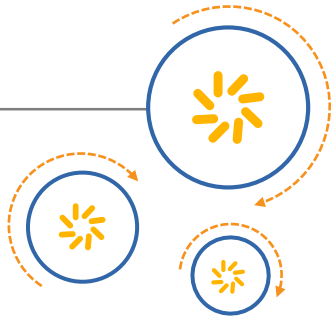
Qualcomm Technologies, Inc.



QMUX V2 Protocol Specification

80-NJ457-1 B

July 22, 2015

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	September 2013	Initial release
B	July 2015	Numerous changes were made to this document; it should be read in its entirety.

Contents

1 Introduction 7

1.1 Purpose 7
1.2 Conventions 7
1.3 Terms and assumptions 7
1.3.1 Nodes 7
1.3.2 Edges 7
1.3.3 Ports 8
1.3.4 Port names 8
1.3.5 Transport reliability 8

2 Usage models 9

2.1 Client/server model 9
2.2 Publish/subscribe model 9

3 Interface 10

4 Message header 11

4.1 Packet header 11
4.2 Control flags 12
4.3 Optional header 12
4.4 Packet types 13

5 Link management 14

5.1 Link initialization 14
5.1.1 Extended link initialization with version negotiation 15
5.2 Link termination 17
5.3 Master-slave configuration 18

6 Name space management 20

6.1 NEW_SERVER packet 20
6.2 REMOVE_SERVER packet 21

7 Endpoint lifetime 22

8 Data transmission 23

8.1 Point-to-point data transmission 23
8.2 Multicast data transmission 23

9 Flow control 26

10 Network topology and multi-hop routing 28
10.1 Route discovery 28
10.2 Multi-hop routing (optional) 28
10.2.1 Link coloring 29
10.2.2 Broadcast rules 29
10.2.3 Forwarding rules 30
10.2.4 Multicast rules 30
10.3 Risk of multi-hop routing 30

A REMOVE_CLIENT optimization 31

Figures

Figure 4-1 QMUX V2 packet header 11
Figure 4-2 Control flags 12
Figure 4-3 Optional Header format 12
Figure 5-1 Basic HELLO packet 14
Figure 5-2 Extended HELLO packet 15
Figure 5-3 Version supported bitmask 16
Figure 5-4 Version negotiation example 17
Figure 5-5 BYE packet 18
Figure 5-6 Link initialization between three connected nodes (A – B – C connection) 19
Figure 6-1 NEW_SERVER packet 20
Figure 6-2 REMOVE_SERVER packet 21
Figure 7-1 REMOVE_CLIENT packet 22
Figure 8-1 Data packet 23
Figure 8-2 Multicast messaging with a 32-bit name (short form) 24
Figure 8-3 Multicast messaging with a 64-bit name (long form) 24
Figure 9-1 Flow control algorithm 26
Figure 9-2 RESUME_TX packet 27
Figure 10-1 A fully-meshed network 28
Figure 10-2 Partially-meshed network requiring multi-hop routing 29
Figure A-1 Mode state transition diagram 32

Tables

Table 3-1 Interface operation, description, and parameters 10

1 Introduction

NOTE: Numerous changes were made to this document; it should be read in its entirety.

1.1 Purpose

This document describes the Qualcomm Multiplexing Protocol (QMUX) V2 specification.

QMUX V2 is designed as a peer-to-peer routing protocol and succeeds QMUX V1. QMUX V2 is mostly a rewrite of QMUX V1 to add features such as routing and distributed service discovery. A router is defined as the entity on each processor node implementing the QMUX V2 Protocol. This document uses the terms router and QMUX V2 interchangeably. Most notably, a router is an entity that implements the QMUX V2 Protocol.

This document is intended for software developers who are using the QMUX V2 Protocol.

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Shading indicates content that has been added or changed in this revision of the document.

1.3 Terms and assumptions

1.3.1 Nodes

Nodes are individual processors on an interconnected network. They are identified by a 16-bit integer. Currently, there are no limitations on this address (other than the requirement that each node use a globally unique ID), but it is recommended that the least significant 8 bits be used, and to reserve the most significant 8 bits for future expansion. The Node ID 0xffff can be reserved to be used when the address is unknown or is irrelevant; no implementation shall expect this behavior and must ignore the field in such circumstances.

1.3.2 Edges

Edges are links connecting two communicating nodes. Edges throughout this document are represented as the tuple of the participating nodes. For example, the edge connecting nodes A and B are represented as (A, B). Any descriptive attributes of an edge are represented as a function of the edge. For example, the status of an edge connecting A and B can be represented as STATUS(A,B).

1.3.3 Ports

Ports are endpoints that identify the clients of the router. A unique 16-bit integer is assigned to each port. A user can open multiple ports, similar to a TCP or UDP socket. The <node, port> pair (referred to as the *address*) identifies an endpoint that must be known to communicate over the network. Unlike TCP/UDP, the port number is ephemeral and allocated by the router rather than bound by the user. The Port ID 0xffff or 0 can be reserved to be used when the address is unknown or is irrelevant; no implementation shall expect this behavior and must ignore the field in such circumstances.

1.3.4 Port names

Port names provide location transparency and they are the preferred way of communication over hard-coded addresses. A port name is composed of a 32-bit integer pair in the form of <type, instance>. The instance field can be used to identify different versions of the same service, but no hard rule has been defined. The router can allow a matching function to be passed in to allow custom look-up rules of the instance.

When a server is created, it must register its <type, instance> pair with the router. The information is then forwarded to the rest of the nodes on the network and kept in each router's database in a distributed fashion. The same <type, instance> pair can resolve to multiple endpoints on the network, and the entire list can be returned to the caller.

1.3.5 Transport reliability

The router protocol is designed with the assumptions of a reliable transport; therefore, there is no sequence number and error correction in the header to maximize performance on an embedded system.

When a packet is forwarded, it is possible for the packet to be dropped due to memory constraints on the specific node. When such errors occur, the downstream node is negatively impacted because there is no error correction or retransmission in the protocol. Therefore, it is a requirement for the forwarder to provide reliable transport for the secondary nodes.

2 Usage models

2.1 Client/server model

For the server to be seen across the network, it must first open a port to the router. A unique port ID is assigned to the endpoint. The server then registers a <type, instance> port name with the router. Messages from remote endpoints are provided to the server endpoint along with a source address. A reply can then be sent using the <node, port> pair directly.

For the client to communicate with the server, it must first open a port to the router. A unique port ID is assigned to the endpoint. The client then looks up the server's address using the <type, instance> pair and sends messages directly to the server.

2.2 Publish/subscribe model

For the subscriber to be seen across the network, it must first open a port to the router. A unique port ID is assigned to the endpoint. The subscriber then registers a <type, instance> port name with the router, which is the subscribed group name. Messages from publishers are provided to the subscriber endpoint along with a source address. The publisher shall multicast its messages to the group name <type, instance> rather than a specified address.

3 Interface

Every router within the network is expected to provide an interface following the format specified in Table 3-1.

Table 3-1 Interface operation, description, and parameters

Operation	Description	Parameters
OPEN	Allocates a port ID on the local node and associates it with the returned handle.	**Input:** None
		Output: Handle
REGISTER	Associates a handle with a port name (<type, instance>).	**Input:** Handle, port name <type, instance>
		Output: None
UNREGISTER	(optional) Disassociates a handle from a previously associated port name (<type, instance>).	**Input:** Handle, port name <type, instance>
		Output: None
LOOKUP	Resolves the physical address of a specified port name.	**Input:** Port name <type, instance>
		Output: Address <node, port>
CLOSE	Releases the handle and port ID associated with the handle.	**Input:** Handle
		Output: None
SENDTO	Sends a message to a destination physical address.	**Input:** Handle, destination address <node, port>, message (buffer, length).
		Output: None
MULTICAST	(optional) Sends a message to all addresses associated with the specified name.	**Input:** Handle, destination name <type, instance>, message (buffer, length).
		Output: None
RECVFROM	Receives a message from a remote source.	**Input:** Handle
		Output: Source address <node, port>, message (buffer, length)

4 Message header

4.1 Packet header

A message, or a packet, is a unit of transmission for the router. A practical message size limit can be 64 KB, but the underlying media might still fragment the message into multiple frames, which is outside the scope of this document. Every message should contain the QMUX V2 header (Figure 4-1). Fields greater than a byte are packed in little-endian format.

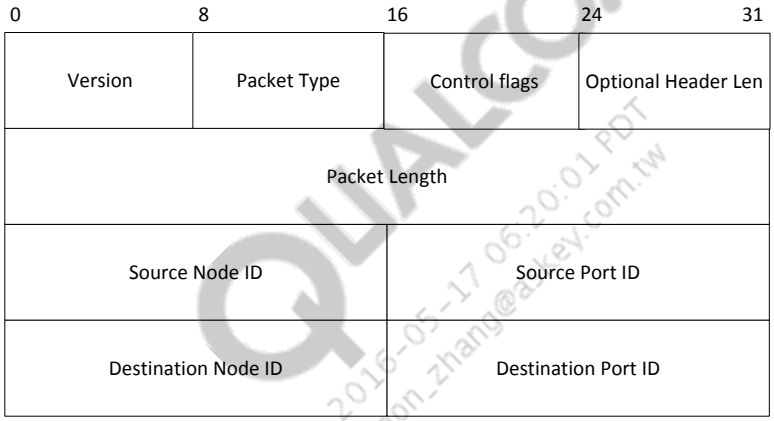


Figure 4-1 QMUX V2 packet header

Header entry	Description
Version	QMUX V2 version is always 3.
Packet Type	Type of packet; see Section 4.4 for details.
Control Flags	Control flags for this packet; see Section 4.2 for details.
Optional Header Len	The combined size of all optional headers present in this packet (in multiples of 4-byte words); can be shortened as *optlen*.
Packet Length	Length of the packet payload (excluding all headers).
Source Node ID	Source node ID.
Source Port ID	Source port ID.
Destination Node ID	Destination node ID.
Destination Port ID	Destination port ID.

4.2 Control flags

Control flags (Figure 4-2) can indicate various statuses of the packet. A router should ignore flags in the control flag that it does not recognize.

Figure 4-2 Control flags

Control flag	Description
Confirm RX	Request confirmation to continue transmission; see Chapter 9 for information on flow control.

4.3 Optional header

If the Optional Header Len field (fourth field in Figure 4-1) is greater than 0, the packet contains one or more optional headers (shown in Figure 4-3) immediately following the packet header. The first two bytes of all optional headers must contain the optional length and optional header type as described in Figure 4-3. This allows routers to skip optional headers (specified by the Optional Type) that they do not support and continue processing the next optional header, if present in the packet. Optional headers are always sized to be multiple of 4 bytes, and hence the length is also specified in number of words.

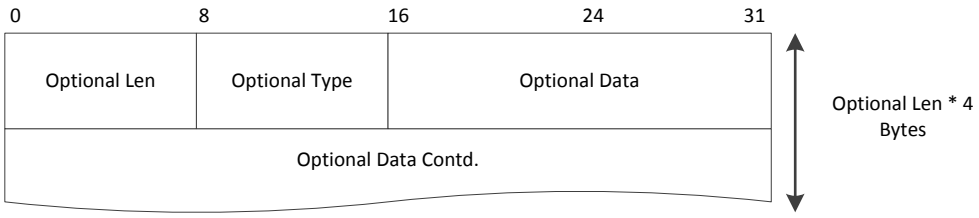


Figure 4-3 Optional Header format

Optional Header format	Description
Optional Len	Length (in multiples of 32-bit words) of the optional header.
Optional Type	Type of the optional header. The router shall ignore and discard the optional header if the type is unknown.
Optional Data	For use by the defined optional header.

4.4 Packet types

Based on the Packet Type field in the packet header, a packet can be one of eight types. A router shall drop a packet if it is not one of the implemented message types.

Packet type	Description
1	DATA – Normal data packet (Section 8.1).
2	HELLO – Link initialization control packet (Section 5.1).
3	BYE – Link termination control packet (Section 5.2).
4	NEW_SERVER – Control packet for new name association (Section 6.1).
5	REMOVE_SERVER – Control packet for new name disassociation (Section 6.2).
6	REMOVE_CLIENT – Endpoint termination control packet (Chapter 7).
7	RESUME_TX – Flow control acknowledgment control packet (Chapter 9).
8	MCAST – Multicast data packet (optional, Section 8.2).

5 Link management

A link is formally defined as a transport that provides a direct path between two neighboring nodes. The protocol allows for dynamic neighbor discovery, and each node shall not implicitly assume that a link to a neighbor is active. The next neighbor management is accomplished by the HELLO and BYE control packets.

5.1 Link initialization

Upon initialization, each node shall send a HELLO packet (Figure 5-1) over each link to the remote router with its node ID in the source node ID field of the routing header. The destination address in the routing header and the port field of the source address should be ignored by the receiver. Upon receiving a HELLO packet, a router shall reply with a list of all the servers in its database (see Section 6.1). A link shall be considered active if, and only if, a HELLO has been sent and received over the link. No packet shall be transmitted over the link until the exchange of HELLO has occurred. If a packet other than the HELLO is received over an inactive link, it should be ignored. Any padding or extra bytes in the HELLO packet should be ignored.

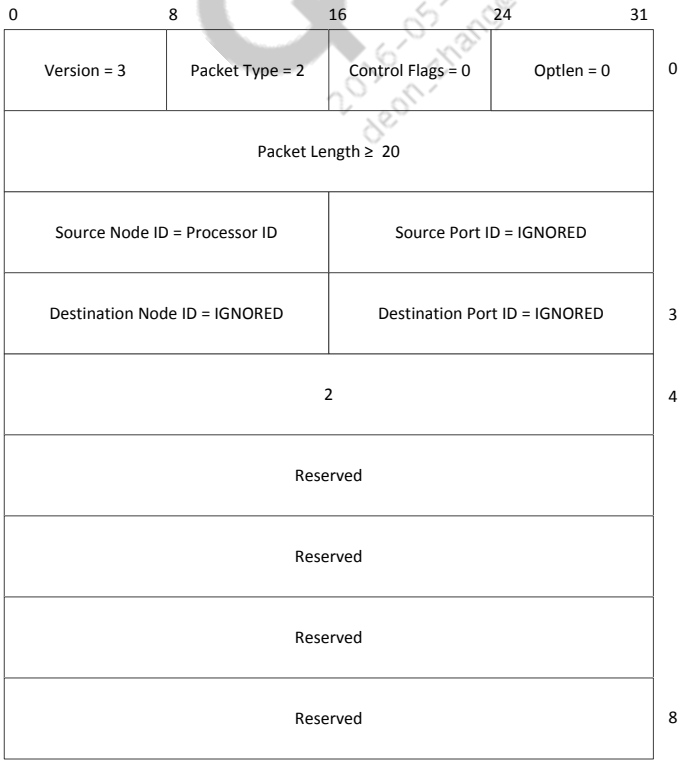


Figure 5-1 Basic HELLO packet

5.1.1 Extended link initialization with version negotiation

The HELLO packet has been extended (as shown in Figure 5-2) to include version and capabilities negotiation. The extended HELLO packet is fully backwards compatible with the one described in Figure 5-1.

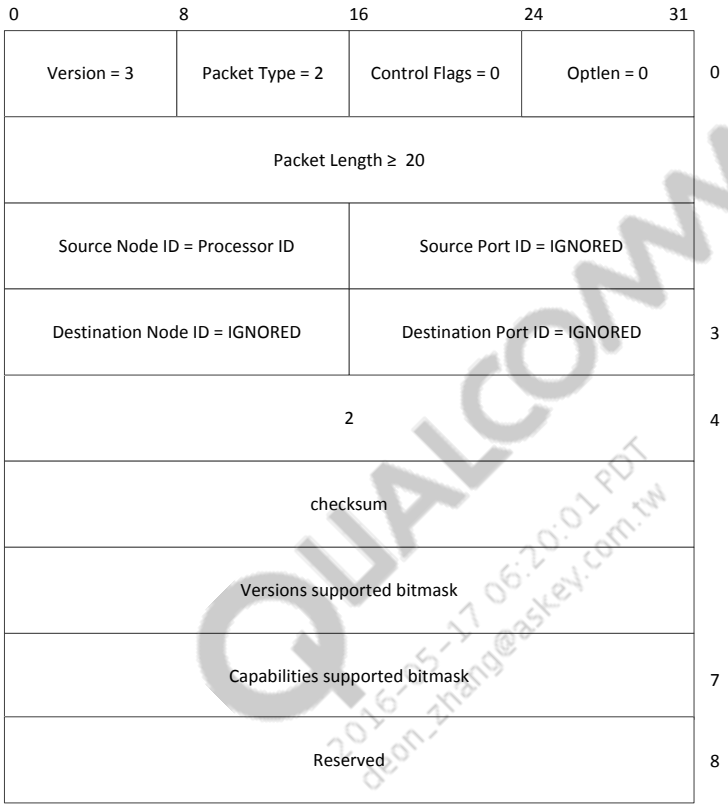


Figure 5-2 Extended HELLO packet

Field Name	Description
checksum	Checksum of the packet payload as described in Section 5.1.1.3.
Versions supported bitmask	Bitmask of all the versions supported by the source router.
Capabilities supported bitmask	Bitmask of all the capabilities supported by the source router.

5.1.1.1 Version negotiation

The versions supported bitmask is shown in Figure 5-3. The first and last bits are reserved for future expansion. Bit 3 is expected to be set for the current version described in this document. If router A supports a set of versions {X} and receives an extended HELLO from router B with the version supported bit field {Y}, the link is automatically upgraded to version MAX(X∩Y) and all future messages use the upgraded header of version MAX(X∩Y).

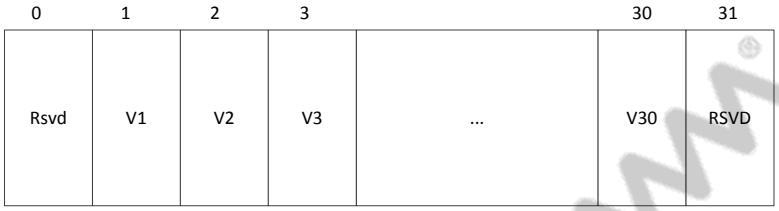


Figure 5-3 Version supported bitmask

5.1.1.2 Capabilities supported

This field is currently reserved for future use and must be set to 0 by the sender and ignored by the receiver.

5.1.1.3 Checksum

A checksum of the control message body (excluding the 16-byte router header) is added to the payload of the extended HELLO message to determine if the remote router actually intended to initiate version negotiation.

5.1.1.3.1 Computation

Computation occurs as follows:

1. The sum of all 16-bit nibbles of the control message body is computed.
2. The upper and lower nibbles are added until a sum lesser than or equal to 0xFFFF is achieved.
3. The lower 16 bits are inverted.

5.1.1.3.2 Sender

The sender sets the checksum field to the magic number 0xE110 and then computes the checksum, which replaces the checksum field.

5.1.1.3.3 Receiver

The receiver calculates the checksum of the control message body as described in Section 5.1.1.3.1. The extended fields (described in Sections 5.1.1.1 and 5.1.1.2) are considered if, and only if, the computed checksum results in the value 0xE110. Otherwise, the HELLO is considered as the basic version (described in Section 5.1) and version/capabilities negotiation is disabled.

5.1.1.4 Example link initialization sequences

The example in Figure 5-4 shows the sequence between three routers connected in a star configuration (A-B and B-C). A implements just the basic HELLO message, while B and C implement the extended HELLO message and also implement a new hypothetical protocol version 4.

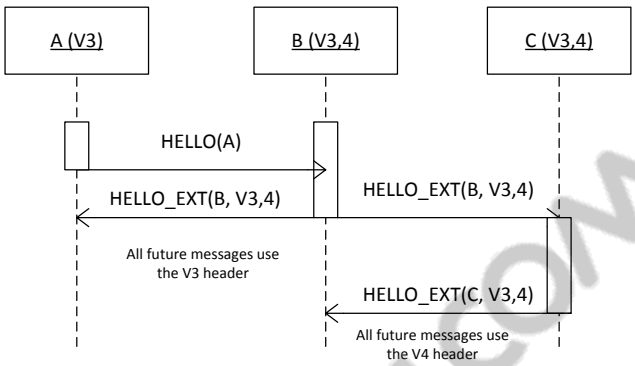


Figure 5-4 Version negotiation example

A ignores the additional fields in the extended HELLO message received from B and continues to use V3 as the router header for packets going toward B. Upon receiving the old HELLO message from A (checksum computation failure), B maintains the version 3 for all packets flowing toward A.

C receives the extended HELLO message from B and chooses 4, the maximum common version supported by both routers. All messages exchanged by B and C contain the hypothetical V4 header.

5.2 Link termination

If a node is required to terminate its relationship with a neighbor over a certain link, it can send a BYE control message. Nodes should ignore all traffic over the link (other than a HELLO message) and mark it inactive after sending or receiving a BYE message. The router should remove all endpoints and name associations known over that link and notify its local ports of such events. Any padding or extra bytes in the packet should be ignored. A BYE packet (Figure 5-5) shall never be forwarded to another node.

In the scenario where a link goes down, the router shall behave as if a BYE message has been received over that link.

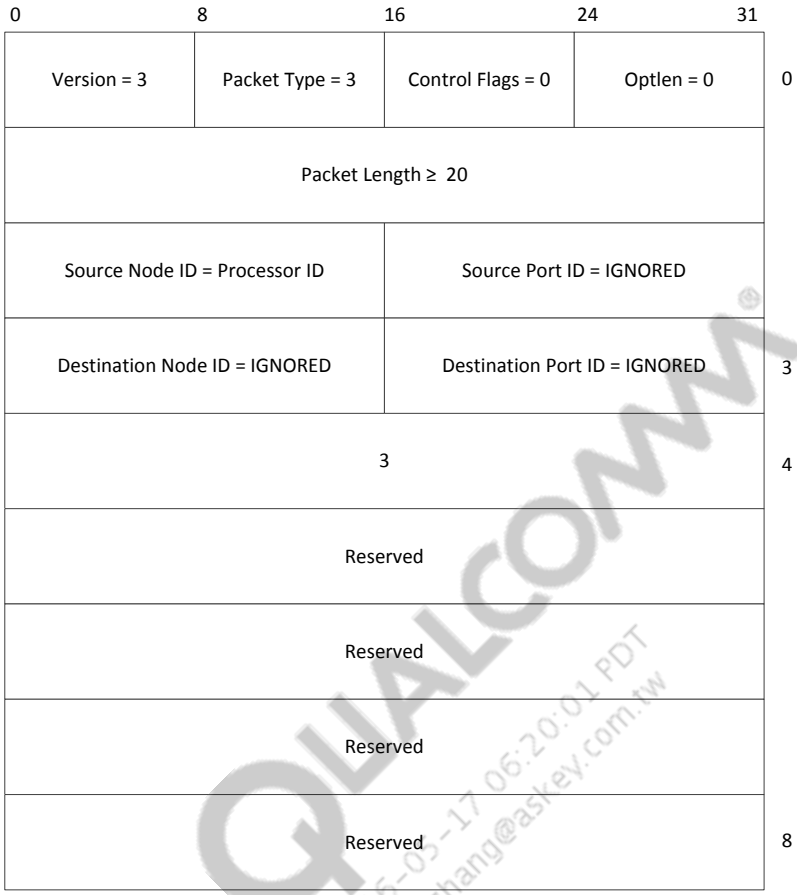


Figure 5-5 BYE packet

5.3 Master-slave configuration

When the underlying link has a master-slave relationship (for example, USB), the link of the node can be explicitly configured either as a master or a slave. A node acting as a master shall always be the first to transmit the HELLO packet. The slave node shall transmit the HELLO packet only as a response to the HELLO from the master.

Figure 5-6 shows three nodes connected in a star configuration. A and B, being the masters on all links connected to them, simultaneously send the HELLO message to their neighbors. Meanwhile, C sends the HELLO message only as a response to the HELLO message sent by B. Note that no ordering should be assumed in a master-master mode, while a slave-slave configuration results in a dead link.

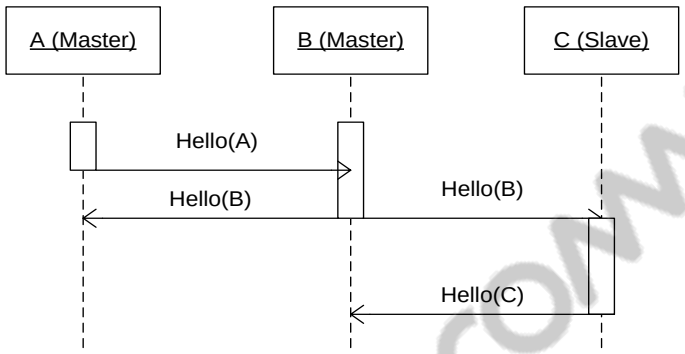


Figure 5-6 Link initialization between three connected nodes (A – B – C connection)

6 Name space management

The router shall provide a name service that functions in a fully distributed manner. Every node in an interconnected cluster knows every associated name <type, instance> present in the system and the address <node, port> of the servers through the NEW_SERVER message. The router shall also allow endpoints to disassociate themselves from the names through the REMOVE_SERVER message.

6.1 NEW_SERVER packet

When a local port associates itself with a *name* (REGISTER, which is described in Chapter 3), the router shall broadcast a NEW_SERVER packet (Figure 6-1).

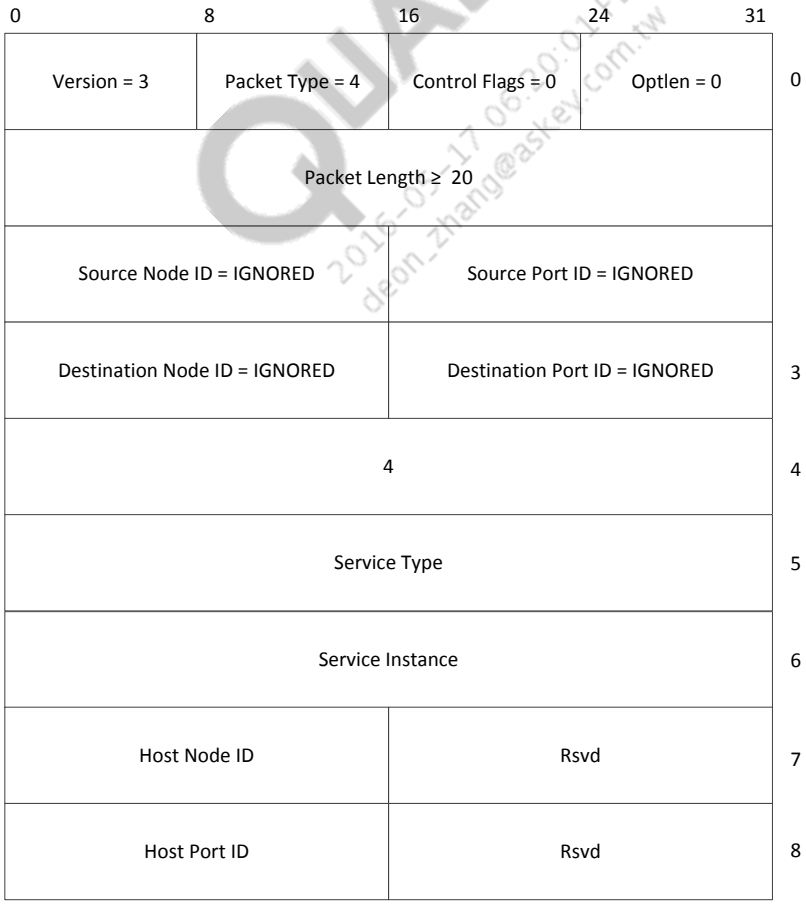


Figure 6-1 NEW_SERVER packet

The router header's destination address shall contain the broadcast port ID, 0xffff, and the recipient shall ignore the field.

Upon receiving a NEW_SERVER packet, the router should update its name space database that maps a service *name* to a physical *address*. If a service already has an entry for the same service type, the following rules apply:

- If the new service has a different address than the existing service of the same type, the router does the following:
  a. Append the new service to the list if it is not from the local node.
  b. Prepend the new service to the list if it is from the local node.
- If the new service has the same *address* and the same *instance ID* as the existing service, the control packet shall be ignored.

6.2 REMOVE_SERVER packet

If an endpoint disassociates itself from a previously associated name, it shall send a REMOVE_SERVER control packet (Figure 6-2) to all connected nodes. Upon receiving the packet, the router shall remove this relationship from its name space database and notify all local endpoints of this event.

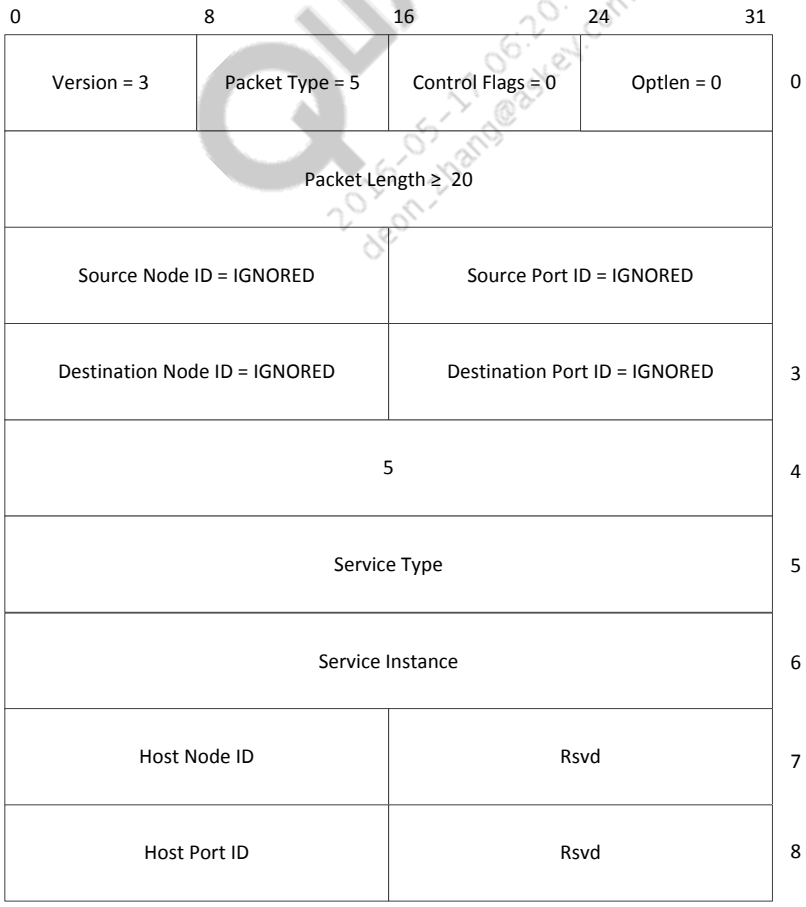


Figure 6-2 REMOVE_SERVER packet

7 Endpoint lifetime

An endpoint is considered terminated when one of the following events occurs:

- A CLOSE method (described in Chapter 3) is called on the endpoint.
- The process hosting the endpoint terminates.
- The endpoint is no longer reachable due to termination of the nodes in the forwarding path.

Upon termination of an endpoint, a REMOVE_CLIENT packet (Figure 7-1) must be broadcast by the node to all its neighbors. The router is also required to clean up all names associated with that address.

An optimization to conserve power is suggested in Appendix A that minimizes the number of nodes receiving the REMOVE_CLIENT packet.

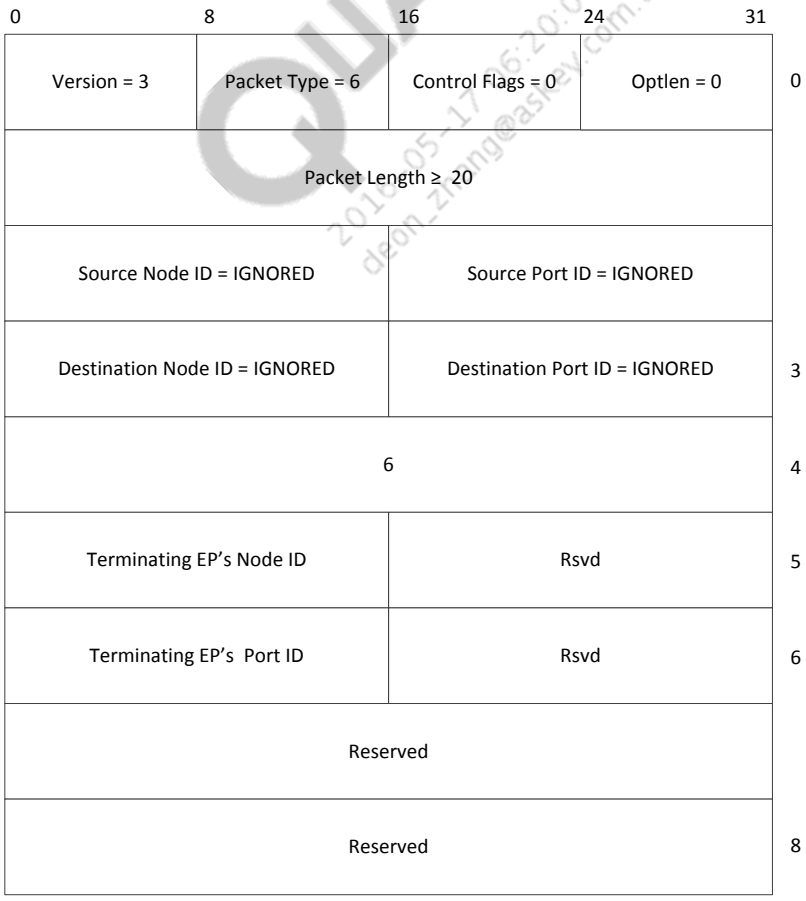


Figure 7-1 REMOVE_CLIENT packet

8 Data transmission

The router shall allow local endpoints to send messages to either a destination *address* or to a multicast group of endpoints associated with a specific *name*.

8.1 Point-to-point data transmission

A message sent by an endpoint to a specified remote endpoint (by destination *address*) is sent as a data packet (Figure 8-1). The packet payload must be word aligned. If the payload length is not a multiple of 4, the packet must be padded. The length field in the router header is *not* updated to contain the padding bytes.

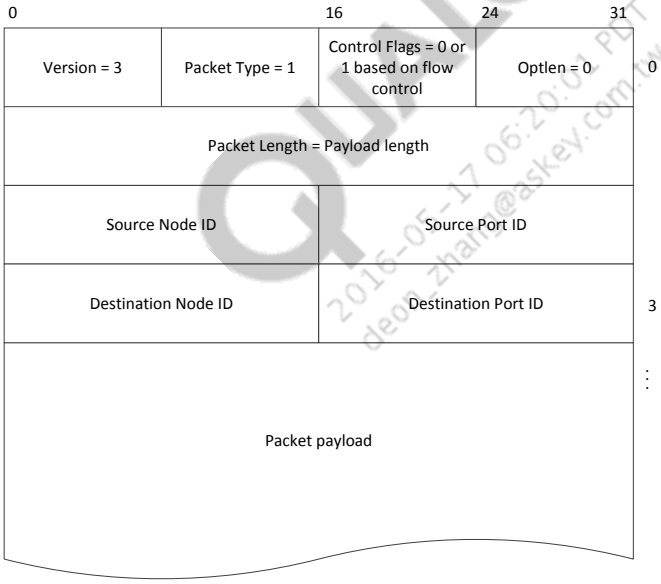


Figure 8-1 Data packet

8.2 Multicast data transmission

A router may support multicast messaging where a packet is destined to a specified *name* rather than a specified address. Routers that do not support multicast messaging should drop the packets. Multiple endpoints that have registered the same name receive the data packets.

Multicast data packets have message type 8, with the destination address field of the router header containing the name of the *service* <*type*, *instance*> in place of the *address* <*node*, *port*>. Flow control (described in Chapter 9) is not supported, so care must be taken to not flood the receivers.

Figure 8-2 shows the simple case where the *name* field fits in the destination address field. Note that for this to be possible, both the *type* and *instance* must fit in the required 16-bit fields. If such packing is not possible, the format shown in Figure 8-3 must be used. Note that the optional header is used with the optlen field in the router set to a value 3.

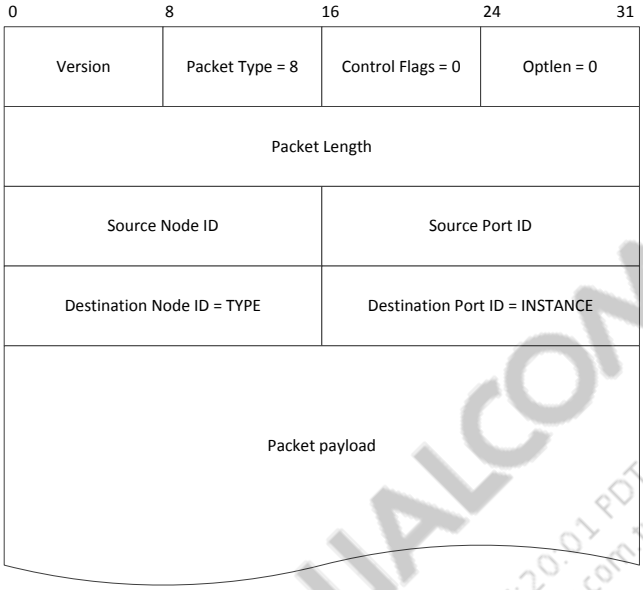


Figure 8-2 Multicast messaging with a 32-bit name (short form)

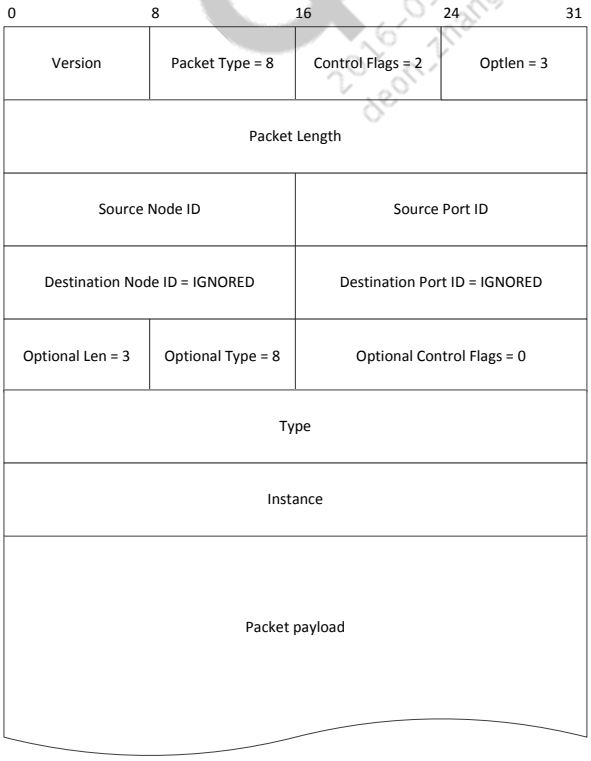


Figure 8-3 Multicast messaging with a 64-bit name (long form)

A router supporting multicast messaging must support reception of both the short and long form of multicast messages. Upon receiving a multicast data message, the packet shall be sent to all local endpoints that have registered the *name* <*type*, *instance*> provided in the destination field.

9 Flow control

To avoid flooding a destination address with packets at a rate faster than it can consume (and potentially exhaust memory at the destination), a simple flow control mechanism is implemented to limit the total packets in flight destined to a specified address.

Each transmitting router defines a *count* of the number of successive unconfirmed messages to a specific address (destination). Each router also defines a low and a high limit, referred to as L and H respectively. Upon *count* reaching L, the packet in discussion must have the CONFIRM_RX bit set in the router header's control flags (see Figure 4-2). When the *count* reaches H, the router should block the transmission and no more packets should be sent to the destination address until a RESUME_TX packet (Figure 9-2) is received with the destination address in question as the payload. Upon receiving the RESUME_TX packet, the *count* is cleared and the transmission must be retried. This process is shown in Figure 9-1.

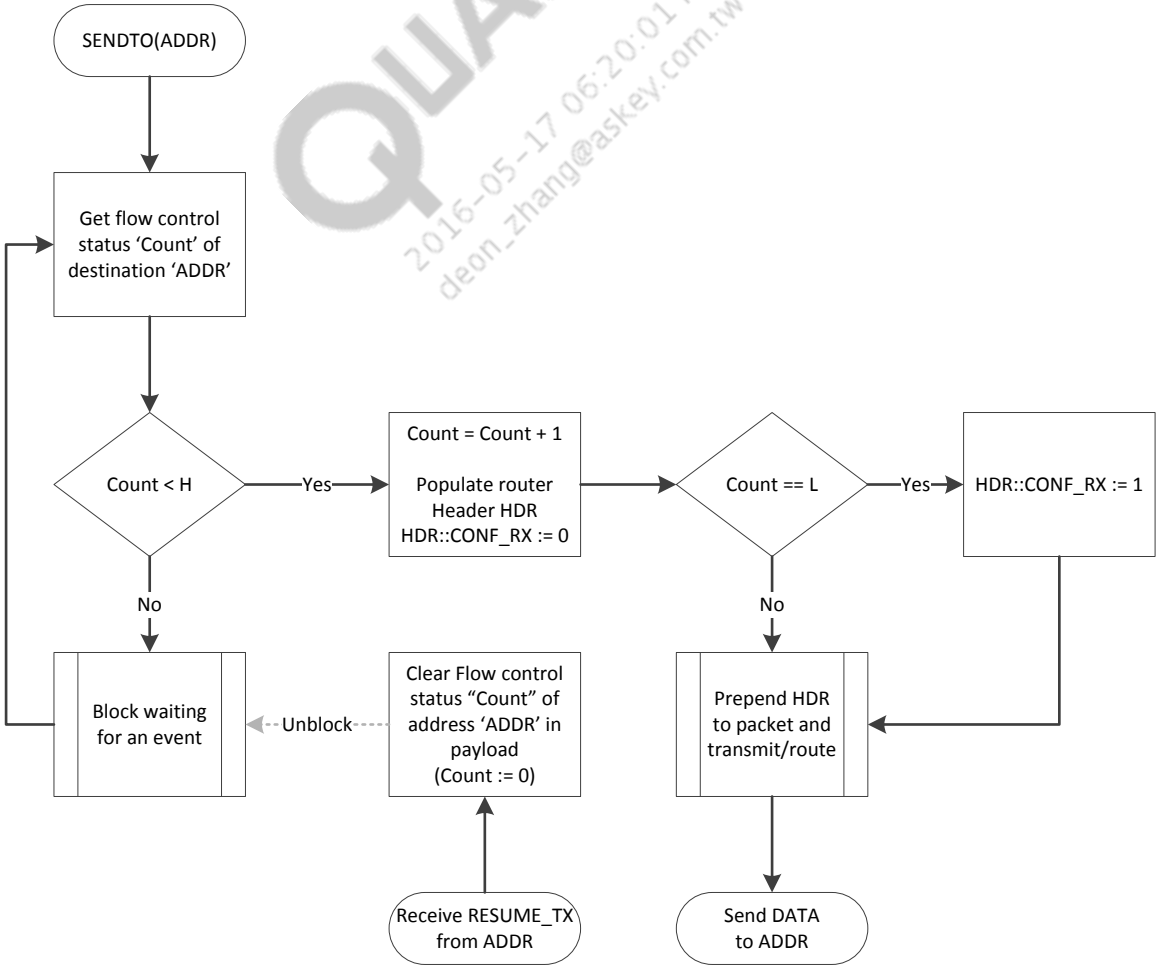


Figure 9-1 Flow control algorithm

On the receiver's end, upon consuming the packet with the CONFIRM_RX bit set in the router header's control flags field, the remote endpoint must reply with a RESUME_TX control packet with its *address* in the control message body (shown in the figure as *Flow controlled EP's Node/Port ID* address) to the source processor.

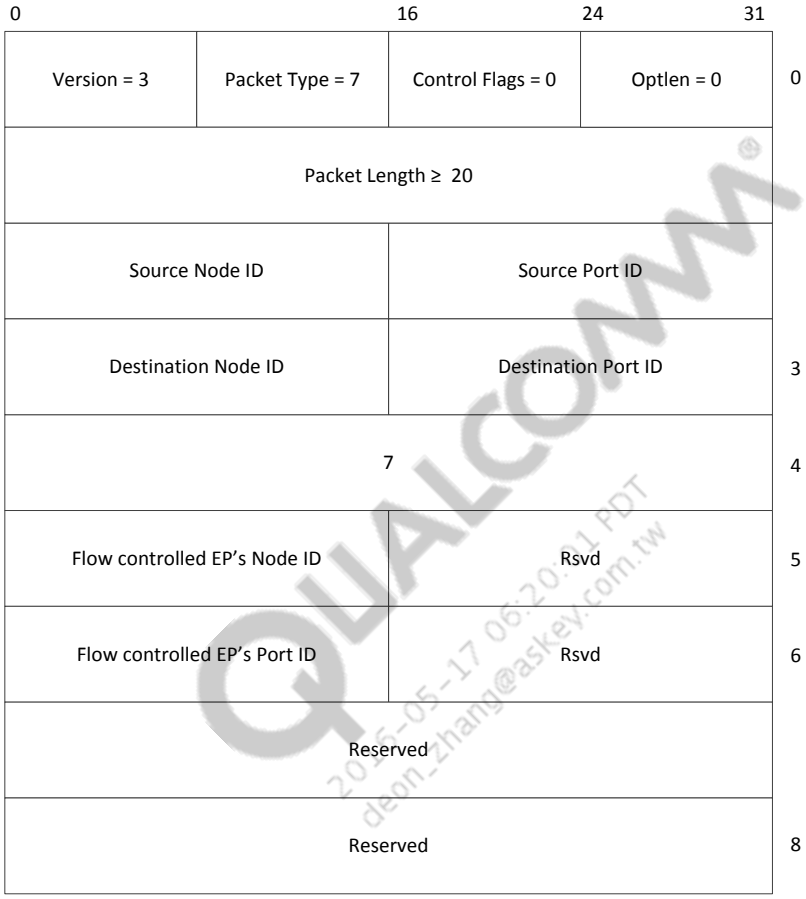


Figure 9-2 RESUME_TX packet

The recommended value for L and H is 5 and 10 respectively, but different values can be used. With the flow control algorithm shown in Figure 9-1, the number of packets in transit is guaranteed, at most, to be $2H - L$. Thus at any specified time, if there are an average N endpoints receiving data in a specific node with an average packet size S, the average memory utilization at the node is $N \times S \times (2H - L)$.

10 Network topology and multi-hop routing

10.1 Route discovery

Implementing the protocol verbatim allows for communication within nodes in a fully meshed network as shown in Figure 10-1.

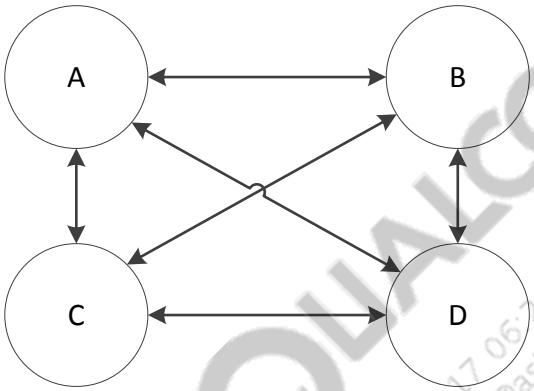


Figure 10-1 A fully-meshed network

Each of the nodes A, B, C, and D discover each other via the HELLO message (Section 5.1), and the names (<type, instance>) hosted on each node (Chapter 6) are distributed across the network via the broadcast NEW_SERVER messages.

Upon receipt, each router of a NEW_SERVER, DATA, or MCAST message associates the source node ID (or source address) with the link over which the message was received. Each node discovers the route to the destination nodes dynamically (and future messages destined to the specific node ID can be routed over the stored edge). Note that more than one node can be reachable via the same link; each router implementation must store all associations and make no assumptions otherwise.

10.2 Multi-hop routing (optional)

Even though preferred from a reliability and power standpoint, there are occasions when a fully meshed network is not viable from an implementation standpoint. In such cases, it might be necessary to extend one or more nodes to act as intermediaries. A simplified network requiring multi-hop routing is shown in Figure 10-2.

A router implementation that will never act as a forwarding node can skip this section.

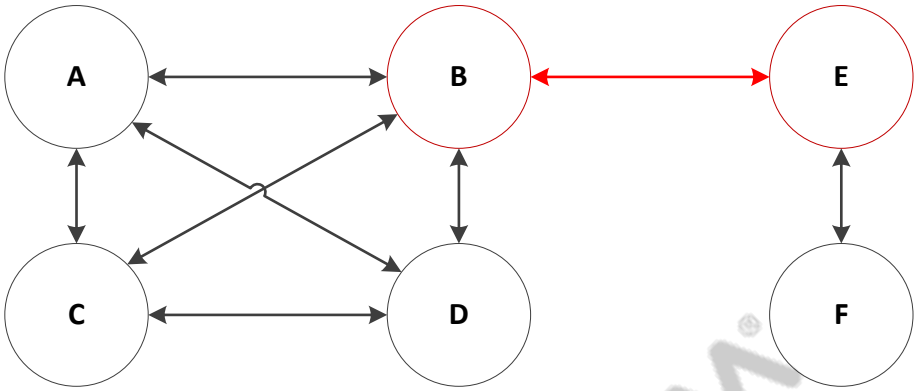


Figure 10-2 Partially-meshed network requiring multi-hop routing

In Figure 10-2 nodes B and E act as forwarding nodes bridging between nodes A, C, D, and F. The following subsections describe rebroadcast and forwarding rules to avoid routing loops.

10.2.1 Link coloring

Each node maintains a static identification for the link connecting them to their immediate neighbors. This is pictorially represented as the color of the links in Figure 10-2 and represented as COLOR(A, B) in subsequent sections. This coloring allows a node to distinguish between non-overlapping neighbors. For example, from the perspective of node B, the links (B, A), (B, C), and (B, D) are colored black while (B, E) is colored red. This allows node B to recognize that it is in the forwarding path between node E and the other three nodes (A, C, and D).

10.2.2 Broadcast rules

Name association changes (NEW_SERVER, REMOVE_SERVER, and REMOVE_CLIENT) within a node are broadcast to all immediate neighbors. For example, when an endpoint invokes the REGISTER method within node B (Figure 10-2), a NEW_SERVER is broadcast to all neighbors of node B. The same applies for methods UNREGISTER and CLOSE, which initiate the broadcast of REMOVE_SERVER and REMOVE_CLIENT messages, respectively.

Each router X, upon receiving a broadcast message (NEW_SERVER, REMOVE_SERVER, and REMOVE_CLIENT) via a link (X, Source), shall do the following:

1. Associate the host node ID with the link over which the message was received in case of a NEW_SERVER message.
2. Perform the required updates to the name server database, as follows:
   - NEW_SERVER – Add the name association to the name server database
   - REMOVE_SERVER – Remove the name association from the database
   - REMOVE_CLIENT – Remove the address from the routing table

   If the update fails, abort the processing and drop the broadcast message, thus mitigating routing loops which might occur when there are more than one route between two nodes. A node refrains from rebroadcasting a previously received broadcast message.

3. Notify all local endpoints of the name server update.
4. Rebroadcast message over all neighbors disjointed from the source. That is, the broadcast message is sent over all links (X, Dest) such that COLOR(X, Dest) ≠ COLOR(X, Source).

For example, if node B receives a NEW_SERVER message from node A, it updates the name and routing tables after which it should rebroadcast the NEW_SERVER message to node E (COLOR(B, A) ≠ COLOR(B, E)) while refraining from rebroadcasting it to nodes C and D (COLOR(B, A) = COLOR(B, {C, D})). Upon receiving this NEW_SERVER message, node E does the same and rebroadcasts the message to node F (COLOR(E, B) ≠ COLOR(E,F)).

Each router X upon completing link initialization (X, L) shall do the following:

1. Send a NEW_SERVER message for each name association local to node X.
2. Send a NEW_SERVER message for each name hosted by a remote node reachable via (X, Dest) such that COLOR(X, L) ≠ COLOR(X, Dest).

10.2.3 Forwarding rules

Each router X, upon receiving a non-broadcast message (DATA, RESUME_TX) via link (X, Source), shall do the following:

1. Associate the source node ID with the link over which the message was received.
2. If the destination node ID matches the local node ID, then post the message to the associated endpoint.
3. If the destination node ID differs from the local node ID, look up the link (X, Dest) in the routing table and forward over that link. The packet must be dropped if COLOR(X, Source) = COLOR(X, Dest).

10.2.4 Multicast rules

Each router X (if it supports multicast messaging), upon receiving a multicast message (MCAST Chapter 8) from link (X, Source) destined to a name <type, instance>, shall do the following:

1. Post the message to all local endpoints associated with the destination name.
2. Send the multicast message over all links (X, Dest) such that the following conditions are met:
   a. COLOR(X, Source) ≠ COLOR(X, Dest)
   b. At least one endpoint is known over the link associated with the destination name <type, instance>.

10.3 Risk of multi-hop routing

The risk in routing packets across an intermediate node is that a routing loop can occur because the protocol is not designed to handle redundant paths between nodes and doing so might violate packet ordering guarantees provided to the users. The neighboring node can also be powered down, causing reachability issues or extra power consumption if the node must be powered up for an incoming packet not destined for it. However, for a system to be scalable, multi-hop routing may be needed in some cases, but should be avoided if possible.

A REMOVE_CLIENT optimization

In most practical applications, an unnamed endpoint would be communicating with, at most, one remote endpoint. Thus, upon termination, only one router needs to be informed via the REMOVE_CLIENT message, instead of waking up every node in the network. An optimization is provided in this section that mitigates such a scenario.

Each endpoint known to a node is assigned a state: Null, Single, or Multi- mode as described below. Note that this optimization requires the creation of entries in the routing table for each local and remote endpoint address (<node, port>).

Null mode

All local endpoints are created in the Null mode. This mode implies that the endpoint is unknown to the rest of the system. When an endpoint in Null mode terminates, no REMOVE_CLIENT messages are sent and no users are notified of this event. Only local endpoints can be in the Null state and all invocations to OPEN create an endpoint in the Null mode.

Single mode

Endpoints that communicate with their peers via only one link are in the Single mode. Upon termination, the REMOVE_CLIENT message must be sent over that specific link. This is achieved by associating endpoints in Single mode with the link over which their peers can be reached. An endpoint in Null mode is transitioned to Single mode upon its first transmission or reception of a packet to or from another endpoint. Remote endpoints are always created in Single mode.

Multi-mode

Endpoints communicating with their peers reachable via two or more links are in Multi-mode. Instead of maintaining states for multiple links, the REMOVE_CLIENT message is broadcast across the network. Upon association with a name, endpoints are transitioned from Null or Single mode to Multi-mode.

Figure A-1 shows the transition between the states and the behavior when an endpoint is terminated in each state.

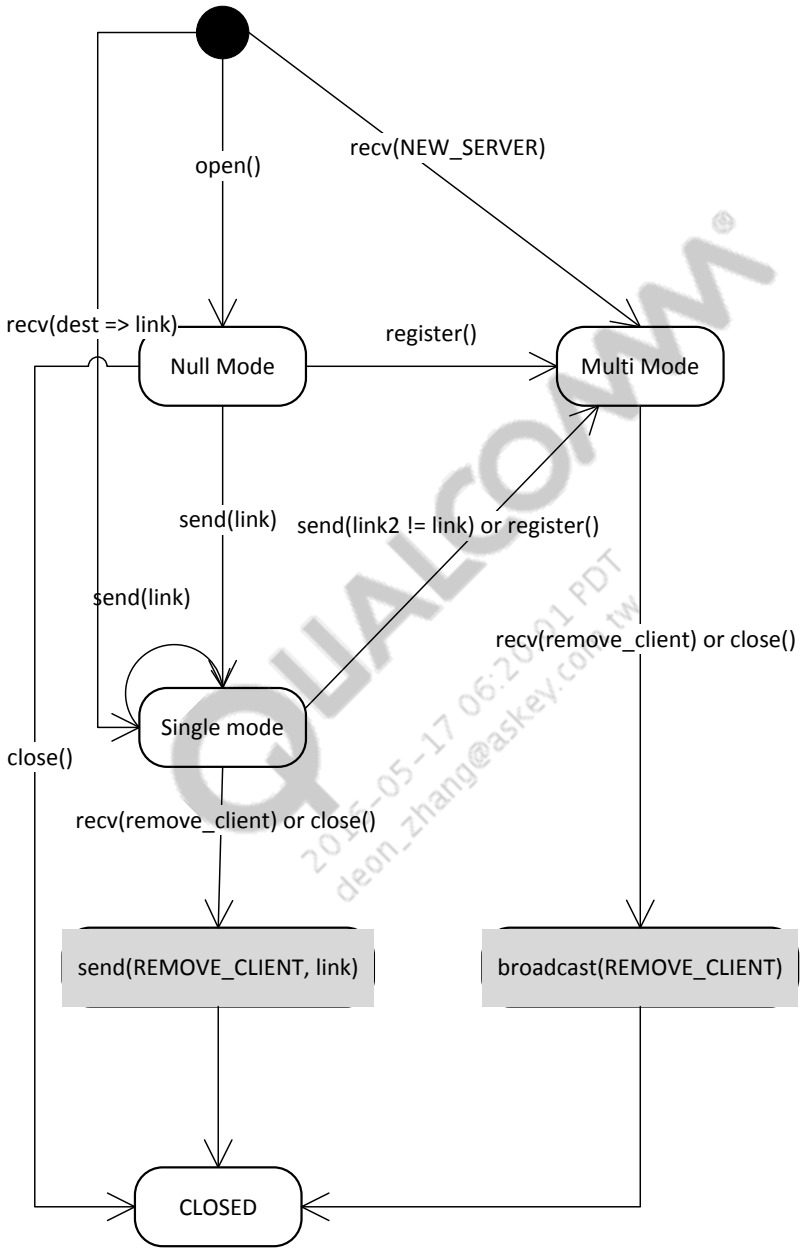


Figure A-1 Mode state transition diagram