3 QP: Message Transfer Mechanism

The Previous Chapter

The previous chapter introduced the concept of device attributes, managers, management agents (MAs), and management datagrams (MADs).

This Chapter

This chapter introduces the concept of the Queue Pair (QP), the message transfer engine that lies at the heart of the IBA technology. Request and response packets, Packet Sequence Numbers (PSNs), and the Verb Layer (a quasi-API used to control an IBA HCA) are introduced. The four IBA QP types are introduced and the concept of the QP Context and its contents are defined. Finally, there is a rather detailed example of a message transfer from one CA to another.

The Next Chapter

The next chapter provides an introduction to the four IBA transport service types (RC, UC, RD, and UD QPs), as well as the two non-IBA transport service types (Raw IPv6 and Raw EtherType QPs) that permit packets associated with virtually any other network protocol to be tunneled through an IBA network encapsulated in "raw" IBA packets.

Introduction

Figure 3-4 on page 58 and Figure 3-5 on page 59 should be viewed as two halves of one large illustration. This split illustration is referenced heavily in the following discussions. Note that many of the elements in the illustration are

tagged with reference letters for ease of reference in the discussions that follow. Also keep in mind that, while there are several types of QPs, the split figure and the examples in this section describe message passing as it occurs on one type of QP (referred to as the Reliable Connected, or RC, type).

A QP Is a Bi-Directional Message Transport Engine

QP Consists of Two Queues

Each CA implements a number of bi-directional message transport engines, each of which is referred to as a Queue Pair (QP). It is called a QP because it consists of a pair of queues:

- **Send Queue (SQ)**. See items J and R. Message transfer requests are posted to this queue by software. As each is executed, the SQ Logic transmits an outbound message transfer request to a remote QP's RQ Logic.
- **Receive Queue (RQ)**. See items H and S. Software posts Work Requests (WRs) to this queue to handle certain types of inbound message transfer requests transmitted to the RQ Logic by a remote QP's SQ Logic.

A CA Contains Multiple QPs

Although the illustration shows a single QP in each of the two CAs (H and J in one; R and S in the other), a CA will implement a minimum of several QPs, and as many as 16M (2^{24}) possible QPs, each of which is capable of sending messages to and receiving messages from one or more QPs in remote CAs.

Request and Response Packets

A QP's SQ Logic transmits a message transfer request to a remote QP's RQ Logic in a series of one or more request packets.

There are several different types of QPs. Depending on the type of QP, the remote QP's RQ Logic may respond to the receipt of a message transfer request by transmitting a series of one or more response packets back to the other QP's SQ Logic.

Packet Sequence Numbers

Each request packet generated by a QP's SQ Logic contains a PSN that identifies this request packet. Upon receipt of each request packet, a QP's RQ Logic verifies that the packet contains the next expected PSN (ePSN).

Each response packet generated by a QP's RQ Logic contains a PSN that associates it with a specific request packet that was received from the remote QP's SQ Logic. Upon receipt of each response packet, a QP's SQ Logic verifies that the packet's PSN is associated with a previously issued request.

Overview of QP Types

There are a number of different QP types that may be implemented in a CA. Because a QP is a message transport engine, the type of QP is referred to as the transport service type.

RC (Reliable Connected) QP

QP Setup. When it is set up by software, a RC QP is initialized with:

— The port number on the local CA through which it will send and receive all messages.
— The QP Number (QPN) that identifies the RC QP that it is married to in a remote CA.
— The port address of the remote CA port behind which the remote RC QP resides.

Private Comm Channel. The RC QP can only send messages to and receive messages from its companion RC QP in another CA.

Ack/Nak Protocol. The destination RC QP's RQ Logic is required to respond to each message request packet sent by the source QP's SQ Logic. The possible response types are:

— A positive Acknowledge (**Ack**) packet is returned to signal the successful receipt and processing of a Send or RDMA Write Request packet.
— A Negative Acknowledge (**Nak**) packet is returned to signal one of the following:
  – A temporary Receiver Not Ready (**RNR Nak**) condition. Upon receipt of this type of Nak, the sending QP's SQ Logic may retry the

transmission of the affected request packet repeatedly either until the request packet is successfully accepted and processed or until the sender's Retry Count is exhausted. An error is reported to the sender's software if the count is exhausted.
  - A **PSN Sequence Error Nak** is returned if one or more missing packets are detected. Upon receipt of this type of Nak, the sending QP's SQ Logic rewinds its SQ Logic to the first missing packet and re-transmits from that point forward. It will repeatedly rewind and resend until either the missing request packet is received by the destination QP's RQ Logic or until its retry count is exhausted. An error is reported to the sender's software if the count is exhausted.
  - A **fatal Nak error code** is returned if the destination QP's RQ Logic detects a fatal error either in a request packet or upon attempting to execute it. The operation is not retried and an error is reported to the sender's software.
- An **RDMA Read Response packet** is the appropriate response to an RDMA Read Request. The requested read data is returned in a series of one or more RDMA Read Response packets.
- An **Atomic Response packet** is the appropriate response to an Atomic Request packet. The requested read data is returned in a single Atomic response packet.

This service is referred to as "reliable" for the following reasons:

- The destination QP's RQ Logic verifies the PSN in each request packet to ensure that all of the message's request packets are received in order, that none are missing, and, if any duplicate request packets are received, that they are only processed once. There is one exception; upon receipt of a duplicate memory read, the data is read from memory again.
- The sending QP's SQ Logic verifies that the appropriate response is received for each request packet issued. In addition, upon receipt of either an RNR Nak or a PSN Sequence Error Nak, the SQ Logic automatically attempts recovery without involving software.

Bandwidth Use. Due to the generation of Acks and Naks, the RC protocol consumes a substantial amount of IBA bandwidth.

Message Length. Each message transferred can contain from 0 to 2GB of data.

RC CA Support Requirements. HCAs are required to support the RC QP type. It is optional whether or not a TCA implements the RC QP type.

UC (Unreliable Connected) QP

QP Setup. When it is set up by software, an UC QP is initialized with:

— The port number on the local CA through which it will send and receive all messages.
— The QP Number (QPN) that identifies the UC QP that it is married to in a remote CA.
— The port address of the remote CA port behind which the remote UC QP resides.

Private Comm Channel. The UC QP can only send messages to and receive messages from its companion UC QP in another CA.

No Ack/Nak Protocol. Unlike the RC service type, the remote QP's RQ Logic does not respond to each request packet with an Ack or Nak. The message sender therefore has no guarantee that all request packets of a message have been received correctly by the target QP's RQ Logic (hence, the term "unreliable").

Bandwidth Use. Since the target QP's RQ Logic does not generate Acks and Naks, the UC protocol consumes significantly less IBA bandwidth than the RC protocol.

Message Length. Each message transferred can contain from 0 to 2GB of data.

UC CA Support Requirements. HCAs are required to support the UC QP type. It is optional whether or not a TCA implements the UC QP type.

RD (Reliable Datagram) QP

General. A RD QP can send messages to and receive messages from any number of RD QPs located in one or more other CAs. It does so through one or more "pipelines" that are established between the local CA and one or more remote CA(s). Each "pipeline" is referred to as a Reliable Datagram Channel (RDC) and acts as the conduit through which multiple local client RD QPs send messages to and receive messages from RD QPs residing in the remote CA. The QP sends the request packets that comprise a message transfer request to an RDC in its local CA. That RDC is programmed to send and receive all packets through a specific local CA port. The RDC is also programmed with the address of a port on the remote CA behind which the other end of the RDC resides, as well as the address of the other end of that RDC.

Multiple-Destination Comm Channel. The RD QP can send messages to and receive messages from multiple RD QPs residing in other CAs.

Ack/Nak Protocol. The RD QP transport service type uses the same Ack/Nak protocol as the RC type. It is therefore referred to as "reliable."

Bandwidth Use. Due to the generation of Acks and Naks, the RD protocol consumes a substantial amount of IBA bandwidth.

Message Length. Each message transferred can contain from 0 to 2GB of data.

RD CA Support Requirements. It is optional whether or not HCAs and TCAs implement the RD QP type.

UD (Unreliable Datagram) QP

General. An UD QP can send messages to and receive messages from any number of UD QPs located in one or more other CAs. Unlike RD, no Ack or Nak is returned for each request packet received (hence, the term "unreliable").

QP Bound To a Local CA Port. When it is set up by software, an UD QP is initialized with the port number on the local CA through which it will send and receive messages. *It can therefore only send and receive messages with remote UD QPs that are reachable through that port.*

Multiple-Destination Comm Channel. The UD QP can send messages to and receive messages from multiple UD QPs residing in other CAs.

No Ack/Nak Protocol. Unlike the RC and RD service types, the remote QP's RQ Logic does not respond to each request packet with an Ack or Nak. The message sender therefore has no guarantee that a request packet has been received correctly by the target QP's RQ Logic (hence, the term "unreliable").

Bandwidth Use. Since the target QP's RQ Logic does not generate Acks and Naks, the UD protocol consumes significantly less IBA bandwidth than the RC and RD protocols.

Message Length. *Each message transferred must fit in the data payload field of a single packet.*

UD CA Support Requirements. HCAs are required to support the UD QP type. This includes general purpose UD QPs as well as the special UD QPs (QP0 and QP1) that are used to send and receive MADs. TCAs are required to support the special UD QPs (QP0 and QP1) that are used to send and receive MADs, but it is optional whether or not they support general-purpose UD QPs.

Raw QP

A Raw QP is used to send and receive message packets for a protocol other than IBA (e.g., IPv6 or Ethernet packets). The non-IBA packet encapsulates the non-IBA message within itself and permits the non-IBA message to be transported through the IBA fabric to its destination. Switches and routers are responsible for routing the encapsulated non-IBA packet to its ultimate destination.

How Many CA Ports Can a QP Be Associated With?

When initially set up by software, each QP is associated with only one local CA port. This QP/port association is established when software creates the QP.

How Many QPs Can Be Associated With One Port?

Multiple QPs and/or EECs (see "Reliable Datagram (RD) Service Type" on page 67) can be associated with each local CA port.

QP's SQ Logic Responsibilities

A QP's SQ Logic (item L or P) has the following basic responsibilities:

- It processes message transfer requests (referred to as Work Requests, or WRs) that were posted to the SQ by software. Once a WR is posted to the SQ, it is referred to as a Work Queue Entry (WQE, or "wookie"). The WQEs are processed by the QP's SQ Logic one-at-a-time in the same order that they were posted.
- It handles the transmission of message transfer request packets and, if the request packet recipient is expected to return Acknowledge packets (RC service type), the SQ Logic waits for the Ack packets and, upon receipt, verifies that they are in the correct order and do not indicate any errors.

- It handles inbound RDMA Read response packets returning previously requested read data. The data is written to the CA's local memory using the address pointer contained in the SQ WQE specifying the RDMA Read message transfer.
- It handles an inbound data item being returned in response to a previously issued Atomic RMW request. The data is written to the CA's local memory using the address pointer contained in the SQ WQE specifying the Atomic RMW message transfer.

QP's RQ Logic Responsibilities

A QP's RQ Logic (item K or Q) awaits incoming message transfer request packets sent by the remote QP's SQ Logic. A QP's RQ Logic has the following basic responsibilities:

- Upon receipt of a request packet, the RQ Logic verifies that the PSN in the packet is the next expected PSN (ePSN).
- Upon receipt of a request packet, if the request packet recipient is expected to return Acknowledge packets (RC service type), the RQ Logic sends an Ack packet back to the remote QP's SQ Logic. The Ack packet's PSN is the same as the PSN in the request packet being acknowledged.
- If the request packet is a Send packet, the request packet's data payload is written to the CA's local memory (by the QP's RQ Logic; see item C or W) using the Scatter Buffer List specified in the RQ WQE currently at the top of the RQ (item H or S). Upon receipt of the last or only packet of a Send, the RQ Logic retires the top entry from the RQ. In addition, it posts a Completion Queue Entry (CQE, or cookie) to the Completion Queue (CQ; see item F or T) associated with the QP's RQ and, if the optional 32-bit immediate data value (more on this later) is present in the last or only request packet of the Send, the RQ Logic stores the immediate data value in the newly posted CQE.
- If the request packet is an RDMA Write request packet, the request packet's data payload is written to the CA's local memory (by the QP's RQ Logic) using the address pointer supplied in the first request packet of that RDMA Write message transfer.
- If the request packet is the last or only request packet of an RDMA Write message transfer, and, if the packet contains the optional 32-bit immediate data item, the RQ Logic retires the top entry from the RQ. In addition, it posts a Completion Queue Entry (CQE, or cookie) to the Completion Queue (CQ; see item F or T) associated with the QP's RQ and stores the immediate data value in the newly posted CQE.

- If the request packet is an RDMA Read Request packet, the QP's RQ Logic reads the requested data from the area of its local memory indicated by the start memory address supplied in the RDMA Read request packet (the request packet also specifies how much data is to be read). The requested read data is supplied back to the remote QP's SQ Logic in a series of one or more RDMA Read response packets.
- If the request packet is an atomic RMW request, the RQ Logic performs the operation on the local memory location specified in the packet and returns the data read from the location to the SQ Logic in the remote QP.

Verb Layer Is an OS-Independent API

Software applications do not interact with the HCA interface directly. Rather, the software application (item B) makes calls to a software entity referred to as the Verb Layer (item D). Think of the Verb Layer as an API loosely defined by the specification and implemented by each OS vendor in a vendor-specific manner. For each verb (think of it as a function call), the specification defines:

- Its input parameters.
- Its return, or output, parameters.
- The actions taken by the verb.

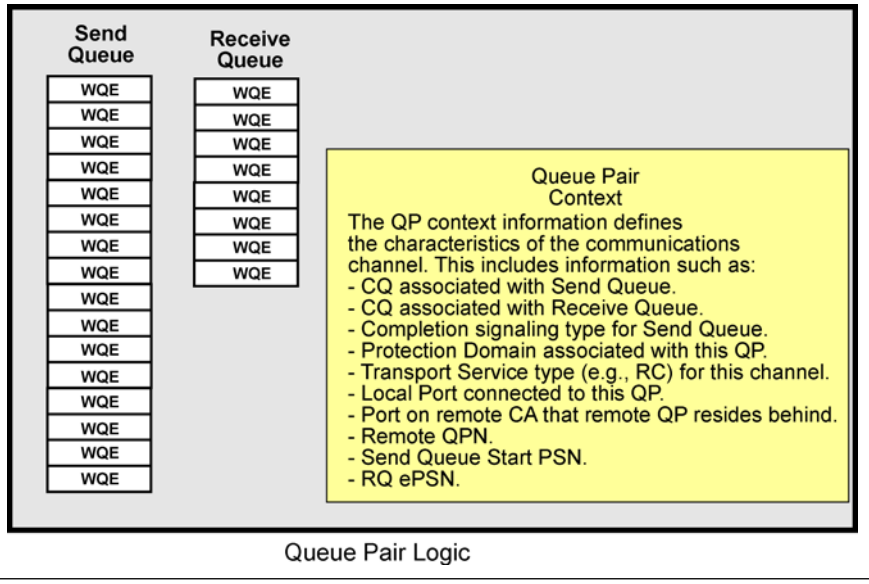
The Verb Layer, in turn, can access the HCA hardware interface (i.e., its register set; item E) to accomplish the desired action within the HCA. In addition, the Verb Layer also:

- makes calls to the OS (item A) when required (e.g., it can call the OS memory management facility to have it allocate pages of physical main memory for a verb's use or for the HCA's use).
- accesses main memory if it must do so to accomplish an action requested via a verb call. As an example, in a particular OS environment, a software application may build a message transfer Work Request (WR) in main memory and then execute a *Post Send Request* verb call to post the WR to a QP's SQ for processing. It would supply the start address of the WR in main memory as one of the input parameters to the *Post Send Request* verb. The verb would then access main memory to read the WR, which it then posts to the targeted QP's SQ.

QP Context Defines QP's Operational Characteristics

Refer to Figure 3-1 on this page. Before a QP can be used to send and/or receive messages with a QP of the same type in another CA, software creates the QP and supplies it with the basic operational characteristics that it will use to send and receive messages with its companion QP in the remote CA. In this example, the QP Context of each of the two companion QPs have been programmed (when the QP was setup by software) with the information described in the sub-sections that follow. *The QP Context items described in the following subsections assume that the QP type is Reliable Connected (RC).*

Figure 3-1: QP Context



Local Port Number

The local CA port number through which the QP will send and receive message packets is programmed into the QP's context during QP setup.

QP Type

The type of QP must be specified when the QP is created. There are five types:

- **RC**. Reliable Connected.
- **UC**. Unreliable Connected.
- **RD**. Reliable Datagram.
- **UD**. Unreliable Datagram.
- **Raw**. Used to send and receive non-IBA packets that are encapsulated within an IBA packet.

SQ Start PSN (Packet Sequence Number)

When the SQ Logic starts transmitting request packets, this is the PSN that will be inserted in the first packet generated. This start number is stored in the QP Context. The SQ's current PSN (cPSN; see items L and P) is the PSN that will be inserted in the current request packet to be transmitted. Initially this is set equal to the start PSN and is then updated as each request packet is issued by the SQ Logic.

RQ Logic's Expected PSN (ePSN)

See items K and Q. When using the RC protocol, upon receiving a request packet from the SQ Logic in the remote CA, the RQ Logic in the target QP is required to validate that the packet's PSN is the next expected PSN.

- If not, then one or more request packets may have been lost somewhere along the flight path (e.g., a switch or a router may have dropped the packet due to a CRC error) and a PSN Sequence Error Nak is returned.
- If the packet's PSN is correct (i.e., it equals the RQ Logic's ePSN), then the RQ Logic returns an Ack (Acknowledge) back to the sender's SQ Logic.

It should be obvious that the RQ Logic in each QP must be initialized with the start PSN that will be used by the SQ Logic in the other QP.

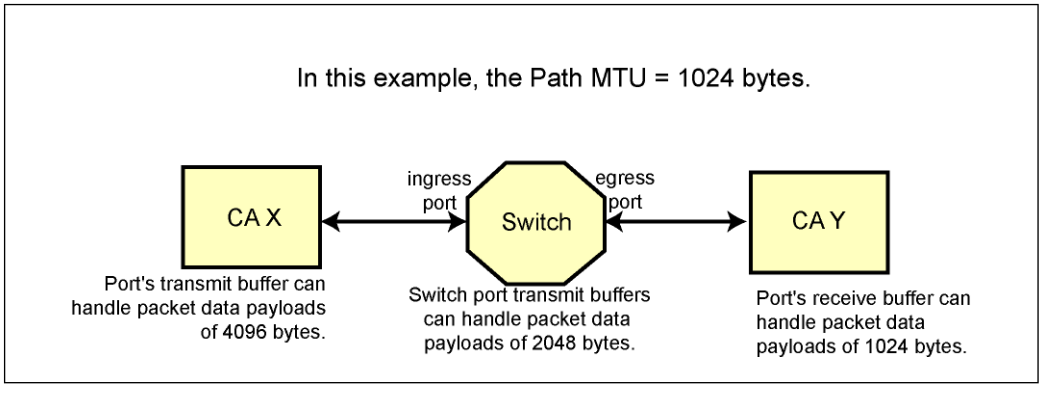
Maximum Data Payload Size

Refer to Figure 3-2 on this page. As a packet traverses the link(s) between the source and destination CA ports, it is sent from a transmit buffer in one port to the receive buffer in the port on the other end of that link. Although the maximum allowable size of a packet's data payload field is 4KB, one or more of the transmit and/or receive buffers along the path between the source and destination ports may not be able to handle packets with 4KB data payloads. The buffers implemented in a port may be limited to a maximum packet data payload size of:

- 256 bytes.
- 512 bytes.
- 1024 bytes.
- 2048 bytes.
- 4096 bytes.

During configuration, the configuration software (i.e., the SM) discovers whether this limitation exists. When a pair of QPs are subsequently created in the two CAs, the QP Context of each is initialized with the maximum permissible data payload size that can be used in packets. This is referred to as the Path Maximum Transfer Unit, or PMTU. Additional information can be found in "Packet Length Field (PktLen)" on page 614.

Figure 3-2: Example Path Maximum Transfer Unit Size



Destination Local ID (DLID) Address

This is the destination Local ID address of the port on the other CA behind which the remote QP resides. In the illustration, the QP Context of the QP in HCA "X" (item M) is initialized with the LID address assigned to the port on HCA "Y" behind which its companion QP resides. Likewise, the QP Context of the QP in HCA "Y" (item N) is initialized with the LID address assigned to the port on HCA "X" behind which its companion QP resides. The DLID address stored in a QP's Context is inserted in each request packet generated by the QP's SQ Logic.

Desired Local Quality of Service

In order for a request packet to get from the QP SQ that generates it to the target QP's RQ Logic, it may have to traverse a series of links interconnected by switches. Some applications may require that messages be transferred through the network as quickly as possible in order to achieve performance adequate to the task at hand. In other words, some applications require a high QoS (Quality of Service).

On the other hand, another application may not require that the network expedite transmission of its messages through the fabric to their destination. Said another way, the application doesn't require a high QoS.

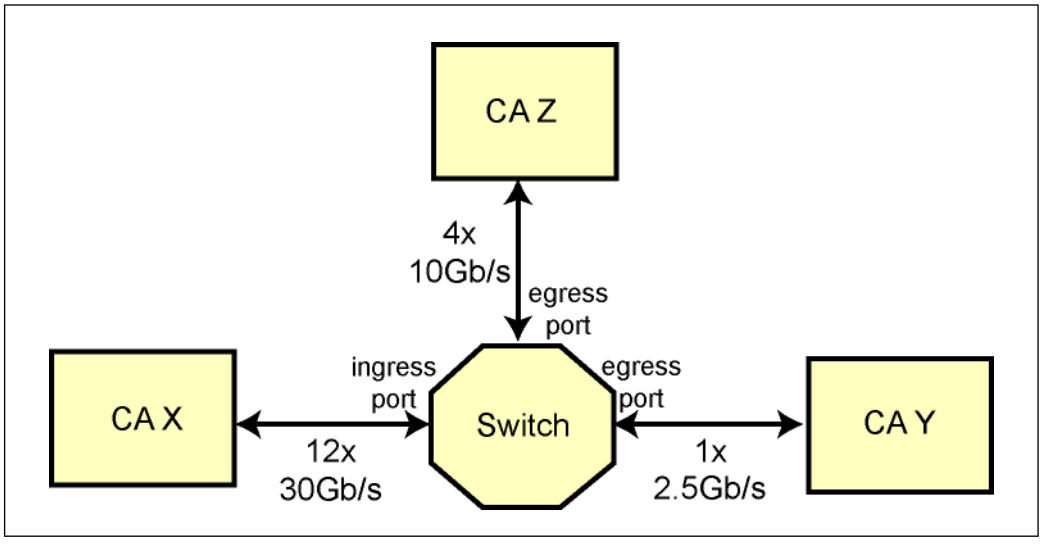
When a QP is initially set up by software, the programmer indicates the desired QoS by specifying a desired Service Level (SL). This 4-bit value is placed in each packet generated by the QP. As will be seen later, the packet's SL value determines how quickly the packet will begin transmission from the source port. Likewise, if the packet arrives at a switch, the switch looks at the packet's SL value to determine how quickly the packet must be forwarded out through one of the switch's egress ports.

Packet Injection Delay

Packet injection delay is also referred to as the Inter-Packet Delay, or IPD, and as the Maximum Static Rate. Refer to Figure 3-3 on page 44. Some links implement only one transmit/receive signal pair, while others may implement four or twelve signal pairs. It should be obvious that a wider link can receive data significantly faster than a thinner link. The path between the source and destina-

tion CA ports may encompass a number of links and multiple switches. The links are not necessarily the same width. To prevent faster links from overrunning slower links with traffic, a QP is supplied with an IPD that defines the interval that must be observed between sending packets to the destination QP.

Figure 3-3: Packet Injection Delay Example



Ack Receipt Timeout (Local Ack Timeout)

The Local Ack Timeout value assigned to a QP when it is set up defines the amount of time the QP's SQ Logic will wait for an Ack packet before it retries the transmission of the corresponding request packet.

Ack Timeout and Missing Packet(s) Retry Counter

The official name of this item is the Retry Count and it defines the number of times the QP's SQ Logic will retry the transmission of a request packet due to any of the following:

- a timeout while awaiting an Ack packet,
- the receipt of an RDMA Read response packet with a PSN higher than the expected PSN. This indicates that one or more RDMA read response packets were lost in the fabric.

- the receipt of a response packet with a PSN higher than that of an expected Atomic response packet. This indicates that an Atomic response packet was lost in the fabric.
- the receipt of a Nak packet from the remote QP's RQ indicating that it detected a missing packet (i.e., a request packet was received with a PSN > the ePSN).

Receiver Not Ready (RNR) Retry Count

This value is supplied by the remote QP's RQ Logic when the two QPs are first set up. It defines how many times this QP's SQ Logic should retry the transmission of a request packet that receives an RNR Nak from the remote QP's RQ Logic. The remote QP's RQ Logic will respond to a request packet with an RNR Nak if it is temporarily unable to handle that request. A classic example would be if a request packet is received and there are currently no WQEs posted to the receiver's RQ to handle the inbound request.

Source Port's LID Address

At a minimum, the configuration software (i.e., the SM) will assign a single Local ID (LID) address to a port. This is referred to as its base LID address. The configuration software may optionally assign a range of LID addresses to a port by assigning its base LID address and also telling it how many LIDs starting at the base LID are assigned to it.

When a QP's SQ Logic sends a request packet to its assigned port for transmission, it must indicate to the port which of the port's assigned LID addresses to insert in the packet's Source LID (SLID) field. This item is supplied in the form of an offset from the source port's base LID address. It is programmed into the QP Context of a RC, UC, or UD QP when the QP is set up and is referred to as the Source Path Bits.

Global Source/Destination Addresses

Introduction

If the destination CA is not in the same subnet as the source CA, then the packets generated by the QP will have to traverse one or more routers to get to the destination CA. In this case, the packet must contain a Global Router Header

(GRH) that contains the 128-bit Source Global ID (SGID) of the source CA port and the 128-bit Destination Global ID (DGID) of the destination CA port.

DGID Address Identification

When a QP is set up, software provides its QP Context with the DGID assigned to the destination CA port behind which the remote QP resides. This DGID address is inserted in the DGID field in each request packet generated by the QP's SQ Logic.

SGID Address Identification

Background. Each port has at least one 64-bit GUID address assigned to it by the device manufacturer (see "Port's Default GUID Is Hardwired" on page 156). This GUID address resides in entry 0 of the port's *GUIDInfo* attribute. The *GUIDInfo* attribute is a table with one or more entries. The SM can use entries 1 through *n* of the table to assign additional GUIDS to the port.

SGID Selection. The upper 64 bits of the SGID is provided from the attribute that contains the port's 64-bit subnet ID (the *GIDPrefix* attribute). When a RC, UC, or UD QP is set up, software indicates which of the GUIDs assigned to the local port must be inserted into the SLID field of each packet generated by the QP. This information is supplied to the QP Context in the form of an index into the port's *GUIDInfo* attribute.

Additional Global Address Information

In the case of a global destination, software also supplies the QP Context with the following items (which are inserted into their respective fields in each request packet's GRH):

- **Traffic Class** (**TClass**). This value indicates the QoS desired as the request packets cross multiple subnets to get to the destination CA port.
- **Flow Label**. If non-zero, this instructs all routers along the path to ensure that all packets with the same Flow Label value are delivered in the correct order to the destination CA port.
- **Hop Limit**. Each router along the path decrements the Hop Limit and, if it is exhausted, the packet is discarded (because it has lost its way and is wandering throughout the global fabric).

Sending a Message to a Destination CA

Definition of Requester and Responder

In the specification, the QP SQ Logic that is sending request packets is referred to as the requester, while the QP RQ Logic that is receiving them is referred to as the responder.

Example Scenario Assumptions

This example scenario assumes that the QPs being used were created earlier in time. It also makes the following assumptions:

- The companion QPs in the two CAs are both of the Reliable Connected (RC) type (see "QP Type" on page 41).
- The start PSN assigned to the SQ Logic in CA "X" is 100, while the start PSN assigned to the SQ Logic in CA "Y" is 2000—see "SQ Start PSN (Packet Sequence Number)" on page 41.
- The two QPs have just been created and set up and have not yet sent any packets to each other.
- The ePSN assigned to the RQ Logic in CA "X" is therefore 2000, and the ePSN assigned to the RQ Logic in CA "Y" is 100—see "RQ Logic's Expected PSN (ePSN)" on page 41.
- The example operation will cause the transmission of a message from the local memory of CA "X" to the local memory of CA "Y".
- The message is 5KB in size and the maximum data payload size is 2KB—see "Maximum Data Payload Size" on page 42. This means that the entire message will require the transmission of three Send request packets: a "Send First" request packet with a data payload of 2KB, one "Send Middle" request packet with a data payload of 2KB, and a "Send Last" request packet with a data payload of 1KB.
- The Ack Retry Count assigned to the SQ Logic in both QPs = 7—see "Ack Timeout and Missing Packet(s) Retry Counter" on page 44.
- The RNR Retry Count assigned to the SQ Logic in both QPs = 7—see "Receiver Not Ready (RNR) Retry Count" on page 45.
- The destination CA is in the same subnet as the source CA. Packets therefore only require a Local Router Header (LRH) containing the SLID and DLID addresses. They do not require a Global Route Header (GRH) containing a SGID and DGID—see "Global Source/Destination Addresses" on page 45.

Step One: Posting the Message Receive Request

The process of creating a QP is described later in this book. In this example scenario, the requester QP's SQ Logic (item L) in CA "X" will read a 5KB message from its local memory and send it to the target responder QP's RQ Logic (item Q). Upon receipt of the message data, the responder RQ Logic will use the top entry (WQE) currently posted to its RQ (item S) to determine where to write the incoming message data in its CA's local memory. This obviously presupposes that software associated with the requester QP in CA "Y" had posted a WR to the local QP's RQ. The software application (item V) takes the following actions to post such a request:

1. The software application makes a call to the OS memory management routine to request the allocation of a buffer (or multiple buffers) to hold the next expected incoming message. This brings up the question of how it would know in advance the size of the next inbound message. The specification doesn't cover this issue, but it can be handled in a number of ways:
   — This information could have been received in a previous message.
   — The application may have device- or application-specific knowledge of the entity with whom it will trade messages and would "know" the size of messages to be expected.
   — Perhaps the software application associated with the example CA "Y" QP may not have any idea of the size of yet-to-arrive incoming messages and will therefore not post any WRs to the RQ in advance. Instead, when the QP's RQ Logic detects the first packet of an inbound Send operation (as an example), it could return an RNR Nak packet—see "Receiver Not Ready (RNR) Retry Count" on page 45—to the sender's SQ Logic. HCA "Y" could then set a status bit in a CA-specific register to indicate that a WR should be posted to that QP's RQ to handle the expected retry (i.e., retransmission) of the Send request packet. Software associated with CA "Y" is invoked by an interrupt, checks status, and, as a result, posts a WR to the indicated QP's RQ to handle the retry.
2. The example scenario assumes that software will post a WR to the QP's RQ (item S) prior to the receipt of the first request packet of the Send operation. The WR is supplied to CA "Y" by executing a *Post Receive Request* verb call and supplying the following input parameters:
   — The QP handle that identifies the QP to whose RQ the WR is to be posted. The handle was returned when the *Create QP* verb was called.
   — A unique 64-bit WR ID. This WR ID will be deposited in the Completion Queue Entry (CQE; pronounced "cookie") that is created on the

RQ's CQ (Completion Queue; item T) once all of the message data has been written to CA "Y's" local memory.
   — The operation type: in this example, it is a Receive operation.
   — A Scatter Buffer List identifying the main memory buffer(s) to which the inbound message data will be written.
3. Upon receipt of the WR, the *Post Receive Request* verb causes the WR to be posted to the next entry in the QP's RQ (item S). Once posted on the RQ, the WR is referred to as a WQE (Work Queue Entry; pronounced "wookie").

Step Two: Posting the Message Send Request

Software can cause a QP's SQ in a CA to transmit a message transfer request to a QP's RQ in another CA in the following manner:

1. The software application (item B) builds the message to be transferred in memory local to the CA (item C). In the illustration, the software application writes the data that comprises the message into one or more buffers (memory ranges) in main memory.
2. The software application then executes a *Post Send Request* verb call (item D) and supplies the verb with a Work Request (WR) consisting of the following parameters (note that this is not an all-inclusive list):
   — A QP identifier in the form of the handle returned when the *Create QP* verb was called earlier.
   — A unique 64-bit WR ID. This WR ID will be deposited in the CQE that is created once the message Send operation has been completed.
   — The operation type: in this example, it is a Send operation.
   — A Gather Buffer List identifying the main memory buffer(s) that hold the example 5KB message to be sent to the remote CA.
   — Optionally, a 32-bit immediate data value can be supplied. Upon receipt of the entire 5KB message, the remote CA's QP RQ Logic (item Q) deposits this value in the CQE it creates on the RQ's associated CQ (item T). This data value can be used to inform software associated with the destination CA regarding the nature of the message it just received.
3. Upon receipt of the WR, the *Post Send Request* verb causes the WR to be posted to the next entry in the QP's SQ (item J). Once posted on the SQ, the WR is referred to as a WQE.

Step Three: 'Send First' Request Packet Sent

1. The QP's SQ Logic (item L) starts processing the top SQ entry. The WQE specifies a multi-packet Send operation to send the 5KB message from this HCA's local memory (i.e., main memory) to a QP within a remote CA.

2. The SQ Logic determines the maximum amount of data that can be placed in each request packet by checking the Maximum Data Payload Size value (aka PMTU) stored in the QP's Context. In this case, it is 2KB. The SQ Logic uses the WR's Gather Buffer List and reads the first portion of the message (2KB of data) from main memory.
3. The SQ Logic sets the Opcode field in the first request packet to "Send First", thereby indicating to the remote QP's RQ Logic that this is the first packet of a multi-packet Send operation. It should be noted that the target QP's RQ Logic will not know the length of the Send operation until it has received the "Send Last" request packet.
4. The SQ Logic sets the PSN field in the first request packet to the Send Logic's cPSN. Our example assumes that this is the first packet sent by this QP since the QP was created. The cPSN is therefore set to the start PSN assigned to the SQ Logic, 100—see "Example Scenario Assumptions" on page 47.
5. The SQ Logic sets the Destination QP (DestQP) field in the first request packet to the destination QP's QPN (supplied from the local QP's Context).
6. When the QP's SQ Logic forwards the request packet to port X for transmission, it supplies the port with the offset to add to its base LID address in forming the LID address to be placed in the packet's SLID field. See "Source Port's LID Address" on page 45.
7. The SQ Logic sets the DLID field in the first request packet to the DLID address of the destination CA port behind which the target QP resides. This LID address is supplied from this QP's Context.
8. The SQ Logic sets the SL (Service Level) field in the first request packet to the SL from the QP Context—see "Desired Local Quality of Service" on page 43.
9. If the requested operation (in this case, a Send operation) had a global destination (i.e., the destination CA is in a different subnet and the packets therefore have to cross one or more routers), the SQ Logic would insert the following global address information into the respective fields of the first request packet's GRH. The values would be supplied from the QP Context (see "Global Source/Destination Addresses" on page 45):
   — TClass.
   — Flow Label.
   — Hop Limit.
   — Index into HCA port's *GUIDInfo* attribute table. The index is supplied to the port's Link Layer so it can choose the desired SGID address from the port's GUID list (stored in the port's *GUIDInfo* attribute table).
   — The DGID (see "Global Source/Destination Addresses" on page 45) of the destination CA port (item Y).

In this example, however, it should be noted that both the source and destination CAs reside in the same subnet. The request and response packets will therefore not contain a GRH.

10. The first request packet of the Send operation is sent to the Network Layer and then forwarded to the Link Layer of the HCA port (port X) identified in the QP Context. In addition, at this point the SQ Logic also does the following:
    - Updates its nPSN by incrementing it from 100 to 101. This is the PSN that will be inserted in the next request packet sent.
    - Begins awaiting the receipt of the "Send First" request packet's corresponding Ack packet. The receipt of the first Ack packet is covered in "Step Four: First Ack Packet Returned" on page 53.
    - Begins to form the next request packet (a "Send Middle") to send to the Link Layer for transmit. The transmission of the "Send Middle" request packet is covered in "Step Five: 'Send Middle' Request Packet Sent and Ack Returned" on page 55.
11. Upon receipt of the first packet from the Network Layer, the port's Link Layer takes following actions:
    - It adds the offset from the port's base LID address (supplied from the QP Context) to the base LID address and inserts the resulting LID address in the request packet's SLID field (see "Source Port's LID Address" on page 45).
    - If the destination CA resides in a different subnet (i.e., it's a global destination), then the 128-bit SGID address is formed as follows:
      - The port's Link Layer forms the SGID address by combining the port's assigned 64-bit *GIDPrefix* attribute (aka Subnet ID) with the 64-bit GUID selected by the index into HCA port's *GUIDInfo* attribute table (the index is sourced from this QP's Context; see "Global Source/Destination Addresses" on page 45).
    - Uses the SL—supplied from the QP context; see "Desired Local Quality of Service" on page 43—to perform a lookup in the port's *SLtoVLMappingTable* attribute. The entry selected (1-of-16) identifies which of the port's Link Layer transmit buffers (referred to as a Virtual Lane, or VL) the packet is placed in. As will be seen later (in "QoS within the Subnet: SL and VLs" on page 617), during configuration the SM set up the port's *SLtoVLMappingTable* attribute table to map each of the 16 possible SL values to a specific Link Layer transmit buffer. The SM had also set up an arbitration scheme that assigns a level of importance to each of these transmit buffers. This defines in what order the transmit buffers get to transmit packets to the port's Physical Layer.
    - The request packet is posted in the Link Layer VL transmit buffer selected in the previous step.

   - When that VL transmit buffer's turn for packet transmission has come (based on the port's VL transmit buffer arbitration mechanism), the port's Link Layer forwards the request packet to the port's Physical Layer for transmission. The packet's VL (Virtual Lane) field identifies the respective VL receive buffer that is to receive the packet on the other end of the physical link immediately connected to this port (either a port on an intervening switch or router, or the target port on the destination CA).

12. The request packet byte stream from the port's (item X) Link Layer is encoded into 10-bit characters by the port's Physical Layer, is converted into a serial bit stream, and is transmitted over the wire.
13. The request packet traverses one or more links until it arrives at the destination port (item Y). Each switch along the way uses the packet's DLID field to perform a lookup in its Forwarding Table to determine through which of its ports the packet must be transmitted to move it towards the destination port (item Y). A switch's Forwarding Table is set up by the configuration software at startup time.
14. The destination port's Physical Layer deserializes the data, decodes the 10-bit characters into an 8-bit byte stream, and sources the request packet's byte stream to the port's Link Layer.
15. The port's Link Layer address decode logic decodes the request packet's DLID field and determines that this is the destination port.
16. The port's Link Layer accepts the packet's byte stream into the VL receive buffer indicated by the packet's VL field. This is the VL that was chosen by the Link Layer at the other end of the physical link (item X) that is connected to this port.
17. The request packet is forwarded to the port's Network Layer.
18. The Network Layer passes the packet to the RQ Logic (item Q) of the QP targeted by the request packet's DestQP field.
19. The QP's RQ Logic compares the request packet's PSN (100 in this case) to its current ePSN value (100) to determine if the request packet has the expected PSN.
    - Assuming that the packet's PSN = ePSN, packet processing continues.
    - If the packet's PSN is greater than the ePSN, the RQ Logic schedules a PSN Sequence Error Nak packet to be sent back to the remote QP's SQ Logic and the requested operation (a Send in this case) is not executed (i.e., the packet's data payload is not written to this CA's local memory) by the receiving QP's RQ Logic.
    - If the packet's PSN falls within the range of PSNs for request packets that were previously received (i.e., it's a duplicate request packet), the RQ Logic does not re-write the packet's data payload to memory, but it does schedule an Ack packet to be returned.

20. The QP's RQ Logic checks the packet's opcode to determine if a RQ WQE is required to determine where the message is to be written in CA "Y's" local memory. If it is a Send (which this is) or an RDMA Write With Immediate (covered later), a RQ WQE is required. If the QP's RQ currently does not have any WQEs posted, the RQ Logic schedules an RNR Nak packet to be sent back to the sender's SQ Logic (item L) and the requested operation (in this case, a Send) is not executed by the receiving QP's RQ Logic. Assuming that there is at least one WQE posted to the RQ (and in this case, there is), the packet's processing continues.
21. The QP's RQ Logic (item Q) checks the packet's opcode to ensure it makes sense. In this case, it should be a "Send First", not a "Send Middle" or some other nonsense opcode that is out of sequence. Assuming the opcode makes sense, the packet's processing continues.
22. The RQ Logic (item Q) schedules a positive Ack packet to be returned to the sender's QP's SQ Logic (item L). Its transmission and subsequent arrival back at the sender's SQ Logic is covered in "Step Four: First Ack Packet Returned" on this page.
23. The RQ Logic uses the information in the WQE at the top of the RQ (item S) to determine where to write the packet's data payload in the CA's local memory. Earlier in time ("Step One: Posting the Message Receive Request" on page 48), software associated with this CA (i.e., CA "Y") executed a *Post Receive Request* verb call and posted a WR to the QP's RQ.
24. The request packet's data payload is written to the CA's local memory (item W) using the Scatter Buffer List from the RQ WQE.
25. The RQ Logic updates the memory address pointer in the RQ WQE to point to where it left off (and, therefore, where the data payload of the next packet of the Send operation is to be written when it arrives).
26. The RQ Logic updates its ePSN (currently = 100) to ePSN + 1 and awaits the arrival of the next packet of the Send operation.

Step Four: First Ack Packet Returned

Upon receipt of the "Send First" request packet, the responder QP's RQ Logic (item Q) schedules a positive Ack packet to be sent back to the requester QP's SQ Logic (item L). The Ack packet's PSN is the same one contained in the "Send First" request packet. The transmission of the Ack packet involves the following steps:

1. The responder QP's RQ Logic forwards the Ack packet to the CA's Network Layer which, in turn, forwards it to the Link Layer of the port that received the request packet (item Y).

2. The Ack packet does not contain a data payload field. It does, however, contain the Acknowledge opcode and an Acknowledge Extended Transport Header (AETH) field. The opcode indicates that this is an Ack packet, and the AETH indicates the type of Ack packet: positive Ack, or negative Ack (Nak). If it's a Nak packet, the reason for the Nak is also indicated.
3. The Ack packet's DestQP field is loaded with the QPN (QP Number) that identifies the QP that sourced the request packet (this QPN is sourced from the receiving QP's Context).
4. In this example, the AETH indicates that this is a positive Ack packet.
5. The SL used in the Ack packet must be the same as the one used in the request packet.
6. The SLID and DLID fields received in the request packet are swapped in the Ack packet for the return journey.
7. Using the Ack packet's SL value, the port's (item Y) Link Layer performs a lookup in its *SLtoVLMappingTable* attribute to determine which of the Link Layer's VL transmit buffers to post the Ack packet in for transmission.
8. The Ack packet is posted in the selected VL transmit buffer.
9. When that VL transmit buffer's turn for packet transmission has come (based on the VL transmit buffer arbitration mechanism), the port's Link Layer forwards the Ack packet to the port's Physical Layer for transmission. The Ack packet's VL (Virtual Lane) field identifies the respective VL receive buffer that is to receive the packet on the other end of the physical link immediately connected to this port (either a port on an intervening switch or the target port on the destination CA).
10. The Ack packet's 8-bit byte stream from the port's (item Y) Link Layer is encoded into 10-bit characters by the port's Physical Layer, is converted into a serial bit stream, and is transmitted over the wire.
11. The Ack packet traverses one or more links until it arrives at the destination port (item X, the port that originally sourced the request packet into the fabric).
12. The destination port's Physical Layer deserializes the data, decodes the 10-bit characters into an 8-bit byte stream, and sources the Ack packet's byte stream to the port's Link Layer.
13. The port's Link Layer address decode logic decodes the DLID field and determines that this is the destination port.
14. Port's Link Layer accepts the Ack packet byte stream into the VL receive buffer indicated by the Ack packet's VL field. This is the VL chosen by the Link Layer at the other end of the physical link connected to this port.
15. The Ack packet is forwarded to the Network Layer.
16. The Network Layer passes the Ack packet to the SQ Logic (item L) of the QP targeted by the Ack packet's DestQP field. This is the same SQ Logic that originally generated the corresponding request packet.

17. The SQ Logic examines the AETH to determine if this is a positive or negative Ack. In this case, it is a positive Ack.
18. The SQ Logic compares the Ack packet's PSN to determine which of the following is true (in the example, the first case is true; the other three possibilities are covered later in this book):
    — Ack packet's PSN = PSN of the oldest unAck'd request packet (i.e., the Ack packet's PSN = 100). In this example it is equal, so the requester QP's SQ Logic rolls up the lower end of its unAck'd request window by one (in other words, the oldest unAck'd request packet is now = 101).
    — Ack packet's PSN is > the SQ Logic Start PSN but < the PSN of the oldest unAck'd request packet (i.e., it's a duplicate Ack packet).
    — Ack packet's PSN is > the PSN of the oldest unAck'd request packet but less than the high-water mark the SQ Logic has reached in issuing new request packets.
    — Ack packet's PSN is < the SQ Logic Start PSN or > the high-water mark the SQ Logic has reached in issuing new request packets (i.e., it's an invalid Ack packet).

Step Five: 'Send Middle' Request Packet Sent and Ack Returned

It should be noted that the requester QP's SQ Logic doesn't wait for the Ack for the just-issued request packet to arrive before it launches the next request packet into the fabric.

The requester QP's SQ Logic continues as follows:

1. Using the top entry on the SQ (item J), it reads the next 2KBs of message data from main memory using the Gather Buffer List in the WQE on top of the SQ.
2. It adjusts the WQE's read pointer to point to where it left off.
3. It places the next sequential PSN(101) in the "Send Middle" request packet.
4. It transmits the request packet to the responder QP's RQ Logic with a "Send Middle" opcode.

Upon receipt of the request packet, the responder QP's RQ Logic (item Q) takes the following actions:

1. The responder QP's RQ Logic compares the request packet's PSN (101 in this case) to its current ePSN value (101) to determine if the request packet has the expected PSN:
    — Assuming that the packet's PSN = ePSN, the packet's processing continues (see step 2 below).

   - If the packet's PSN is greater than the ePSN, the RQ Logic schedules a PSN Sequence Error Nak packet to be sent back to the remote QP's SQ Logic and the requested operation (a Send in this case) is not executed (i.e., the packet's data payload is not written to this CA's local memory) by the receiving QP's RQ Logic.
   - If the packet's PSN falls within the range of PSNs for request packets that were previously received (i.e., it's a duplicate request packet), the RQ Logic does not re-write the packet's data payload to memory, but it does schedule an Ack packet to be returned.
2. The QP's RQ Logic (item Q) checks the packet's opcode to ensure it makes sense. In this case, it should be a "Send Middle" or a "Send Last", not a "Send First" or some other nonsense opcode. Assuming the opcode makes sense, the packet's processing continues.
3. The packet's 2KB data payload is written to the CA's local memory (item W) using the updated Scatter Buffer List pointer from the top RQ WQE.
4. The RQ Logic updates the memory address pointer in the RQ WQE to point to where it left off (and, therefore, where the data payload of the next request packet of the Send operation is to be written when it arrives).
5. The RQ Logic updates its ePSN to ePSN + 1 (102) and awaits the arrival of the next packet of the Send operation.

The responder QP's RQ Logic schedules a positive Ack packet to be sent back to the requester QP's SQ Logic. The Ack packet's PSN is the same one contained in the "Send Middle" request packet just received. If this were a longer message Send operation, the steps listed in this section would be repeated for each of the "Send Middle" packets.

Step Six: 'Send Last' Request Packet Sent

The requester QP's SQ Logic (item L) continues as follows:

1. It reads the final 1KB of message data from the last buffer identified in the Gather Buffer List of the WQE on top of the SQ (item J). The data payload in the last packet of a message Send operation will contain 1KB of data.
2. The next sequential PSN (102) is placed in the request packet.
3. It sends the request packet with a "Send Last" opcode.

Upon receipt of the "Send Last" request packet, the responder QP's RQ Logic (item Q) takes the following actions:

1. The responder QP's RQ Logic compares the request packet's PSN (102 in this case) to its current ePSN value (102) to determine if the request packet has the expected PSN.

   - — Assuming that the packet's PSN = ePSN, the packet's processing continues (see step 2 below).
   - — If the packet's PSN is greater than the ePSN, the RQ Logic schedules a PSN Sequence Error Nak packet to be sent back to the remote QP's SQ Logic and the requested operation (a Send in this case) is not executed (i.e., the packet's data payload is not written to this CA's local memory) by the receiving QP's RQ Logic.
   - — If the packet's PSN falls within the range of PSNs for request packets that were previously received (i.e., it's a duplicate request packet), the RQ Logic does not re-write the packet's data payload to memory, but it does schedule an Ack packet to be returned.
2. The QP's RQ Logic (item Q) checks the packet's opcode to ensure it makes sense. In this case, it should be a "Send Middle" or a "Send Last" (which it is), not a "Send First" or some other nonsense opcode. Assuming the opcode makes sense, the packet's processing continues.
3. The packet's 1KB data payload is written to the CA's local memory (item W) using the updated pointer in the Scatter Buffer List from the top WQE on the RQ (item S).
4. All packets of the message send operation have now been received and written to the CA's local memory, so the RQ Logic updates its ePSN to ePSN + 1 (103) and awaits the arrival of the first request packet of the next message transfer operation.
5. The top WQE is retired from the RQ (item S) and a CQE is created on the CQ associated with the RQ (item T). This CQE contains the completion status of the message receive operation. In addition, if the "Send Last" request packet contained an ImmDtETH (Immediate Data Extended Transport Header), the 32-bit immediate data value it contains is stored in the CQE.
6. This completes the receipt of the message (but the final Ack is yet to be sent; see the next section).
7. CA "Y" could be designed to generate an interrupt whenever a CQE is posted to a CQ associated with any QP's SQ or RQ.

Step Seven: Final Ack Returned

The responder QP's RQ Logic schedules a positive Ack packet to be sent back to the requester QP's SQ Logic (item L). The Ack packet's PSN (102) is the same one contained in the "Send Last" request packet. Upon arrival at the requester QP's SQ Logic, the SQ Logic takes the following actions:

1. The SQ Logic examines the Ack packet's AETH field to determine if this is a positive or negative Ack. In this case, it is a positive Ack.

2. Since this Ack is ack'ing the "Send Last" request packet, the SQ Logic takes the following actions:
   — The top WQE is retired from the SQ (item J).
   — A CQE is created on the CQ associated with the SQ (item G). This CQE contains the completion status of the message send operation.
3. This completes the message send operation.

Figure 3-4: Example Scenario (left half of illustration)

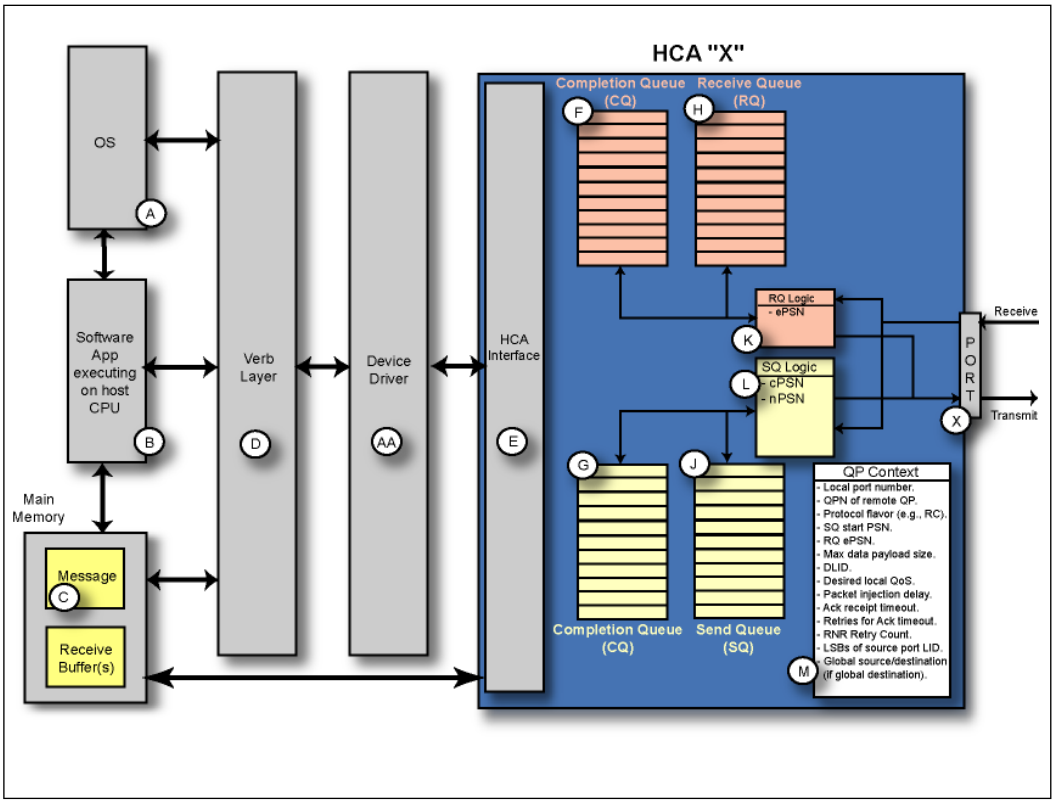


Figure 3-5: Example Scenario (right half of illustration)