"The 'must-have' PC architecture reference set."

—*PC Magazine's* "Read Only" column

InfiniBand Network Architecture

MindShare, Inc.

Tom Shanley

PC SYSTEM ARCHITECTURE SERIES

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

world-class technical training

Are your company's technical training needs being addressed in the most effective manner?

MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students' time away from work and provides cost-effective alternatives. MindShare offers many flexible solutions to meet those needs. Our courses are taught by highly-skilled, enthusiastic, knowledgeable and experienced instructors. We bring life to knowledge through a wide variety of learning methods and delivery options.

training that fits your needs

MindShare recognizes and addresses your company's technical training issues with:

- Scalable cost training
- Just-in-time training
- Training in a classroom, at your cubicle or home office
- Customizable training options
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training

MindShare training courses expand your technical skillset

- PCI Express 2.0®
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals

... and many more.

All courses can be customized to meet your group's needs. Detailed course outlines can be found at **www.mindshare.com**

bringing life to knowledge.

real-world tech training put into practice worldwide

*PCI Express ® is a registered trademark of the PCISIG

MindShare Learning Options

MindShare Classroom

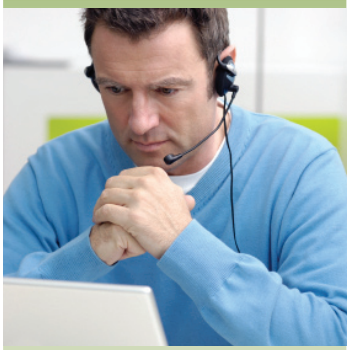
In-House Training

Public Training

Classroom Training

Invite MindShare to train you in-house, or sign-up to attend one of our many public classes held throughout the year and around the world. No more boring classes, the 'MindShare Experience' is sure to keep you engaged.

MindShare Virtual Classroom



Virtual In-House Training

Virtual Public Training

Virtual Classroom Training

The majority of our courses live over the web in an interactive environment with WebEx and a phone bridge. We deliver training cost-effectively across multiple sites and time zones. Imagine being trained in your cubicle or home office and avoiding the hassle of travel. Contact us to attend one of our public virtual classes.

MindShare eLearning

Intro eLearning Modules

Comprehensive eLearning Modules

eLearning Module Training

MindShare is also an eLearning company. Our growing list of interactive eLearning modules include:

- **Intro to Virtualization Technology**
- **Intro to IO Virtualization**
- **Intro to PCI Express 2.0 Updates**
- **PCI Express 2.0**
- **USB 2.0**
- **AMD Opteron Processor Architecture**
- **Virtualization Technology**

...and more

MindShare Press

Books

eBooks

MindShare Press

Purchase our books and eBooks or publish your own content through us. MindShare has authored over 25 books and the list is growing. Let us help make your book project a successful one.

Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to "Bring Your Knowledge to Life." Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual classroom settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

ADAPTEC • AMD • AGILENT TECHNOLOGIES • APPLE • BROADCOM • CADENCE • CRAY • CISCO • DELL • FREESCALE
GENERAL DYNAMICS • HP • IBM • KODAK • LSI LOGIC • MOTOROLA • MICROSOFT • NASA • NATIONAL SEMICONDUCTOR
NETAPP • NOKIA • NVIDIA • PLX TECHNOLOGY • QLOGIC • SIEMENS • SUN MICROSYSTEMS SYNOPSYS • TI • UNISYS

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

4285 SLASH PINE DRIVE COLORADO SPRINGS, CO 80908 USA
M 1.602.617.1123 O 1.800.633.1440 ravi@mindshare.com

www.mindshare.com

InfiniBand Network Architecture

First Edition

InfiniBand Network Architecture

First Edition

MINDSHARE, INC.

TOM SHANLEY

TECHNICAL EDIT BY
JOE WINKLES

ADDISON-WESLEY DEVELOPER'S PRESS

Reading, Massachusetts • Harlow, England • Menlo Park, California
New York • Don Mills, Ontario • Sydney
Bonn • Tokyo • Amsterdam • Mexico City • Seoul
San Juan • Madrid • Singapore • Paris • Taipei • Milan

Library of Congress Cataloging-in-Publication Data

ISBN: 0-201-726823zz

Sponsoring Editor:
Project Manager:
Cover Design:
Set in 10 point Palatino by MindShare, Inc.

1 2 3 4 5 6 7 8 9-MA-999897
First Printing, September 2000zz

Addison-Wesley books are available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (800) 238-9682.

Find A-W Developers Press on the World Wide Web at:
http://www/aw.com/devpress/

At-a-Glance Table of Contents

Part 1, **Core Concepts**, introduces basic concepts and terminology and consists of the following chapters:

Part 2, QP Creation and Operation, begins the portion of the book that provides detailed information about the InfiniBand technology. Part 2 provides a detailed descripion of the creation of, the management of, and the basic operation of the various types of Queue Pairs. Part 2 consists of the following chapters:

Part 3, Protection Mechanisms, introduces memory management, memory protection, and the various other protection mechansims available in the InfiniBand environment. Part 3 consists of the following chapters:

Part 4, Detailed Description of the Transport Services, provides a detailed description of the four IBA transport services as well as the Raw IPv6 and Raw EtherType services. In addition, it also provides a detailed description of UD Multicasting, Automatic Path Migration, and Static rate Control. Part 4 consists of the following chapters:

Part 5, Link and Physical Layer Descriptions, provides a detailed description of the Link Layer and the Physical Layer and consists of the following chapters:

Part 6, The SM and the SA, provides a detailed description of the SM and the SA, as well as the interfaces to the SM and the SA: the SMI and the GSI. In addition, it discusses the issues related to multiple SMs in a subnet and the discovery (i.e., the configuration) process. Part 6 consists of the following chapters:

Part 7, General Services, provides a detailed description of the General Services and consists of the following chapters:

Chapter 1: About This Book

Part 1: Core Concepts

Chapter 2: Basic Terms and Concepts

Chapter 3: Intro to Attributes and Managers

Chapter 4: QP: Message Transfer Mechanism

Chapter 5: Intro to Transport Types

Chapter 6: Intro to Send/Receive Operations

Chapter 7: Division of Labor

Chapter 8: Subnet-Local Addressing

Chapter 9: Global Addressing

Chapter 10: Intro to the Managers

Chapter 11: Intro to Connection Establishment

Chapter 12: PSN Usage

Part 2: QP Creation and Operation

Chapter 13: QP Verbs and QP State Machine

Chapter 14: WRs, WQEs, and CQEs

Chapter 15: Asynchronous Events and Errors

Part 3: Protection Mechanisms

Chapter 16: Memory Protection

Chapter 17: Other Protection Mechanisms

Part 4: Detailed Description of the Transport Services

Chapter 18: RC Transport Service

Chapter 19: UC Transport Service

Chapter 20: RD Transport Service

Chapter 21: UD Transport Service

Chapter 22: Raw Transport Service Types

Chapter 23: Multicasting

Chapter 24: Automatic Path Migration

Chapter 25: Static Rate Control

Part 5: Link and Physical Layer Descriptions

Chapter 26: Detailed Description of the Link Layer

Contents

Chapter 27: Detailed Physical Layer Description

Part 6: The SM and SA

Chapter 28: The SMI

Chapter 29: Detailed Description of MADs

Chapter 30: SM Methods and Attributes

Chapter 31: Multiple SMs

Chapter 32: Discovery

Chapter 33: The GSI

Chapter 34: Detailed Description of SA

Chapter 36: Performance Management

Chapter 37: Communications Management

1 Basic Terms and Concepts

This Chapter

This chapter provides:

- Advantages of the InfiniBand network architecture.
- An introduction to basic terminology.
- Packet addressing basics.
- The basic roles of channel adapters, routers, switches, and repeaters.
- An introduction to message passing.

The Next Chapter

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

Definition of the Acronym "IBA"

In the InfiniBand specification, the acronym IBA stands for InfiniBand Architecture.

Packet Field Documentation Convention

Many of the fields contained in a packet are comprised of subfields. When the specification refers to a subfield, it is frequently represented as:

FieldName:SubFieldName.

As an example, LRH:SL is referring to the Service Level subfield within the packet's Local Route Header field.

InfiniBand Advantages

Some of the major advantages offered by the InfiniBand network architecture include the following:

- Supports multiple protocols, including tunneling packets associated with virtually any non-InfiniBand protocol through an InfiniBand fabric.
- Throughput of 2.5Gb/s, 10Gb/s, or 30Gb/s is achievable (depending on the link width implemention).
- Non-privileged applications can send and receive messages without causing a kernel privilege mode switch.
- The processor is not involved in message passing. Rather, each network message transfer is handled by hardware DMA transfer engines within the channel adapter.
- The InfiniBand protocol includes message transfer commands that permit direct memory-to-memory message transfers between the local memories of two channel adapters.
- The majority of the protocol layers can be implemented in silicon, thereby minimizing the burden placed on software and the processor.

Some Preliminary Terminology

- **Processor Node**. See Figure 1-1 on page 11. A group of one or more processors and their associated main memory. It is interfaced to the IBA fabric via one or more **HCAs** (**Host Channel Adapter**), each of which implements one or more IBA ports.
- **Port**. A bi-directional interface that connects an IBA device to an IBA link.
- **Link**. The bi-directional, high-speed connection between two ports on two IBA devices. At a minimum, it is implemented using one high-speed serial transmission line in each direction capable of transmitting at 2.5Gb/s, yielding 250MB/s throughput (note that each 8-bit character is converted to a 10-bit character before transmission). Optionally, a link may be implemented with four or twelve transmission lines in each direction, yielding 1GB/s or 3GB/s throughput, respectively.
- **IOU** and **IOC**. Refer to Figure 1-2 on page 11. An IO Unit (IOU) is comprised of:
  - The **TCA** (**Target Channel Adapter**) that interfaces it to the IBA fabric.
  - One or more IO Controllers (IOCs) providing the interface to various IO devices. An example of an IOC would be a mass storage array.
- **CA**. The term Channel Adapter is abbreviated as CA throughout the remainder of this book.

Figure 1-1: Host Channel Adapter

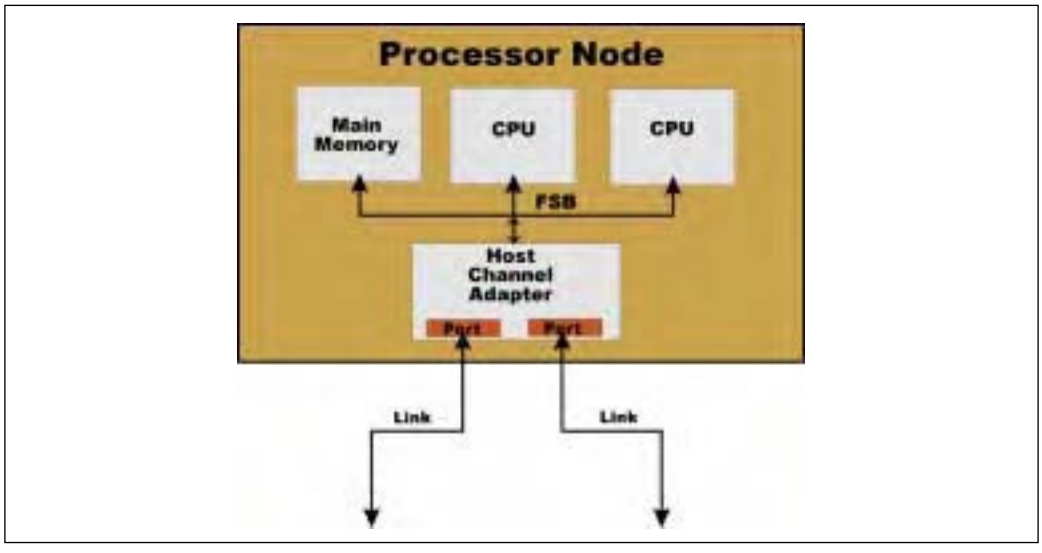
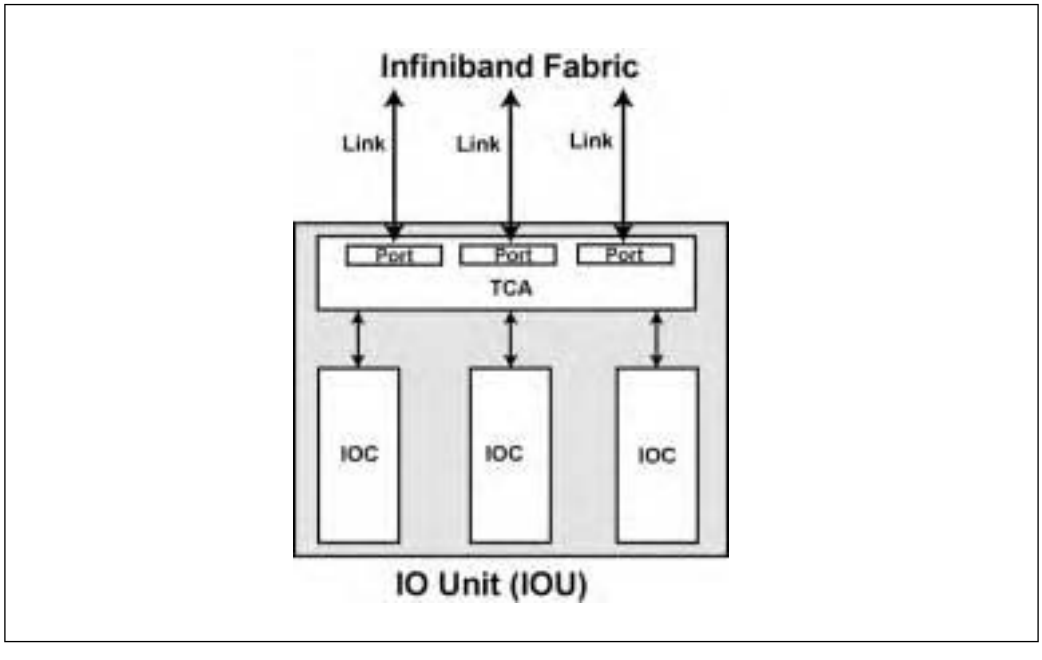


Figure 1-2: Target Channel Adapter



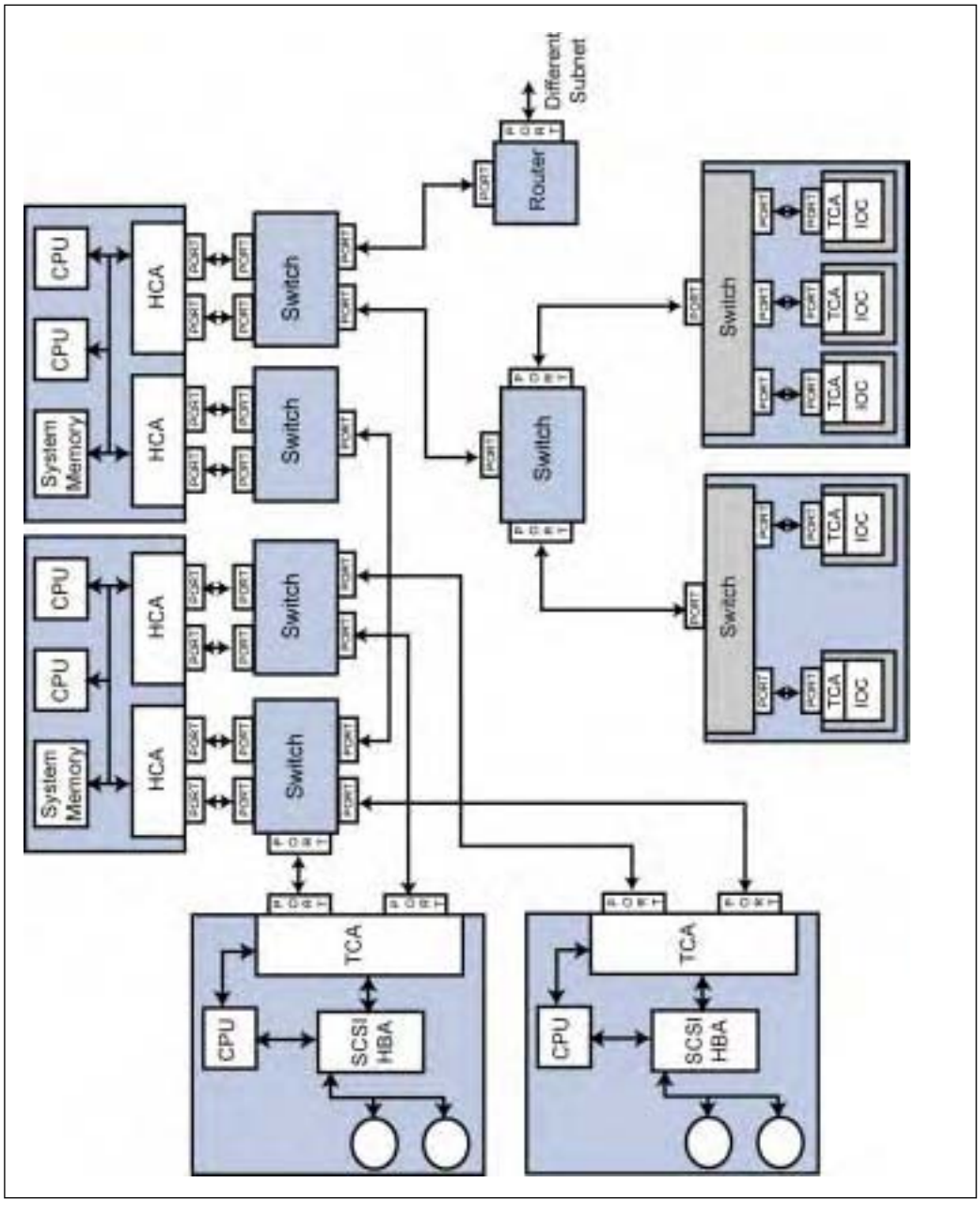
Definition of a Subnet

The specification defines a subnet as follows: a set of ports and associated links with a common **Subnet ID** and managed by a common **Subnet Manager (SM)**.

The SM is the entity that discovers all of the devices in the subnet at startup time, configures them, and then performs a periodic sweep of the subnet to detect any changes to the subnet's topology.

Refer to Figure 1-3 on page 13. In the illustration, all of the CA and router ports, as well as switch management ports (switch port 0—typically an internal port with no connector for a physical link to attach to) that can exchange packets solely by traversing switches (not routers) are said to reside in the same subnet. During configuration, the SM assigns each of these ports a unique Local ID address (referred to as the LID address) as well as a common Subnet ID (also referred to as the **Subnet Prefix**) that identifies the subnet that a port resides in. Subnets may be connected to each other through routers.

Figure 1-3: Example System



2 Intro to Attributes and Managers

The Previous Chapter

The previous chapter provided:

- An introduction to basic terminology.
- Packet addressing basics.
- The basic roles of channel adapters, routers, switches, and repeaters.
- An introduction to message passing.

This Chapter

This chapter introduces the concept of device attributes, managers, management agents (MAs), and management datagrams (MADs).

The Next Chapter

The next 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.

Why Talk About Attributes and Managers Now?

At this point in the book, it becomes increasing difficult to avoid discussion of the various device attributes. This being the case, it would seem like a good idea to define what they are, who accesses them, and why.

Definition of an Attribute

With the exception of a repeater (a repeater is programmatically invisible; it has no attributes), all IBA devices contain a series of attributes that various managers perform operations upon (e.g., reads or writes) for various reasons. For example:

- To discover the existence of an IBA device.
- To discover the IBA device type (e.g., a CA, a switch, or a router).
- To ascertain the device's current status.
- To determine the number of ports implemented on the device.
- To control the device's operational characteristics.
- etc.

Who Accesses Attributes?

IBA defines a series of managers, each of which is responsible for various aspects of an IBA device's operation. Some examples are:

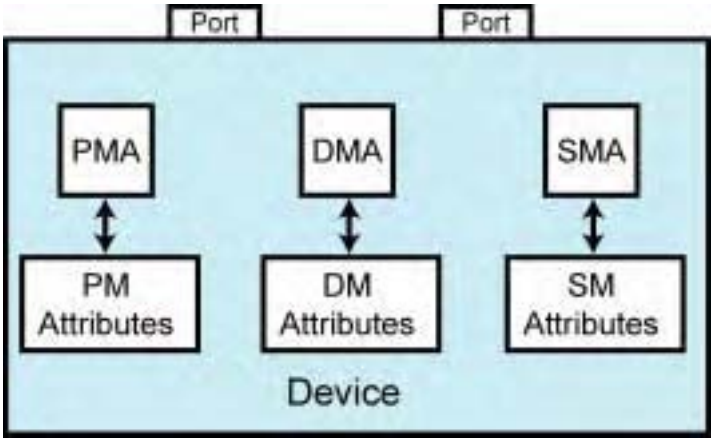
- Subnet Manager (SM).
- Performance Manager (PM).
- Device Manager (DM).
- Communications Manager (CM).

MAs Handle Access Requests

Refer to Figure 2-1 on page 27. Each IBA device contains a series of management agents (MAs), each of which handles attribute access requests issued by their respective managers. Some examples are:

- The Subnet Management Agent (SMA) handles requests issued by the SM.
- The Performance Management Agent (PMA) handles requests issued by the PM.
- The Device Management Agent (DMA) handles requests issued by the DM.

Figure 2-1: Each MA Accesses Its Manager's Attributes



MA's Response

When a MA within a device receives an attribute access request packet from its respective manager, it performs the requested operation on the specified attribute and, in most cases, returns the results in a response packet.

Managers Use Special Packets Called MADs

Request MAD

The various managers use special packets called Management Datagrams (MADs) to request that an operation (i.e., a method) be performed on a device attribute. A request MAD has the following basic characteristics:

- The MAD message is wholly contained in a single packet's data payload field and the data payload field always contains exactly 256 bytes.

- Some of the elements contained in the MAD are:
  - **Management Class**: Identifies the manager that issued the packet and therefore the management agent within the device that is to process the request MAD.
  - **Method**: Specifies the type of operation the target management agent is to perform on the specified attribute. As an example, a Get method performs an attribute read, while a Set method performs an attribute write operation.
  - **Attribute ID**: Specifies the attribute to be acted upon (e.g., to read or written).
  - **Attribute Modifier**: Attribute-specific; not required for many attribute/method combinations. Specifies additional information regarding the targeted attribute. As an example, if the manager is targeting the *PortInfo* attribute on a CA, router, or switch, the Modifier specifies the target port number.
  - **Data area**: Content depends on the Method and Attribute. Some examples would be:
    - If the Method is a Set operation, the data area contains the data to be written to the specified attribute.
    - If the Method is a Get operation, the data area contents is undefined in the request MAD, but contains the requested attribute's contents in the corresponding response MAD returned by the device.

Response MAD

Upon receipt of a request MAD from its respective manager, the MA performs the operation on the indicated attribute and returns the results in a response MAD. The response MAD contains (among other things):

- **Status field**: This 16-bit field indicates the operation's completion status. The Status field is only valid in a response MAD generated by the target in response to a request MAD. It is formatted as follows:
  - Lower 8 bits = status common to all classes. These are:
    - Busy, MAD discarded (not an error).
    - Redirect required (not an error).
    - Class not supported.
    - Method not supported.
    - Method/Attribute combination not supported.
    - One or more Attribute fields contain bad values.
  - Upper 8 bits = class-specific status.

- **Data field**. The content of this field is defined by the type of request MAD this is a response to. Some examples would be:
  - If the request was an attribute read (i.e., a Get), the data field contains the data read from the attribute indicated in the request MAD.
  - If the request was an attribute write (i.e., a Set), the data field contains the data read from the attribute indicated in the request MAD after the Set operation (i.e., the attribute write) was performed.

Attribute Format and Documentation Conventions

General

The IBA specification defines many different types of attributes:

- An attribute may be very simple and consists of a single item. An example would be the *LEDInfo* (Light-Emitting Diode Information) attribute.
- An attribute may take the form of a table. An example would be the *LinearForwardingTable* attribute in a switch.
- An attribute may consists of data structure containing a number of elements. An example would be the *PortInfo* attribute.

Attribute and Element Designation

As just discussed, many attributes are actually comprised of a series of elements (the specification frequently refers to them as components). As an example, the *PortInfo* attribute is a data structure containing 46 elements (see Table 29-5 on page 825). The specification references a particular element of a multi-element attribute using the following format:

AttributeName.ElementName

Some examples would be:

- *PortInfo.M_Key*
- *PortInfo.LID*
- *PortInfo.Capabilitymask*
- *SMInfo.GUID*
- *SMInfo.ActCount*
- *SwitchInfo.LinearFDBCap*
- *SwitchInfo.DefaultPort*

In this book, the author has italicized all instances of attributes and their elements.

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.

4 Intro to Transport Types

The Previous Chapter

The previous chapter introduced 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) were introduced. The four IBA QP types were introduced and the concept of the QP Context and its contents were defined. Finally, there was a rather detailed example of a message transfer from one CA to another.

This Chapter

This 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 within "raw" IBA packets.

The Next Chapter

The next chapter provides an introduction to the five types of message transfer requests that can posted to and executed by a QP's SQ logic. It also introduces the single type of message transfer request that can be posted to and executed by a QP's RQ logic.

Four IBA Transfer Protocol Flavors

QPs (in separate CAs) that will be sending messages to each other must each be set up to use the same IBA protocol "flavor." In the specification, the four IBA transfer types are referred to as service types, service classes, or transport types.

Connected Service Types

General

Earlier Example Assumed RC Service Type. The example scenario described in "Sending a Message to a Destination CA" on page 47 and pictured in Figure 3-4 on page 58 and Figure 3-5 on page 59 made the assumption that the QPs in the two CAs had been set up to use the Reliable Connected (RC) IBA transport service.

Two Connected Service Types. The specification defines two types of connected service types:

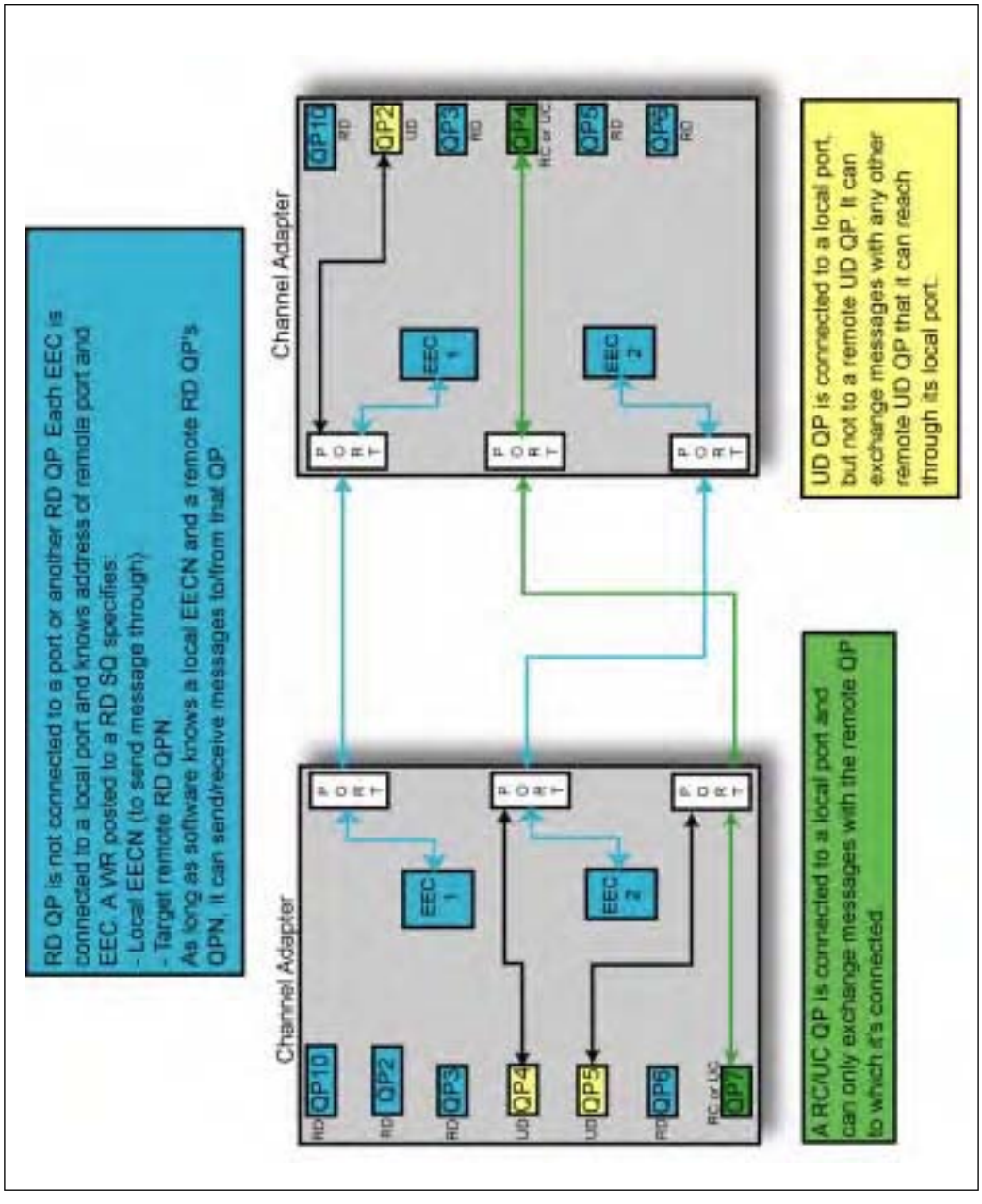
— Reliable Connected (**RC**). The responder QP's RQ Logic must respond to each request packet with an Ack or Nak packet. See "Reliable Connected Service Type" on page 64.
— Unreliable Connected (**UC**). UC works the same as RC except that the responder QP's RQ Logic does not send back a response to each request packet. See "UC Transport Service" on page 443.

RC/UC Necessitate Initial QP Connection Establishment. In both cases, the two QPs are initialized at setup time with information about each other and this information is stored in the QP Context of each of the two QPs.

RC/UC Are Private, Point-to-Point Comm Channels. Figure 4-1 on page 63 illustrates a pair of RC or UC QPs, one QP in each of the illustrated CAs (QP7 and QP4). When software associated with a RC or UC QP wishes to send a message from its local CA's memory to the memory of the CA containing the its companion RC or UC QP, it posts a WR to the SQ of its local RC or UC QP. It's important to note the following: *the WR does not specify the destination CA port or QP to which the message is to be sent*. This is because, once it has been set up, the QP inherently knows the addresses of the target port and QP in the other CA (from the information stored in its QP Context).

The two QPs (in two separate CAs) using the RC or UC service type to communicate with each other are therefore limited in the following manner: *they can only transfer messages to each other (and not with any other QP)*.

Figure 4-1: QP Basics



Reliable Connected Service Type

This service type provides a high degree of reliability, but, due to the required response packets, it generates a substantial amount of traffic over the network. The basic characteristics of the RC service type are:

- **Connection needed**. Before any messages may be transferred, a connection must be established between the RC QPs in the two CAs, and the QP Contexts of the two QPs are each programmed with the identity of the remote QP as well as the address of the port behind which the remote QP resides.
- **Private communications channel**. The two RC QPs may then be used to send messages to each other (but not to any other QP in the same or any other target adapter).
- **Message size**. Each message transfer WR can specify a message transfer anywhere from zero to 2GB in size. Messages larger than one PMTU are segmented into multi-packet transfers.
- An **Ack/Nak protocol** permits the requester (i.e., the QP SQ Logic sending the message) to verify that all packets are delivered to the responder QP's RQ Logic. It also permits the responder QP's RQ Logic to detect missing, duplicate, or invalid packets.
- **Software completion notification**. Software will not be alerted that a message transfer has completed until the QP's SQ Logic has received the Ack indicating that the message's final request packet (in the case of a Send or RDMA Write), or final RDMA Read response packet (in the case of an RDMA Read), or the Atomic response packet (in the case of an Atomic RMW).
- **High traffic**. Because the responder QP's RQ Logic must Ack or Nak each request packet received, this service class generates considerably more network traffic than the "unreliable" service types.
- **Packet PSN checking**. Each request packet contains a PSN that the target QP's RQ Logic uses to verify that all request packets are received in order (and that each is only processed once, even if it should be received multiple times; more on this later).
- Two **CRC** fields in each packet are used to verify the integrity of the packet.
- **Completes on receipt of final response**. The SQ Logic cannot create a CQE on the SCQ (Send Completion Queue) to signal the completion of a message transfer until the response for the final packet of the message transfer has been received.
- **Operations supported**. Supports the following types of message transfer operations (they are explained in the next chapter, "Intro to Send/Receive Operations" on page 77):
  - — RDMA Read support is required.
  - — RDMA Write support is required.

  - It is optional whether or not the CI (see *Channel Interface* in the glossary) supports Atomic operations. If it does, then it must support both the Atomic Fetch and Add and Atomic Compare and Swap If Equal operations on the RC transport service type.
  - Send support is required.
  - Bind Memory Window support is required (memory windows are covered in "Memory Windows" on page 308).

Unreliable Connected Service Type

UC is a subset of the RC protocol. The responder's RQ Logic doesn't Ack or Nak each request packet it receives.

- **Disadvantage**: The requester has no verification that each request packet has been received by the remote QP's RQ Logic.
- **Advantages**:
  - This protocol generates considerably less network traffic than RC.
  - From the sender's perspective, the message transfer completes quickly (because it doesn't have to wait for the receipt of all of the responses for the transfer to complete).

The basic characteristics of the UC service type are:

- **Connection needed**. Just like the RC protocol, before any messages may be transferred a connection must be established between the UC QPs in the two adapters, and the QP Contexts of the two QPs are each programmed with the identity of the remote QP as well as the address of the port behind which it resides.
- **Private communications channel**. Just like the RC protocol, the two UC QPs may then be used to send messages to each other (but not to any other other QP in the same or any other target adapter).
- **Message size**. Just like the RC protocol, each message can be anywhere from 0 to 2GB in size. Messages larger than one PMTU are segmented into multi-packet transfers.
- **No Ack/Nak protocol.** Unlike the RC protocol, there is no Ack/Nak protocol. The requester (i.e., the QP SQ Logic sending the message) therefore cannot verify that each packet is delivered to the responder (the target QP's RQ Logic).
- **Software completion notification**. There are no responses returned by the remote QP's RQ Logic when using the UC transport service type. That being the case, software on the sender's side is alerted that a message transfer has completed immediately upon the transmission of the last request packet of the Send or RDMA Write (RDMA Reads and Atomic requests are not supported).

5 Intro to Send/Receive Operations

The Previous Chapter

The previous chapter provided 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.

This Chapter

This chapter provides an introduction to the five types of message transfer requests that can posted to and executed by a QP's SQ logic. It also introduces the single type of message transfer request that can be posted to and executed by a QP's RQ logic.

The Next Chapter

The next chapter provides an introduction to the layers comprising the IBA stack that handles message transmission and reception. Those layers are:

- The ULP (Upper Layer Protocol). This is actually not part of the IBA stack. Rather, it is comprised of OS and application software that uses the stack to pass messages with remote CAs.
- The Verb Layer.
- The Transport Layer.
- The Network Layer.
- The Link Layer.
- The Physical Layer.

How Is a Message Transmitted or Received?

To Transmit a Message

To send a message from the SQ of one QP to the RQ Logic of another QP, software posts a WR to the requester QP's SQ by executing the *Post Send Request* verb call.

To Handle Inbound Send or RDMA Write With Immediate

To prepare a QP's RQ Logic to handle an incoming Send or RDMA Write With Immediate message to be written to its local memory, software posts a WR to the QP's RQ by executing a *Post Receive Request* verb call.

SQ Operation Types

Introduction

The types of send operation WRs that can be posted to a SQ are:

- Message read request.
- Message write request.
- Locked RMW (Read/Modify/Write) request.
- It could also be a purely local operation (i.e., the **Memory Bind** operation).

Specifically, the SQ WR operation types are:

- **Send**. The requester QP's SQ Logic sends a block of data from its local memory to the responder QP's RQ Logic. The WR at the head of the responder QP's RQ tells the responder QP's RQ Logic where to write the data in its local memory. The Send operation is supported on all four IBA transport types and both non-IBA raw transport types.
- **RDMA Read**. The requester QP's SQ Logic issues a memory read request to the responder QP's RQ Logic. The responder QP's RQ Logic reads the requested data from its local memory and transmits it back to the requester

QP's SQ Logic. On receiving the requested read data, the SQ Logic writes the data into the local memory buffer(s) specified in the active SQ WQE. The RDMA Read operation is only supported on the RC and RD transport types.

- **RDMA Write**. The requester reads data from its local memory and specifies where the responder is to write the data in its local memory. The RDMA Write operation is supported on the RC, UC, and RD transport service types.
- **Atomic RMW**. The requester QP's SQ Logic issues a RMW request to the responder QP's RQ Logic. The responder QP's RQ Logic performs the requested RMW on the specified local memory location and transmits the data read from the location back to the requester QP's SQ Logic. On receiving the read data, the SQ Logic writes the data into the local memory buffer specified in the active SQ WQE. Atomic operations are only supported on the RC and RD transport service types.
- **Memory Window Bind**. More on this later.

Large Messages Are Segmented into Multiple Packets

A message may be too large to fit into the data payload field of a single packet (i.e., the message size exceeds the PMTU specified in the QP Context—for RC and UC, or EEC—for RD). In this case, the QP divides it up into multiple packets for transmission. A message therefore is transmitted as one of the following:

- A single packet (referred to as an "Only" packet).
- Two packets: a "First" and a "Last."
- Three or more packets consisting of:
  - A "First" packet,
  - one or more "Middle" packets,
  - followed by a "Last" packet.

Packet Type Specified in Each Packet

Refer to Figure 5-1 on page 80. Every packet of each message contains the Base Transport Header (BTH) field. The BTH:Opcode field specifies what type of message transfer request packet this is. Table 5-1 on page 81 defines the packet Opcode types. The 8-bit Opcode field is divided into two subfields:

- Bits 7:5 define the transport service type (RC, UC, RD, or UD).
- Bits 4:0 define the message transfer operation type (e.g., Send, RDMA Write, etc.) as well as whether the packet is the "Only", a "Middle", or the "Last" packet of a message.

The last column in the table defines the types of additional headers immediately following the packet's BTH field, the presence of which are defined by the packet type as indicated by the BTH:Opcode field. Note that ETH stands for Extended Transport Header. The definition of each of the ETHs can be found later in this book (as well as in the "Glossary" on page 1133).

Figure 5-1: The Base Transport Header

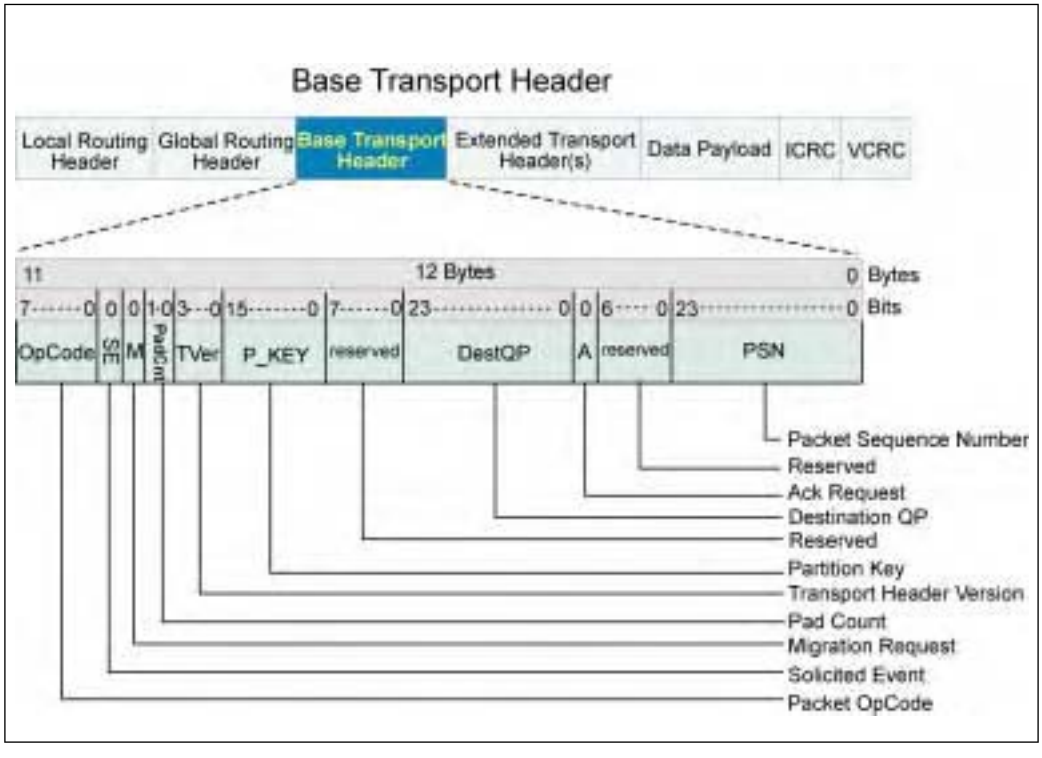


Table 5-1: Packet Opcodes

Bits 7:5	Bits 4:0	Packet Type	Fields That Follow BTH
000 **RC**	00000	Send First	Data PayLoad
	00001	Send Middle	Data PayLoad
	00010	Send Last	Data PayLoad
	00011	Send Last with Immediate	Immediate Data, Data PayLoad
	00100	Send Only	Data PayLoad
	00101	Send Only with Immediate	Immediate Data, Data PayLoad
	00110	RDMA Write First	RDMA ETH (RETH), Data PayLoad
	00111	RDMA Write Middle	Data PayLoad
	01000	RDMA Write Last	Data PayLoad
	01001	RDMA Write Last with Immediate	Immediate Data, Data PayLoad
	01010	RDMA Write Only	RETH, Data PayLoad
	01011	RDMA Write Only with Immediate	RETH, Immediate Data, Data PayLoad
	01100	RDMA Read Request	RETH
	01101	RDMA Read Response First	Acknowledge ETH (AETH), Data PayLoad
	01110	RDMA Read Response Middle	Data PayLoad
	01111	RDMA Read Response Last	AETH, Data PayLoad
	10000	RDMA Read Response Only	AETH, Data PayLoad
	10001	Acknowledge packet	AETH
	10010	ATOMIC Acknowledge packet	AETH, AtomicAckETH
	10011	CmpSwap packet	AtomicETH
	10100	FetchAdd packet	AtomicETH
	10101 - 11111	Reserved	undefined

6 Division of Labor

The Previous Chapter

The previous chapter provided an introduction to the five types of message transfer requests that can posted to and executed by a QP's SQ logic. It also introduced the single type of message transfer request that can be posted to and executed by a QP's RQ logic.

This Chapter

This chapter provides an introduction to the layers comprising the IBA stack that handles message transmission and reception. Those layers are:

- The ULP (Upper Layer Protocol). This is actually not part of the IBA stack. Rather, it is comprised of OS and application software that uses the stack to pass messages.
- The Verb Layer.
- The Transport Layer.
- The Network Layer.
- The Link Layer.
- The Physical Layer.

The Next Chapter

The next chapter defines how ports on channel adapters, switches, and routers are numbered. It then defines the Local ID (LID) address space, the purpose of the LID address, and the SM's assignment of a unique LID to each port. The SM may optionally assign a range of LID addresses to a port. This chapter defines how this is done as well as the value of assigning more than one address to a port. It describes how the QP indicates which of the local ports assigned LID addresses will be inserted into the LRH:SLID field when the QP sources a packet to the port for transmission. The concept of the SM's path database is introduced.

Introduction

Refer to Figure 6-1 on page 109 and Figure 6-2 on page 110. The task of creating a message to be sent, dividing it into packets and delivering them to their ultimate destination is performed by the following layers:

- Upper Layer Protocols (ULPs).
- Verbs Layer.
- Transport Layer (i.e., QPs and EECs).
- Network Layer.
- Link Layer.
- Physical Layer.

The following sections provide an introduction to the layers.

Brief Layer Descriptions

Upper Layer Protocols (ULP—i.e., software)

A software application or driver builds the message data and transfer request in system memory. It then posts a message send or receive request to a local QP's SQ or RQ. It subsequently handles notice of successful completion or any errors that may be reported back by the verbs or by the transport layer.

Verb Layer

A software application calls the verbs to perform operations such as:

- HCA management.
- QP management.
- EEC management.
- Memory Region and Window management.
- Post work requests to send and receive queues.
- Handle message transfer completions.
- Handle unexpected events.

Transport Layer

The Transport Layer is responsible for sending and receiving messages across the fabric between QPs. When necessary, it segments a message into multiple packets (when the message's total length is greater than the PMTU of the path between the source and destination ports). The QP on the receiving end reassembles the message data into the specified data buffer(s) in its memory.

Network Layer

The Network Layer in a router handles the routing of a packet between subnets (using the packet's GRH). A packet contains a GRH when the source and destination are in different subnets (or if it's a multicast UD packet). In a CA, the Network Layer performs the following functions:

- **On packet transmission**: When a request packet is received from the Transport Layer for transmission and the request packet's destination resides in a different subnet, the CA's Network Layer inserts the GRH in the packet. Otherwise, it just passes the packet to the source port's Link Layer for transmission.
- **On packet receipt**: When a request or a response packet is received from the Link Layer, the Network Layer silently discards (with the exception of incrementing any applicable performance counters) any packet with a GRH that does not meet either of the following conditions:
  - The GRH:IPVer field must contain 6 (indicating Internet Protocol Version 6).
  - The packet's GRH:DGID must be one of the GID addresses assigned to the port that received the packet.

  If both of these conditions are met, the Network Layer presents the packet to the Transport Layer for processing. If a packet does not contain a GRH, the Network Layer just passes the packet to the Transport Layer (i.e., to the destination QP or EEC).

Link Layer

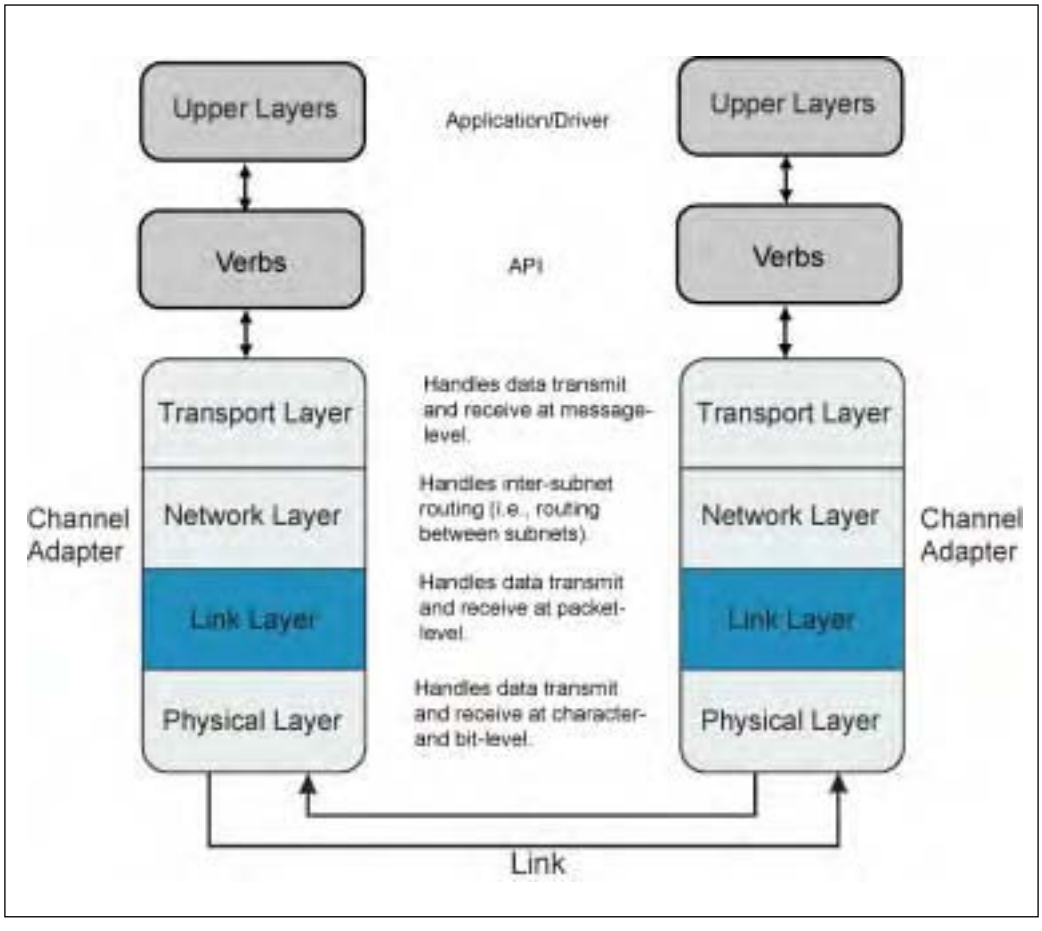
The Link Layer is responsible for sending and receiving data across the fabric at the packet, rather than the message, level. It has the following responsibilities:

- **Addressing**:
  - — On transmission, the Link Layer receives the packet from the CA's Network Layer and uses the Source Path Bits supplied by the Transport Layer to insert the correct port LID address into the packet's LRH:SLID field.
  - — On receipt of a packet from the port's Physical Layer, the port's Link Layer performs an address decode of the packet's DLID field to determine if it is the target port. If it is, the packet is passed to the CA's Transport Layer for processing.
- **Buffering**. A port's Link Layer contains transmit and receive buffer pairs (referred to as Virtual Lane buffer pairs).
- **Flow Control**. The port's Link Layer handles the issuance of link-level Flow Control packets from its data VL receive buffers. Its data VL transmit buffers receive and process link-level Flow Control packets sent by the data VL receive buffers on the other end of the link (a detailed description of link-level Flow Control can be found in "Link-Level Flow Control" on page 637).
- **Error detection**.
- **Packet switching**. In a switch, each port's Link Layer performs a lookup in the switch's Forwarding Table attribute using the packet's DLID field to determine through which of its ports the packet must be forwarded.

Physical Layer

The physical layer is responsible for sending and receiving data across the fabric at the character- and bit-level (rather than the message- or packet-level). It defines how bits are placed on the wire to form symbols (i.e., characters) and defines the symbols used for framing (i.e., the start-of-packet and end-of-packet symbols), for data characters, and the fill characters used between packets (Idles). It defines the symbol encoding, the proper alignment of the packet framing symbols, disallows invalid and non-data symbols between the start- and end- packet delimiters, and defines disparity errors, as well as the methods used to sync up the transmitter and receiver on a link.

Figure 6-1: CA Layer Overview



7 Subnet-Local Addressing

The Previous Chapter

The previous chapter provided an introduction to the layers comprising the IBA stack that handles message transmission and reception. Those layers are:

- The ULP (Upper Layer Protocol). This is actually not part of the IBA stack. Rather, it is comprised of OS and application software that uses the stack to pass messages.
- The Verb Layer.
- The Transport Layer.
- The Network Layer.
- The Link Layer.
- The Physical Layer.

This Chapter

This chapter defines the addresses that are used to route a packet from its source port to its destination port within an IBA subnet. It defines how ports on CAs, switches, and routers are numbered. It then defines the Local ID (LID) address space, the purpose of the LID address, and the SM's assignment of a unique LID to each port. The SM may optionally assign a range of LID addresses to a port. This chapter defines how this is done as well as the value of assigning more than one address to a port. It describes how the QP indicates which of the local port's assigned LID addresses will be inserted into the LRH:SLID field when the QP sources a packet to the port for transmission. The concept of the SM's path database is introduced.

The Next Chapter

The next chapter describes the 128-bit global address used to route a packet from its source CA port in one subnet to the destination CA port in another subnet. IPv6 (Internet Protocol version 6) addresses are introduced (both unicast

and multicast) as well as the documentation convention used for IPv6 addresses. Each 128-bit global address consists of a 64-bit subnet prefix and a 64-bit globally unique ID (GUID). This chapter describes how each port in a subnet is assigned the same subnet ID as well as one or more GUIDs. The concept of a global multipathing is introduced.

Port Numbering

The following rules govern the number of (and the numbering of) ports that may be implemented on CAs, switches, and routers:

- **CA**:
  - — As few as one or as many as 255 ports can be implemented on a CA.
  - — They are numbered starting at one.
  - — Each CA port must be assigned at least one 16-bit LID address by the SM at startup time. Each port must have a LID address that is unique within the subnet. The SM may optionally assign a range of LID addresses to a CA port. For more information, see "Why Assign a LID Range to a Port?" on page 137.
- **Router**:
  - — A router may implement as few as two or as many as 255 ports. Although the specification doesn't say the lower limit is two ports, a router with one port would be a deadend. Consider that a router's job is to route packets received on a port attached to a link in one subnet through a port attached to a link within another subnet.
  - — As with a CA, the ports are numbered starting at one.
  - — Each router port must be assigned a 16-bit LID address by the SM at startup time. Each port must have a LID address that is unique within the subnet. The SM may optionally assign a range of LID addresses to a router port.
- **Switch**:
  - — A switch may implement as few as three or as many as 255 ports. Although the specification doesn't say the lower limit is three ports, consider the following:
    - – It is a requirement that all switches must implement port 0. This is the switch's management port through which managers (e.g., the SM) can access the switch's attributes. It's typically implemented as an internal port with no physical attachment to a link. The specification, however, doesn't rule out implementing a physical link as well.

  - A switch's job is to relay packets received from a port attached to one link in the subnet through a port attached to another link in the same subnet. This being the case, a minimalist switch design would have two ports (in addition to the management port, port 0).
- — As with a CA, the ports (other than port 0) are numbered starting at one.
- — *The only switch port that is assigned a LID address is its management port, port 0.*
- — *Switch port 0 is only assigned a single LID address. It is never assigned a range of LID addresses.*

LID Address Space

The LID assigned to a port is a 16-bit value assigned by the SM. This means there are 64K LIDs available for assignment to the ports that populate a specific subnet. The 64K overall LID address range is subdivided as follows:

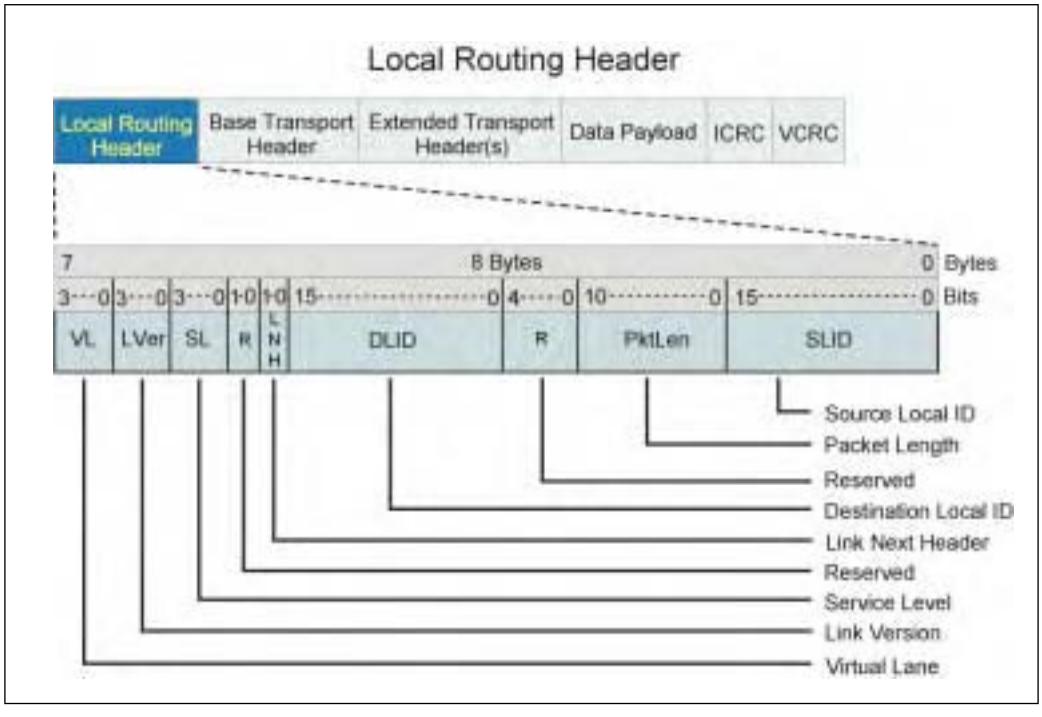
- LID address 0000h is **reserved** and must never be used.
- LID addresses **0001h–BFFFh** (a 48K minus 1 address range) are set aside for assignment as **unicast addresses**. A packet with a unicast DLID is always delivered to the single target port that is assigned that address. It is never delivered to more than one port. When a switch receives a packet wherein the DLID field contains a unicast LID address, it consults its unicast Forwarding Table and forwards the packet through the port indicated by the matching entry. For more detail on unicast packet forwarding, refer to "Switch Unicast Packet Forwarding" on page 669.
- LID addresses **C000h–FFFEh** (a 16K minus 1 address range) are set aside for assignment as **multicast addresses**. Usage of a multicast DLID permits a packet to be broadcast to multiple destinations. When a switch receives a packet wherein the DLID field contains a multicast LID address, it consults its Multicast Forwarding Table and forwards the packet through all of the ports indicated by the selected table entry. A detailed description of multicast operations can be found in "Switch Multicast Packet Forwarding" on page 675 and "Multicasting" on page 563.
- LID address **FFFFh** is referred to as the **Permissive LID (PLID)** and has a special use during the discovery process. A detailed description of PLID usage can be found in the chapter entitled "Discovery" on page 871.

LID's Purpose: Packet Routing Within Subnet

Assigning a LID to a port permits it to be addressed as the target of a packet. In Figure 7-2 on page 136 a packet is sent from a port on one TCA to a destination port on another TCA:

1. The source TCA composes the packet including a Local Route Header (LRH; see Figure 7-1 on page 135) with the DLID and SLID values set as follows:
   — LRH:DLID set to one of the LID addresses assigned to the destination TCA port.
   — LRH:SLID set to one of the LID addresses assigned to the TCA port sourcing the packet into the fabric.
2. The source TCA injects the packet onto the link.
3. The packet arrives at a port on the first switch in the path to the destination port.
4. The switch port's Link Layer examines the packet's LRH:DLID to determine if the address is a unicast or multicast address:
   — If it's a unicast address (it is within the range from 0001h–BFFFh), then the port's Link Layer performs a lookup in its Forwarding Table (an attribute that was set up by the SM earlier in time). The DLID selects a table entry that indicates the switch port through which the packet is to be output.
   — If it's a multicast address (it is within the range from C000h–FFFEh), then the port's Link Layer behaves as described in "IBA and Raw Packet Multicast" on page 124.
5. As can be seen in the illustration, the packet traverses a total of five switches before arriving at the destination TCA port.
6. Upon arrival at the destination TCA port, the port's Link Layer logic decodes the packet's DLID field and determines that it is the destination port. The packet is therefore passed to the TCA's Network Layer for additional processing.

Figure 7-1: Intra-Subnet Packet Format



8 Global Addressing

The Previous Chapter

The previous chapter defined the LID addresses that are used to route a packet from its source port to its destination port within an IBA subnet. It defined how ports on CAs, switches, and routers are numbered. It then defined the Local ID (LID) address space, the purpose of the LID address, and the SM's assignment of a unique LID to each port. The SM may optionally assign a range of LID addresses to a port. This chapter defined how this is done as well as the value of assigning more than one address to a port. It described how the QP indicates which of the local port's assigned LID addresses will be inserted into the LRH:SLID field when the QP sources a packet to the port for transmission. The concept of the SM's path database was introduced.

This Chapter

This chapter describes the 128-bit global address used to route a packet from its source CA port in one subnet to the destination CA port in another subnet. IPv6 (Internet Protocol version 6) addresses are introduced (both unicast and multicast) as well as the documentation convention used for IPv6 addresses. Each 128-bit global address consists of a 64-bit subnet prefix and a 64-bit globally unique ID (GUID). This chapter describes how each port in a subnet is assigned the same subnet ID as well as one or more GUIDs. The concept of global multipathing is introduced.

The Next Chapter

The next chapter introduces the Subnet Manager (SM), the Subnet Management Agent (SMA), the General Services Managers (GSMs), the General Services Agents (GSAs), the Subnet Administrator (SA), the Subnet Management Interface (SMI), the General Services Interface (GSI), Subnet Management Packets (SMPs), and General Services Management Packets (GMPs). The possibility of multiple SMs in one subnet is introduced, along with the concept of GMP redirection, how a trap can deliver an event notification to a manager, and how an entity can subscribe to a manager for the forwarding of event notifications.

Global Routing: Source/Destination CAs in Different Subnets

Refer to Figure 8-2 on page 144. In this example scenario, the packet is originated by a CA in one subnet and is targeting a CA in another subnet. The packet must transit two routers to get to its destination.

LID Only Permits Addressing Port in Same Subnet

Using a LID to address the destination port only permits another port within the same subnet to be addressed. In the example, however, the destination port resides in another subnet. It should be obvious that additional address information is required when a packet targets a port in another subnet. The packet's headers not only need to supply the address of the destination port, but must also identify the subnet within which that port resides.

Global Addressing Must Be Used

Refer to Figure 8-1 on page 143. A packet that targets a port in another subnet must, in addition to the LRH, include a Global Route Header (GRH). While the LRH contains the SLID and DLID, the GRH contains the Source Global ID (SGID) and Destination Global ID (DGID):

- **DGID**: identifies the port that is the ultimate target of the packet (item F in the example), as well as the subnet within which it resides.
- **SGID**: identifies the port that initially injected the packet into the fabric (item A in the example), as well as the subnet within which it resides.

Figure 8-1: Inter-Subnet Packet Format

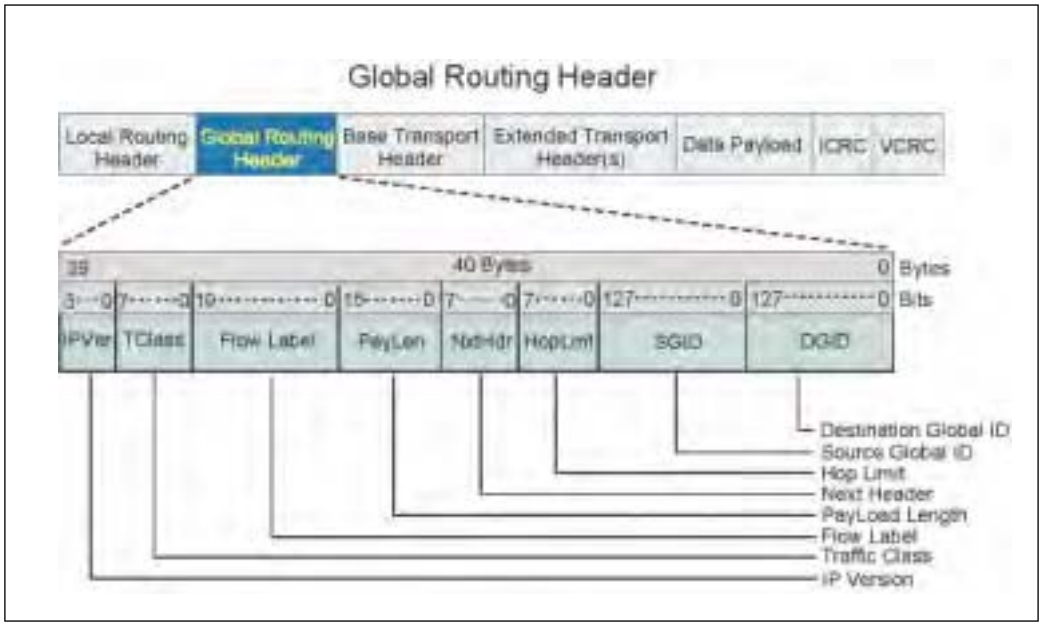
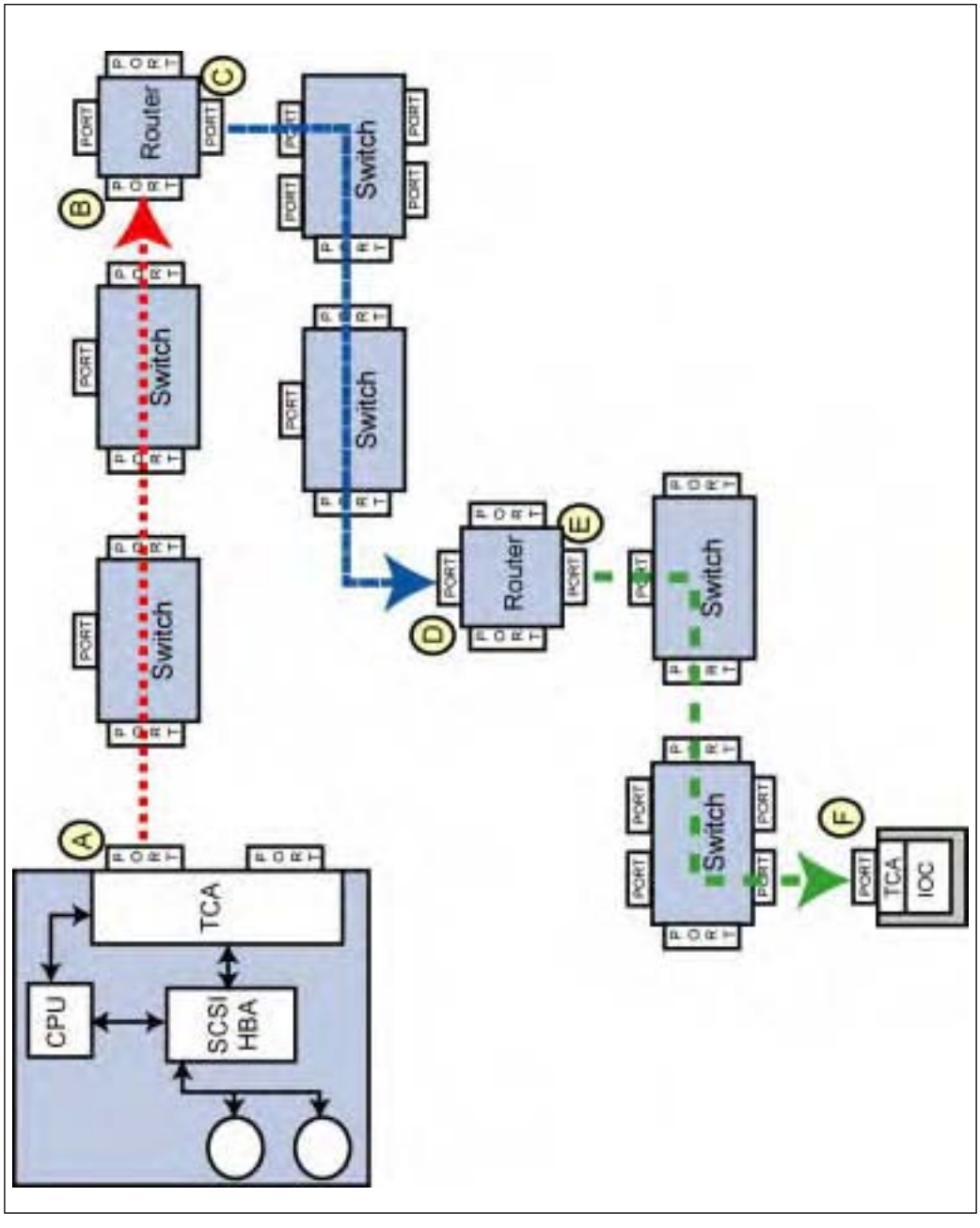


Figure 8-2: Inter-Subnet Packet Example



An Inter-Subnet Example

The example case (in Figure 8-2 on page 144) assumes that this is a unicast, rather than a multicast, operation. The following describes the packet's travel through the fabric:

1. When the TCA's port (item A) injects the packet onto the first link, it sets the addresses as follows:
   - LRH:SLID = LID of the CA's source port.
   - LRH:DLID = LID of the ingress port on the first router (item B). It acts as the packet's destination within the first subnet.
   - SGID =
     - Upper 64 bits is the Subnet ID of the subnet within which the source port (item A) resides.
     - Lower 64 bits is the GUID (Globally Unique ID; more on this later) of the source port.
   - DGID =
     - Upper 64 bits is the Subnet ID of the subnet within which the destination port (item F) resides.
     - Lower 64 bits is the GUID of the destination port.
2. Using the DLID, the Forwarding Tables within the first two switches relay the packet to the destination port within this subnet (i.e., item B, the ingress port on the first router).
3. Upon arrival at the target port on the first router, the router ingress port's logic decodes the DLID and recognizes that it is the packet's target within this subnet. The router takes the following actions:
   - Using the GRH:DGID field, the router performs a lookup in its internal Routing Table:
     - **Case 1**: If the target subnet (indicated by the Subnet ID portion of the DGID) is not directly connected to one of the router's ports, the Routing Table lookup is used to determine which of the router's ports is in the path to the target subnet. The lookup also produces the LID of an ingress port on the next router along the path to the target subnet. *This is the case in the current scenario*.
     - **Case 2**: If the target subnet (indicated by the Subnet ID portion of the DGID) matches the Subnet ID of one of the subnets connected directly to the router, the Routing Table lookup mechanism also uses the GUID portion of the DGID to determine the LID of the CA port targeted by the DGID address.

9 Intro to the Managers

The Previous Chapter

The previous chapter described the 128-bit global address used to route a packet from its source CA port in one subnet to the destination CA port in another subnet. IPv6 (Internet Protocol version 6) addresses were introduced (both unicast and multicast) as well as the documentation convention used for IPv6 addresses. Each 128-bit global address consists of a 64-bit subnet prefix and a 64-bit globally unique ID (GUID). This chapter described how all ports in a subnet are assigned the same subnet ID as well as one or more GUIDs. The concept of a global multipathing was introduced.

This Chapter

This chapter introduces the Subnet Manager (SM), the Subnet Management Agent (SMA), the General Services Managers (GSMs), the General Services Agents (GSAs), the Subnet Administrator (SA), the Subnet Management Interface (SMI), the General Services Interface (GSI), Subnet Management Packets (SMPs), and General Services Management Packets (GMPs). The possibility of multiple SMs in one subnet is introduced, as is the concept of GMP redirection, how a trap can deliver an event notification to a manager, and how an entity can subscribe to a manager for the forwarding of event notifications.

The Next Chapter

The next chapter starts by introducing the concept of services, how a CA is located, and the discovery of what services that CA provides. Having located a service provider, a communications channel must be established between the local application program and the service provider. The chapter provides a basic description of the process used to establish a communications channel between a RC, UC, RD, or UD QP type in the local CA and a QP of the same type in the CA that will provide the desired service.

The SM

Role of Subnet Manager

The SM is the software entity that performs the following functions:

- Discovers the topology of the subnet that it manages.
- Assigns a common Subnet ID (aka Subnet Prefix) to all of the ports in its subnet.
- Assigns an address to each port in its subnet (an address that is unique within that subnet). This address is referred to as a port's Local ID, or LID.
- Establishes the possible paths between all endnodes in the subnet.
- Sweeps (i.e., scans) its subnet on a regular basis looking for topology changes (devices added or removed, changes in the operational state of links, etc.).

Where Does the SM Reside?

The SM may reside within or behind any device in the subnet, but most typically would consist of a program running on a processor node.

How the SM Communicates With Devices

Subnet Management Interface and SMPs

Refer to Figure 9-3 on page 166. A special QP, QP0, is used by the SM:

- To send Subnet Management request Packets (SMPs) to QP0 on the target device. The request packet supplies the target device with:
  - — **Attribute ID**. Identifies the device attribute that some action (i.e., method) is to be performed on.
  - — **Method**. Specifies the action to be taken on the indicated attribute.
  - — **Attribute Modifier**. Some attribute/method combinations also require the specification of an attribute modifier. As an example, when sending a request to access a port's *PortInfo* attribute, the attribute modifier identifies the port on the destination device.

  - — **Data**. The usage of this field is defined by the attribute/method combination. As an example, on an attribute *Set* operation (i.e., a write to an attribute), the data field contains the data to be written into the attribute.
- To receive Subnet Management response Packets (SMPs). Upon receipt of an SMP request packet, the destination device performs the specified method on the specified attribute and then returns an SMP response packet to the SM. The response packet indicates the results of the requested action.

QP0 is referred to as the Subnet Management Interface (SMI). An SMP is a form of Management Datagram, or MAD.

Multiport CAs and Routers Implement Multiple SMIs

Each port on a CA or router implements an SMI (i.e., QP0) to receive SMP request packets and to transmit SMP response packets. If a CA or a router implements an SM either within or behind itself, the SM uses the SMI on one of the device's ports to send and receive SMPs.

Switch Implements One SMI on Port 0

On a switch, only switch port 0 (its management port) implements an SMI. An SMP request packet received on any other switch port with its DLID set to port 0's LID address is internally forwarded to port 0's SMI for processing. If the packet's DLID does not match port 0's LID address, the SMP packet is treated like any packet enroute to its final destination. The switch performs a lookup in its Forwarding Table using the packet's DLID to determine through which of its ports the packet will be forwarded.

It should be noted that an SM may be implemented within a switch. In this case, the SM uses the switch management port's SMI to send and receive SMPs.

Subnet Management Agent

Each CA, router, and switch implements one Subnet Management Agent (SMA). When an SMP request packet is received on any of the device's SMIs, the SMP is passed to the device's SMA for processing of the request. The SMA performs the operation defined by the SMP and then returns an SMP response packet to the SMI that had received the request SMP. The SMI transmits the response SMP back to the SM.

In the case where another SM sends an SMP to access the *SMInfo* attribute of the SM residing within a device, the SMI passes the SMP directly to the device's SM (rather than to its SMA) for processing.

An Example SM Attribute Access

The SM performs an attribute access in the following manner (refer to Figure 9-1 on page 164, Figure 9-2 on page 165, and to Figure 9-3 on page 166):

1. **Build MAD in memory**. The SM builds a 256-byte MAD in main memory (item C). The content of this MAD specifies the following:
   - Class = Subn (indicating that this MAD is to be delivered to the target port's SMA for processing).
   - Method = the action to be performed on the targeted attribute (e.g., Get or Set).
   - Attribute ID = ID of attribute to be acted upon.
   - Attribute Modifier (if applicable).
   - Data (if it's an attribute write).
2. **Post WR to QP0's SQ**. The SM executes a *Post Send Request* verb call (item D) to post a WR to the SQ (item J) of the QP0 associated with the port that it will use to transmit the SMP. QP0 is a special-purpose QP tasked with sending and receiving SMPs. The WR specifies the start memory address that contains the MAD to be sent. It also specifies the LID address of the destination port to which the SMP is to be delivered and to which QP behind that port (SMPs must be delivered to the port's QP0; this is the port's Subnet Management Interface, or SMI, QP).
3. **Post WR to QP0's RQ**. The SM will also execute a *Post Receive Request* verb call to the QP's RQ to handle the corresponding inbound response SMP that will be returned by the destination device's SMA.
4. **Packet Transmitted**. The QP0 SQ Logic (item L) reads the 256-byte MAD from main memory and places it in the data payload field of an SMP. The SMP is forwarded to the HCA port associated with this QP0 for transmit.
   - The BTH:Opcode field indicates that this is a Send-Only UD packet (all MADs are UD packets).
   - The packet's DLID is set to the LID address of the destination port.
   - The BTH:DestQP is set to QP0 (the destination port's SMI).
   - The VL field is set to 15 (SMPs always travel over VL15).
5. **Packet arrival at destination port**. The SMP traverses one or more links until it arrives at the destination port.
6. **Sent to port's SMI**. The packet is delivered to the RQ Logic (item Q) of the port's SMI (it's QP0).
7. **Sent to SMA**. The SMI sees that the MAD's Class = Subn, so it delivers the 256-byte MAD to the device's SMA (Subnet Management Agent; see Figure 9-3 on page 166) for processing.
8. **Attribute action performed**. The SMA performs the designated method (i.e., action) on the attribute indicated in the request SMP MAD.
9. **Response packet formed**. The SMA then forms an SMP response packet

and sends it back to the receiving port's SMI for transmission. The SLID and DLID fields that were in the SMP request packet are reversed in the response packet's LRH:SLID and LRH:DLID fields for the return journey. The BTH:DestQP field is set to QP0 (the HCA port's SMI QP).

10. **Response delivered to SMI**. The packet is output onto the link, travels the same path back to the HCA port, and is delivered to that port's SMI (its QP0) RQ Logic for processing.
11. **Response deposited in memory**. The SMI's RQ Logic uses the Scatter Buffer List in the top RQ WQE to determine where the 256-byte MAD response is to be deposited in memory (item C) and a CQE is created on the CQ (item F) associated with QP0's RQ. The completion status of the operation can be found in the CQE's Status field:
    - **Status field**: 16-bit field. Operation completion status. Only valid in a response MAD generated by the target in response to a request:
        - Lower 8 bits = status common to all classes. These are:
            - Busy, MAD discarded (not an error).
            - Redirect required (not an error).
            - Class not supported.
            - Method not supported.
            - Method/Attribute combination not supported.
            - One or more Attribute fields contain bad values.
        - Upper 8 bits = class-specific status.
12. **SM informed**. An interrupt is generated by the HCA to tell the SM that the response is in memory.

10 Intro to Connection Establishment

The Previous Chapter

This chapter introduced the Subnet Manager (SM), the Subnet Management Agent (SMA), the General Services Managers (GSMs), the General Services Agents (GSAs), the Subnet Administrator (SA), the Subnet Management Interface (SMI), the General Services Interface (GSI), Subnet Management Packets (SMPs), and General Services Management Packets (GMPs). The possibility of multiple SMs in one subnet was introduced, the concept of GMP redirection, how a trap can deliver an event notification to a manager, and how an entity can subscribe to a manager for the forwarding of event notifications.

This Chapter

This chapter starts by introducing the concept of services, how a CA is located, and the discovery of what services that CA provides. Having located a service provider, a communications channel must be established between the local application program and the service provider. This chapter provides a basic description of the process used in establishing a communications channel between a RC, UC, RD, or UD QP type in the local CA and a QP of the same type in the CA that will provide the desired service.

The Next Chapter

The next chapter provides a detailed discussion of how PSNs in request packets are generated by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic) and how they are verified by the responder QP's RQ Logic (or, in RD, the responder EEC's Receive Logic). It describes how the Start PSN is assigned to one QP's SQ Logic (or, in RD, one EEC's Send Logic) and the expected PSN (ePSN) is assigned to the other QP's RQ Logic (or, in RD, the other EEC's Receive Logic). It describes how, when using RD, RC, and UC the

PSN in each response packet is verified by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic). The concepts of valid and invalid PSNs, as well as duplicate requests and responses are introduced. This chapter concludes the introductory portion of the book (i.e., Part 1).

Questions Addressed in This Chapter

An earlier chapter, "Intro to Transport Types" on page 61, provided an introduction to the operational characteristics of the four IBA transport types: RC, UC, RD, and RD. In the descriptions of the RC, UC, and RD transport types, reference is made to the requirement for establishing a connection (i.e., a communications channel) between two CAs before messages can be transferred. For example, the following is the first item from "Reliable Connected Service Type" on page 64:

> "Before any messages may be transferred, a connection must be established between the RC QPs in the two CAs, and the QP Contexts of the two QPs are each programmed with the identity of the remote QP as well as the address of the port behind which the remote QP lives. "

The following important questions are addressed in this chapter:

- Why exchange messages with a CA?
- How can a specific CA be located?
- Having located a CA, how do you find out what services it provides?
- How is a communications channel established with a specific service the CA provides? Specifically:
  - — How does software set up a local QP and, if using RD, a local EEC?
  - — How does software cause the remote CA to create and setup a QP (of the same type), and if RD, an EEC?
  - — For RC/UC, how are the QP Contexts of the local and remote QPs each programmed with the identity of the other CA's QP, as well as the address of the port behind which the other CA's QP resides?
  - — For RD, how are the EECs in the two CAs each programmed with the identity of the EEC in the other CA, as well as the address of the port behind which the other CA's EEC lives?

A CA Is a Provider of Services

A CA exists for a reason. It provides the interface between the IBA fabric and back-end logic designed to provide one or more types of services to other enti-

ties. In this context, it can be thought of as a Server. This concept breeds a series of questions:

- How do other entities (CAs acting as Clients, so to speak) issue requests to a CA? The answer is: by passing messages to it.
- How does the CA supply the results back to the client CA? The answer is: by sending it messages.
- How are messages passed between the two? As we already know, messages are exchanged using QPs within the two CAs.
- If a CA provides more than one type of service to its clients, then it probably provides each specific type of service through a specific QP (and the CA would implement multiple QPs).

Locating a Specific CA

During its initial probing of the subnet, the SM discovers each device, its operational characteristics, and its identity (see "Each Device Has Device-Level Identifiers" on page 158). After assigning addresses to each port on a CA or router, and to the management port on a switch, the SM builds the topology database known as the SA. When software wishes to determine if a particular device exists in the subnet, it may issue a query to the SA in the form of a *SubnAdmGet(NodeRecord)* GMP. In the GMP, software provides the SA with the device GUID (i.e., *NodeInfo.NodeGUID*) that it is looking for. If the SA has one or more *NodeRecords* that contain the desired *NodeGUID* value, it returns those *NodeRecords* to software. It should be noted that if the device has multiple ports, then the SA will return one *NodeRecord* for each port. Each *NodeRecord* contains the following elements:

- The device's *NodeInfo* attribute.
- The device's *NodeDescription* attribute.
- The LID address assigned to the port.

Discovering Services a CA Provides

Step One: Discover Occupied IO Controller Slots in IOU

Assuming that the CA just discovered is a TCA within an IOU, software can then issue a *DevMgtGet(IOUnitInfo)* GMP (Device Management read of the

IOUnitInfo attribute) to discover the number of slots that exists in the IOU and which ones are currently occupied by IOCs.

Step Two: Obtain Number of Services IOC Supports

This is accomplished by reading the *IOControllerProfile.ServiceEntries* attribute for each of the IOCs that are present.

Step Three: Obtain List of Services Supported by Each IOC

This is accomplished by reading the *ServiceEntries* attribute for each of the IOCs that are present. This attribute consists of a list of all the services provided by the respective IOC. Each entry contains:

- The *ServiceName*. This the 40-byte text name of the service.
- The *ServiceID*. This is the 64-bit ID of that service.

Step Four: ServiceID Provided in Connection Request

As will be seen later in this chapter, the ServiceID of the service to establish a communications channel with is supplied to the remote CA during the connection establishment process.

RC/UC Connection Establishment

Figure 10-1 on page 193 illustrates the process of establishing a connection between two RC or UC QPs. It should be noted that, although the dialogue shown is 100% correct, it does not include all of the information that is exchanged in the REQ and REP messages.

Local QP Initial Creation and Setup

Refer to Figure 3-4 on page 58.

CQ(s) and PD Created Before Creating the Local QP

CQ Creation. Before creating the local QP, software must first cause the HCA to create the CQ(s) to be associated with the SQ and RQ of the QP about to be created. The QP's SQ and RQ can have separate CQs or may share a CQ. To create one or two CQs, software executes the *Create CQ* verb call once or twice. When calling this verb, software provides the following input parameters:

— HCA identifier (referred to as the HCA Handle).
— The desired size of the CQ to be created.

The verb layer commands the HCA to create a CQ of the requested size. The verb then returns the following output parameters:

— Handle that identifies the newly created CQ.
— The actual size of the CQ (the HCA may or may not support a CQ of the requested size; if it doesn't, it creates a CQ of a smaller size and the verb returns the actual size to the caller).

PD Creation. In addition to the CQ(s) to be associated with the QP about to be created, the PD (Protection Domain) that the QP will belong to must also have been created. A PD is created by executing the *Allocate PD* verb. The only input parameter is the HCA handle, and the only output parameter is the PD value.

Create the Local QP

See step one in Figure 10-1 on page 193. Software causes a QP to be created in a HCA by executing a *Create QP* verb call. In response, the verb layer commands the HCA Interface to create a QP of the desired type. In this case, a RC or UC QP is created.

Create QP Input Parameters. When calling the *Create QP* verb, the following information is provided as input parameters (note that this a partial list):

— The HCA Handle.
— The type of QP (in this case, RC or UC).
— Handles of the CQs to be associated with the QP's SQ and RQ.

11 PSN Usage

The Previous Chapter

The previous chapter started by introducing the concept of services, how a CA is located, and the discovery of what services that CA provides. Having located a service provider, a communications channel must be established between the local application program and the service provider. This chapter provided a basic description of the process used in establishing a communications channel between a RC, UC, RD, or UD QP type in the local CA and a QP of the same type in the CA that will provide the desired service.

This Chapter

This chapter provides a detailed discussion of how PSNs in request packets are generated by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic) and how they are verified by the responder QP's RQ Logic (or, in RD, the responder EEC's Receive Logic). It describes how the Start PSN is assigned to one QP's SQ Logic (or, in RD, one EEC's Send Logic) and the expected PSN (ePSN) is assigned to the other QP's RQ Logic (or, in RD, the other EEC's Receive Logic). It describes how, when using RD, RC, and UC the PSN in each response packet is verified by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic). The concepts of valid and invalid PSNs, as well as duplicate requests and responses, are introduced. This chapter concludes the introductory portion of the book (i.e., Part 1).

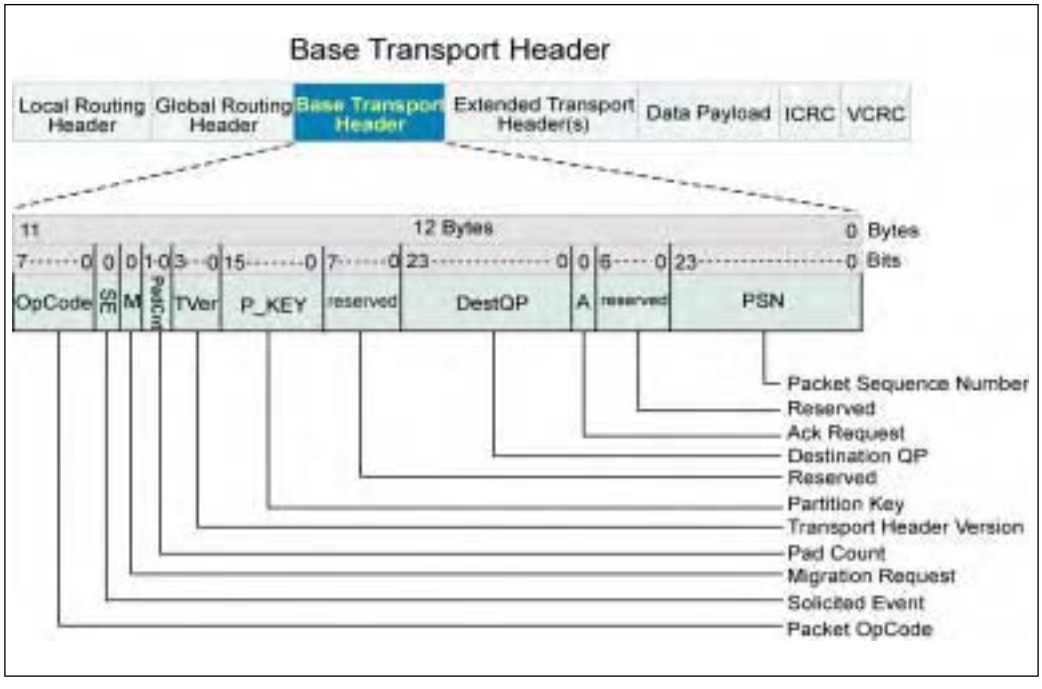
The Next Chapter

The next chapter provides a detailed description of the verbs used to create a QP, modify its operational characteritics, obtain its current operational characteristics, and destroy it. It provides a detailed description of the creation of a QP and the steps required to program it. The QP state machine is described in detail.

Overall Size of PSN Range

Refer to Figure 11-1 on this page. As mentioned earlier, each packet contains an identifier known as the PSN. It is a 24-bit field, so the overall size of the PSN space is 16M (2^{24}).

Figure 11-1: Base Transport Header



Requester QP's SQ Logic PSN Generation and Verification

Start PSN Assignment

General

Refer to Figure 11-2 on page 212. When a QP is created, one of the values that must be set is the SQ Logic's Start PSN value. This is the PSN that will be inserted into the first request packet generated by the SQ Logic. It may be any value in the range from 000000h through FFFFFFh.

Select a Start PSN Value That Precludes Stale Packets

The specification stresses that a value should be selected that:

> "minimizes the chance that a packet from a previous connection could fall within the valid PSN window."

The specification is referring to either a request packet received by a QP's RQ Logic or a response packet received by a QP's SQ Logic in response to a request packet issued at an earlier time. The "valid PSN window" is the range of PSNs up to 2^{23} (8M) in size consisting of the Ack'd PSN range (also referred to as the duplicate response region) plus the unAck'd region (i.e., up to the PSN of the request packet most recently transmitted by the SQ Logic).

When a QP is created, the verb layer assigns it a 24-bit QP Number (QPN). It is possible that a QP that had been in use earlier in time had the same QPN assigned as a new one just created and in the process of being set up. Alternately, the connection management software might use the *Modify QP* verb call to reset a local and remote QP that have been actively sending messages to each other.

The SQ Logic of earlier QPs had been assigned Start PSNs and, during their lifetimes, had transmitted a number of request packets with ascending PSNs in each packet. Subsequently, those QPs may have been deleted (i.e., destroyed) or reset and then set up again while they still had some request packets, or their respective responses, in flight. These orphan request and/or Ack packets are referred to as "stale" packets.

When a new QP and a remote QP that it communicates with are then created (or are reset) and put into play, if their respective SQ Logics were assigned Start PSNs that fell within the range of PSNs of any in-flight orphan requests and Acks, the orphan requests or Acks could be mistaken for valid requests or valid responses to requests issued by this new QP. The specification therefore urges software to assign a Start PSN to a new QP's SQ Logic so that any possible orphan requests or Acks that may be received by a QP's RQ Logic or SQ Logic will fall within the Invalid range (i.e., before the Start PSN or after the PSN in the most recently issued request packet).

No Protection From Stale Packets

If the connection management software were to tear down (i.e., destroy) the connection between two QPs and then chooses to reuse the two QPs too soon (i.e., before enough time has passed for any stale request and/or response pack-

ets to be dropped in the fabric), stale request or response packets with PSNs that fall within the valid areas might arrive at a QP's SQ Logic or RQ Logic. They will be perceived as valid request or response packets and will mortally screw things up.

If this is permitted to happen, there is no protection against it. It is the responsibility of the connection management software to avoid reusing QPs for a period long enough to ensure that there is no possibility that a stale packet could arrive at the SQ Logic or RQ Logic. The specification refers to this period of time as the "Time Wait" period.

Request Packet PSN Generation

As it transmits request packets, the SQ Logic assigns each request packet a PSN based on the following rules:

- The first request packet's PSN = the Start PSN assigned to the SQ Logic.
- For each subsequently issued request packet:
  - — If the previously issued request packet was not an RDMA Read request packet, then the PSN inserted in the current request packet = the previous PSN + 1.
  - — If the previously issued request packet was an RDMA Read request packet, then the PSN inserted in the current request packet = PSN of the RDMA Read request packet + the number of expected RDMA Read response packets.

Ack'd (aka Duplicate) Region

As the SQ Logic issues request packets and subsequently receives responses for each of them, the upper end of the Ack'd region grows upward.

Maximum Number of Outstanding Requests

As it issues each request packet, the RC QP's SQ Logic (or EEC Send Logic) doesn't wait for the respective response packet before issuing the next request packet. It continues to stream request packets and, eventually, it will begin to receive the stream of corresponding response packets. The maximum number of outstanding unAck'd request packets (of all types—Sends, RDMA Reads and Writes, and Atomic requests) that the SQ Logic is permitted to transmit before it

must stall is 8M (2^{23}; see the unAck'd PSN range in Figure 11-2 on page 212). This is based on a worst-case scenario with the following characteristics:

- The requester QP's SQ Logic initiates an RDMA Write or a Send operation to write a 2GB message into the remote CA's local memory.
- Packets transmitted between the source and destination ports can not have a data payload bigger than that specified as the path's PMTU (see "Maximum Data Payload Size" on page 42.
- Assuming that the path PMTU were 256 bytes, a 2GB message transfer would consist of 8,388,608 request packets (for a Send or RDMA Write) or 8,388,608 RDMA Read response packets.
- The requester QP's SQ Logic starts transmitting the request packets.
- If the requester had transmitted all 8,388,608 request packets associated with the message and had not yet received any Acks for the message, it must stall at that point and not issue any additional request packets (for a new message transfer) until it starts receiving Acks.

Ack Verification

As the SQ Logic receives each response packet, it checks to make sure that it falls within the unAck'd region. Assuming it does, the SQ Logic advances the upper end of its Ack'd region accordingly.

Detecting Duplicate Acks

General

As will be seen later in the book (e.g., see "Packet Delivery Delays" on page 389), in certain circumstances the SQ Logic will retransmit (i.e., retry) a request packet. As a result, the remote QP's RQ Logic may receive a duplicate copy of a request after it had already issued a response to the first copy. The SQ Logic would end up receiving a second copy of the response packet with the same PSN as that received in the first copy of the response packet. Upon receipt of the first copy of the response, the SQ Logic would have advanced the upper end of its Ack'd region, so the second copy of the response packet has a PSN that falls within the Ack'd region (i.e., the duplicate response region). The SQ Logic is required to handle a duplicate response in a graceful manner by quietly dropping it.

12 QP Verbs and QP State Machine

The Previous Chapter

The previous chapter provided a detailed discussion of how PSNs in request packets are generated by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic) and how they are verified by the responder QP's RQ Logic (or, in RD, the responder EEC's Receive Logic). It described how the Start PSN is assigned to one QP's SQ Logic (or, in RD, one EEC's Send Logic) and the expected PSN (ePSN) is assigned to the other QP's RQ Logic (or, in RD, the other EEC's Receive Logic). It described how, when using RD, RC, and UC the PSN in each response packet is verified by the requester QP's SQ Logic (or, in RD, the requester EEC's Send Logic). The concepts of valid and invalid PSNs, as well as duplicate requests and responses, were introduced. The previous chapter concluded the introductory portion of the book (i.e., Part 1).

This Chapter

This chapter provides a detailed description of the verbs used to create a QP, modify its operational characteristics, obtain its current operational characteristics, and destroy it. It provides a detailed description of the creation of a QP and the steps required to program it. The QP state machine is described in detail.

The Next Chapter

The next chapter provides a detailed description of how Work Requests (WRs) are posted to a QP's SQ or RQ, as well as a detailed description of the WR content. Once a WR has been posted to a QP's SQ or RQ, it is referred to as a WQE (Work Queue Entry; pronounced as "wookie"). This chapter describes the ordering rules that govern:

- WQE execution order.
- WQE completion order.

- RD RQ WQE completion order. The reason why RD RQ WQEs can complete out of order is described.
- RDMA Read relaxed ordering rules.

When a message transfer request WQE completes execution, a Completion Queue Entry (CQE; pronounced as "cookie") is created on the respective work queue's Completion Queue (CQ). This chapter provides a detailed description of the verbs used to create a CQ, resize it, obtain its current operational characteristics, and destroy it. It also covers:

- The completion event handler.
- The *Set Completion Event Handler* verb.
- The *Poll for Completion* verb.
- The *Request for Completion Notification* verb.
- The detailed content of a CQE.
- Solicited and unsolicited events.

QP-Related Verbs

General

The verbs related to QP creation, management, and destruction are:

- *Allocate RDD.*
- *Create QP.*
- *Modify QP.*
- *Query QP.*
- *Destroy QP.*
- *Deallocate RDD.*

The sections that follow provide a description of each verb.

Allocate and *Deallocate RDD* Verbs

Execution of the *Allocate RDD* verb is only necessary with relation to RD QPs. Refer to "Allocate RDD Verb" on page 506 and "Deallocate RDD Verb" on page 513.

Create QP Verb

The *Create QP* verb is executed to create a QP that supports one of the four IBA transport service types. It's **input parameters** are:

- The HCA handle that identifies which HCA the QP is to be created on. The handle is returned upon execution of the *Open HCA* verb.
- The handle(s) of the CQ(s) to be associated with QP's SQ and RQ. The handle is returned upon execution of the *Create CQ* verb.
- The maximum number of outstanding WRs software expects to post to the SQ and the RQ.
- The maximum number of Scatter Buffers or Gather Buffers software will specify in any WR posted to the SQ or the RQ.
- RDD to be associated with this QP (note that this is only applicable to a RD QP).
- The Signaling Type must be specified for the QP's SQ. The valid types are:
  - — All Work Requests posted to the SQ always generate a CQE on completion.
  - — Software specifies in each WR posted to the SQ whether or not to generate a CQE for a successful completion.
- The PD that the QP is a member of (note that the PD must have been previously created by executing the *Allocate PD* verb).
- The IBA transport service type requested for this QP. The valid service types are RC, RD, UC, and UD.

The **output parameters** returned by the verb:

- The handle for the newly created QP. The handle is used as an input parameter for subsequent executions of the *Modify QP, Query QP*, or *Destroy QP* verbs.
- QP number (QPN). This is the 24-bit QPN assigned to the QP by the verb. The QPN is used for the following purposes:
  - — During the connection establishment process for the RC and UC transport service types, the QPN of the HCA QP is provided to the remote CA's CM in the communications REQ message.
  - — When posting a WR to the SQ of a RD or UD QP, the QPN of the destination QP is provided in the WR.
- The actual number of outstanding SQ WQEs supported. If an error is not returned, this is guaranteed to be greater than or equal to the number requested (this may require software to increase the size of the SQ's CQ).
- The actual number of outstanding RQ WQEs supported. If an error is not returned, this is guaranteed to be greater than or equal to the number

requested (this may require software to increase the size of the RQ's CQ).
- The actual maximum number of Scatter or Gather Buffers that can be specified in a WR submitted to the SQ or the RQ. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.

Modify QP Verb

General

The *Modify QP* verb is executed for one of the following reasons:

- To provide a newly created QP with operational parameters.
- To update the operational characteristics of an already operational QP, but leave the QP's current state unchanged.
- To programmatically change the operational state of a QP. At the same time, software may optionally alter one or more of the QP's current operational characteristics.

The *Modify QP* verb's input parameters are listed in Table 12-1 on page 223.

Rules Regarding Software-Initiated State Change

When software executes the *Modify QP* verb, the QP's current state and the next state specified in the verb call define some input parameters as required and others as optional. Table 12-1 on page 223 defines the optional versus required input parameters. The following list provides a table summary and some additional clarification (note that in all cases, the HCA handle, the QP handle, and the Next State input parameters are required):

- **Reset-to-Init** state change. Some parameters are required (see Table 12-1 on page 223) and no optional parameters are allowed.
- **Init-to-RTR** state change. There are both required and optional parameters.
- **RTR-to-RTS** state change. There are both required and optional parameters.
- **Stay in RTS** state (no change). There are no required input parameters and there are some optional parameters.
- **SQE-to-RTS** state change. There are no required input parameters and there are some optional parameters.
- **RTS-to-SQD** state change. There are no required or optional input parameters.
- **SQD-to-RTS** state change. There are no required input parameters and there are some optional parameters.

- **Any state-to-Error** state change is permitted. There are no required or optional input parameters.
- **Any state-to-Reset** state change is permitted. There are no required or optional input parameters.
- **Error-to-Reset** state change. This is the only state change software is permitted to command after a QP has entered the Error state. There are no required or optional input parameters.

Table 12-1: Modify QP Verb Input Parameters

Parameter	Applicable Service Type	Comments	Required, Optional, or Not Allowed?
HCA handle	All	Identifies the HCA on which the QP resides. Returned by an earlier execution of the *Open HCA* verb.	Required.
QP handle	All	Identifies QP whose characteristics are to be updated. Returned by the execution of the *Create QP* verb.	Required.
Next QP state	All	• If the state specified is the same as the QP's current state, then the QP's state remains unchanged. The specified QP attributes are modified. • If the next state is the SQD state, then the caller may also specify whether or not the Asynchronous Affiliated Event handler is to be called once the SQ has completed draining.	Required, but can be the same as the QP's current state.

13 WRs, WQEs, and CQEs

The Previous Chapter

The previous chapter provided a detailed description of the verbs used to create a QP, modify its operational characteristics, obtain its current operational characteristics, and destroy it. It provided a detailed description of the creation of a QP and the steps required to program it. The QP state machine was described in detail.

This Chapter

This chapter provides a detailed description of how Work Requests (WRs) are posted to a QP's SQ or RQ, as well as a detailed description of the WR content. Once a WR has been posted to a QP's SQ or RQ, it is referred to as a WQE (Work Queue Entry; pronounced as "wookie"). This chapter describes the ordering rules that govern:

- WQE execution order.
- WQE completion order.
- RD RQ WQE completion order. The reason why RD RQ WQEs can complete out of order is described.
- RDMA Read relaxed ordering rules.

When a message transfer request WQE completes execution, a Completion Queue Entry (CQE; pronounced "cookie") is created on the respective work queue's Completion Queues (CQ). This chapter provides a detailed description of the verbs used to create a CQ, resize it, obtain its current operational characteristics, and destroy it. It also covers:

- The completion event handler.
- The *Set Completion Event Handler* verb.
- The *Poll for Completion* verb.
- The *Request for Completion Notification* verb.

- The detailed content of a CQE.
- Solicited and unsolicited events.

The Next Chapter

The next chapter defines what an asynchronous error or event is, as well as the difference between an affiliated and an unaffiliated event or error. It describes how to specify an Asynchronous Event Handler that will automatically be called whenever an asynchronous event or error is detected.

Once Posted to SQ or RQ, WR Is Called a WQE

Once a WR is posted to a QP's RQ or SQ, it is commonly referred to as a Work Queue Entry. The SQ and RQ are sometimes referred to as Work Queues (WQs).

WRs

WRs Posted to SQ

Purpose of a SQ WR

Software posts a WR to a QP's SQ in order to initiate a message transfer with a QP of the same type in a remote CA.

Posting a WR to the SQ

A WR is posted to the SQ by executing the *Post Send Request* verb call. Refer to "Content of SQ WRs" on page 262 for a description of the WR's content.

Types of SQ Operations

The types of WRs that can be posted to a QP's SQ depends on the type of QP:

- **Message Send operation**. Supported on RC, RD, UC, UD, and Raw QPs.
- **RDMA Write operation**. Supported on UC, RC, and RD QPs.
- **RDMA Read operation**. Supported on RD and RC QPs.
- **Atomic operations**. Supported on RD and RC QPs.
- **Memory Window Bind operation**. Supported on RC, UC, and RD QPs.

Resync operation is atypical. Supported on RD QPs. It should be noted that this operation is automatically initiated by the EEC hardware in the event of an error. It is not initiated by software posting a WR to the SQ of a QP. When a Resync is required, the EEC's Send Logic automatically initiates it. For a detailed description, refer to "Resync Operation" on page 499.

Note that only the Send operation is supported on Raw QPs.

WRs Posted to RQ

Purpose of a RQ WR

Software posts WRs to a QP's RQ in order to handle two types of inbound message transfer requests issued by the remote QP's SQ Logic (see "Inbound Message Types Handled by RQ WQEs" on this page).

Posting a WR to the RQ

A WR is posted to the RQ by executing the *Post Receive Request* verb. Refer to "Content of RQ WRs" on page 276 for a description of the WR's content.

Only One Type of RQ Operation

Only one type of operation may be specified in a WR posted to the RQ. It is the Receive operation.

Inbound Message Types Handled by RQ WQEs

Software posts WRs to a QP's RQ (once posted, it's referred to as a WQE) to handle the receipt of one of two types of messages sent by the remote QP's SQ Logic:

- An **inbound message Send operation**. In this case, the top RQ WQE contains the Scatter Buffer List defining the series of one or more buffers in the CA's local memory to which the message will be written. When the message's final request packet data payload has been written to the CA's local memory, the top entry on the RQ is retired and a CQE is created on the RCQ. If the final packet includes the ImmDtETH containing a 32-bit immediate data value, the 32-bit immediate data value is stored in the CQE. This data value can be used to inform software associated with the destination QP regarding the nature of the message just sent to it.

- An **inbound RDMA Write With Immediate message transfer operation**. In this case, the WR is really a dummy WR. The final request packet of the RDMA Write operation includes the ImmDtETH containing the 32-bit immediate data value. When the data payload of the final request packet has been written to the CA's local memory, the top entry on the RQ is retired and a CQE is created on the RCQ. The 32-bit immediate data value is stored in the CQE. This data value can be used to inform software associated with the destination QP regarding the nature of the message just sent to it.

Inbound Message Types Not Handled by RQ WQEs

RQ WQEs do not handle the following types of inbound operations sent by the remote QP's SQ Logic:

- **RDMA Write Without Immediate** message transfer operation. Although this operation is handled by the destination QP's RQ Logic, a RQ WQE is not used to define where the inbound message is written in the CA's local memory. Rather, the start memory address, transfer length, and a remote access key (R_Key) are supplied by the remote QP's SQ Logic in the first request packet of the message transfer.
- **RDMA Read message transfer operation**. This type of inbound operation is posted in a special queue (within the RQ Logic) reserved to handle inbound RDMA Read and Atomic operations.
- **Atomic operations (**see the previous bullet item)**:**
  - — **Atomic Fetch and Add operation**.
  - — **Atomic Compare and Swap If Equal operation**.

WR Content

Content of SQ WRs

Send Operation WR. The message Send operation is supported on all QP types. The input parameters supplied when executing the *Post Send Request* verb are defined in Table 13-1 on page 263.

RDMA Write Operation WR. The RDMA Write operation is supported on UC, RC, and RD. The input parameters supplied when executing the *Post Send Request* verb are defined in Table 13-2 on page 266.

RDMA Read Operation WR. The RDMA Read operation is supported on RC and RD. The input parameters supplied when executing the *Post Send Request* verb are defined in Table 13-3 on page 268.

Atomic Fetch and Add WR. The Atomic Fetch and Add operation is supported on RC and RD. The input parameters supplied when executing the *Post Send Request* verb are defined in Table 13-4 on page 270.

Atomic Compare and Swap If Equal WR. The Atomic Compare and Swap If Equal operation is supported on RC and RD. The input parameters supplied when executing the *Post Send Request* verb are defined in Table 13-5 on page 272.

Memory Window Bind WR. The Memory Window Bind operation is supported on RC, UC, and RD. Unlike the other SQ operations, it is not posted to the SQ using the *Post Send Request* verb. Rather, the *Bind Memory Window* verb is used. The input parameters supplied when executing the *Bind Memory Window* verb are defined in Table 13-6 on page 274. A detailed description of memory windows can be found in "Memory Windows" on page 308.

Table 13-1: WR Content for a Send Operation

WR Element	QP Type Applicability	Required ?	Description
HCA Handle	All	Yes	Identifies the target HCA. Returned by the *Open HCA* verb.
QP Handle	All	Yes	Identifies the target HCA QP. Returned by the *Create QP* verb.
WR ID	All	Yes	64-bit, software-assigned unique identifier for this WR. It is stored in the CQE created at the end of the operation.
Operation type	All	Yes	Operation type = Send.
CQE generation indicator	All	Yes, if...	...QP was set up to only generate SQ CQE when requested in WR.

14 Asynchronous Events and Errors

The Previous Chapter

The previous chapter provided a detailed description of how Work Requests (WRs) are posted to a QP's SQ or RQ, as well as a detailed description of the WR content. This chapter described the ordering rules that govern:

- WQE execution order.
- WQE completion order.
- RD RQ WQE completion order. The reason why RD RQ WQEs can complete out of order is described.
- RDMA Read relaxed ordering rules.

This chapter provided a detailed description of the verbs used to create a CQ, resize it, obtain its current operational characteristics, and destroy it. It also covered:

- The completion event handler.
- The *Set Completion Event Handler* verb.
- The *Poll for Completion* verb.
- The *Request for Completion Notification* verb.
- The detailed content of a CQE.
- Solicited and unsolicited events.

This Chapter

This chapter defines what an asynchronous error or event is, as well as the difference between an affiliated and an unaffiliated event or error. It describes how to specify an Asynchronous Event Handler that will automatically be called whenever an asynchronous event or error is detected. This chapter concludes the part of the book that provides a detailed description of QPs, WRs, WQEs, CQEs, and asynchronous errors or events (i.e., Part 2).

The Next Chapter

The next chapter describes the various types of memory access violations that could take place and the protection mechanisms used to prevent them. It provides a detailed description of memory regions, memory handles, and protection domains (PDs).

Why Asynchronous?

When referring to these types of events or errors, the specification refers to them as "asynchronous." This is because the event or error is not associated with any particular message transfer.

Registering a Handler

Software executes the *Set Asynchronous Event Handler* verb to register a handler to be called when an asynchronous event or an asynchronous error is encountered by the HCA.

Whenever one of these events or errors is encountered by the HCA, it alerts its device driver (typically via an interrupt). The parameters passed to the driver are:

- The HCA handle.
- An Event record. This contains information indicating the resource type and the RID (Record ID), as well as which event occurred (RIDs are covered in "Detailed Description of SA" on page 917).

The driver, in turn, calls the Asynchronous Event Handler and passes the HCA handle and the event record to it.

Affiliated Asynchronous Events

The term "affiliated" is used because the event is associated (or affiliated) with a specific QP or EEC. The term "event" is used because it is not an error, but rather an event that software may find useful. Since the event is not considered to be an error, the state of the associated QP or EEC remains unchanged. The currently defined affiliated asynchronous events are:

- **Path Migrated**. A connection has been migrated to an alternate path. Refer to "Automatic Path Migration" on page 575 for a detailed description of Automatic Path Migration.
- **Communication Established**. This indicates that the first request packet has arrived at a QP's or EEC's RQ Logic while the QP or EEC is still in the RTR state. This automatically transitions the QP or EEC from the RTR to the RTS state. The handle of the destination QP or EEC is returned in the event record. This event may be useful to the HCA's Communication Manager (CM).
- **SQ Drained**. This indicates that the SQ of the specified QP has completed transmission of all outstanding messages that were in progress when software commanded the QP or EEC to transition from the RTS to the SQD state. For RC and RD, the event indicates that the QP or EEC SQ Logic has received all of the responses associated with all of the message request packets that were sent to the destination QP or EEC. This event informs software that it is now safe to execute the *Modify QP* or *Modify EEC* verb to modify the operational characteristics of the QP or EEC and to transition it back to the fully operational RTS state.

Affiliated Asynchronous Errors

The term "affiliated" is used because the event is associated (or affiliated) with a specific SQ, RQ, CQ, or EEC. The term "error" is used because it is an error, but the HCA is unable to report the error in a CQE. The affected QP or EEC is transitioned to the Error state. The currently defined error types are:

- **CQ Error**. An error occurred when the HCA was attempting to write a CQE to a CQ. If any SQ or RQ associated with the CQ should subsequently attempt to post a CQE to that CQ, it results in the reporting of this error. CQEs can no longer be posted to the CQ, and it is not guaranteed that CQEs already present in the CQ at the time that the error occurred can be retrieved successfully. Possible causes include a CQ overrun or a CQ protection error.
- **Local Work Queue Catastrophic Error**. An error while accessing or processing WQEs on a SQ or RQ prevents the reporting of completions. This error is generated when a SQ or RQ associated with the CQ that caused the CQ Error (see the previous bullet item) attempts to use that CQ.
- **Local EEC Catastrophic Error**. An error while accessing or processing an EEC prevents the successful posting of CQEs to RD QP CQs.
- **Path Migration Request Error**. An incoming path migration request to this QP or EEC was not accepted. Refer to "Automatic Path Migration" on page 575 for a detailed description of Automatic Path Migration.

Unaffiliated Asynchronous Errors

The term "unaffiliated" is used because the error cannot be associated (or affiliated) with a specific QP or EEC. How the error is handled is defined by the error type:

- **Local Catastrophic Error**. The error cannot be associated with any resource. The behavior of the HCA and/or its driver are indeterminate.
- **Port Error**. This error is reported when a link becomes unavailable. It is generated when the link associated with an HCA port transitions from the available to the unavailable state. Refer to "Link State Machine" on page 602 for a detailed description of the possible port states:
  - **Unavailable states**: Down, Initialize, and Armed.
  - **Available states**: Active and ActDefer.

Part 3 Protection Mechanisms

The Previous Part

Part 2 began the portion of the book that provides detailed information about the InfiniBand technology. Part 2 provided a detailed description of the creation, management, and the basic operation of the various types of Queue Pairs. Part 2 consisted of the following chapters:

- Chapter 12—QP Verbs and QP State Machine.
- Chapter 13—WRs, WQEs, and CQEs.
- Chapter 14—Asynchronous Events and Errors.

This Part

Part 3 introduces memory management, memory protection, and the various other protection mechanisms available in the InfiniBand environment. Part 3 consists of the following chapters:

- Chapter 15—Memory Protection.
- Chapter 16—Other Protection Mechanisms.

The Next Part

Part 4 provides a detailed description of the four IBA transport services as well as the Raw IPv6 and Raw EtherType services. In addition, it also provides a detailed description of UD Multicasting, Automatic Path Migration, and Static Rate Control. Part 4 consists of the following chapters:

- Chapter 17—RC Transport Service.
- Chapter 18—UC Transport Service.

15 Memory Protection

The Previous Chapter

The previous chapter defined what an asynchronous error or event is, as well as the difference between an affiliated and an unaffiliated event or error. It described how to assign an Asynchronous Event Handler that will automatically be called whenever an asynchronous event or error is detected. This chapter concluded the part of the book that provided a detailed description of QPs, WRs, WQEs, CQEs, and asynchronous errors or events (i.e., Part 2).

This Chapter

This chapter describes the various types of memory access violations that could take place and the protection mechanisms used to prevent them. It provides a detailed description of memory regions, memory handles, and protection domains (PDs).

The Next Chapter

The IBA specification defines a number of mechanisms to prevent various types of unauthorized access. While this chapter covers the memory-oriented protection mechanisms, the next chapter provides a detailed description of the remaining protection mechanisms:

- Preventing unauthorized access to UD QPs with PDs.
- Partition keys (P_Keys).
- Management keys (M_Keys).
- The Subnet Management key (SM_Key).
- Reliable Datagram Domains (RDDs).
- Queue keys (Q_Keys).
- Baseboard Management keys (B_Keys).

The Problems

This chapter focuses on the mechanisms that prevent an entity from completing one of the following actions:

- **Local access violation**. Logic (e.g., a QP) within a CA may attempt to access an area of that CA's local memory that it does not have authorization to access.
- **Remote access violation**. A remote CA's QP may attempt to access an area of the local CA's memory that it does not have authorization to access.
- **Access rights violation**. Although local logic within a CA or a remote CA may have permission to access a specific area of the local CA's memory, it may attempt to access the memory area in a manner for which it does not have permission. As an example, it may attempt to write to a memory area for which it only has permission to read.
- **Unauthorized access to a remote UD QP**. Software posts a WR to the SQ of a local UD QP. The WR specifies the target remote UD QP to whom the message transfer request is to be sent. The message sender may not have authorization to send a message to the specified destination QP. This subject is covered in "PDs and UD Service" on page 318.

The Solutions

Three mechanisms are provided to enforce memory access restrictions:

- Memory regions.
- Memory windows.
- Memory Protection Domains (PDs).

The sections that follow provide a detailed discussion of these protection mechanisms.

Virtual-to-Physical Page Mapping Background

Understanding regions requires an understanding of virtual-versus-physical memory addressing. For background on this subject, refer to the following MindShare publications (all published by Addison-Wesley):

- **Protected Mode Software Architecture**, First Edition. Specifically, the chapter entitled "Virtual Paging."
- **Pentium Pro and Pentium II System Architecture**, Second Edition. Specifically, the chapter entitled "Paging Enhancements."
- **ISA System Architecture**, Third Edition. Specifically, the section entitled, "Page Translation" in the chapter entitled "80386Dx and SX Microprocessors."
- **PowerPC System Architecture**, First Edition. Specifically, the chapters entitled "Address Translation Overview" and "Virtual Paging."

Memory Regions

Definition of a Memory Region

A memory region is a collection of memory pages within the local HCA's memory. When the region is created by software (via a verb call), the caller defines a combination of local and, possibly, remote access permissions within the memory region being created:

- **Local access permission** grants logic within the local CA (e.g., QPs) authority to perform the defined access types within the region.
- **Remote access permission** grants QPs within a remote CA authority to perform the defined access types within the region. When a remote QP sends a request to access the local memory associated with the destination QP, the destination QP must have been granted the appropriate access rights to access local memory on behalf of the remote QP. In other words, in order for remote QPs to access this CA's local memory, the local CA must also have been granted local access rights.

Before QPs within a HCA can access its memory, one or more memory regions must be set up that permit local and, possibly, remote access.

Virtual Memory Regions

Definition of a Virtual Memory Region

When software tells the verbs to create a virtual memory region, it is left up to the OS memory management routine to select which physical memory pages are assigned to the region (for the alternative, see "Physical Memory Regions" on page 307).

Creating a Virtual Region

A virtual region is created by performing a *Register Memory Region* verb call. Some of the input parameters provided by software include:

- The desired start virtual memory address (64 bits):
  - — It can be any byte-aligned 64-bit address.
  - — It does not have to be a page-aligned start address.
  - — If the start address isn't page-aligned, the region starts at an offset within the first physical page assigned to the region by the OS memory management routine.
  - — Note that if the requested virtual address range is already in use (i.e., the OS memory management routine has already mapped it to physical pages in memory for another requester), the OS will assign a different virtual address range to this region.
- Region length (in bytes):
  - — This defines the end address of the virtual address range assigned to the region.
  - — The region may end in the middle of the last page assigned to region.
- Access control attributes:
  - — Local read access is always automatically granted.
  - — Enable/disable local write access.
  - — Enable/disable remote write access (requires enabling of local write access as well).
  - — Enable/disable remote read access.
  - — Enable/disable remote atomic operation access (requires enabling of local write access as well).
  - — Enable/disable memory window binding (window binding is described in "Memory Windows" on page 308).

Some of the output parameters returned from the verb call include:

- **L_Key**. Local access key (aka Region ID). This key must be specified when local logic within the HCA wishes to access the region. When specifying the Gather Buffer List or Scatter Buffer List in a WR, each entry in the supplied list is specified by start virtual memory address, buffer length, and the L_Key assigned to that memory area when that memory region was created.
- **R_Key**. Remote access key (aka Region ID). This key is only returned if remote access rights were requested. The R_Key is supplied in an RDMA or Atomic request packet sent by a remote QP when it desires access to the region.
- **Region handle**. The handle must be specified when making any subsequent verb calls to manipulate this region's characteristics.

Actions Taken by Verb to Create a Region

Refer to Figure 15-1 on page 304. The following steps are performed in order to create a virtual memory region:

1. Software executes the *Register Memory Region* verb to create a region.
2. The *Register Memory Region* verb in turn calls the OS memory allocation routine to request a buffer of the requested size.
3. The OS memory allocation routine locates pages in physical main memory whose sum total size ≥ the requested size of the region.
4. The pages allocated by the OS memory management routine almost certainly do not occupy a contiguous physical memory address range.
5. When calling the OS memory management routine, the verb must specify that the pages allocated to the region must be pinned in memory by the OS. This prevents the OS from swapping these pages out to mass storage.
6. The OS memory management routine returns the start physical address of each page to the verb.
7. The verb creates (within the HCA or the HCA's local memory) a table mapping the VA (Virtual Address) range defined (by the start VA and the region length) to the series of physical pages assigned to the newly created region.

Region Windowing May Be Enabled When Region Is Created

In order to enable the binding of windows (see "Memory Windows" on page 308) to a region, this must be specified when the region is created (using the *Register Memory Region* or *Register Physical Memory Region* verbs).

Characteristics of a Region

- Multiple regions may partially or fully overlap the same physical memory locations.
- A region is only related to the HCA to which it is registered (not to any other HCA).
- The region registration verbs must issue a pin request to the OS memory management routine each a time region is registered (even if some of the pages were previously allocated and pinned by previous region registrations).
- When all of the regions utilizing a page have been deregistered, the OS must unpin the page. This means that the OS memory management routine must track how many pin requests were received for a page.

16 Other Protection Mechanisms

The Previous Chapter

The previous chapter described the various types of memory access violations that could take place and the protection mechanisms used to prevent them. It provided a detailed description of memory regions, memory handles, and protection domains (PDs).

This Chapter

The IBA specification defines a number of mechanisms to prevent various types of unauthorized access. While the previous chapter covered the memory-oriented protection mechanisms, this chapter provides a detailed description of the remaining protection mechanisms:

- Preventing unauthorized access to UD QPs with PDs.
- Partition keys (P_Keys).
- Management keys (M_Keys).
- The Subnet Management key (SM_Key).
- Reliable Datagram Domains (RDDs).
- Queue keys (Q_Keys).
- Baseboard Management keys (B_Keys).

This chapter concludes the part of the book that focuses on protection mechanisms (i.e., Part 3).

The Next Chapter

The next chapter provides a detailed description of the RC transport service.

The IBA Protection Mechanisms

Refer to Table 16-1 on this page. IBA provides a number of mechanisms that prevent unauthorized operations from being performed. This chapter provides a detailed explanation of each of these mechanisms.

Table 16-1: Who Provides Protection from What?

Problem	Handled by	Support Required?
Unauthorized local access to a memory area.	PD	Yes
Logic within HCA (e.g., a local QP) attempts unauthorized access to system memory.	L_Key	Yes
QP on remote CA attempts unauthorized RDMA or Atomic access to an area of CA's local memory.	R_Key	Yes, if RDMA and Atomic operations are supported.
Port's acceptance of packet from port that is not a member of the same partition.	P_Key	• For QPs: Yes, except for Raw Datagram QPs. • Optional for switches and routers.
Unauthorized SM attempts to access port's or device's configuration locations (i.e., its attributes).	M_Key	• Optional whether Master SM assigns *M_Keys* to ports. If it doesn't, ports do not check *M_Key*. • Based on setting of *M_KeyProtectBits*, port may or may not check *M_Key* before allowing SMP read attempt. • If *M_Keys* assigned, port must validate *M_Key* before allowing SMP write attempt.

Table 16-1: Who Provides Protection from What? (Continued)

Problem	Handled by	Support Required?
Unauthorized Baseboard Manager (BM) attempts to access port's or device's configuration locations (i.e., its attributes).	B_Key	• Optional whether BM assigns *B_Keys* to ports. If it doesn't, ports do not check *B_Key*. • Based on setting of *B_KeyProtectBit*, port may or may not check *B_Key* before allowing BM read attempt. • If *B_Keys* assigned, port must validate *B_Key* before allowing BM write attempt.
Passing SM mastership to SM not authorized to manage this subnet.	SM_Key	Yes
Unauthorized access to a destination UD QP.	PD	Yes
Acceptance of a datagram (by QP's RQ Logic) from unauthorized sender.	Q_Key	• Yes for RD and UD. • Not applicable for SMPs, but is applicable to GMPs.
• On the sender's end, a RD QP isn't a member of the same RDD as the local EEC to whom it is passing a request packet to transmit. • On the receiving end, the EEC is not a member of the same RDD as the local RD QP to whom it passes a received request packet.	RDD	Yes for RD.

Memory Access Protection (PD, L_Key, and R_Key)

Protection from unauthorized memory accesses is provided by L_Keys, R_Keys, and PDs, and was covered in the chapter entitled "Memory Protection" on page 297.

PDs and UD Service

Background On Address Handles

In order to send request packets to an UD QP on a remote CA, the following actions must be accomplished:

1. Create a local UD QP and assign it to a PD.
2. Prior to any attempt to send any messages to an UD destination, software must first execute a *Create Address Handle* verb call, providing the following as input parameters (note that this is a partial list):
   — The LID or GID of the destination port behind which the target UD QP resides.
   — The PD that is being assigned to this destination port.
   — The handle that identifies the HCA on which the Address Handle is to be created.
3. To send a message to a remote UD QP, software then posts a WR to a local UD QP's SQ and supplies the following as input parameters (note that this is a partial list):
   — The Address Handle that identifies the remote port behind which the destination UD QP resides.
   — The QPN of the destination remote UD QP.
4. The local UD QP's SQ will not accept the WR (i.e., it will generate an error) if the local QP's PD (assigned when the local QP was created) does not match the PD that was assigned to the destination Address Handle contained in the WR (the PD was assigned to that destination when the Address Handle was created). If the PDs match, the WR is accepted and the WQE will be executed when it reaches the head of the SQ.

PDs and Address Handles

The following rules apply regarding the association of PDs and Address Handles:

- The PD must be created (i.e., allocated) before the creation of any Address Handle that will be a member of that PD.
- Each Address Handle is a member of a single PD.
- Multiple Address Handles can be members of the same PD.
- A PD cannot be destroyed if any Address Handle (or QP, window, or region) is still a member of that PD.

Partition Key (P_Key)

Definition of a Partition

A partition is defined as a collection of CA, router, and switch ports that are permitted to communicate with one another. A port may be a member of multiple partitions simultaneously.

Who Enrolls a Port in a Partition?

Refer to Figure 16-1 on page 322. A partition is represented by a 15-bit partition ID. Theoretically, this permits up to 32K partitions to be set up. Partition values assigned to a port are stored in the port's *P_KeyTable* attribute. The attribute consists of a series of P_Key entries.

One or more P_Keys are assigned to a port by the subnet's PM (Partition Manager). The PM is typically implemented as a subset of the Master SM. The *P_KeyTable* attribute is read and written using SMPs issued by the PM.

A Port Can Be a Member of Multiple Partitions

The size of a port's *P_KeyTable* attribute (and therefore the number of P_Keys which can be assigned to the port) is device design-specific (but its maximum possible size is 32K entries). The actual size of the *P_KeyTable* attribute implemented for each port on a CA or router and for the switch management port is found in the *NodeInfo.PartitionCap* attribute element. If a switch implements the optional *P_KeyTable* attribute at each external port, the table size is found in the *SwitchInfo.PartitionEnforcementCap* attribute element. For a detailed description of the *P_KeyTable*, refer to "P_KeyTable Attribute" on page 837).

Which P_Key Is Inserted in Packets and Checked?

For RC, UC, and UD

General. When using the RC, UC, or UD service types, the QP is connected to one of the HCA's ports (during QP set up). When the QP is created and

17 RC Transport Service

The Previous Chapter

The IBA specification defines a number of mechanisms to prevent various types of unauthorized access. The previous chapter provided a detailed description of the protection mechanisms (other than the memory protection mechanisms):

- Preventing unauthorized access to UD QPs with PDs.
- Partition keys (P_Keys).
- Management keys (M_Keys).
- The Subnet Management key (SM_Key).
- Reliable Datagram Domains (RDDs).
- Queue keys (Q_Keys).
- Baseboard Management keys (B_Keys).

The previous chapter concluded the part of the book that focused on the protection mechanisms (i.e., Part 3).

This Chapter

This chapter provides a detailed description of the RC transport service.

The Next Chapter

The next chapter provides a detailed description of the UC transport service.

RC Support Requirement

HCAs are required to implement the RC transport service type. TCAs may optionally support the RC transport service type.

RC Basic Operational Characteristics

"Reliable Connected Service Type" on page 64 provided a very basic overview of the RC transport service's operational characteristics. For convenience sake, that information is repeated here:

- **Connection needed**. Before any messages may be transferred, a private connection must be established between the two RC QPs in the two CAs, and the QP Contexts of the two QPs are each programmed with the identity of the remote QP as well as the address of the port behind which the remote QP resides.
- **Private communications channel**. The two RC QPs may then be used to send messages to each other (but not to any other QP in the same or any other target adapter).
- **Message size**. Each message transfer WR can specify a message transfer anywhere from zero to 2GB in size. Large messages (larger than the path's PMTU) are segmented into multi-packet transfers.
- An **Ack/Nak protocol** permits the requester (i.e., the QP SQ Logic sending the message) to verify that all packets are delivered to the responder QP's RQ Logic.
- **Packet PSN**. Each request packet contains a PSN that the responder QP's RQ Logic uses to verify that all request packets are received in order (and that each is only processed once, even if it should be received multiple times; more on this later).
- Two **CRC** fields in each packet are used to verify the integrity of the packet.
- **High-traffic**. Because the responder QP's RQ Logic must Ack or Nak each request packet received, this service class generates considerably more network traffic than the "unreliable" service types.
- **Completes on receipt of final Ack**. The SQ Logic cannot create a CQE on the SCQ (Send Completion Queue) to signal the completion of a message transfer until the Ack for the final packet of the message transfer has been received.
- **Operations supported**. Supports the following types of message transfer operations:
  - — RDMA Read support is required.
  - — RDMA Write support is required.
  - — It is optional whether or not the CI supports Atomic operations. If it does, then it must support both the Atomic Fetch and Add and Atomic Compare and Swap If Equal operations on the RC transport service type.
  - — Send support is required.
  - — Bind Memory Window support is required.

RC Connection Establishment

The process of establishing a connection between a local and a remote RC QP was described in "RC/UC Connection Establishment" on page 186. Additional detail on connection establishment can be found in "Communications Management" on page 1069.

Packet Opcodes

Table 17-1 on page 354 shows the valid opcodes that can be inserted in the BTH:Opcode field of each outbound request packet generated by the requester QP's SQ Logic and in the BTH:Opcode field of each response packet transmitted by the responder QP's RQ Logic.

RC Message Transfer Primer

The example message Send operation described in "Sending a Message to a Destination CA" on page 47 provided a fairly detailed introduction to the RC transport service.

Table 17-1: RC Packet Opcodes

Bits 7:5	Bits 4:0	Packet Type	Fields That Follow BTH
000 **RC**	00000	SEND First	Data PayLoad
	00001	SEND Middle	Data PayLoad
	00010	SEND Last	Data PayLoad
	00011	SEND Last with Immediate	Immediate Data, Data PayLoad
	00100	SEND Only	Data PayLoad
	00101	SEND Only with Immediate	Immediate Data, Data PayLoad
	00110	RDMA WRITE First	RDMA ETH (RETH), Data PayLoad
	00111	RDMA WRITE Middle	Data PayLoad
	01000	RDMA WRITE Last	Data PayLoad
	01001	RDMA WRITE Last with Immediate	Immediate Data, Data PayLoad
	01010	RDMA WRITE Only	RETH, Data PayLoad
	01011	RDMA WRITE Only with Immediate	RETH, Immediate Data, Data PayLoad
	01100	RDMA READ Request	RETH
	01101	RDMA READ response First	Acknowledge ETH (AETH), Data Pay-Load
	01110	RDMA READ response Middle	Data PayLoad
	01111	RDMA READ response Last	AETH, Data PayLoad
	10000	RDMA READ response Only	AETH, Data PayLoad
	10001	Acknowledge packet	AETH
	10010	ATOMIC Acknowledge packet	AETH, AtomicAckETH
	10011	CmpSwap packet	AtomicETH
	10100	FetchAdd packet	AtomicETH
	10101 - 11111	Reserved	undefined

Structure of This Discussion

This chapter provides a detailed description of the RC transport mechanism. For ease of understanding, the author has chosen to divide the discussion into the following sections:

1. **QP State Before Any Messages Transferred**. It is always helpful to establish the state of things just before normal operation begins. This section initiates the discussion by establishing the state of the local and remote QPs after they have been created and set up, but before any messages have been transferred between the two.
2. **Standard Operation in Fast, Error-Free Environment**. This section describes how messages are transferred in a "perfect" environment. It assumes the following set of conditions:
   - **No delay in request packet delivery**. The request packets issued by the requester QP's SQ Logic make their way to the destination QP's RQ Logic in an expeditious manner.
   - **Quick generation of responses**. The responder QP's RQ Logic processes each request packet rather quickly and issues the corresponding response packet back to the requester QP.
   - **No delay in response packet delivery**. The response packet makes its way back to the requester QP's SQ Logic quickly.
   - **No packets lost**. No request or response packets are "lost" in the fabric due to errors of any sort.
   - **No error conditions are encountered,** neither within the two CAs nor in the packet flight path.
   - **No traffic reduction**. The responder QP's RQ Logic does not implement Acknowledge coalescing (more on this later). In other words, the responder QP's RQ Logic is designed to issue a response (or, possibly, multiple responses in the case of an RDMA Read request) for each request packet received.
3. **Traffic Reduction**. This section describes the Ack coalescing mechanism that can be used to decrease the amount of response traffic injected into the fabric. Support for Ack coalescing is optional for the RQ Logic and mandatory for the SQ Logic.
4. **Packet Delivery Delays**. The next phase of the discussion assumes the following:
   - **No request or response packets are lost in transit**. This section assumes that all request packets generated by the requester QP's SQ Logic eventually arrive at the responder QP's RQ Logic. Likewise, it assumes that all responses generated by the responder QP's RQ Logic eventually arrive at the requester QP's SQ Logic.

18 UC Transport Service

The Previous Chapter

The previous chapter provided a detailed description of the Reliable Connected (RC) transport service.

This Chapter

This chapter provides a detailed description of the UC transport service.

The Next Chapter

The next chapter provides a detailed description of the RD transport service.

UC Support Requirement

HCAs are required to implement the UC transport service type. TCAs may optionally support the UC transport service type.

In RC, Responses Are Expected

In the RC transport service type, the requester QP's SQ Logic expects to receive a confirmation from the responder QP's RQ Logic that each request packet of the message transfer has been successfully received and acted upon by the responder QP's RQ Logic. The requester QP's SQ Logic cannot signal a message transfer completion to software until it has received the Ack corresponding to the message's final request packet (in the case of a Send or RDMA Write), or RDMA response packet (in the case of an RDMA Read), or the Atomic response packet, in the case of an Atomic RMW).

UC Is a Subset of RC

Quite simply, the UC transport service type is a subset of the RC transport service type. In the UC transport service type, no responses are expected. Software on the sender's end is therefore alerted that a message transfer has completed immediately upon the transmission of the last request packet of the Send or RDMA Write. RDMA Reads and Atomic requests are not supported.

UC Transport Service Type's Basic Characteristics

UC is a subset of the RC protocol. The responder's RQ Logic doesn't Ack or Nak each request packet it receives.

- **Disadvantage**: The requester has no verification that each request packet has been received by the remote QP's RQ Logic.
- **Advantages**:
  - — This protocol generates considerably less network traffic than RC.
  - — From the sender's perspective, the message transfer completes quickly (because it doesn't have to wait for the transfer to be acknoweldged).

The basic characteristics of the UC service type are:

- **Connection needed**. Just like the RC protocol, before any messages may be transferred, a connection must be established between the UC QPs in the two adapters, and the QP Contexts of the two QPs are each programmed with the identity of the remote QP as well as the address of the port behind which it resides. Refer to "RC/UC Connection Establishment" on page 186 for a description of the connection establishment process. Also refer to "Communications Management" on page 1069.
- **Private communications channel**. Just like the RC protocol, the two UC QPs may then be used to send messages to each other (but not to any other other QP in the same or any other CA).
- **Message size**. Just like the RC protocol, each message can be anywhere from 0 to 2GB in size. Messages large than one PMTU are segmented into multi-packet transfers. It should be noted that a 0-byte Send or RDMA Write is permitted.
- **No Ack/Nak protocol.** Unlike the RC protocol, there is no Ack/Nak protocol. The requester (i.e., the QP SQ Logic sending the message) therefore cannot verify that each packet is delivered to the responder (the target QP's RQ Logic).

- **Software completion notification**. There are no responses returned by the remote QP's RQ Logic when using the UC transport service type. That being the case, software on the sender's side is alerted that a message transfer has completed immediately upon the transmission of the last request packet of the Send or RDMA Write. RDMA Reads and Atomic requests are not supported.
- **Low traffic**. Because the responder QP's RQ Logic doesn't Ack or Nak each request packet received, this service class generates considerably less network traffic than the "reliable" service types.
- **Packet PSN checking.** Just like the RC protocol, each request packet contains a PSN that the target QP's RQ Logic uses to verify that all request packets are received in order (and that each is only processed once even if it should be received multiple times). Unlike the RC protocol, however, if the responder QP's RQ Logic detects out-of-order request packets (i.e., one or more missing request packets), it is not permitted to send a Nak back to the requester QP's SQ Logic. If this should happen:
  - The responder QP's RQ Logic ignores the remaining request packets of the current message.
  - The responder QP's RQ Logic awaits the beginning of a new message (as signaled by a request packet with a BTH:Opcode of the "First" or "Only" type).
  - The responder QP's RQ Logic may (or may not) inform its local client (e.g., software) of the problem.
- Just like the RC protocol, two **CRC** fields in each packet are used to verify the integrity of the packet.
- **Operations supported**. Supports the following types of message transfer operations:
  - RDMA Write support is required.
  - Send support is required.
  - Bind Memory Window support is required.

Requester QP's SQ Logic Operation

Start PSN Assignment and PSN Management

As previously described in "RC/UC Connection Establishment" on page 186, the SQ Logic's Start PSN is assigned during QP setup.

As the SQ Logic generates each outbound request packet, it sets the BTH:PSN to the previous PSN + 1.

Request Packet Opcodes and Packet Sequences

The valid UC request packet opcodes are shown in Table 18-1 on this page. Since there are no response packets used in this transport service type, no response opcodes are defined.

The valid request packet transmission sequences are defined in Table 18-2 on page 447.

Table 18-1: Packet UC Request Packet Opcodes

Bits 7:5	Bits 4:0	Packet Type	Fields That Follow BTH
001 UC	00000	SEND First	Data Payload
	00001	SEND Middle	Data Payload
	00010	SEND Last	Data Payload
	00011	SEND Last with Immediate	Immediate Data, Data Payload
	00100	SEND Only	Data Payload
	00101	SEND Only with Immediate	Immediate Data, Data Payload
	00110	RDMA WRITE First	RETH, Data Payload
	00111	RDMA WRITE Middle	Data Payload
	01000	RDMA WRITE Last	Data Payload
	01001	RDMA WRITE Last with Immediate	Immediate Data, Data Payload
	01010	RDMA WRITE Only	RETH, Data Payload
	01011	RDMA WRITE Only with Immediate	RETH, Immediate Data, Data Payload
	01100 - 11111	Reserved	Undefined

Table 18-2: Valid Request Packet Transmission Sequences

Previous Request Packet Opcode	Valid Opcode to Be Placed in Current Request Packet
The QP has just been set up and no messages have been transmitted yet. Therefore there was no previous request packet.	One of the following: • Send First. • Send Only Without Immediate. • Send Only With Immediate. • RDMA Write First. • RDMA Write Only Without Immediate. • RDMA Write Only With immediate.
Send First	One of the following: • Send Middle. • Send Last Without Immediate. • Send Last With Immediate.
RDMA Write First	One of the following: • RDMA Write Middle. • RDMA Last Without Immediate. • RDMA Write Last With Immediate.
Send Middle	One of the following: • Send Middle. • Send Last Without Immediate. • Send Last With Immediate.
RDMA Write Middle	One of the following: • RDMA Write Middle. • RDMA Last Without Immediate. • RDMA Write Last With Immediate.
Send Last Without Immediate or Send Last With Immediate	One of the following: • Send First. • Send Only Without Immediate. • Send Only With Immediate. • RDMA Write First. • RDMA Write Only Without Immediate. • RDMA Write Only With immediate.

19 RD Transport Service

The Previous Chapter

The previous chapter provided a detailed description of the Unreliable Connected (UC) transport service.

This Chapter

This chapter provides a detailed description of the RD transport service.

The Next Chapter

The next chapter provides a detailed description of the UD transport service.

RD Support Requirement

Support for the RD transport service type is optional on both HCA and TCA designs.

Introduction

The basic characteristics of the RD transport service were introduced in "Datagram QPs Can Exchange Messages With Multiple QPs" on page 66, "RD Requires Initial Connection Establishment, UD Does Not" on page 67, and "Reliable Datagram (RD) Service Type" on page 67.

Many Similarities to RC

The protocol used to transmit packets between the two EECs that comprise a RDC is in most ways identical to that used in transmitting packets between two

RC QPs. This chapter defines in what ways the RD transport service differs from the RC transport service.

RD Basic Operational Characteristics

For convenience sake, the basic characteristics of the RD service type, which were provided in "Reliable Datagram (RD) Service Type" on page 67, are replicated here:

— **Connection needed**. Before any messages may be transferred, a connection must be established between the EECs in the two CAs, and the EECs are each programmed with the identity of the remote EEC as well as the address of the port behind which the remote EEC resides. This is necessary so the two EECs can transmit packets to each other.
— **Message size.** Each message can be anywhere from zero to 2GB in size. Large messages are segmented into multi-packet transfers.
— **Ack/Nak protocol.** An Ack/Nak protocol permits the requester (i.e., the Send Logic of the EEC sending the message) to verify that all packets are delivered to the remote EEC.
— **High traffic**. Because the responder EEC must Ack or Nak each request packet received, this service class generates considerably more network traffic than the "unreliable" service types.
— **Packet PSN checking.** Each packet contains a PSN that the target EEC's Receive Logic uses to verify that all packets are received in order (and that each is only received and processed once) and that there are no missing packets.
— Two **CRC** fields in each packet are used to verify the integrity of the packet.
— **Local EEC forwards request packets to EEC in remote CA.** Upon receipt of each request packet from a local RD QP's SQ Logic, the local EEC's Send Logic inserts the packet's PSN and transmits the packet to the EEC in the remote CA.
— **Remote EEC forwards request packets to remote QP**. Upon receipt of each request packet, the EEC's Receive Logic in the remote CA validates the request packet's PSN and forwards the request packet to the targeted remote RD QP's RQ Logic for processing.
— **Operations supported.** Supports the following types of message transfer operations:
  – RDMA Read support is required.
  – RDMA Write support is required.

- It is optional whether or not the CI supports Atomic operations. If it does, then it must support both the Atomic Fetch and Add and the Atomic Compare and Swap If Equal operations.
- Send support is required.
- Bind Memory Window support is required.

The Major Differences from RC

Connection Establishment

In RC, Two QPs Are Connected Together

When using the RC transport service, a pair of RC QPs are connected to each other and transmit messages to and from each other and no one else.

In RD, Two EECs Are Connected Together

When using the RD transport service, on the other hand, RD QPs do not communicate directly with each other. Rather, each WR posted to the SQ of a RD QP identifies a local EEC to be used as the conduit (i.e., the RDC) in sending messages to and receiving messages from RD QPs in a remote CA. The WR also specifies the QPN of the destination remote RD QP.

Obviously, before a RDC can be used to transmit and receive messages, it must be created. This requires the creation of an EEC in each of the two CAs. The procedure used to create a local and a remote EEC and to establish a connection between them was covered in "RD Connection Establishment" on page 194. That section also described the procedure for creating local and remote RD QPs. Additional detail on connection establishment can be found in "Communications Management" on page 1069.

Message Transfer Engine

For Other Service Types, QP Is the Message Transfer Engine

The QP's SQ Logic and RQ Logic provides the message transfer engine for the RC, UC, and UD service types. As already covered in the chapter entitled "QP Verbs and QP State Machine" on page 219, the QP actually consists of a state machine with the following states:

- **Reset** state. The QP has just been created and is not ready to send or receive message packets. For more information, refer to "Reset State" on page 237.
- **Initialize** state. WRs can be posted to the RQ, but the QP is not yet ready to send or receive message packets. For more information, refer to "Initialized State" on page 240. The QP has been provided with:
  - The CA port number that it will use to send and receive message packets.
  - The P_Key Index—for more information, refer to "Partition Key (P_Key)" on page 319.
  - For an RD or UD QP, the Q_Key assigned to the QP [for more information, refer to "Queue Key (Q_Key)" on page 333].
- **RTR** state. WRs can be posted to the RQ and inbound message transfer request packets can be handled. SQ WQE processing is disabled. Posting WRs to the SQ results in the return of an immediate error. For more detail, refer to "Ready to Receive State" on page 244.
- **RTS** state. WRs can be posted to both the SQ and RQ. The QP is fully enabled to transmit and receive packets. For more detail, refer to "Ready to Send State" on page 247.
- **SQD** state. This state is entered under software control if software needs to alter the operational characteristics of a QP that is already fully operational (i.e., it is in the RTS state). For more detail, refer to "SQ Drain (SQD) State" on page 249.
- **SQE** state. This state is implemented on all QP types except for RC QPs. The SQE state is entered on a SQ completion error. It can be entered either from the RTS state or the SQD state. For more detail, refer to "SQ Error State" on page 253.
- **Error** state. This state is entered upon detection of a fatal error. For more detail, refer to "Error State" on page 257.

EEC Is Message Transfer Engine for Client RD QPs

In the RD service type, the EEC (rather than the RD QP) acts as the surrogate message transmit/receive engine for its client RD QPs. As previously described, the QP's message transfer engine is implemented as a state machine. Likewise, the EEC is implemented as a state machine.

EEC Has Same State Machine as QPs

When using transport service types other than RD, the QP's SQ Logic transmits request packets and, in the case of the RC type, receives and processes response packets. The QP's RQ Logic receives and processes incoming request packets and, in the case of the RC type, transmits response packets back to the requester QP's SQ Logic.

When using the RD transport service type, the EEC provides the Send and Receive Logic engines that perform those same operations. That being the case, it should be no surprise that the EEC state machine is very similar to the state machine of non-RD QPs.

The state machine diagram pictured in Figure 12-1 on page 235 applies to EECs as well as to QPs. There are some differences, however, and they are defined in "EEC States" on page 513. It should be noted that EECs do not implement the SQE state, while their client RD QPs do implement it.

EEC State and QP States Are Not Related

At a given moment in time, an EEC and its client RD QPs are not necessarily in the same state. As an example, while the EEC might remain in the RTS state (i.e., the fully operational state), one or more of its client QPs may be in the Error state as a result of QP-related errors that were incurred during attempted message transfers.

EEC, Not QP, Is Associated with a Local Port

In the RC, UC, UD, and Raw transport services, the QP Context of each QP is supplied with the port number of the local CA port it is to use to transmit and receive packets. In the case of RC and UC, it is also programmed with the LID address of the port on the remote CA behind which its companion QP resides.

In the RD transport service, the QP Context is not supplied with this information. Rather, each EEC is supplied with the port number of the local CA port it is to use to transmit and receive packets. The EEC is also programmed with the LID address of the port on the remote CA behind which its companion EEC resides.

Multiple EECs Can Use Same CA Port

Just as multiple RC, UC, or UD QPs can use the same CA port to transmit and receive packets, multiple EECs can also do so. In fact, a port can be used by a combination of multiple non-RD QPs, as well as by one or more EECs. It should be obvious that the CA designer must implement arbitration logic to decide in what order the port is to accept the various outbound packet streams generated by these packet sources. That portion of the implementation is outside the scope

20 UD Transport Service

The Previous Chapter

The previous chapter provided a detailed description of the Reliable Datagram (RD) transport service.

This Chapter

This chapter provides a detailed description of the Unreliable Datagram (UD) transport service.

The Next Chapter

The next chapter provides a detailed description of the Raw IPv6 and Raw EtherType transport services.

UD Support Requirement

HCAs are required to implement the UD transport service type. TCAs are required to support the special, management UD QPs on each port (QP0—the SMI, and QP1—the GSI). A TCA design may also implement additional UD QPs to provide message-passing capability to support TCA design-specific services.

No Responses Expected

In the UD transport service type, no responses are expected. Software on the sender's end is therefore alerted that a message transfer has completed immediately upon the transmission of the message's (note that a UD message must fit entirely in a single request packet) single "Send-Only" request packet.

The Only Operation Supported Is Send

An UD QP only supports the Send operation. RDMA and Atomic operations are not supported.

Maximum Message Length Is One PMTU

An UD message cannot be longer than PMTU bytes in size and is delivered in a single request packet. There is no such thing as a multi-packet UD message (with one exception; see "Reliable Multi-Packet Transaction Protocol" on page 949).

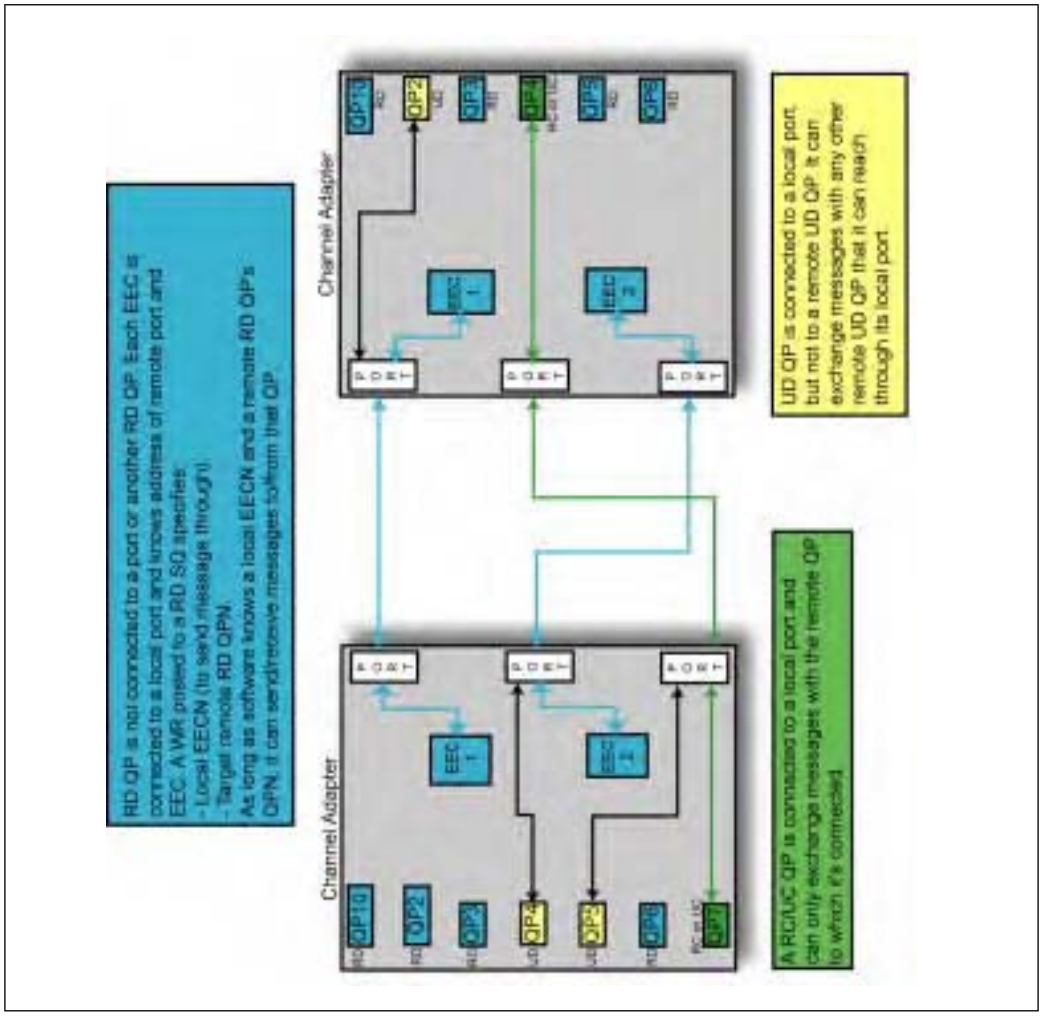
Basic Operational Characteristics

In Figure 20-1 on page 525, refer to QPs 4 and 5 in one CA and QP 2 in the other CA. The UD service has the following basic characteristics:

- **No connection setup.** No initial connection setup with a remote QP is necessary prior to sending or receiving messages.
- **QP attached to specific port.** The QP is associated with a specific local CA port through which it sends and receives messages with remote UD QPs.
- **Port association may limit destinations.** The QP can only send and receive messages with remote UD QPs that can be reached through the local CA port with which the QP is associated.
- **The destination is specified in each WR.** Each WR posted to the QP's SQ can target a different remote UD QP.
- **Message size.** Each message must fit in the data payload field of a single packet (and cannot exceed PMTU bytes in size).
- **No Ack/Nak protocol.** There is no Ack/Nak protocol, so there is no guarantee that packets are delivered.
- **Software completion notification**. There are no responses returned by the remote QP's RQ Logic when using the UD transport service type. That being the case, software on the sender's side is alerted that a message transfer has completed immediately upon the transmission of the last request packet of the Send. RDMA Writes, RDMA Reads, and Atomic requests are not supported.
- **No packet PSN checking.** Although there is a PSN in each packet, it's not meaningful because the entire message is encapsulated in a single packet.
- **CRCs** protect each packet.

- **Support mandatory.** Because management messages (referred to as Management Datagrams, or MADs) are sent using UD, all CAs, switches and routers must support the UD protocol on the QPs used to send and receive management messages. The data payload field of a MAD is always exactly 256 bytes in length.
- **Operations supported.** Only supports the Send message transfer operation (using either the "Send Only" or "Send Only With Immediate" opcode).

Figure 20-1: UD QPs



Messaging with the Desired Remote Service

"UD Connection Issues" on page 202 provides a description of how to locate a desired service in a remote CA that communicates via an UD QP.

Address Handles

Address Handle Specifies Destination Port's Address

When a WR is posted to SQ of an UD QP, the address of the destination port is specified in the form of an Address Handle. The WR also specifies the destination UD QP's QPN.

Create Address Handle Verb

An Address Handle is created using the *Create Address Handle* verb call. It's input parameters specify the destination port's local and, possibly, global address as follows:

- The **HCA Handle** input parameter was returned by an earlier call to the *Query HCA* verb. It identifies the HCA on which the Address Handle is to be created.
- The **Protection Domain** (**PD**) input parameter assigns this Address Handle to the specified PD.
- The **address vector** specifies all of the information necessary to deliver the request packet to the destination port. It consists of the following input parameters:
  - **Service Level**. This the 4-bit SL value to be inserted in the request packet LRH:SL field. It defines the desired QoS within the local subnet.
  - **DLID** address:
    - If the destination CA port is in the same subnet, this is the LID address of the destination CA port.
    - If the destination CA port is in another subnet, this is the LID address of a router ingress port.

- If the destination port is in a different subnet, or if this is a UD multicast operation, the **Send Global Routing Header Flag** bit is set to one, otherwise it is cleared to zero. If it is set to one, this indicates that the following input parameters are also valid:
  - **Flow Label**. If this is a non-zero value, it indicates to routers that all packets with this Flow Label must be delivered to the destination port in order.
  - **Hop Limit**. Defines the number of routers the packet is permitted to hop before it will be dropped (because it has lost its way and has become a wandering soul).
  - **Traffic Class**. It defines the desired QoS end-to-end from the source CA port to the destination CA port.
  - **Source GID index**.
- If the destination port is in a different subnet or if it contains a multicast global address, the **DGID address** of the destination port or the destination multicast group is also supplied.
- **Maximum Static Rate** (i.e., the IPD). See "Packet Injection Delay" on page 43.
- **Source Path Bits**. See "Source Port's LID Address" on page 45.

Other Address Handle-Related Verbs

The other verbs that are used to manipulate an Address Handle are:

- *Query Address Handle* verb. Returns the current contents of the specified Address Handle.
- *Modify Address Handle* verb. Permits an Address Handle's contents to be modified.
- *Destroy Address Handle* verb. Self-explanatory.

PD Check Performed before Message Is Processed

When a Send WR is posted to the QP's SQ, the CA logic compares the PD of the Address Handle specified in the WR to the PD assigned to the QP. If they do not match, one of the following actions is taken:

- The *Post Send Request* verb does not post the WR to the QP's SQ and returns an immediate error, indicating an Invalid Address Handle was specified.
- Alternatively, the following actions may be taken:
  - The WR is not posted to the QP's SQ.
  - An error CQE is created on the SQ's CQ indicating a "Local Operation Error."

21 Raw Transport Service Types

The Previous Chapter

The previous chapter provided a detailed description of the Unreliable Datagram (UD) transport service.

This Chapter

This chapter provides a detailed description of the Raw IPv6 and Raw EtherType transport services.

The Next Chapter

The next chapter provides a detailed description of UD multicasting. This includes creating a multicast group, joining and leaving a group, and group pruning. It also covers multicast loopback and multicast packet distrbution within a receiving CA.

Goal: Tunneling Non-IBA Packets through IBA Network

Assume that software associated with an IBA CA wishes to send a message to a device outside of the IBA fabric and the message must be transmitted in a packet that uses a protocol other than IBA. The destination device resides in another subnet that is not an IBA-environment (e.g., it resides within an Ethernet network). The packet that the IBA CA transmits would therefore have to be an Ethernet packet. This presents the problem of how the packet manages to traverse the IBA subnet (or subnets) until it arrives at a router that is connected to the destination Ethernet network.

The reverse situation may also occur: a non-IBA device may wish to send a non-IBA packet to a software application associated with an IBA CA.

Solution: Disguise It as Special-Purpose IBA Packet

The solution is to encapsulate the Ethernet packet (or whatever type of non-IBA packet it may be) in a special-purpose IBA packet.

Raw QPs Are Used to Transmit/Receive Non-IBA Packets

The IBA specification defines two types of QPs that may be used by software to send and receive non-IBA packets. They are:

- The IPv6 raw datagram QP.
- The EtherType raw datagram QP.

Both QP types are referred to as raw QPs.

Raw QP Support Requirement

It is optional whether or not an HCA or TCA implements the raw QP types. Software local to an HCA can determine the HCA's capabilities by executing the *Query HCA* verb. On return, the following information is provided to the caller:

- The maximum number of Raw IPv6 Datagram QPs supported by this HCA. This value is zero if Raw IPv6 Datagrams are not supported.
- The maximum number of Raw Ethertype Datagram QPs supported by this HCA. This value is zero if Raw Ethertype Datagrams are not supported.
- Ability of this HCA to support raw packet multicast.

To discover whether or not a remote CA supports raw QPs, the CM may send a *Get(ClassPortInfo)* GMP packet to each port's GSI on the remote CA. Bit 11 in the *ClassPortInfo.CapabilityMask* indicates whether the CA supports raw QPs on the port. It should be noted, however, that it doesn't indicate which type of raw QPs are supported (IPv6 or EtherType).

Raw Transport Services Are Unreliable

The raw QPs are a special form of UD QP. No response packets are returned when a request packet is transmitted.

Send and Receive Are Only Supported Operation Types

The only type of operation that can be specified in a WR posted to the SQ is the Send operation (resulting in the transmission of a Send Only request packet). No other operations, including Send Only With Immediate, are supported.

As with any other QP type, the only operation supported on the RQ is the Receive operation.

Basic Operational Description

Each Message Must Fit in One Packet

When using the raw QPs, there is no such thing as a multi-packet message. When a message transfer request is posted to the SQ of a raw QP, the WR's Gather Buffer List must specify a message that fits into the data payload field of a single request packet.

To Activate a Raw QP

Assuming that the HCA supports one or more raw QPs, software activates a raw QP of the desired type (IPv6 or EtherType) by first creating the CQ(s) to be associated with the QP's SQ and RQ (using the *Create CQ* verb) and then executing the *Get Special QP* verb with the following input parameters:

- **HCA Handle**. Identifies the HCA on which the QP is to be activated.
- **HCA port number**. Identifies the HCA port that the QP will use to send and receive message packets.
- **QP type requested**. The allowed types are:
  - SMI QP (QP0).
  - GSI QP (QP1).
  - Raw IPv6.
  - Raw Ethertype.
- **Protection Domain**. Identifies the PD that the QP is to be a member of.
- The handle(s) of the **CQ(s)** to be associated with the SQ and RQ.
- **Requested SQ and RQ size**. The maximum number of outstanding WRs software expects to post to the SQ and RQ.

- **Requested SQ Gather Buffer List size**. The maximum number of Gather Buffer List elements software expects to specify in any WR posted to the SQ.
- **Requested RQ Scatter Buffer List size**. The maximum number of Scatter Buffer List elements software expects to specify in any WR posted to the RQ.
- The **Signaling Type** for the SQ. The valid types are:
  — All WRs posted to the SQ always generate a CQE.
  — Software must specify in each WR posted to the SQ whether to generate a CQE for a successful transfer or only if there is an error.

The output parameters returned to the caller are:

- **QP handle**. This handle must used when making any subsequent *Query QP*, *Modify QP*, or *Destroy QP* verb calls.
- The **actual size of the SQ**. If an error is not returned, this is guaranteed to be greater than or equal to the number requested (note that this may require software to increase the size of the SQ's CQ).
- The **actual size of the RQ**. If an error is not returned, this is guaranteed to be greater than or equal to the number requested (note that this may require software to increase the size of the RQ's CQ).
- The **actual maximum** number of **Gather Buffer List elements** that can be specified in WRs posted to the SQ. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- The **actual maximum** number of **Scatter Buffer List elements** that can be specified in WRs posted to the RQ. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.

To Send Non-IBA Message Packets

Software accomplishes the transmission of a raw request packet in the following manner:

1. Build the message to be transmitted in local memory.
2. Execute the *Post Send Request* verb to post a WR to the QP's SQ specifying:
   — The Gather Buffer List defining the buffer(s) in local memory that contain the message to be transmitted. Each element in the list supplies the virtual start address, the buffer length, and the L_Key needed to access that buffer.
   — Send Without Immediate operation. This is the only operation type supported on the SQ of a raw QP.
   — If this an EtherType raw QP, the EtherType must be specified in the WR.

   - The DLID address of the router ingress port to which the packet is being sent.
   - SL value indicating the desired QoS within this subnet.
   - Offset from the HCA port's base LID address. This selects which of the port's assigned LID addresses is inserted into the packet's SLID field.
   - Maximum Static Rate (i.e., the IPD). Refer to "Static Rate Control" on page 589.
3. When that WQE gets to the head of the SQ, the SQ Logic reads the message from local memory using the Gather Buffer List.
4. The SQ Logic encapsulates the message in an IBA raw EtherType request packet and forwards it to the HCA port for transmission.
5. The packet is transmitted.
6. The SQ WQE is retired and a good completion CQE is created (assuming that the Signaling Type specified that CQEs are always created or that the WR says a CQE must be created on a good completion).

To Receive Non-IBA Message Packets

1. In preparation for the receipt of an non-IBA message, software posts a WR to the RQ using the *Post Receive Request* verb. The WR specifies the Scatter Buffer List defining the local memory buffer(s) to which the message data should be written when it is received.
2. When an inbound non-IBA packet arrives at the RQ Logic, the RQ Logic uses the Scatter Buffer List in the top WQE on its RQ to write the request packet's data payload field into local memory.
3. The RQ WQE is then retired and a CQE is created indicating a good completion.
4. Software examines the CQE, determines that it was a good completion, and processes the non-IBA packet that is in local memory.

Raw Datagrams Do Not Have an ICRC

Raw datagrams do not have an ICRC because the fields within the packet that will not change as the packet transits switches and routers depend on the protocol of the encapsulated packet. This varies greatly from protocol to protocol, so the packet only has a VCRC.

22 Multicasting

The Previous Chapter

The previous chapter provided a detailed description of the Raw IPv6 and Raw EtherType transport services.

This Chapter

This chapter provides a detailed description of UD multicasting. This includes creating a multicast group, joining and leaving a group, and group pruning. It also covers multicast loopback and multicast packet distribution within a receiving CA.

The Next Chapter

The next chapter provides a detailed description of automatic path migration, including possible reasons for migration, priming QPs or EECs for migration, and how a path migration is triggered.

Definition Of Multicasting

A multicast message is, by definition, one packet in length. When a QP transmits a multicast message at the behest of a local application program, the packet's destination address defines to which ports (the multicast group members) the message is delivered.

As the packet arrives at each switch, the switch uses the packet's multicast DLID address to perform a lookup in its Multicast Forwarding Table (see "Multicast Forwarding Table Implemented" on page 675). The table lookup determines through which of the switch's exit ports (see Figure 22-1 on page 564) the packet is to be forwarded.

When the packet arrives at a CA port (see Figure 22-2 on page 565), it is delivered to all QPs within the CA that are members of the group.

If the packet should arrive at a router port, the router uses the packet's GRH:DGID address to perform a lookup in its routing table. There are two possibilities:

- **Link-Local Subnet ID.** If the DGID address in the GRH is FF02:0:0:0:0:0:0:1, the packet is not to leave the subnet. This GID is used as the destination address when multicasting to the QPs that are members of the All CAs Multicast Group.
- **Other Subnets Participate In the Group**. If the DGID address is a multicast GID and a lookup in the router's routing tables indicates that other subnets beyond this router are participating in the group represented by that GID, the router will pass the packet through one or more of its ports towards participating subnets.

Figure 22-1: Switch Performing a Multicast

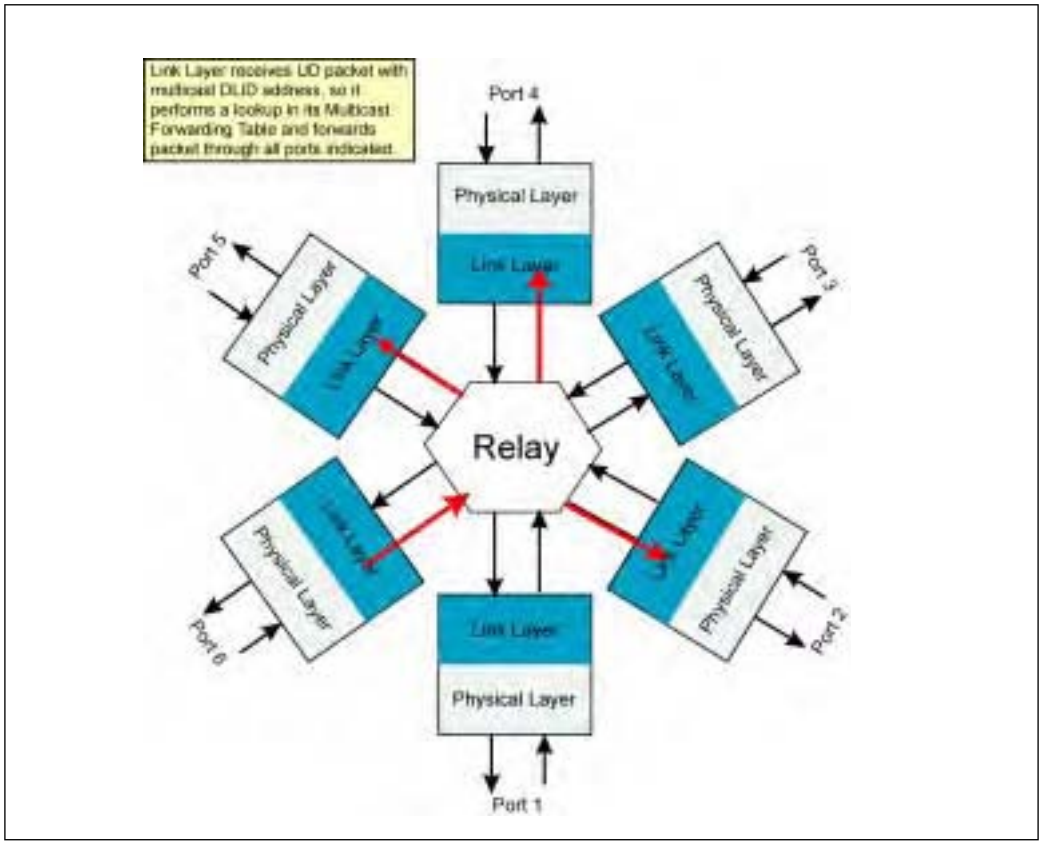
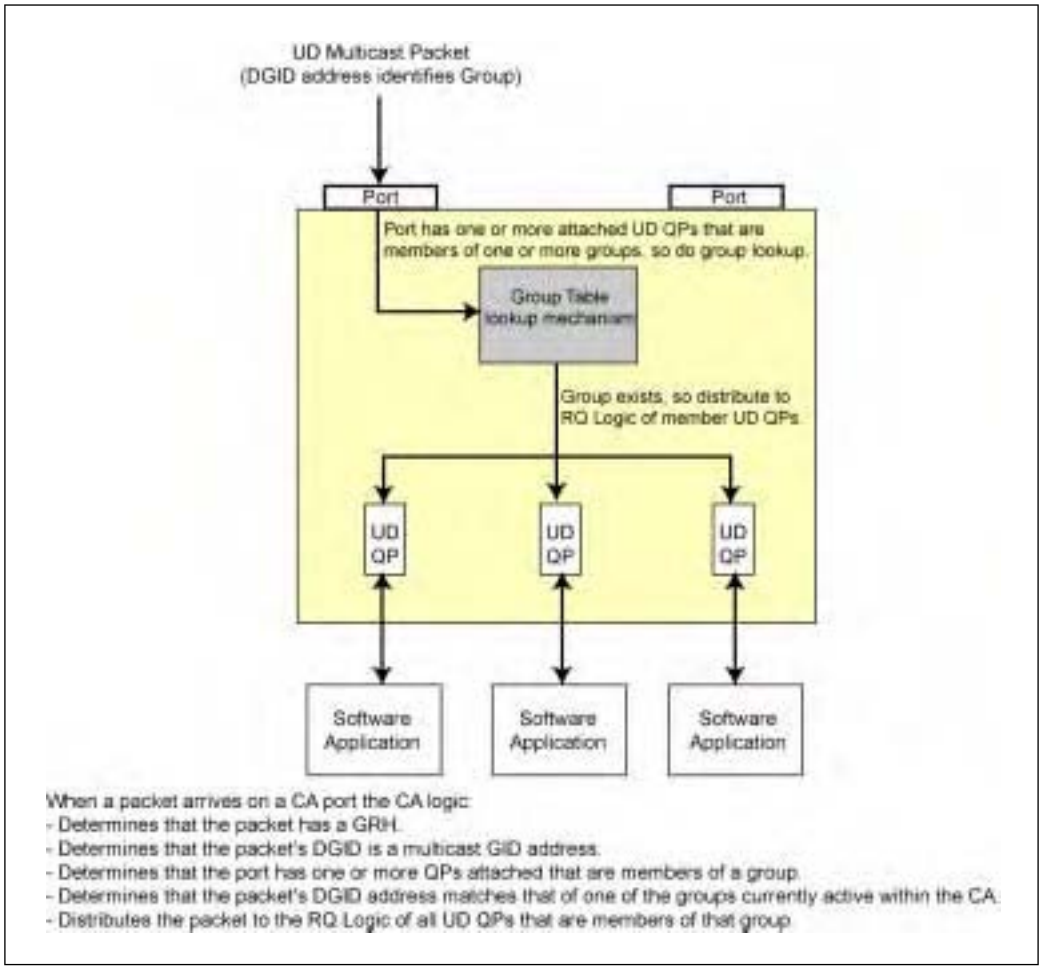


Figure 22-2: Multicast UD Packet Arrives at a CA Port



Support for Multicasting Is Optional

Support for multicasting is defined as follows:

- A switch may or may nor implement the Multicast Forwarding Table. If it does not, then the switch actions are those defined in “When the Table Lookup Fails” on page 677.

- A router may or may not support multicast packet routing. How this is determined is outside the scope of the specification.
- A HCA may or may not support multicasting. This can be determined by performing a *Query HCA* verb call. The following parameters related to multicasting are returned:
  - The maximum number of multicast groups supported by the HCA. This is zero if the HCA does not support UD multicast.
  - The maximum number of QPs that can be attached to multicast groups for this HCA. This value is zero if the HCA does not support IBA unreliable multicast.
  - The maximum number of QPs per multicast group supported by the HCA. This is zero if the HCA does not support UD multicast.
  - Ability of the HCA to support raw packet multicast.
- TCAs may or may not support multicast operations.

If the CI supports UD multicast, it must support at least one multicast group.

Relative to CAs, the specification contains the following statement:

> "It is expected that any implementation of the Unreliable Datagram transport service will trivially support the *generation* of multicast packets."

The author believes that this refers to the fact that a multicast message (i.e., a message with a multicast destination address) can be sent from any UD QP (even one that is not a member of a multicast group).

Only UD and Raw QPs Can Participate in Multicasting

Packet multicasting only applies to the UD and raw packet types. RD, RC, and UC packets always utilize unicast DLID and, when the destination port is in a different subnet, unicast DGID addresses.

UD Multicasting

Creating a Group

Before any UD QPs can become members of a multicast group, the group must be created. A UD multicast group is created in the following manner:

1. An administrative application (outside the scope of the specification) creates a UD multicast group by creating an *MCGroupRecord* in the SA database ("MCGroupRecord" on page 969 defines how this is accomplished). When creating the database record, the application may either specify the multicast GID address that will act as the group identifier or it may leave it to the SA to assign an available multicast GID address.
2. When the application instructs the SA to create the group record in its database, the SA automatically assigns a multicast LID address identifier for the group and returns it to the administrative application. It also returns the assigned multicast GID address that identifies the group.
3. When the new group record is created:
   — The administrative application causes the creation of the group within the HCA. Exactly how this is accomplished is HCA implementation-specific. Basically, the administrative application would instruct the HCA device driver to create a group data structure within (or accessible by) the HCA (refer to the item labelled "Group Table" in Figure 22-2 on page 565). Whenever an incoming multicast UD packet arrives at any of the CA ports, the HCA logic consults this data structure to determine which, if any, UD QPs are members of the group specified by the packet's GRH:DGID field.
   — In addition, software may notify the appropriate routers on the subnet of the new group (the manner in which this is accomplished is outside the scope of the specification). The router update protocol used to accomplish this should determine whether this multicast group is already in operation within another subnet. If so, the router returns the PMTU of the existing multicast group to determine whether the create is allowed or not.

Attaching an UD QP and Its Respective Port to a Group

Accomplished by a Verb Call

A UD QP in the local HCA is made a member of a UD multicast group on the HCA by calling the *Attach QP to Multicast Group* verb. If the HCA doesn't support multicast UD groups, this verb does not have to be implemented. When called, the following input parameters are supplied to the verb:

- HCA Handle.
- Multicast group DLID (this was returned by the SA when it created the group).

23 Automatic Path Migration

The Previous Chapter

The previous chapter provided a detailed description of UD multicasting. This included creating a multicast group, joining and leaving a group, and group pruning. It also covered multicast loopback and multicast packet distrbution within a receiving CA.

This Chapter

This chapter provides a detailed description of automatic path migration, including possible reasons for migration, priming QPs or EECs for migration, and how a path migration is triggered.

The Next Chapter

The next chapter introduces the problem for which Static Rate Control is the solution, and then provides a detailed description of how Static Rate Control works.

Definition of APM

Automatic Path Migration (APM) causes a local and a remote connected QP (RC or UC) or a local and a remote EEC (RD) to cease using the current path to transmit request and response packets and switch to an alternate path.

CA Support Is Optional

Support for APM is optional for both HCAs and TCAs:

- Software may determine whether or not an HCA supports APM by executing the *Query HCA* verb.

- During the connection establishment communications message exchange, the CM sending the REQ message may supply alternate path information to the remote CA's CM. If the remote CA supports APM, the REP message's 2-bit Failover Accepted field will contain 00b, indicating that the alternate path information was accepted and the remote CA therefore supports APM.

A CA that doesn't support APM may ignore the alternate address information if software supplies it in the QP's or EEC's setup.

Causes of a Path Migration

There are two possible causes for a path migration:

- **Software command**. Software commands the two QPs or two EECs to perform a path migration. Some of the circumstances that might cause software to trigger a path migration are:
  - — **Load-balancing**. After examining the various performance counters (see "Performance Management" on page 1019) throughout the fabric, software might trigger a path migration to ease traffic congestion in the fabric.
  - — **Excessive soft failures**. Software may determine (by examining various performance counters) that a specific link or a switch or router is experiencing frequent soft (i.e., correctable) errors that indicate a possible imminent hard failure.
- **Repetitive hardware failure**. A repetitive hardware failure causes the two QPs or two EECs to automatically migrate to the alternate path.

APM-related Elements

The following elements are associated with APM:

- Each packet contains a BTH:**MigReq bit**.
- Each RC and UC **QP** contains an **APM state machine** (see Figure 23-1 on page 583) that may be in one of three possible states:
  - — **Migrated** state. This is the state during normal operation before APM has been enabled.
  - — **ReArm** state.
  - — **Armed** state.
- Each **EEC** contains an **APM state machine** (with the same three states as the QP's APM state machine).
- To use APM, the **software** that sets up the local and remote QP or the local and remote EEC must support APM.

Normal Operation before APM Enabled

Assuming that APM has not been enabled, the circumstances are as follows:

- The APM state machine of both QPs or EECs are in the Migrated state.
- The BTH:MigReq bit is set to one in every request and response packet generated by the two QPs or EECs.
- Each of the two QPs or two EECs has been programmed (during QP or EEC setup) with primary path information but not with alternate path information.

Enabling APM

Software enables APM in the following manner (this discussion assumes that a local and remote QP are just being created and set up by software):

1. The local QP or EEC is created and programmed (via the *Modify QP* or *Modify EEC* verb) with both the primary and alternate path information specified in Table 23-1 on page 578 or Table 23-2 on page 580.
2. The local CA's CM is instructed to send a REQ message to the remote CA's CM requesting the creation of a new QP or EEC. The REQ message contains both the primary and alternate path information to be assigned to the new QP or EEC. A non-zero value in the Alternate Local Port LID field of the REQ message indicates to the receiving CM that APM activation has been requested.
3. The remote CA's CM creates the new QP or EEC and returns the communications REP message to the local CA's CM. The newly created QP's or EEC's state machine is in the RTR state and its APM state machine is in the Migrated state.
4. Local software uses the information contained in the REP message to complete the setup of the local QP or EEC. The local QP's or EEC's state machine is in the fully operational RTS state and its APM state machine is in the Migrated state.
5. The remote CA's QP or EEC transitions to the fully operational RTS state under either of the following circumstances:
   — When the local CA's CM sends a communications RTU (Ready-to-Use) message to the remote CA's CM, the remote CA's CM transitions the newly created QP or EEC to the RTS state.
   — When the remote QP or EEC receives the first message transfer request packet from the local QP or EEC, if the remote QP or EEC is still in the RTR state, it automatically transitions to the RTS state.

6. Using the *Modify QP* or *Modify EEC* verb, software commands the local QP or EEC to transition its APM state machine from the Migrated state to the ReArm state (see Figure 23-1 on page 583).
7. The next packet (either request or response packet) that the local QP or EEC transmits to the remote QP or EEC will have its BTH:MigReq bit = 0.
8. Upon receipt of that packet, the remote CA's QP or EEC transitions from the Migrated to the ReArm state. Both QPs or EECs are now in the ReArm state.
9. The next packet (request or response) transmitted by the QP or EEC in either CA will have BTH:MigReq = 0.
10. When a QP or EEC currently in the ReArm state receives a packet with BTH:MigReq = 0, it automatically transitions to the Arm state. Both QPs or EECs end up in the Arm state. APM is now fully enabled and a path migration can be triggered either when:
    — Software commands (via the *Modify QP* or *Modify EEC* verb) either QP or EEC to transition from the Arm state to the Migrated state.
    — Either QP or EEC experiences Retry Count Exhaustion.

Table 23-1: RC and UC QP Primary and Alternate Path Address Elements

Address Element	Description
P_Key Index	• On packet transmit, the P_KeyIndex selects which of the local CA port's assigned partition keys (from its *P_KeyTable* attribute) is inserted in the packet's BTH:P_Key field. • On packet receive, the P_KeyIndex selects which of the receiving port's assigned partition keys (in its *P_KeyTable* attribute) the inbound packet's BTH:P_Key is compared to.
Local CA Port Number	Specifies the local CA port that the local QP will use to send and receive request and response packets.
Service Level (SL)	Specifies the SL (i.e., QoS) value that is placed in the LRH:SL field of all outbound packets generated by the local QP.

Table 23-1: RC and UC QP Primary and Alternate Path Address Elements (Continued)

Address Element	Description
Destination Port's DLID Address	• If the destination CA port is in the same subnet, the DLID address is set to the address of the destination CA port. • Otherwise, the DLID address is set to the LID address of a router ingress port in the path to the destination CA port.
Maximum Static Rate	Specifies the amount of time that the local QP will delay after transmitting a packet before transmitting the next packet (i.e., IPD value; for more information, see "Static Rate Control" on page 589).
Local Ack Timeout	Applicable only to RC (not UC) QPs. The local QP uses this value to compute its Transport Timer timeout (for more information, see "SQ Logic's Transport Timer" on page 381).
Retry Count	Applicable only to RC (not UC) QPs. This is the local QP's Retry Count for Transport Timer timeouts, PSN Sequence Error Naks, and missing RDMA Read or Atomic Ack packets. *It should be noted that a revision is being made to the 1.0a specification stating that this item is no longer supplied as part of the alternate path information.*
RNR Retry Count	Applicable only to RC (not UC) QPs. When the remote QP's SQ Logic sends a Send or an RDMA Write With Immediate request packet and the local QP's RQ doesn't have a RQ WQE posted to handle it, an RNR Nak is returned with this value in the Timer subfield within the Nak packet's Syndrome field. *It should be noted that a revision is being made to the 1.0a specification stating that this item is no longer supplied as part of the alternate path information.*
Source Path Bits	This specifies which of the local CA port's assigned LID addresses will be inserted in the LRH:SLID field of each packet generated by the local QP.

24 Static Rate Control

The Previous Chapter

The previous chapter provided a detailed description of automatic path migration, including possible reasons for migration, priming QPs or EECs for migration, and how a path migration is triggered.

This Chapter

This chapter introduces the problem for which Static Rate Control is the solution, and then provides a detailed description of how Static Rate Control works. This chapter concludes the part of the book that focuses on the transport service types (i.e., Part 4).

The Next Chapter

The next chapter provides a detailed description of the Link Layer. This includes:

- Link Layer functional overview.
- Link state machine.
- Detailed description of LRH.
- QoS within the subnet: SL and VLs.
- Detailed description of VL arbitration.
- Link-Level flow control.
- Packet CRCs.
- Intro to the packet delimiters.
- Packet receive state machine.
- Data packet check.
- Link packet (Flow Control Packet) check.
- Switch performs packet forwarding.
- Overview of router port's link layer.

How Fast Can a Port Transmit Packets?

The rate at which a CA port can transmit packets is defined by two factors:

- The speed at which a bit stream can be transmitted over the wire (or fiber). The only speed currently supported is 2.5Gb/s (Gigabits/second), but future versions of the specification may permit faster transmit speeds.
- The number of transmit paths (i.e., lanes) the port implements. A port implementation may support one (1x), four (4x), or 12 (12x) lanes.

This yields one of three transmit speeds:

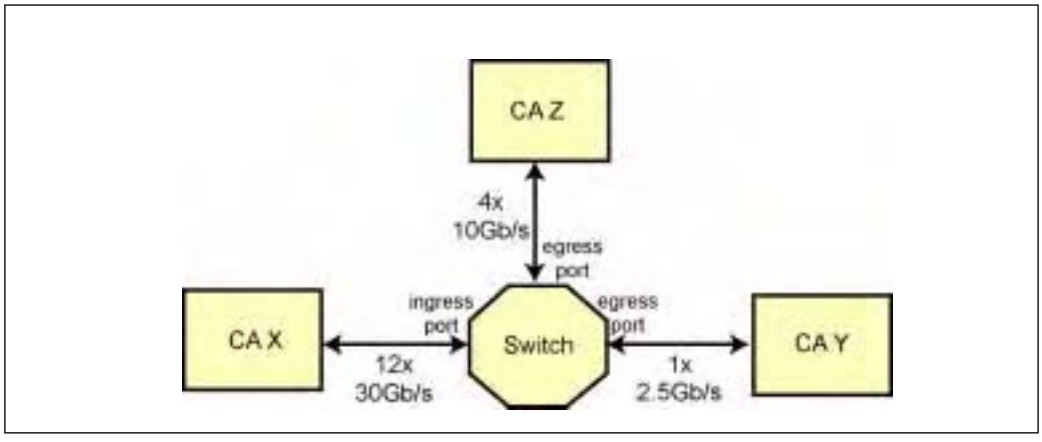
- A 1x port can transmit packets at 2.5Gb/s.
- A 4x port can transmit packets at 10Gb/s.
- A 12x port can transmit packets at 30Gb/s.

Problem: Fast CA Port Can Cause Problems

Refer to Figure 24-1 on page 591. The port on CA X is capable of transmitting data at 30Gb/s. If it were permitted to transmit back-to-back packets to a destination (such as CA Y or CA Z), the result would be as follows:

1. The switch ingress port's VL receive buffer [see "Virtual Lanes" on page 119 and "QoS within the Subnet: SL and VLs" on page 617] selected by the packet's LRH:SL would fill up very rapidly.
2. The ingress port's Link Layer would very quickly fill up the egress port's selected VL transmit buffer.
3. The speed at which the egress port's Link Layer can unload its VL transmit buffer is severely limited by its link speed (in the example, either 3x or 12x slower than the ingress port's link speed).
4. Upon achieving a buffer full condition, the Flow Control Packets (see "Link-Level Flow Control" on page 637) sent back to the corresponding VL transmit buffer in CA X's port Link Layer by the switch ingress port's VL receive buffer will indicate the switch's VL receive buffer is full (i.e., no credits available). The specification refers to this as "back pressure."
5. CA X's VL transmit buffer is prevented from transmitting any additional packets from that VL transmit buffer due to the lack of buffer space in the switch ingress port VL's receive buffer.
6. CA X's VL transmit buffer quickly becomes full and stops accepting packets for transmission to *any* destination CA port from *any* QPs and/or EECs within CA X whose specified SL selects the same VL transmit buffer.

Figure 24-1: Paths with Non-Uniform Transfer Speeds



Solution

This problem is solved by a mechanism referred to as Static Rate Control that prevents a QP's SQ Logic or an EEC's Send Logic from forwarding back-to-back packets to the same local CA port for transmission to the same destination port (as defined by the packet's DLID address).

Rather than streaming back-to-back packets to the same port for delivery to the same destination DLID address, the QP or EEC must delay a programmed amount of time (referred to as the Inter-Packet Delay—IPD) before forwarding another packet to the same local CA port for delivery to the same destination DLID address. The ultimate CA port address may be in the same or a different subnet, but if the destination port within the local subnet is the same (as defined by the DLID address), the Static Rate Control mechanism must be used.

The specification mandates that the Static Rate Control mechanism must be used when a QP or EEC transmits packets through a local CA port connected to a 4x or 12x link and configured to utilize all four or all 12 lanes when transmitting packets.

The maximum transfer rate a port supports is defined by the following two *PortInfo* attribute elements:

- *PortInfo.LinkWidthSupported*:
  - 01h = 1x.
  - 02h = Reserved.

  - 03h = 1x or 4x.
  - 04h-10h = Reserved.
  - 11h = 1x, 4x or 12x.
  - 12h-FFh = Reserved.
- *PortInfo.LinkSpeedSupported*:
  - 0h = Reserved.
  - 1h = 2.5Gb/s.
  - 2h-Fh = Reserved.

The actual transmit rate a CA port is using is defined by how the SM configured the following two attributes:

- *PortInfo.LinkWidthActive*:
  - 00h = Reserved.
  - 01h = 1x.
  - 02h = 4x.
  - 03h-07h = Reserved.
  - 08h = 12x.
  - 09h-FFh = Reserved.
- *PortInfo.LinkSpeedActive:*
  - 0h = Reserved.
  - 1h = 2.5Gb/s.
  - 2h-Fh = Reserved.

IPD Calculation and Source

IPD Calculation

The time to wait before transmitting a subsequent packet through the same local CA port to the same DLID address is based on the time it takes to transmit the current packet.

The second packet isn't forwarded to the local CA Port until Ts has elapsed since the previous packet was forwarded to the port for transmission. Ts is calculated as:

(IPD + 1) X time to transmit the previous packet

where IPD is an 8-bit value defined as per Table 24-1 on page 594, and the time it takes to transmit a packet is calculated as:

(LRH:PktLen X 4)/Lr

where Lr = the local CA port speed calculated using its *LinkWidthActive* and *LinkSpeedActive* attribute elements. The PktLen value is multiplied by four to yield the number of bytes in the packet.

IPD Source

The IPD value to be used for a path between the source port and a given destination port (as defined by its DLID address) is based upon a value obtained from the SA's *PathRecord* (refer to "PathRecord" on page 975 for more information) that defines the speed of the overall path (as determined by the SM during subnet discovery and configuration). The 6-bit *PathRecord.Rate* field defines the path's overall speed as follows:

- **1 = 1Gb/s**. The IPD to be used depends on the speed of the source port:
  - If the source port speed is 2.5Gb/s, the IPD = 2. This would yield a source port transmit speed of 2.5Gb/s ÷ 3 (IPD + 1) = 833Mb/s.
  - If the source port speed is 10Gb/s, the IPD = 9. This would yield a source port transmit speed of 10Gb/s ÷ 10 (IPD + 1) = 1Gb/s.
  - If the source port speed is 30Gb/s, the IPD = 29. This would yield a source port transmit speed of 30Gb/s ÷ 30 (IPD + 1) = 1Gb/s.
- **2 = 2.5Gb/s.** The IPD to be used depends on the speed of the source port:
  - If the source port speed is 10Gb/s, the IPD = 3. This would yield a source port transmit speed of 10Gb/s ÷ 3 (IPD + 1) = 2.5Gb/s.
  - If the source port speed is 30Gb/s, the IPD = 11. This would yield a source port transmit speed of 30Gb/s ÷ 12 (IPD + 1) = 2.5Gb/s.
- **3 = 10 Gb/s**. The IPD to be used depends on the speed of the source port:
  - If the source port speed is 10Gb/s, the IPD = 0. This would yield a source port transmit speed of 10Gb/s ÷ 1 (IPD + 1) = 10Gb/s.
  - If the source port speed is 30Gb/s, the IPD = 2. This would yield a source port transmit speed of 30Gb/s ÷ 3 (IPD + 1) = 10Gb/s.
- **4 = 30 Gb/s**. In this case, every link between the source and destination port are 12x links, so the IPD is 0 (i.e., no delay is necessary between the transmission of packets to that destination through the local CA port).
- **5-64** = Reserved.

While all 256 possible IPD values are valid, Table 24-1 on page 594 defines the minimum number of values that must be supported by a CA design. The specification contains the following statement:

> "If a CA is requested to use a value that it doesn't support, the CA will use a supported value, and return that value in the appropriate MADs or verbs."

25 Detailed Description of the Link Layer

The Previous Chapter

The previous chapter introduced the problem for which Static Rate Control is the solution, and then provided a detailed description of how Static Rate Control works. It concluded the part of the book that focuses on the transport service types (i.e., Part 4).

This Chapter

This chapter provides a detailed description of the Link Layer. This includes:

- Link Layer functional overview.
- Link state machine.
- Detailed description of LRH.
- QoS within the subnet: SL and VLs.
- Detailed description of VL arbitration.
- Link-Level Flow Control.
- Packet CRCs.
- Intro to the packet delimiters.
- Packet receive state machine.
- Data packet check.
- Link packet (Flow Control packet) check.
- Switch performs packet forwarding.
- Overview of router port's link layer.

The Next Chapter

The next chapter provides a detailed description of the Physical Layer and includes:

- Module basics.
- Port types.
- Signal naming conventions.
- Electrical signaling and copper cable.
- Link Layer to Physical Layer interface.
- Transmit logic functions.
- Receiver logic functions.
- Link training.
- Physical Layer error handling.
- Repeaters.
- Performance counters.

Link Layer Functional Overview

The Purpose of the Link Layer

The Link Layer is responsible for sending and receiving data across the fabric at the packet level rather than the message level (that is the responsibility of the Transport Layer). The Link Layer defines the packet format (other than the GRH which is handled at the Network Layer and is only present when the source and destination are in different subnets), as well as how packets are switched within a subnet between the source and destination ports. It also handles flow control (transmit/receive buffer management) at the physical link level between the transmitter on one end and the receiver on the other end of the physical link. The Link Layer's functions include:

- Buffering.
- Flow control.
- Error detection.
- Routing the packet within the local subnet.
- Address decode.

This chapter provides a detailed description of the Link Layer's functionality.

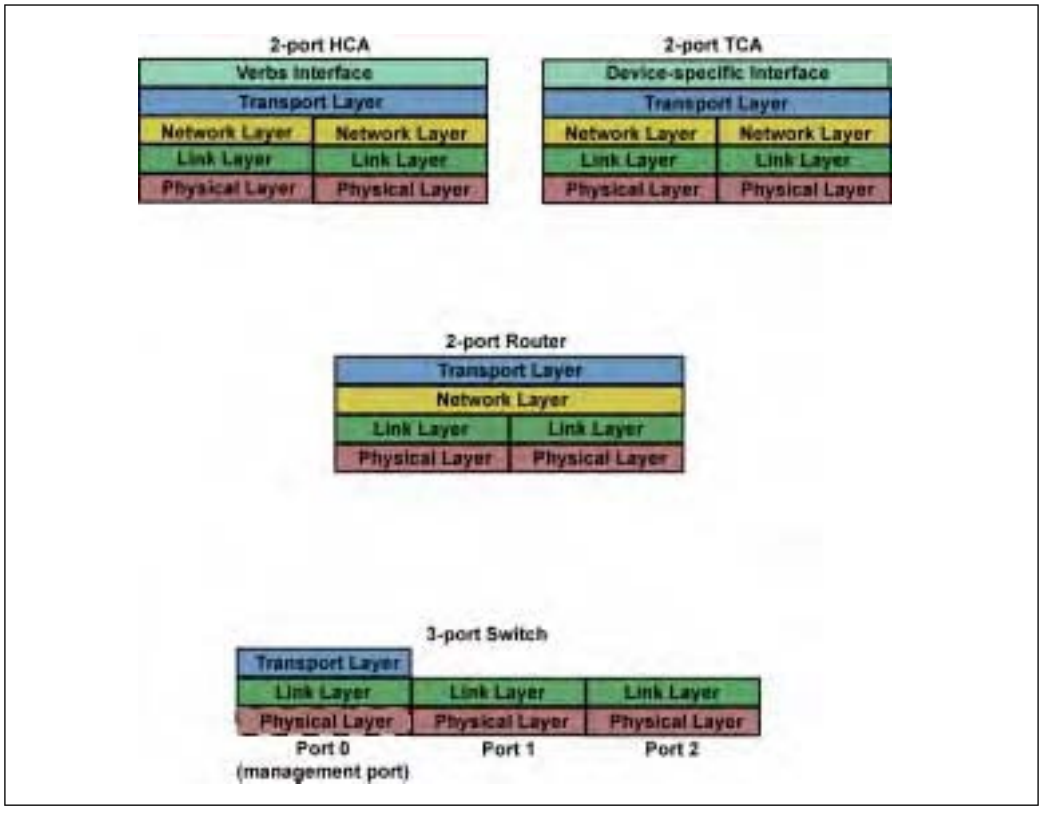
Devices That Implement the Link Layer

Refer to Figure 25-1 on this page. The Link Layer is employed in:

- **CAs**. The Link Layer associated with each CA port handles buffering, flow control, error detection, and port address decode (on packet reception).
- **Switches**. The Link Layer associated with each switch port handles buffering, flow control, error detection, and packet switching (i.e., routing) within the local subnet.
- **Routers**. The Link Layer associated with each router port handles buffering, flow control, error detection, and port address decode. It passes the packet to the router's Network Layer to determine to which attached subnet(s) the packet must be routed.

The Link Layer is not used in repeaters.

Figure 25-1: The Link Layer in a CAs, Switches, and Routers



Link State Machine

Introduction

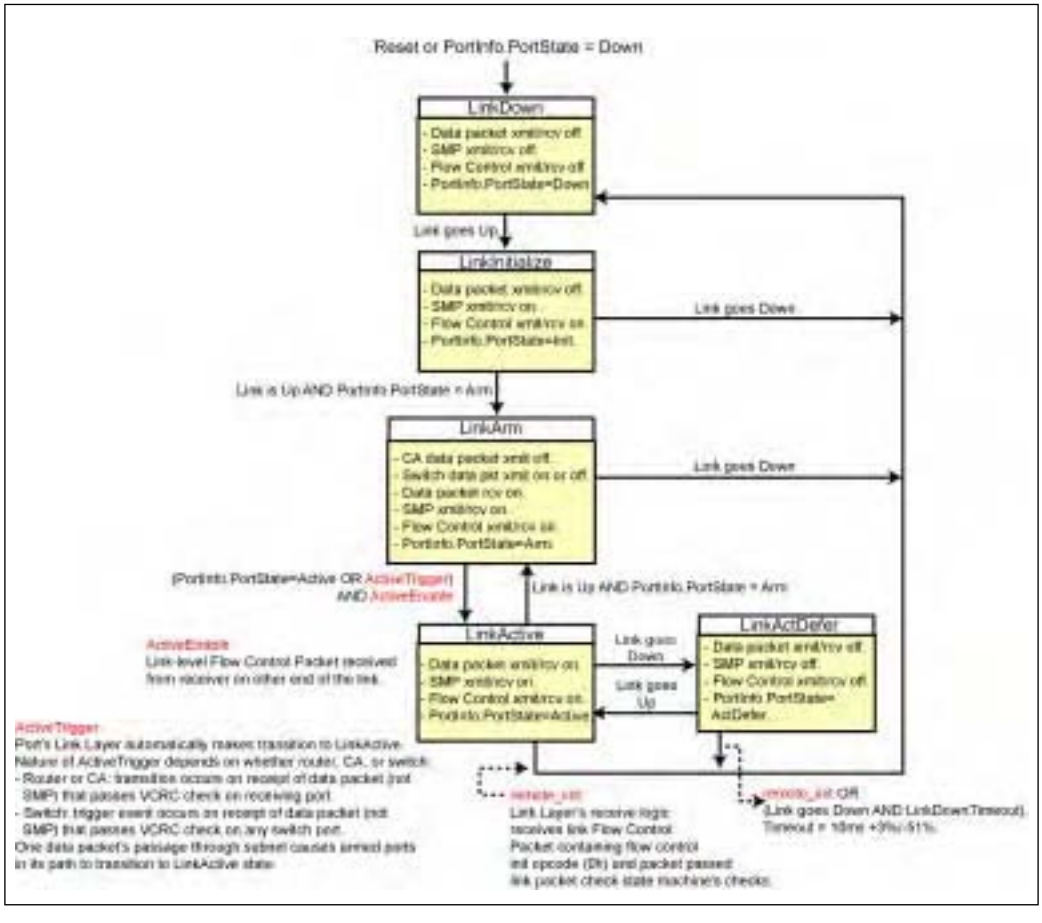
Figure 25-2 on page 603 illustrates the Link Layer state machine. It should be noted that the Physical Layer signals its current state (up or down) to the Link Layer. The actual signal name is immaterial, but the specification uses the name PhyLink. At a given moment in time, the Link Layer is in one of the following states:

- **LinkDown state**. While in the LinkDown state, the Link Layer has the following operational characteristics:
  - The Link Layer's ability to transmit and receive data packets is disabled. Data packets are defined as all packet types other than SMPs or link-level Flow Control Packets (FCPs). See "Link-Level Flow Control" on page 637.
  - The Link Layer's ability to receive and send SMPs is disabled.
  - The Link Layer's ability to receive and send Link Layer Flow Control Packets (FCPs) is disabled.
- **LinkInitialize state**. Entered when the Physical Layer signals that it is operational. While in the LinkInitialize state, the Link Layer has the following operational characteristics:
  - The Link Layer's ability to transmit and receive data packets is disabled.
  - The Link Layer's ability to receive and send SMPs is enabled.
  - The Link Layer's ability to receive and send Link Layer Flow Control Packets (FCPs) is enabled.
- **LinkArm state**. Entered on receipt of an SMP commanding the state change. While in the LinkArm state, the Link Layer has the following operational characteristics:
  - In a CA, the port's ability to transmit data packets is disabled. In a switch design, the port may or may not be able to transmit data packets.
  - The Link Layer's ability to receive data packets is enabled.
  - The Link Layer's ability to receive and send SMPs is enabled.
  - The Link Layer's ability to receive and send Link Layer Flow Control Packets (FCPs) is enabled.
- **LinkActive state**. This is the fully operational state. While in the LinkActive state, the Link Layer has the following operational characteristics:
  - The Link Layer's ability to transmit and receive data packets is enabled.
  - The Link Layer's ability to receive and send SMPs is enabled.
  - The Link Layer's ability to receive and send Link Layer Flow Control Packets (FCPs) is enabled.

- **LinkActDefer state**. If the Physical Layer indicates that the external link has gone down, the Link Layer temporarily exits LinkActive and goes to LinkActDefer (Link Active Defer) until either the Physical Layer indicates that the external link is back up or until the LinkDownTimeout has expired (whichever occurs first). While in the LinkActDefer state, the Link Layer has the following operational characteristics:
  — The Link Layer's ability to transmit and receive data packets is disabled.
  — The Link Layer's ability to receive and send SMPs is disabled.
  — The Link Layer's ability to receive and send Link Layer Flow Control Packets (FCPs) is disabled.

The subsections that follow provide a detailed description of each state.

Figure 25-2: The Link State Machine



26 Detailed Physical Layer Description

The Previous Chapter

The previous chapter provided a detailed description of the Link Layer. This included:

- Link Layer functional overview.
- Link state machine.
- Detailed description of LRH.
- QoS within the subnet: SL and VLs.
- Detailed description of VL arbitration.
- Link-Level Flow Control.
- Packet CRCs.
- Intro to the packet delimiters.
- Packet receive state machine.
- Data packet check.
- Link packet (Flow Control packet) check.
- Switch performs packet forwarding.
- Overview of router port's Link Layer.

This Chapter

This chapter provides a detailed description of the Physical Layer and includes:

- Module basics.
- Port types.
- Signal naming conventions.
- Electrical signaling and copper cable.
- Link Layer to Physical Layer interface.
- Transmit logic functions.
- Receiver logic functions.
- Link training.
- Physical Layer error handling.

- Repeaters.
- Performance counters.

This chapter concludes the part of the book that focuses on the Link and Physical Layers (i.e., Part 5).

The Next Chapter

The next chapter provides a detailed descripion of the Subnet Management Interface (SMI). It includes:

- Purpose of the SMI (QP0).
- The SMI on a CA and a router.
- The SMI on switches.
- Detailed switch handling of SMPs.
- Detailed CA or router handling of SMPs.
- SM accessing an attribute in its local device.
- SM can reside in a switch.
- SMI is a privileged resource.
- SMI only communicates with other SMIs.
- The port states in which SMPs can be sent and received.
- SMPs never leave the subnet.
- Setting up an HCA port's SMI.
- How the SM sends a message and handles a response.
- An SMP's source and destination.
- The SMI and the Q_Key.
- The SMI and partitions.
- Additional reference material.

Module Basics

A 3U or 6U chassis contains a backplane with connectors into which modules plug:

- IBA module mechanical elements:
  - — Module board with electronics.
  - — Module cover.
  - — Module ejector and latch.
- Hot plug support:
  - — Staged power pins on connector.
  - — OS power management support.

Definition of an IBA module:

- Minimally consists of at least one IB board, a carrier module, and a cover.
- Installs into IB chassis slot.
- Two module heights and widths:
  - — Standard single-width module.
  - — Standard double-width module (occupies two IB chassis slots).
  - — Tall module (single-width).
  - — Tall double-width module.

General

Some general physical characteristics of the physical link are:

- 2.5Gb/s (wire speed).
- A link is a point-to-point connection between ports on two devices.
- A link can have 1x, 4x, or 12x "data lanes." Each data lane consists of a transmit wire and a receive wire.
- In a multi-lane implementation, the data is byte-striped across the lanes.
- The link is a dual-simplex (i.e., full-duplex) connection permitting simultaneous data transmission in both directions.
- Data is transmitted over a transmit signal pair and received over a receive signal pair.
- There is no carrier sense or collision detect (as in Ethernet).

Port Types

A physical port is a set of signals on a connector interface. The physical port types are:

- **Backplane Port** (see Figure 26-1 on page 684). An IBA module plugs into this port type on the chassis backplane. This port type is electrical rather than fiber optic, and it contains all four signal groups.
- **Cable Port** (see Figure 26-2 on page 685). This electrical port is implemented on the front edge of an IBA module and is implemented as a connector into which an electrical cable plugs. It only implements the Signaling group (not the Hardware Management Group, the Bulk Power Group, or the Auxiliary Power Group).
- **Fiber Optic Port** (see Figure 26-3 on page 685). This fiber optic port is implemented on the front edge of an IBA module and is implemented as a connector into which a fiber optic cable plugs. It only implements the Signaling group (not the Hardware Management Group, the Bulk Power Group, or the Auxiliary Power Group).

Signal Naming Conventions

Refer to Figure 26-1 on this page, Figure 26-2 on page 685, and Figure 26-3 on page 685. The specification uses the following documentation convention in signal names:

- "w" in a signal name is replaced by:
  - "p" for the positive rail of a differential pair.
  - "n" for the negative rail of a differential pair.
- "x" in a signal name is replaced by the port number.
  - Can range from 0 to n–1, where n is number of connectors/ports on the board edge or the backplane.
- "s" in the signal name:
  - On a board, "s" is replaced by:
    - "t" for an IO plate connector or
    - "b" for a backplane connection.
  - On a backplane, "s" is replaced by the slot number.
- "I" = input signal and "O" = output signal.

Figure 26-1: Backplane Port Signals

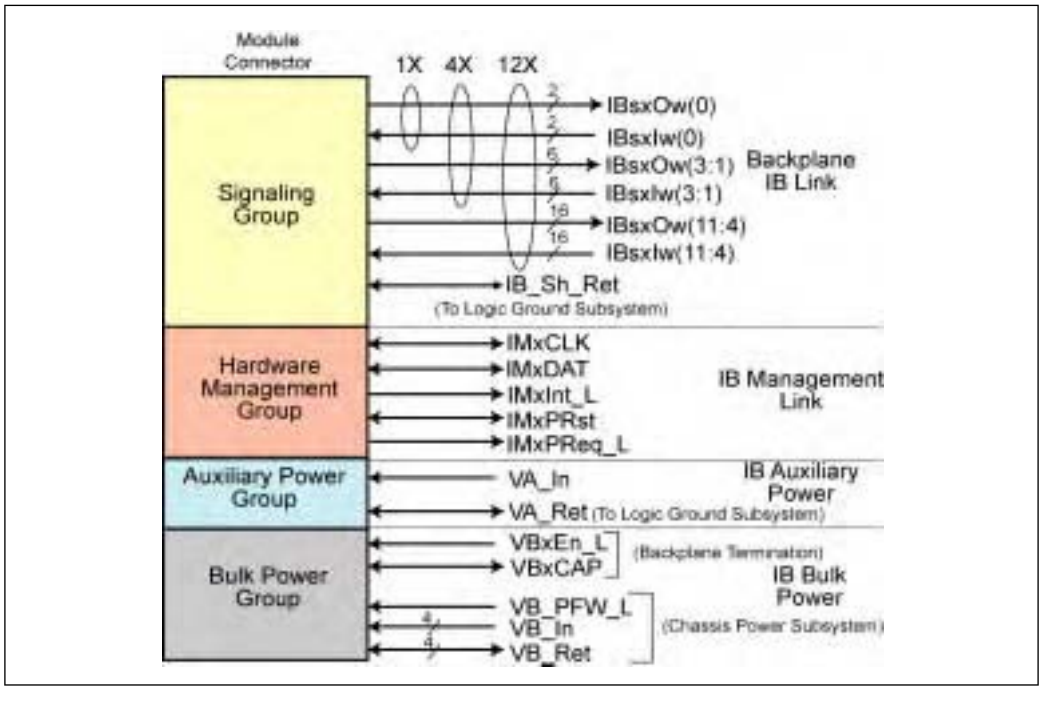


Figure 26-2: Cable Port Signals

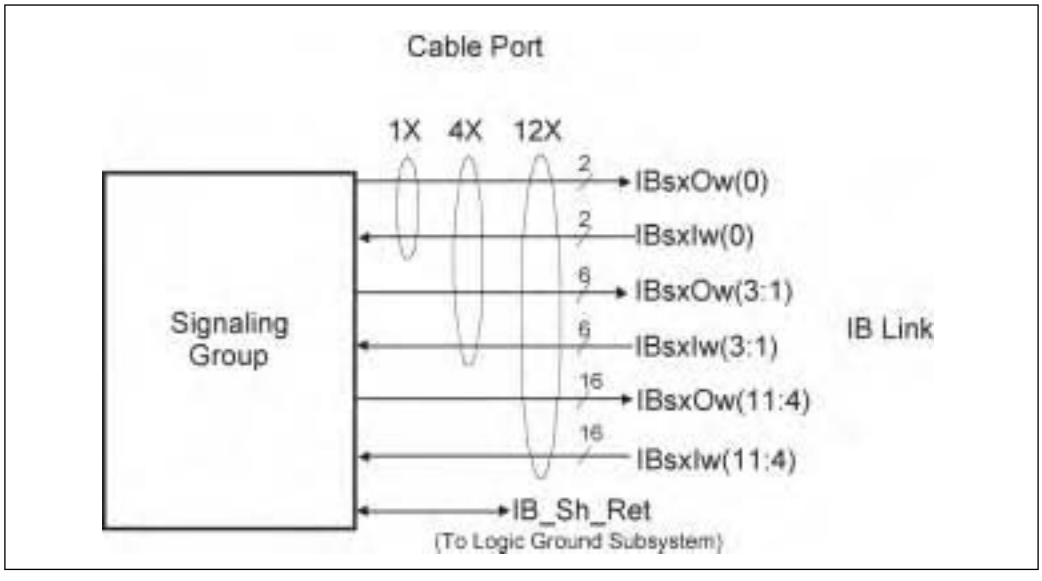
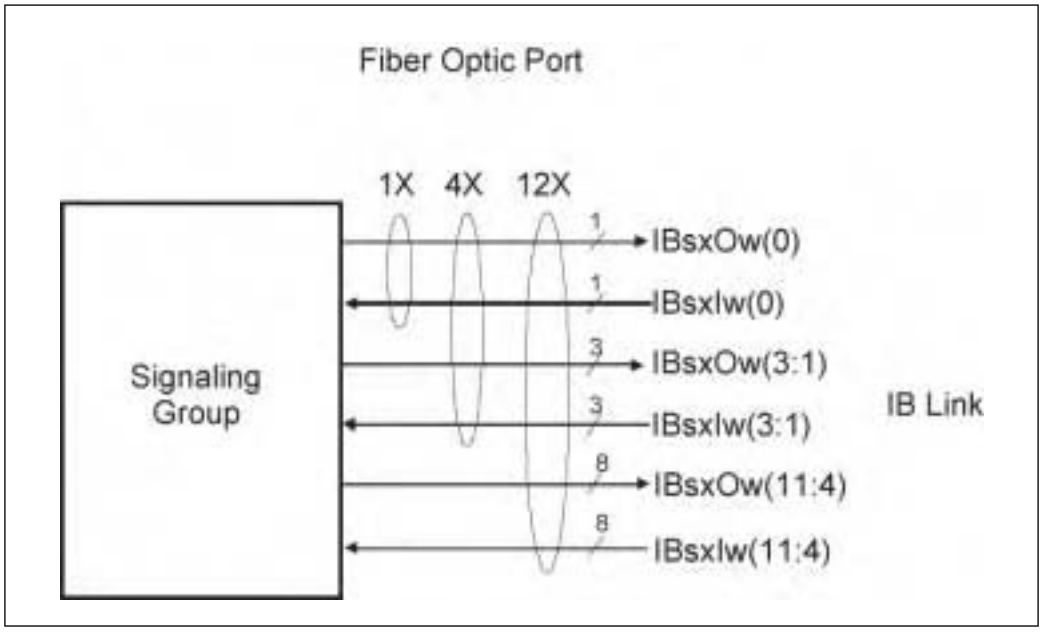


Figure 26-3: Fiber Optic Port Signals



27 The SMI

The Previous Chapter

The previous chapter provided a detailed description of the Physical Layer and included:

- Module basics.
- Port types.
- Signal naming conventions.
- Electrical signaling and copper cable.
- Link Layer to Physical Layer interface.
- Transmit logic functions.
- Receiver logic functions.
- Link training.
- Physical Layer error handling.
- Repeaters.
- Performance counters.

It concluded the part of the book that focused on the Link and Physical Layers (i.e., Part 5).

This Chapter

This chapter provides a detailed descripion of the Subnet Management Interface (SMI). It includes:

- Purpose of the SMI (QP0).
- The SMI on a CA and a router.
- The SMI on switches.
- Detailed switch handling of SMPs.
- Detailed CA or router handling of SMPs.
- SM accessing an attribute in its local device.
- SM can reside in a switch.
- SMI is a privileged resource.
- SMI only communicates with other SMIs.
- The port states in which SMPs can be sent and received.

- SMPs never leave the subnet.
- Setting up an HCA port's SMI.
- How the SM sends a message and handles a response.
- An SMP's source and destination.
- The SMI and the Q_Key.
- The SMI and partitions.
- Additional reference material.

The Next Chapter

The next chapter provides a detailed description of the various types of management datagrams (MADs) and includes:

- Definition of a MAD.
- Software times the return of a MAD's response.
- Basic MAD contents.
- SMP MADs.
- GMP MADs.
- Traps.
- Event subscription and event forwarding.
- The Notice queue.

Purpose of the SMI (QP0)

The SMI is a special UD QP. A separate SMI (QP0) is associated with each port on a CA or a router. On a switch, the SMI is only implemented on the switch management port (port 0).

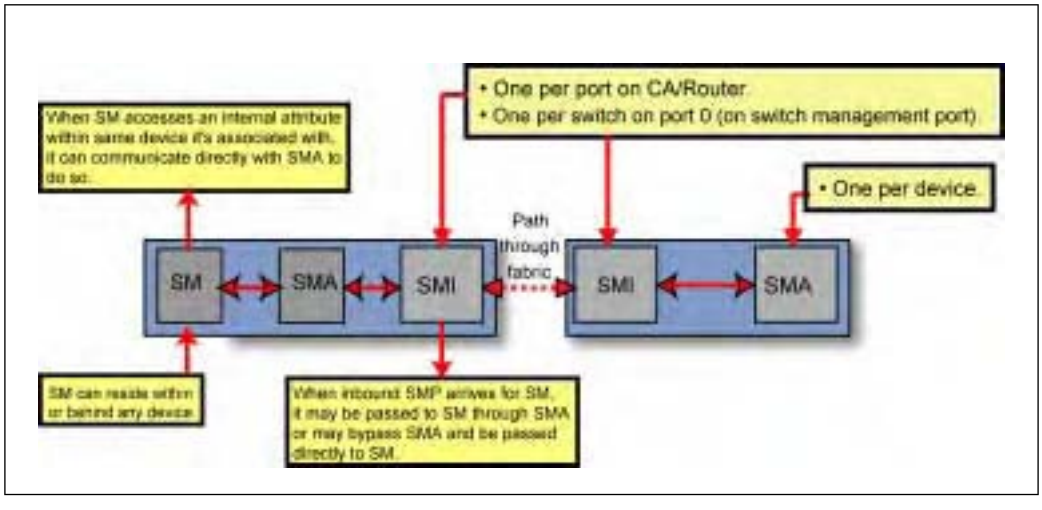
The SMI was introduced in "The SM" on page 160. It only has one purpose: it is used by the SM and the SMAs within CAs, routers, and switches to send and receive SMPs. Refer to Figure 27-1 on page 763. "SMP MADs" on page 787 provides a detailed description of SMPs. SMPs are used in the following manner:

1. The SM uses the SMI of a port on the device it is associated with to transmit a request SMP to a device's SMA instructing it to perform a specified action (referred to as a method) on a specified device attribute. As noted in the figure, the SM may either:
   — pass the SMP to the local device's SMA which, in turn, passes it to the port's SMI for transmission, or
   — pass the SMP directly to the port's SMI for transmission.
2. The port's SMI SQ Logic transmits the SMP to the SMI associated with the destination CA. Upon packet receipt and validation, the port's SMI delivers the SMP to the SMA within the destination device.

3. The SMA performs the specified method on the specified attribute.
4. In most cases, as defined by the method/attribute combination, the SMA is required to return a response SMP back to the SM. To do so, the SMA reverses the addresses that were in the request packet's LRH:SLID and LRH:DLID fields and instructs the receiving port's SMI to send the response SMP back to the SM.
5. Upon receipt of the response SMP, the SMI on the receiving port of the SM's associated device passes the response SMP back to the SM. As noted in the illustration, it may either pass it directly to the SM or may pass it to the device's SMA which, in turn, passes it to the SM.

In some circumstances, a device's SMA may send an unsolicited *SubnTrap(Notice)* SMP to the SM to notify it of an event that occurred within the device. Refer to "Traps" on page 790 for more information on Trap SMPs and also refer to "Switch Transmission of a Trap(Notice) SMP" on page 766 and "CA or Router Transmission of SubnTrap(Notice) SMP" on page 769.

Figure 27-1: Relationship of the SM, SMIs, and SMAs



The SMI on a CA and Router

A separate SMI (i.e., QP0) and GSI (QP1) are implemented on each port of a CA or a router. This permits the CA or router to transmit and receive SMPs and GMPs through any port.

The SMI on Switches

Each switch implements a special management port, port 0. This is the only switch port that implements a Transport Layer as it is the only switch port that implements QPs. Those QPs are QP0 (the SMI) and QP1 (the GSI). Switch port 0 has some other special qualities:

- It is typically implemented as a strictly internal port with no Physical Layer.
- It is the only switch port that is assigned a LID address by the SM. It should be noted that although a base LID address is assigned to switch port 0 by the SM, this port does not implement the *PortInfo.LMC* field, so it cannot be assigned a range of LID addresses.

The other switch ports are not addressable and merely act as conduits for receiving and forwarding packets. When a switch port other than port 0 receives an inbound SMP or GMP, the packet's LRH:DLID address is used to direct the packet. See "Switch Handling of an Inbound Request SMP" on this page.

Detailed Switch Handling of SMPs

Switch Handling of an Inbound Request SMP

When a port's Link Layer receives an inbound request SMP from the SM, it is processed as follows:

1. The SMP packet is received into the port's VL15 receive buffer. It should be noted that if the buffer is currently full—the minimum required buffer depth is one SMP packet—then the inbound SMP is discarded.
2. The SMP's DLID address is used to route the packet. Based on the address in the DLID field, there are three possible cases:
   — **DLID = PLID address**. This is a directed-route SMP. The receiving port passes the directed-route SMP to port 0's SMI RQ Logic for processing. A detailed description of directed-route SMPs can be found in "Discovery" on page 871.
   — **DLID contains a unicast address other than switch port 0's assigned address or the PLID address (i.e., FFFFh).** In this case, the switch performs a lookup in its Forwarding Table (either the Linear Forwarding Table—see "Linear Forwarding Table (LFT)" on page 669, or the Random Forwarding Table—see "Random Forwarding Table (RFT)" on page 671) and forwards the packet through the selected exit port. If the

selected exit port is port 0, then the packet is internally forwarded to the RQ Logic of port 0's SMI or GSI for processing (see the next numbered step).

— **DLID contains the LID address assigned to switch port 0**. In this case, the packet is internally forwarded to the RQ Logic of port 0's SMI for processing.

3. Upon the SMP's arrival at switch port 0's SMI RQ Logic, there are two possibilities:
   — **DLID = PLID**. In this case, this is a directed-route SMP, and it its processing is described in "Packet Routing During Discovery" on page 873.
   — **DLID = port 0's assigned LID address**. In this case, the packet processing is described in the steps that follow.
4. The packet's 256-byte data payload (the SM MAD) is passed to the SMA within the switch for processing.
5. Refer to Table 28-5 on page 788 and Table 28-6 on page 789. The SMA performs the method specified in the MAD on the specified attribute. The method in an inbound request SMP can be one of the following:
   — ***Get***. In response, the SMA reads the contents of the indicated attribute and places its contents in the 64-byte data field of a 256-byte response MAD. The SMA then tells port 0's SMI SQ Logic to transmit the response MAD back to the SM. The method in the response MAD is the *GetResp* method. See the next numbered step for the remaining actions taken.
   — ***Set***. In response, the SMA writes the data contained in the MAD's data field into the indicated attribute, reads it back, and places its contents in the data field of a 256-byte response MAD. The SMA then tells port 0's SMI SQ Logic to transmit the response MAD back to the SM. The method in the response MAD is the *GetResp* method. See the next numbered step for the remaining actions taken.
   — ***TrapRepress***. This tells the SMA to stop repeatedly sending a particular *SubnTrap(Notice)* MAD. See "Traps" on page 790 for a description of traps and "Switch Transmission of a Trap(Notice) SMP" on page 766. In response, the SMA stops transmitting the specified *SubnTrap(Notice)* MAD, completing the processing of the MAD. There is no response.
6. The SMA swaps the LID addresses (SLID and DLID) that were in the request SMP, sets the BTH:DestQP field to QP0 (the SMI), and passes the packet back to the SMI, which, in turn, passes it to the switch port that received the packet (automatically captured in the *PortInfo.LocalPortNum* attribute element). That port accepts the packet into its VL15 transmit buffer and transmits the packet. The receiving port's Link Layer accepts the response SMP into its VL15 transmit buffer and forwards the response packet back to the SM.

28 Detailed Description of MADs

The Previous Chapter

The previous chapter provided a detailed descripion of the Subnet Management Interface (SMI). It included:

- Purpose of the SMI (QP0).
- The SMI on a CA and a router.
- The SMI on switches.
- Detailed switch handling of SMPs.
- Detailed CA or router handling of SMPs.
- SM accessing an attribute in its local device.
- SM can reside in a switch.
- SMI is a privileged resource.
- SMI only communicates with other SMIs.
- The port states in which SMPs can be sent and received.
- SMPs never leave the subnet.
- Setting up an HCA port's SMI.
- How the SM sends a message and handles a response.
- An SMP's source and destination.
- The SMI and the Q_Key.
- The SMI and partitions.
- Additional reference material.

This Chapter

This chapter provides a detailed description of the various types of management datagrams (MADs) and includes:

- Definition of a MAD.
- Software times the return of a MAD's response.
- Basic MAD contents.
- SMP MADs.
- GMP MADs.
- Traps.
- Event subscription and event forwarding.
- The Notice queue.

The Next Chapter

The next chapter provides a detailed description of:

- SM MAD formats.
- SM methods.
- SM attributes.
- SM traps.
- SMA Notice support.

Definition of a MAD

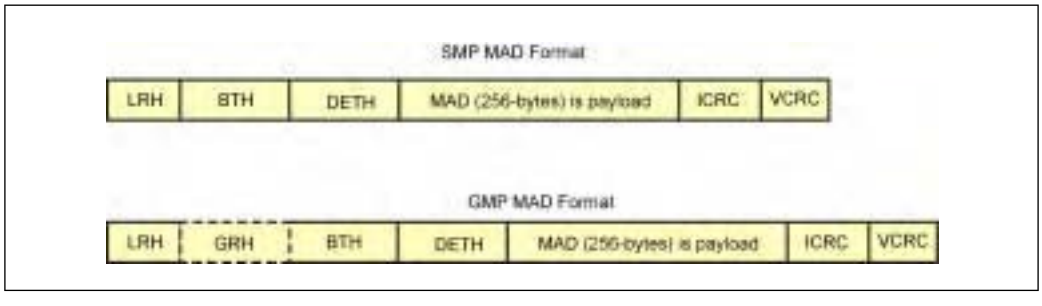
Refer to Figure 28-1 on page 781. A Management Datagram (MAD) is always 256 bytes in length and is carried in the data payload field of either an SMP or a GMP. While an SMP MAD only contains an LRH (it never contains a GRH) and cannot be used to address a port outside of the SM's subnet, a GMP can contain a GRH and can be used to address a port outside of the subnet. This means that a GSM can be used to manage devices in multiple subnets.

A MAD is generated under the following circumstances:

- SMP MADs:
  - — The Master SM issues a SM MAD to a device's SMA requesting that a *SubnGet() or SubnSet()* be performed on the specified device attribute.
  - — A device's SMA issues a *SubnGetResp()* SM MAD to the Master SM as a response to a previously received *SubnGet()* or *SubnSet()* request MAD.
  - — A SM issues a SM MAD to another SM requesting that a *SubnGet(SMInfo)* or a *SubnSet(SMInfo)* be executed.

  - A SM issues a *SubnGetResp(SMInfo)* SM MAD to another SM as a response to a previously received *SubnGet(SMInfo)* or *SubnSet(SMInfo)* request MAD.
  - A device's SMA issues a SM *SubnTrap(Notice)* MAD to inform the SM of an event that has occurred within the device.
  - In response to a series of identical *SubnTrap(Notice)* SM MADs from a specific device, the Master SM issues a *SubnTrapRepress(Notice)* SM MAD to the device's SMA as a cease-and-desist order.
- GMP MADs:
  - A GSM issues a request GMP to the respective GSA within a device requesting that a *Get()* or *Set()* be performed on the specified device attribute.
  - A device's GSA issues a *GetResp()* GMP to its respective GSM as a response to a previously received *Get()* or *Set()* request GMP.
  - A device's GSA issues a *Trap(Notice)* GMP to inform its respective GSM of an event that has occurred within the device.
  - In response to a series of identical *Trap(Notice)* GMPs from a specific device, the respective GSM issues a *TrapRepress(Notice)* GMP to the device's GSA as a cease-and-desist order.
  - A GSA within a device issues a *Send()* GMP to transmit an attribute to either a GSA of the same type within another device or to the respective GSM.

Figure 28-1: A MAD Is Exactly 256 Bytes in Length



Software Times Return of MAD Response

When the SM or a GSM issues a request MAD that should cause a response MAD to be returned, the manager software entity rather than the SMI or GSI is responsible for starting a timer upon issuance of the request MAD. If a response is not received before the timer expires, the software-based management entity

is responsible for retrying the request MAD transmission. How many retries are performed and what actions are taken when retries are exhausted is outside the scope of the specification.

The software management entity derives the timeout from the following:

- Each port on a CA or a router and the management port on a switch implements the following two attribute elements:
  - — *PortInfo.SubnetTimeout* attribute element. This value is programmed by the SM and indicates the maximum time for a response from this port to reach any other port in the subnet. The maximum time is calculated as $4.096us \times 2^{SubnetTimeout}$
  - — *PortInfo.RespTimeValue* attribute element. This hard-wired, read-only value indicates the amount of time from a port's receipt of a request MAD until the corresponding response is sent, or between the port's transmission of successive MADs that are part of a multi-MAD sequence [e.g., in response to the receipt of a *SubnAdmGetBulk()* request MAD]. The time is calculated as $4.096us \times 2^{RespTimeValue}$.
- The maximum expected interval between the transmission of a request MAD and the receipt of the corresponding response MAD = $(2 \times (4.096us \times 2^{SubnetTimeout})) + (4.096us \times 2^{RespTimeValue})$
- The maximum expected interval between the reception of the successive response MADs of a multi-MAD response sequence = $(4.096us \times 2^{SubnetTimeout}) + (4.096us \times 2^{RespTimeValue})$

Basic MAD Contents

Table 28-1 on page 783 illustrates the basic format of a MAD. The first six dwords (24 bytes) are referred to as the base-MAD header and is the same for all MADs. The fields of the base MAD header are defined in Table 28-2 on page 783.

Table 28-1: Base MAD Format

Dword		Byte 0	Byte 1	Byte 2	Byte 3	
0	Base MAD Header	Base Version = 01h	Management Class (see Table 28-3 on page 785)	Class Version = 01h	R	Method (see Table 28-4 on page 786)
1		Status (see Table 27-2 on page 776)		Only used in directed-route SMP (see Table 28-6 on page 789)		
2		Transaction ID				
3						
4		AttributeID		Reserved		
5		AttributeModifier				
6–63		Data field. Contents is defined by the type of MAD.				

Table 28-2: Definition of Base MAD Header Fields

Field	Length (in bits)	Description
Base Version	8	Version of base MAD format. Must be one (01h).
Management Class	8	Indicates the management class of the message. Possible values are listed in Table 28-3 on page 785.
Class Version	8	Version of MAD class-specific format. Must be 01h (except for Vendor class which may be 01h or greater).
R	1	Response bit. It's only set in the response packet returned in response to a request packet.
Method	7	Indicates the action to be performed on the indicated attribute. The Methods that are common to most (but not necessarily all) classes are listed in Table 28-4 on page 786.

29 SM Methods and Attributes

The Previous Chapter

The previous chapter provided a detailed description of the various types of management datagrams (MADs) and included:

- The definition of a MAD.
- Software times the return of a MAD's response.
- Basic MAD contents.
- SMP MADs.
- GMP MADs.
- Traps.
- Event subscription and event forwarding.
- The Notice queue.

This Chapter

This chapter provides a detailed description of:

- SM MAD formats.
- SM methods.
- SM attributes.
- SM traps.
- SMA Notice support.

The Next Chapter

The next chapter describes all of the issues related to multiple SMs in the same subnet. Specifically, it covers:

- Issues related to this subject that are outside the scope of the specification.
- Multi-vendor SM failover is not supported.
- A subnet can have more than one SM.

- How the SM issues requests (using SMPs).
- An introduction to the SM states.
- Standby SMs monitor the Master SM's heartbeat.
- The purpose of SM Control Packets and how they are issued.
- When there are multiple Master SMs.
- A detailed description of the SM states.

SM MAD Formats

A SM SMP MAD has one of two possible formats:

- LID-routed SMP MAD format. See "LID-Routed SMPs" on page 787.
- Directed-route SMP MAD format. See "Directed-Route SMPs" on page 788.

SM Methods

Table 27-1 on page 774 defines all of the methods utilized by both the SM and the SMA within a device.

SM Attributes

General

Table 29-1 on this page defines all of the device attributes that are managed/accessed by the SM through a device's SMA.

Table 29-1: SM Attributes

Name	Attribute ID	Modifier	Description and Implementation Requirement
Notice	0002h	00000000h	• Information regarding the associated event. See Table 28-9 on page 798 for a description of its content. • **Requirement**: SM must support either: – Receipt of Trap,s or – *SubnGet(Notice)* & *SubnSet(Notice).*

Table 29-1: SM Attributes (Continued)

Name	Attribute ID	Modifier	Description and Implementation Requirement
NodeDescription	0010h	00000000h	• 512 bits. Read-only. UTF-8 encoded text string describing device. • **Requirement**: All ports on a CA or router must implement and must return same device description. Switch port 0 also must support it.
NodeInfo	0011h	00000000h	• Read-only. General information about the device. See "NodeInfo Attribute" on page 816 for a description of its content. • **Requirement**: Must be implemented by all ports on a CA or router and on switch port 0. Also see Table 29-3 on page 816.
SwitchInfo	0012h	00000000h	• Information about the Switch. See "SwitchInfo Attribute" on page 818 for a description of its content. • **Requirement**: Only required on switch port 0.
GUIDInfo	0014h	Table block selector	• GUID assignment table. See "Port's GUID(s) Assignment" on page 156 and "Multiple GUID Assignment Permits Router Multipathing" on page 157 for usage and "GUIDInfo Attribute" on page 823 for programming. • **Requirement**: All CA ports must implement it.
VLArbitration	0018h	High/Low table half selector. For a switch, the port selector.	• SM programs this table to prioritize a port's data VL transmit buffers. See "Detailed Description of VL Arbitration" on page 628 for usage and "VLArbitration Attribute" on page 841 for programming. • **Requirement**: Must be implemented on all ports of CAs, routers, and switches that implement more than one data VL.

Table 29-1: SM Attributes (Continued)

Name	Attribute ID	Modifier	Description and Implementation Requirement
PortInfo	0015h	Port selector (see Description)	• Information about the port. See "PortInfo Attribute" on page 824 for a description of its content. • **Requirement**: All ports on CAs, routers, and switches must support it. • Modifier usage for CAs and routers: – 0 = operation is performed on the port that received the SMP. – Otherwise, if ≠ 0 and ≠ port number where the SMP is received, the *PortInfo* attribute is read-only and the M_Key is checked for both the port where the SMP was received and the port selected by the Modifier. • Modifier usage for for switches: •0 selects the management port. •Otherwise, if ≠ 0, a physical port is selected.
P_KeyTable	0016h	Table block selector and, for a switch, port selector.	• Holds the partition keys associated with a port. See "Partition Key (P_Key)" on page 319 for a description of its usage and "P_KeyTable Attribute" on page 837 for programming and inbound and outbound P_Key checking on switches and routers. • **Requirement**: All ports on a CA or router and Switch port 0 must support it. In addition to switch port 0, a switch may optionally implement a *P_KeyTable* attribute for each port other than port 0.

Table 29-1: SM Attributes (Continued)

Name	Attribute ID	Modifier	Description and Implementation Requirement
SLtoVLMapping-Table	0017h	• NA for CA or router. • For switch usage, see description.	• Service Level to Virtual Lane mapping table. See "QoS within the Subnet: SL and VLs" on page 617 for a description of its usage and "SLtoVLMapping Attribute" on page 840 for programming. • **Requirement**: Must be implemented on all ports of CAs, routers, and switches that implement more than one data VL. • For a switch, Modifier selects the input port/output port combination this table applies to: – Modifier bits [15:8] specify input port (1 to n), where n selects port or 0 indicates input port is the management port. – Modifier bits [7:0] specify output port (1 to n), where n selects n port. – Modifier bits [31:16] must be zero.
LinearForwardingTable	0019h	LID block selector	• Switch implements the *LinearForwardingTable* or *RandomForwardingTable* attribute (one or the other, not both) to handle forwarding of packets with unicast DLID addresses. See "Linear Forwarding Table (LFT)" on page 669 for usage and programming. • **Requirement**: Described in the previous paragraph.

30 Multiple SMs

The Previous Chapter

The previous chapter provided a detailed description of:

- SM MAD formats.
- SM methods.
- SM attributes.
- SM traps.
- SMA Notice support.

This Chapter

This chapter describes all of the issues related to multiple SMs in the same subnet. Specifically, it covers:

- Issues that are outside the scope of the specification.
- Multi-vendor SM failover is not supported.
- A subnet can have more than one SM.
- How the SM issues requests (using SMPs).
- An introduction to the SM states.
- Standby SMs monitor the Master SM's heartbeat.
- The purpose of SM Control Packets and how they are issued.
- When there are multiple Master SMs.
- A detailed description of the SM states.

The Next Chapter

The next chapter provides a detailed description of how the SM discovers devices during startup, as well as how it issues attribute accesses to devices that have not yet been assigned LID addresses. It covers:

- Who performs discovery?
- How packets are normally routed (when all devices have already been assigned LIDs).
- Packet routing during discovery (before LIDs have been assigned).
- Accessing device attributes when the path to the device is only partially configured.

Many Issues Are Outside the Scope of the Specification

The following issues related to the discovery process and to the SM are outside the scope of the specification:

- The discovery algorithm.
- How routes are added/deleted.
- The procedure for handling failover from the Master SM to a Standby SM.
- SM priority assignment.
- SM_Key assignment and usage (more later).
- SM-to-OS (or whatever) interface.
- The interval at which the Standby SMs poll the Master SM.

Multi-Vendor SM Failover Is Not Supported

When the author discovered the following statement in the specification, it answered a lot of open questions. Although any SM from any vendor must be able to manage devices from multiple vendors, there is no requirement for interoperability among SMs from different vendors. This explains why the specification says the manner in which the *SMInfo.SM_Key* and the *SMInfo.Priority* (both of which are read-only—see "SMInfo Attribute" on page 842) are assigned is outside the scope of the specification.

Specification statement:

> "The management operations specified herein provide for a level of interoperability such that an SM from any vendor can manage a heterogeneous collection of IBA-compliant channel adapters, switches, and routers from any set of vendors. However, compatibility and interoperability among SMs from different vendors is not supported. Migration from one vendor's SM to another's by way of system reinitialization, i.e., through a planned outage, is supported. Such migration assumes appropriate steps of transferring data between vendors' SMs have been accomplished prior to the reinitialization."

A Subnet Can Have More Than One SM

A subnet may contain more than one SM, but only one SM is permitted to manage the devices in the subnet. This SM is referred to as the Master SM. Any other SMs are in one of two possible states:

- **Standby SM**. Standby SMs do not manage the subnet. Rather, they periodically poll the Master SM—via a *SubnGet(SMInfo)*—to obtain the *ActCount* element to determine if it is still actively managing the subnet or has failed.
- **Inactive SM**. An Inactive SM does not poll the Master SM's ActCount. However, it will respond to a command (referred to as a Control Packet) that causes it to change from the Inactive state to another state (e.g., the Standby state).

How SM Issues Requests

As described in "The SMI" on page 761, each port on a CA or router, as well as the switch management port has an SMI tasked with sending and receiving SMPs. The SMI is always implemented as QP0. As described earlier (see "Managers Use Special Packets Called MADs" on page 27, "Detailed Description of MADs" on page 779, and "SM Methods and Attributes" on page 809), an SMP is a form of MAD and always has a data payload field exactly 256 bytes in length (see Figure 28-1 on page 781). "How the SM Sends a Message and Handles a Response" on page 772 described the basic process the SM uses to construct and send an SMP.

Introduction to the SM States

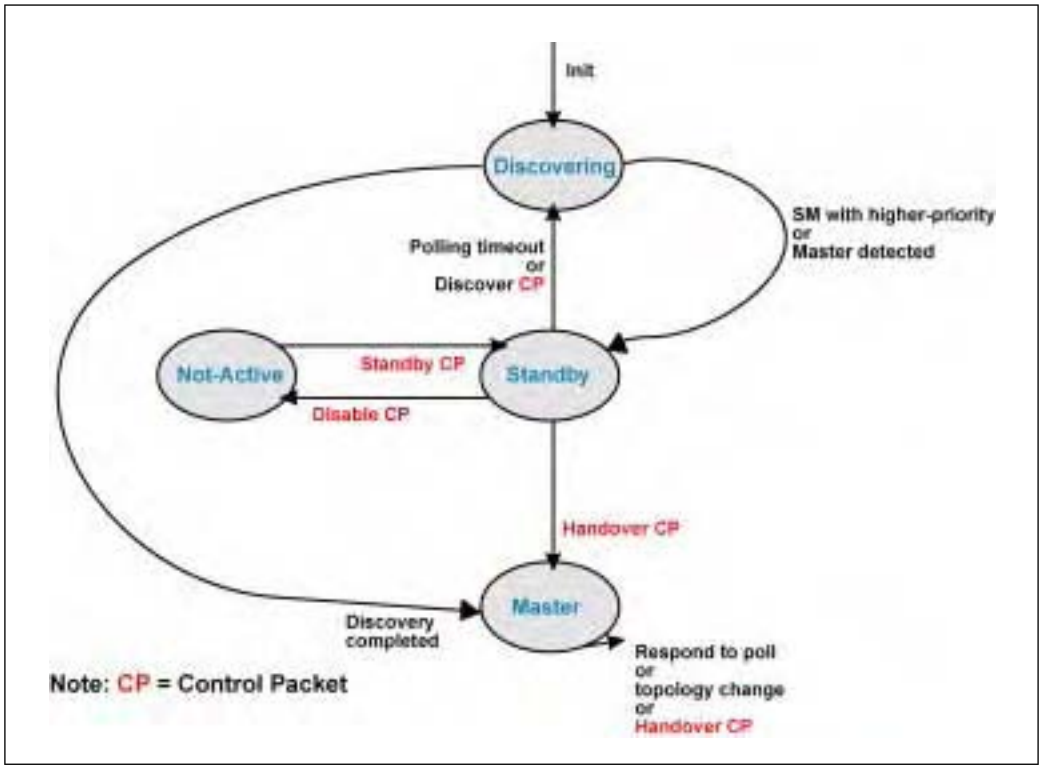
Refer to Figure 30-1 on page 854. A SM can be in one of four possible states:

- **Master state**. This is the state an SM is in when it is actively managing the devices in the subnet. Only one SM may be in the Master state during a given period of time.
- **Standby state**. While in the Standby state, an SM periodically polls the Master SM to determine if it is still actively managing the subnet or has failed. More than one SM in a subnet can be in the Standby state.
- **Discovering state**. This is the startup state for all SMs. While in this state, an SM is actively probing the subnet (described in "Discovery" on page 871) to determine the type of devices that populate the subnet and whether or not there are additional SMs within the subnet.
- **Not-Active state**. A SM currently in the Not-Active state is passive. It does not initiate the sending of any SMPs. Rather, it quietly awaits the receipt of a *SubnGet(SMInfo)* or *SubnSet(SMInfo)* from the Master SM and responds accordingly (more later).

A SM may receive an SMP from another SM regarding an SM state change. This type of SMP is referred to as a Control Packet (CP). The current state of a SM can

be read or changed by accessing the *SMState* element of its *SMInfo* attribute. The remainder of this chapter provides a detailed description of the states and Control Packets.

Figure 30-1: SM State Diagram



Are You Alive?

The Master SM maintains an activity counter in the form of the *SMInfo.ActCount* attribute element.

- *ActCount* increments each time the Master SM issues a request packet or performs any "other" management activities (the specification is vague). This counter is the Master SM's heartbeat.
- Standby SMs periodically issue *SubnGet(SMInfo)* requests to see if the Master SM is still alive (i.e., has the counter incremented since the standby SM last read it).

- The polling interval is set by the OS and is outside the scope of the specification.
- If the Master SM appears dead (i.e., no response is received, or the response indicates that the *ActCount* is not changing), the standby SM transitions back to the Discovering state (more later).

SM Control Packets

Definition

An SM uses Control Packets to command another SM to change state (or to acknowledge receipt of a state change command).

How SM Sends a Control Packet

A Control Packet consists of an SMP specifying the following:

- *SubnSet(SMInfo)* operation.
- The attribute modifier (in the base MAD header—see Table 28-1 on page 783) is set to one of the values indicated in Table 30-1 on this page.

Table 30-1: SubnSet(SMInfo) Attribute Modifier

Modifier Value	Description
1	Identifies this as a **Handover Control Packet**. It is issued by the current Master SM to initiate the process of handing over mastership to a higher-priority Standby or Master SM that was discovered during its periodic sweep or because a trap was received informing the Master of a topology change.
2	Identifies this as an **Acknowledge Control Packet**. It is issued by the new Master SM to acknowledge the handover from the old Master SM.

31 Discovery

The Previous Chapter

The previous chapter described all of the issues related to multiple SMs in the same subnet. Specifically, it covered:

- Issues that are outside the scope of the specification.
- Multi-vendor SM failover is not supported.
- A subnet can have more than one SM.
- How the SM issues requests (using SMPs).
- An introduction to the SM states.
- Standby SMs monitor the Master SM's heartbeat.
- The purpose of SM Control Packets and how they are issued.
- When there are multiple Master SMs.
- A detailed description of the SM states.

This Chapter

This chapter provides a detailed description of how the SM discovers devices during startup, as well as how it issues attribute accesses to devices that have not yet been assigned LID addresses. It covers:

- Who performs discovery?
- How packets are normally routed (when all devices have already been assigned LIDs).
- Packet routing during discovery (before LIDs have been assigned).
- Accessing device attributes when the path to the device is only partially configured.

The Next Chapter

The next chapter provides a detailed description of the General Services Interface (QP1). It is provided at this point in the book because the SA is accessed via the GSI and the next chapter covers the SA. Specifically, this chapter covers the following:

- The GSMs, GSAs, and GSIs.
- How to prepare an HCA GSI for use.
- The required versus optional GSAs.
- The SA and the GSI.
- GMPs use a VL buffer determined by the GMP's SL value.
- GMPs can transit routers.
- QP1 is a controlled-access QP.
- QP1's P_Key insertion and checking.
- Additional reference material.

Who Performs Discovery?

The SM uses directed-route SMPs to discover the devices that populate a subnet.

How Packets Are Normally Routed

Table 31-1 on page 873 illustrates the content of a LID-routed SMP's data payload. After LIDs have been assigned to all ports on CAs and routers and to port 0 on each switch, the port that originates an SMP packet specifies:

- the LID of the destination port in the packet's DLID field and
- its own LID in the packet's SLID field (in case a response is required).

Switches then forward the packet from source to destination by consulting their forwarding tables (which the Master SM set up after it had completed discovery).

This is referred to as a LID-routed SMP.

Table 31-1: LID-Routed SMP MAD Format

Dword		Byte 0	Byte 1	Byte 2	Byte 3	
0	Base MAD Header	Base Version = 01h	Management Class = **Subn**	Class Version = 01h	R	**SM Method** (see Table 27-1 on page 774)
1		Status is only meaningful in response packets. See Table 27-2 on page 776.		Not used		
2		Transaction ID				
3						
4		SM AttributeID		Reserved		
5		AttributeModifier				
6	SMP-specific fields	M_Key				
7						
8–15		32 bytes Reserved				
16–31		64 bytes of SMP Data—Attribute content on a *SubnGetResp()* or a *Subn-Trap(Notice)*				
32–63		64 bytes Reserved				

Packet Routing During Discovery

Problem

Before all ports have been assigned LID addresses and switch Forwarding Tables have been initialized by the Master SM, the SM must use a different method to direct a packet to its destination. It uses a directed-route SMP. There are two basic scenarios:

1. **Scenario**: The SM must be able to deliver a MAD containing an attribute access request to a CA or router port or to a switch's port 0 without knowing its LID address.

2. **Scenario**: While the SM may know the destination port's LID address, the switches along the route may not have all been configured with LID addresses and Forwarding Tables.

Four Possible Scenarios

There are four possible scenarios:

1. The entire path to the target port is unaddressable:
   - A LID address may or may not have been assigned to the port the SM is using to originate the SMP.
   - LID addresses have not been assigned to the management ports of the switches in the path between the source and destination ports.
   - The Master SM has not yet built the Forwarding Table in each switch between the source and destination ports.
   - The destination port has not yet been assigned a LID address.
2. The latter half of the path to the destination port is unaddressable:
   - A LID address has been assigned to the port the SM is using to originate the SMP.
   - A LID address has been assigned to the management port of each switch along the first half of the path to the destination port.
   - LID addresses have not been assigned to the management port of each switch along the latter half of the path to the destination port.
   - The Master SM has built the Forwarding Tables in the switches along the first part of the path to the destination port.
   - The Master SM has not yet built the Forwarding Tables in the switches along the latter part of the path to the destination port.
   - The Master SM has not yet assigned a LID address to the destination port.
3. The initial-half of the path to the target port is unaddressable:
   - A LID address may or may not have been assigned to the port the SM is using to originate the SMP.
   - A LID address has not been assigned to the management port of each switch along the first half of the path to the destination port.
   - LID addresses have been assigned to the management port of each switch along the latter half of the path to the destination port.
   - The Master SM has not yet built the Forwarding Tables in the switches along the first part of the path to the destination port.
   - The Master SM has built the Forwarding Tables in the switches along the latter part of the path to the destination port.
   - The Master has assigned a LID address to the destination port.

4. The middle portion of the path to the target port is unaddressable:
   — A LID address has been assigned to the port the SM is using to originate the SMP.
   — A LID address has been assigned to the management port of each switch along the first part of the path to the destination port.
   — LID addresses have not been assigned to the management port of each switch along the middle part of the path to the destination port.
   — LID addresses have been assigned to the management port of each switch along the final part of the path to the destination port.
   — The Master SM has built the Forwarding Tables in the switches along the first part of the path to the destination port.
   — The Master SM has not yet built the Forwarding Tables in the switches along the middle part of the path to the destination port.
   — The Master SM has built the Forwarding Tables in the switches along the final part of the path to the destination port.
   — The Master SM has assigned a LID address to the destination port.

Solution: Directed-Route SMP

Refer to Table 31-2 on page 877. Upon receipt of an SMP with the DLID field set to FFFFh (the Permissive LID, or PLID), the CA or router port at the other end of the link delivers the packet to its SMI for processing. If the device at the other end of the link is a switch, the PLID address causes the switch port to internally deliver the SMP to the SMI of port 0 for processing. In addition to the PLID in its DLID field, the directed-route SMP originated by the SM contains the following:

- The **SLID** field is set to the LID of the port that originates the packet (or the PLID if a LID address has not yet been assigned to it).
- The **DLID** field is set to one of the following:
  — LID address of the switch management port on the switch at the end of the initial part of the path to the destination port. The Forwarding Tables in one or more switches in the initial part of the path have already been set up by the Master SM. Those switches can therefore use their Forwarding Tables to get the packet to the switch management port at the end of the initial part of the path to the destination port.
  — If the switch Forwarding Tables and the management port's LID in the switches on the first part of the path have not yet been set up, the packet's DLID is set to the PLID address.
- The **DrSLID** (directed-route SLID) field is set to the address of the port that the SM uses to transmit the SMP request packet:

32 The GSI

The Previous Chapter

The previous chapter provided a detailed description of how the SM discovers devices during startup, as well as how it issues attribute accesses to devices that have not yet been assigned LID addresses. It covered:

- Who performs discovery?
- How packets are normally routed (when all devices have already been assigned LIDs).
- Packet routing during discovery (before LIDs have been assigned).
- Accessing device attributes when the path to the device is only partially configured.

This Chapter

This chapter provides a detailed description of the General Services Interface (GSI, aka QP1). It is provided at this point in the book because the SA is accessed via the GSI and the next chapter covers the SA. Specifically, this chapter covers:

- The GSMs, GSAs, and GSIs.
- How to prepare an HCA GSI for use.
- The required versus optional GSAs.
- The SA and the GSI.
- GMPs use a VL buffer determined by the GMP's SL value.
- GMPs can transit routers.
- QP1 is a controlled-access QP.
- QP1's P_Key insertion and checking.
- Additional reference material.

The Next Chapter

The next chapter provides a detailed description of the Subnet Administrator (SA). Specifically, it covers:

- The purpose of the SA.
- The SA is accessed using GMPs.
- The location of the SA.
- Requester access authorization.
- The SA's methods and attributes.
- The definition of a Record Identifier (RID).
- *SubnAdmGet()* operation.
- *SubnAdmSet()* operation.
- The definition of a table.
- *SubnAdmConfig()* operation.
- Database queries using *SubnAdmGetTable()*.
- Fetching the entire database.
- The reliable multi-packet transaction protocol.
- Additional reference information.

The GSMs, GSAs, and GSIs

The General Services Managers (GSMs) were introduced in "Intro to the Managers" on page 159. Refer to Figure 32-1 on page 911. The entities involved in the general services are:

- **GSM**. A series of GSMs have responsibility for managing various aspects of device operation. A GSM may only have responsibility for managing the devices in a single subnet, or may have responsibility for managing the devices residing in multiple subnets.
- **GSAs**. The entity that a specific type of GSM communicates within a device is referred to as its respective General Services Agent (**GSA**). Each device implements a series of GSAs, each of them having responsibility for handling incoming requests issued by their respective GSM.
- **GMPs**. The GSM communicates with its respective GSA by issuing General Services Management Packets (GMPs). Like SMPs, GMPs are a form of MAD (see "GMP MADs" on page 789).
- **GSI**. The GSM issues GMPs using a special QP (QP1) referred to as the General Services Interface (GSI). Likewise, the GMPs are received by the GSI on a port of the destination device. The GSI uses the GMP MAD's Management Class field (see Table 28-1 on page 783) in determining to which of the device's GSAs the GMP must be delivered for processing (see Figure 32-2 on page 911).

- If a device implements both a GSM and the GSA associated with that GSM (as shown in the left-hand device in Figure 32-1 on this page), the GSM may send and receive GMPs with GSAs in remote devices through its local GSA, or it may communicate directly with the local GSI to do so. This is implementation-specific.

Figure 32-1: Relationship of GSMs, GSAs, and GSIs

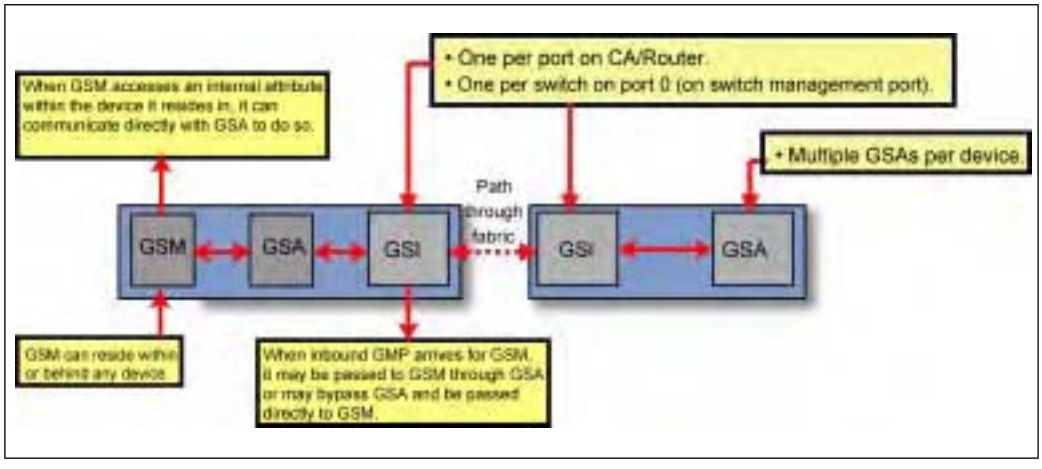
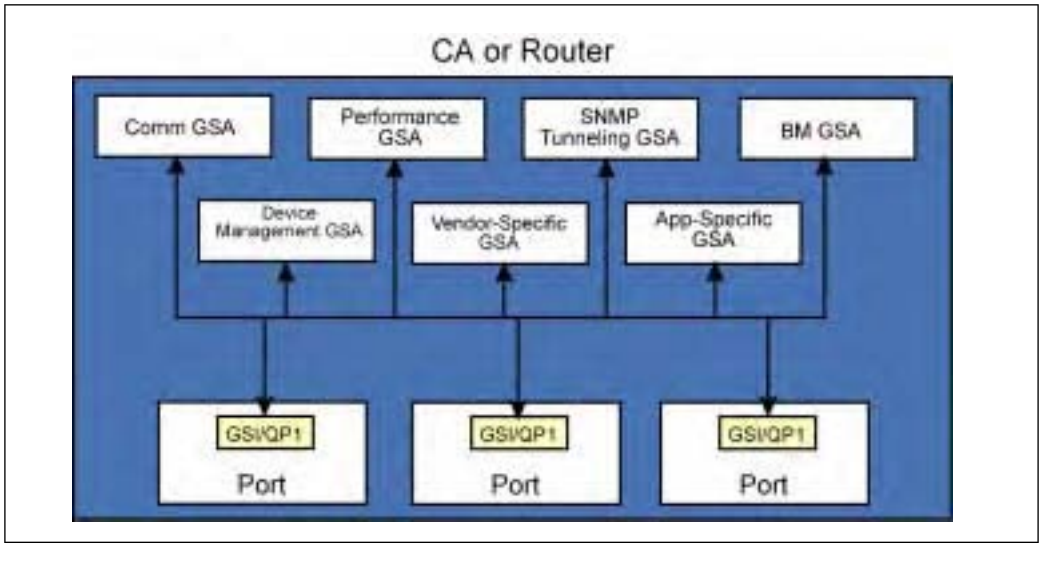


Figure 32-2: GSI Distributes Incoming GMPs to Destination GSA



QP1 Is the GSI

The GSI is implemented as a special UD QP. Unlike other QPs that are created and destroyed on an as-needed basis, these QPs are permanently implemented in the following manner:

- A separate QP1 is implemented on each port of a CA or a router.
- A QP1 is implemented only on the management port (port 0) of a switch.

Preparing an HCA GSI for Use

When a privileged application (i.e., a GSM) wishes to use a port's GSI to transmit and receive GMPs, software activates the port's GSI (i.e., its QP1) by first creating the CQ(s) to be associated with the QP's SQ and RQ (using the *Create CQ* verb). A GSI QP cannot share a CQ with any QP other than another GSI or an SMI. An attempt to do so results in the return of an immediate Invalid CQ Handle error.

After creation of the required CQ(s), software then executes the *Get Special QP* verb with the following input parameters:

- **HCA Handle**. Identifies the HCA on which the QP is to be activated.
- **HCA port number**. Identifies the HCA port whose GSI is being activated.
- **QP type requested**. The allowed types are:
  - SMI QP (QP0).
  - GSI QP (QP1). This is the type indicated in this case.
  - Raw IPv6.
  - Raw Ethertype.
- **Protection Domain**. Identifies the PD that the QP is to be a member of.
- The handle(s) of the **CQ(s)** to be associated with the GSI's SQ and RQ.
- **Requested SQ and RQ size**. The maximum number of outstanding WRs software expects to post to the GSI's SQ and RQ.
- **Requested SQ Gather Buffer List size**. The maximum number of Gather Buffer List elements software expects to specify in any WR posted to the GSI's SQ.
- **Requested RQ Scatter Buffer List size**. The maximum number of Scatter Buffer List elements software expects to specify in any WR posted to the GSI's RQ.
- The **Signaling Type** for the GSI's SQ. The valid types are:
  - All WRs posted to the SQ always generate a CQE.
  - In each WR posted to the SQ, software must specify whether to generate a CQE for a successful transfer or only if there is an error.

The output parameters returned to the caller are:

- **QP handle**. This handle must used when making any subsequent *Query QP*, *Modify QP*, or *Destroy QP* verb calls.
- The **actual size of the SQ**. If an error is not returned, this is guaranteed to be ≥ to the number requested (note that this may require software to increase the size of the SQ's CQ).
- The **actual size of the RQ**. If an error is not returned, this is guaranteed to be greater than or equal to the number requested (note that this may require software to increase the size of the RQ's CQ).
- The **actual maximum** number of **Gather Buffer List elements** that can be specified in WRs posted to the SQ. If an error is not returned, this is guaranteed to be ≥ the number requested.
- The **actual maximum** number of **Scatter Buffer List elements** that can be specified in WRs posted to the RQ. If an error is not returned, this is guaranteed to be ≥ the number requested.

Required/Optional GSAs

Refer to Table 9-1 on page 170.

The SA and the GSI

The GSI, rather than the SMI, is used to communicate with the SA and GMPs used to send and receive information between another device and the one on which the SA resides. For more information on communications with the SA, refer to "Detailed Description of SA" on page 917.

GMPs Use VL Buffer Determined by the GMP's SL Value

Unlike SMPs that are always transmitted using a port's VL15 transmit buffer and are always received into a port's VL15 receive buffer, the VL buffer utilized by a GMP is determined by the SL value specified in the GMP's LRH:SL field.

GMPs Can Transit Routers

The SM is only permitted to manage the devices in its own subnet. For this reason, SMPs never contain a GRH (only an LRH), and they are never permitted to cross a router into another subnet.

33 Detailed Description of SA

The Previous Chapter

The previous chapter provided a detailed description of the General Services Interface (QP1). It was provided at that point in the book because the SA is accessed via the GSI and this chapter covers the SA. Specifically, the previous chapter covered:

- The GSMs, GSAs, and GSIs.
- How to prepare an HCA GSI for use.
- The required versus optional GSAs.
- The SA and the GSI.
- GMPs use a VL buffer determined by the GMP's SL value.
- GMPs can transit routers.
- QP1 is a controlled-access QP.
- QP1's P_Key insertion and checking.
- Additional reference material.

This Chapter

This chapter provides a detailed description of the Subnet Administrator (SA). Specifically, it covers:

- The purpose of the SA.
- The SA is accessed using GMPs.
- The location of the SA.
- Requester access authorization.
- The SA's methods and attributes.
- The definition of a Record Identifier (RID).
- *SubnAdmGet()* operation.

- *SubnAdmSet()* operation.
- The definition of a table.
- *SubnAdmConfig()* operation.
- Database queries using *SubnAdmGetTable()*.
- Fetching the entire database.
- The reliable multi-packet transaction protocol.
- Additional reference information.

The Next Chapter

The next chapter provides an overview of Baseboard Management (BM) and a detailed description of the BM MADs exchanged between the BM and the Baseboard Management Agents (BMAs) within IBA devices. It does not provide a detailed description of all issues related to BM (e.g., hot-install, hot-removal, etc.) nor does it provide a detailed description of the IB-ML bus. These subjects may be covered in a future edition of this book. Specifically, this chapter covers:

- The roles of the other managers.
- The BM reaches behind the IBA front-end.
- A basic definition of a *chassis* and a *module*.
- The chassis Baseboard Management elements.
- Passively managed chassis.
- Module BM elements.
- Non-module IBA devices.
- BM MAD format.
- BM methods.
- BM attributes.
- The BM sending a command to the MME.
- The CME sending a command to the BM.
- BM-related traps.

Purpose of the SA

The SA is the subnet database built by the Master SM. As such, it contains records consisting of:

- The attributes that are read and managed by the Master SM. Table 29-1 on page 810 lists the SM attributes.
- Information delivered to the Master SM in *SubnTrap(Notice)* SMPs regarding events detected by devices within the subnet.
- Some SA database items recorded by administrative software updating the subnet configuration data. Some of these items may have been entered by a

customer (e.g., partition records recording customer's partitioning configuration input).

The SA may be accessed by other SMs (i.e., Standby SMs) to update their own SA databases. In addition, other management software may issue database queries to the SA, or may add, delete, or edit records within the SA database.

Other entities may subscribe to the SA to be notified of events that occur on one or more devices within the subnet.

SA Accessed Using GMPs

Other entities access the SA database using GMPs (refer to Table 33-1 on this page) with the Management Class field set to SubnAdm, and using one of the SA methods (see Table 33-3 on page 925) to act upon one or more SA attributes (see Table 33-4 on page 927). Table 33-2 on page 920 provides a description of each of the MAD fields in an SA GMP.

Table 33-1: SA MAD Format

<table>
<tr><th>Dword</th><th></th><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th colspan="2">Byte 3</th></tr>
<tr><td>0</td><td rowspan="6">Base MAD Header</td><td>Base Version = 01h</td><td>Management Class = SubnAdm</td><td>Class Version = 01h</td><td>R</td><td>SA Method (see Table 33-3 on page 925)</td></tr>
<tr><td>1</td><td colspan="2">Status is only meaningful in response packets. See Table 27-2 on page 776.</td><td colspan="3">Not used</td></tr>
<tr><td>2</td><td colspan="5" rowspan="2">Transaction ID</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td><td colspan="2">SA AttributeID</td><td colspan="3">Reserved</td></tr>
<tr><td>5</td><td colspan="5">AttributeModifier</td></tr>
</table>

Table 33-1: SA MAD Format (Continued)

Dword		Byte 0	Byte 1	Byte 2	Byte 3
6	SA-specific fields	SA_Key			
7					
8		SM_Key			
9					
10		Segment Number			
11		Payload Length			
12		Fragment Flag	Edit Modifier	Window	
13		End RID			
14		Component Mask			
15					
16–63		192-byte Admin Data Area			

Table 33-2: SA MAD Field Descriptions—base MAD fields are described in Table 28-2 on page 783

Field	Description
SA_Key	Refer to "Request Only Table Changes" on page 947.
SM_Key	Refer to "Subnet Manager Key (SM_Key)" on page 330.
Segment Number	• = 0 in a single-packet request or response, = 0. • In multi-packet request or response: – = 1 in 1st packet. – Subsequent packet segment numbers are incremented (except in KeepAlive packets).

Table 33-2: SA MAD Field Descriptions—base MAD fields are described in Table 28-2 on page 783 (Continued)

Field	Description
Payload Length	Payload Length is reserved unless: • First packet of a multi-packet *SubnAdmConfig* request. Length = sum of bytes in the Admin Data fields of all of the packets sent as part of the request. • First and last packet of multi-packet *SubnAdmGetBulk* or *SubnAdmGetTable* response. – In 1st packet of response, length = expected sum of all bytes in Admin Data fields of all response packets. – In last packet of response, length = number of valid bytes in last packet's Admin Data field. – After the first response is sent, the actual payload length may change (due to SM changes to the database). The payload length in the last response packet indicates the number of valid bytes in that packet's Admin Data field, and the Fragment Flag's Last Packet bit is set to one.
Window	Valid only in a multi-packet *SubnAdmGetBulk* or *SubnAdmGetTable* request, and in the Ack packet of a *SubnAdmConfig*. • For a *SubnAdmGetBulk* or *SubnAdmGetTable* request that results in a multi-packet response, this specifies the number of packets the responder may send before it pauses and awaits an Ack packet. When the Ack packet is received, an additional window's worth of packets may be sent. • For a multi-packet *SubnAdmConfig* request: – The implied initial window value is 64d. The requester pauses after sending 64 request packets to await an Ack from the responder. – The responder includes a new window value in each Ack packet. This tells the requester the size of the next request packet window. – If the window value in any Ack packet < 64, the default window value of 64 is used by the requester.

34 Baseboard Management

The Previous Chapter

The previous chapter provided a detailed description of the Subnet Administrator (SA). Specifically, it covered:

- The purpose of the SA.
- The location of the SA.
- Requester access authorization.
- The SA's methods and attributes.
- The definition of a Record Identifier (RID).
- *SubnAdmGet()* operation.
- *SubnAdmSet()* operation.
- The definition of a table.
- *SubnAdmConfig()* operation.
- Database queries using *SubnAdmGetTable()*.
- Fetching the entire database.
- The reliable multi-packet transaction protocol.
- Additional reference information.

This Chapter

This chapter provides an overview of Baseboard Management (BM) and a detailed description of the BM MADs exchanged between the BM and the Baseboard Management Agents (BMAs) within IBA devices. It does not, however, provide a detailed description of all issues related to BM (e.g., hot-install, hot-removal, etc.), nor does it provide a detailed description of the IB-ML bus. These subjects may be covered in a future edition of this book. Specifically, this chapter covers:

- The roles of the other managers.
- The BM reaches behind the IBA front-end.
- A basic definition of a *Chassis* and a *Module*.

- The chassis baseboard management elements.
- Passively managed chassis.
- Module BM elements.
- Non-module IBA devices.
- BM MAD format.
- BM methods.
- BM attributes.
- The BM sending a command to the MME.
- The CME sending a command to the BM.
- BM-related Traps.

The Next Chapter

The next chapter provides a detailed description of the Performance Manager (PM), the Performance Management Agent (PMA), and the PM-related methods and attributes. Specifically, it covers:

- The role of Performance Management (PM).
- The required features.
- The optional features.
- PM MAD format.
- PM methods.
- Mandatory PM attributes.
- Optional PM attributes.

Roles of the Other Managers

All of the other managers are responsible for some aspect of an IBA device's IBA-related operation. In other words, utilizing a set of IBA-defined attributes, they control some aspect of the device's interface to the IBA fabric.

They do not have the ability to manage the operation of device-specific logic that resides behind the CA interface.

The BM Reaches behind the IBA Front-End

The BM, on the other hand, has access to components that reside behind a device's IBA interface (i.e., behind the xCA that interfaces it to the IBA fabric). Example applications would be chassis temperature monitoring and the enablement of graceful hot-plug operations.

Chassis and Module

Refer to Figure 34-1 on page 990. An IBA module is defined as an IBA device that installs in a slot of an IBA chassis. When an IBA module is inserted into an IBA chassis, it connects to the backplane via one or two connectors. Each of these connectors provides the following required signal groups (also see Figure 34-2 on page 991):

- **IBA Link signaling group**. This is an IBA link connection to one of two possible ports on the module.
- The **Bulk Power group**. Powers the module during normal operation. Supplied at 12V and converted on-board the module into the voltages required by the module's electronics.
- The **Auxiliary Power group**. Powers the module management functions (even when Bulk Power is not available). Supplied at 5V and regulated as necessary on-board the module.

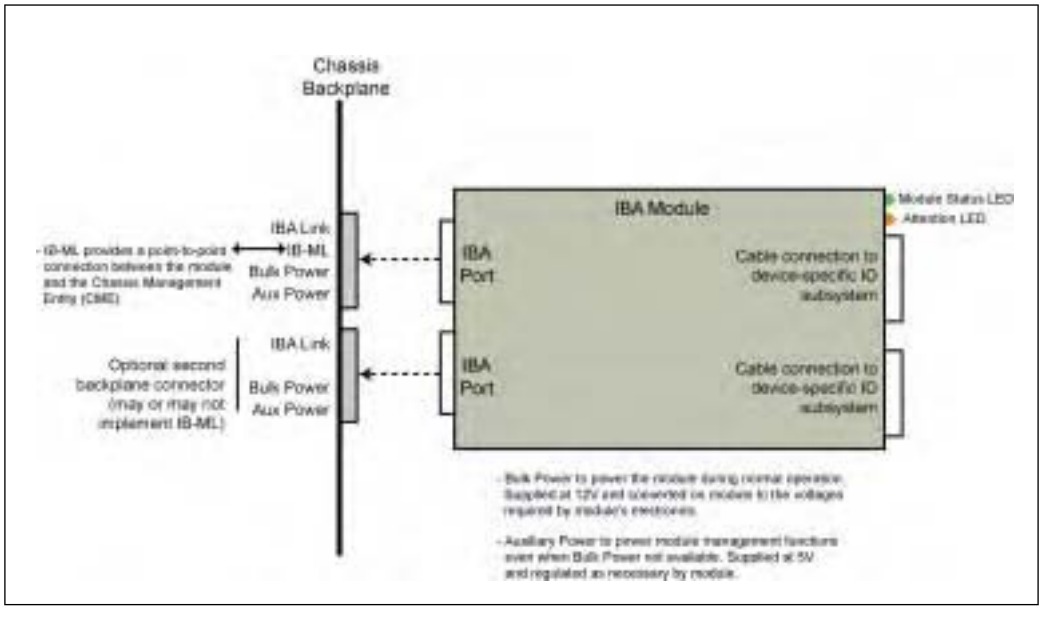
Optionally, the module may also provide the IBA Management Link, or IB-ML, through one of the connectors. IB-ML is a multi-drop, multi-master, two-wire serial bus which uses the SMBus 1.1-based[1] data transfer and arbitration protocols.

Each module has two LEDs:

- Green module Status LED.
- Yellow or amber Attention LED.

1.System Management Bus Specification, Revision 1.1, December 11, 1998; Copyright (c) 1996, 1997, 1998, Benchmarq Microelectronics Inc., Duracell Inc., Energizer Power Systems, Intel Corporation, Linear Technology Corporation, Maxim Integrated Products, Mitsubishi Electric Corporation, National Semiconductor Corporation, Toshiba Battery Co., Varta Batterie AG.

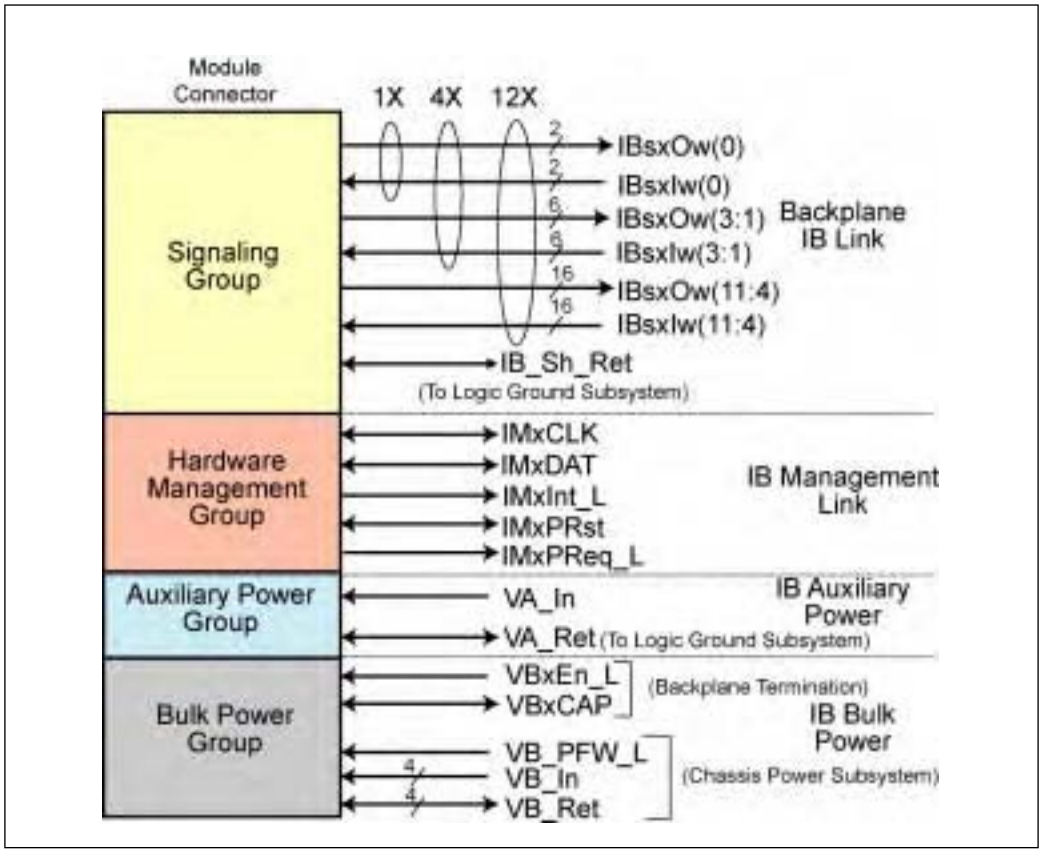
Figure 34-1: IBA Module/Chassis Interconnect



The signal naming conventions used in Figure 34-2 on page 991 are:

- "w" in a signal name is replaced by:
  - "p" for the positive rail of the differential pair.
  - "n" for the negative rail of the differential pair.
- "x" in a signal name is replaced by the port number. Can range from 0 to n-1 where n is the number of connectors/ports on the board edge or the backplane.
- "s" in a signal name:
  - On a board, "s" is replaced by:
    - "t" for an IO plate connector or
    - "b" for a backplane connection.
  - On a backplane, replaced by the slot number.
- "I" = input signal.
- "O" = output signal.

Figure 34-2: Backplane Signal Groups



Chassis Baseboard Management Elements

Figure 34-3 on page 996 illustrates what the specification refers to as an actively managed chassis. It is comprised of the following elements:

- **Embedded switch**. This switch is embedded in the chassis. It is connected to each of the installed IBA modules via one or two IBA links and is connected to the external IBA fabric, in this example, by a single link (there could be more than one external link). The embedded switch incorporates the following entities:

35 Performance Management

The Previous Chapter

The previous chapter provided an overview of Baseboard Management (BM) and a detailed description of the BM MADs exchanged between the BM and the Baseboard Management Agents (BMAs) within IBA devices. It did not provide a detailed description of all issues related to BM (e.g., hot-install, hot-removal, etc.), nor did it provide a detailed description of the IB-ML bus. These subjects may be covered in a future edition of this book. Specifically, this chapter covered:

- The roles of the other managers.
- The BM reaches behind the IBA front-end.
- A basic definition of a *Chassis* and a *Module*.
- The chassis BM elements.
- Passively managed chassis.
- Module BM elements.
- Non-module IBA devices.
- BM MAD format.
- BM methods.
- BM attributes.
- The BM sending a command to the MME.
- The CME sending a command to the BM.
- BM-related traps.

This Chapter

This chapter provides a detailed description of the Performance Manager (PM), the Performance Management Agent (PMA), and the PM-related methods and attributes. Specifically, it covers:

- The role of Performance Management (PM).
- The required features.

- The optional features.
- PM MAD format.
- PM methods.
- Mandatory PM attributes.
- Optional PM attributes.

The Next Chapter

The next chapter provides a detailed description of the Communications Management (CM) MADs, Methods, Attributes, etc. Specifically, it covers:

- CM MAD format.
- CM methods.
- CM attributes.
- CM MADs.
- Definition of *Client* and *Server*.
- Definition of *Active* and *Passive CMs*.
- The three supported models.
- Definition of a stale communications channel.

The Role of Performance Management (PM)

Each IBA device is required to implement a PMA and a minimum set of performance monitoring and error monitoring registers (implemented as PM class attributes). In addition, the specification also defines a set of optional attributes (i.e., registers) permitting the monitoring of additional performance and error counters.

The PM is tasked with retrieving performance and error-related information from these registers. It does this by issuing PM GMP request MADs to a device's PMA. The PMA executes the method on the attribute and, in most cases, returns a result to the PM.

The PM can detect incipient failures (i.e., repetitive soft failures that may become hard failures) and can perform performance analysis. Based on this information, the PM can advise the SM regarding recommended or required path changes (due to errors or poor performance) as well as various load tuning actions that may need to be taken.

Required Features

All protocol-aware IBA devices (i.e., devices other than repeaters) are required to implement the following attributes:

- *ClassPortInfo*. See Table 28-7 on page 794 and the description of *ClassPortInfo* in Table 35-4 on page 1025.
- *PortCounters*. This attribute contains a standard set of required event counters that are always running on each port.
- *PortSamplesControl*. A *PerfSet(PortSamplesControl)* is used to simultaneously:
  - — Select a port on which to take samples.
  - — Select one or more optional operational events on the selected port to be sampled.
  - — Trigger a sampling session of the selected optional events on the selected port.

  An initial execution of a *PerfGet(PortSamplesControl)* returns the number and width of the available counters, the events that can be sampled, and the sampling time interval (called the "tick"). Using this information, the performance application can compute the maximum sample interval that will not cause counter overflow.
- *PortSamplesResult*. Once a sampling session is triggered via the *PerfSet(PortSamplesControl)*, a *PerfGet(PortSamplesResult)* returns the status of the sampling session (Sampling complete, SampleStart timer running, or Sampling underway). If the status indicates that the sampling session is complete, the *PerfGet(PortSamplesResult)* also returns the contents of the sample counters.

Optional Features

A number of performance event counters are optional. A port indicates its support for them in the *PortSampleControls.OptionMask* attribute element (see Table 35-5 on page 1027).

Performance Management MAD Format

Table 35-1 on page 1022 shows the format of the PM MAD. Refer to "Detailed Description of MADs" on page 779 for a detailed discussion of MADs.

Table 35-1: Performance Management MAD Format

Dword		Byte 0	Byte 1	Byte 2	Byte 3	
0	Base MAD Header	Base Version = 01h	Management Class = Perf	Class Version = 01h	R	Perf Method (see Table 35-2 on this page).
1		Status is only meaningful in response packets. See Table 27-2 on page 776. There are no class-specific status bits.		Not used		
2		Transaction ID				
3						
4		Perf AttributeID (see Table 35-4 on page 1025 and Table 35-9 on page 1048).		Reserved		
5		AttributeModifier				
6–15	Perf-specific fields	40 bytes Reserved.				
16–63		192-byte PM MAD payload. Structure and content depend upon Method, Attribute, and Attribute Modifier fields in the header.				

Performance Methods

Table 35-2 on this page lists the methods supported for the PM general services.

Table 35-2: Performance Management Methods

Method	ID	Description
PerfGet()	01h	Read an attribute.
PerfSet()	02h	Write to an attribute.
PerfGetResp()	81h	Response to a *PerfGet()* or *PerfSet()* operation.

Mandatory Performance Attributes

General

Refer to Table 35-4 on page 1025 and Table 35-3 on page 1024. As indicated earlier in this chapter, the following attributes are mandatory and must be implemented on each port:

- *ClassPortInfo*. See Table 28-7 on page 794 and the description of *ClassPortInfo* in Table 35-4 on page 1025.
- *PortCounters*. This attribute contains a standard set of required event counters that are always running on each port.
- *PortSamplesControl*. A *PerfSet(PortSamplesControl)* is used to simultaneously:
  - — Select a port on which to take samples.
  - — Select one or more optional operational events on the selected port to be sampled.
  - — Trigger a sampling session of the selected optional events on the selected port.

  An initial execution of a *PerfGet(PortSamplesControl)* returns the number and width of the available counters, the events that can be sampled, and the sampling time interval (called the "tick"). Using this information, the performance application can compute the maximum sample interval that will not cause a counter overflow.
- *PortSamplesResult*. Once a sampling session is triggered via the *PerfSet(PortSamplesControl)*, a *PerfGet(PortSamplesResult)* returns the status of the sampling session (Sampling complete, SampleStart timer running, or Sampling underway). If the status indicates that the sampling session is complete, the *PerfGet(PortSamplesResult)* also returns the contents of the sample counters.

Initiating a Sampling Session

To initiate a sampling session, the application takes the following actions:

1. Select a random value for *PortSamplesControl.SampleStart* that defines how soon after the *PerfSet(PortSamplesControl)* is executed the sampling session begins. The SampleStart timer allows a performance application to randomize the sample start time and ensure decoupling from node or network events. Values used will typically be tens of milliseconds.

36 Communications Management

The Previous Chapter

The previous chapter provided a detailed description of the Performance Manager (PM), the Performance Management Agent (PMA), and the PM-related mtehods and attributes. Specifically, it covered:

- The role of Performance Management (PM).
- The required features.
- The optional features.
- PM MAD format.
- PM methods.
- Mandatory PM attributes.
- Optional PM attributes.

This Chapter

This chapter provides a detailed description of the Communications Management (CM) MADs, Methods, Attributes, etc. Specifically, it covers:

- CM MAD format.
- CM methods.
- CM attributes.
- CM MADs.
- Definition of *Client* and *Server*.
- Definition of *Active* and *Passive CMs*.
- The three supported models.
- Definition of a stale communications channel.

The Next Chapter

The next chapter provides a detailed description of the Device Manager (DM), Device Management Agents (DMAs), DM Methods, DM Attributes, and so on. Specifically, it covers:

- Definition of IOU and IOCs.
- Each IOU contains a DMA.
- DM MAD format.
- DM Methods.
- DM Attributes.

Introduction

The Communications Manager (CM) and the actions it takes to set up, manage, and destroy connections were introduced in "Intro to Connection Establishment" on page 183.

This chapter provides a detailed description of each of the Communications Management MADs.

CM MAD Format

Table 36-1 on this page defines the format of a CM MAD. A detailed description of MADs can be found in "Detailed Description of MADs" on page 779.

Table 36-1: CommMgt MAD Format

<table>
<tr><th>Dword</th><th></th><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th colspan="2">Byte 3</th></tr>
<tr><td>0</td><td rowspan="6">Base MAD Header</td><td>Base Version = 01h</td><td>Management Class = ComMgt</td><td>Class Version = 01h</td><td>R</td><td>ComMgt Method (see Table 36-2 on page 1071).</td></tr>
<tr><td>1</td><td colspan="2">Status is only meaningful in response packets. See Table 27-2 on page 776. Note that there are no CM-specific status bits.</td><td colspan="3">Not used</td></tr>
<tr><td>2</td><td colspan="5" rowspan="2">Transaction ID</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td><td colspan="2">CM AttributeID (see Table 36-3 on page 1072).</td><td colspan="3">Reserved</td></tr>
<tr><td>5</td><td colspan="5">AttributeModifier (not used for any of the CM attributes)</td></tr>
</table>

Table 36-1: CommMgt MAD Format (Continued)

Dword		Byte 0	Byte 1	Byte 2	Byte 3
6–63	CM-specific fields	• In a *ComMgtGet(ClassPortInfo)*, this data area is unused. • In a *ComMgtSet(ClassPortInfo)*, this data area contains the data to be written to the *ClassPortInfo* attribute. • In a *ComMgtGetResp(ClassPortInfo)*, this data area contains the returned *ClassPortInfo* attribute. • In a *ComMgtSend()*, this data area contains the CM MAD to be sent to the other CA's CM.			

CM Methods

Table 36-2 on this page lists the CM-related methods.

Table 36-2: CM Methods

Method	ID	Description
ComMgtGet(ClassPort-Info)	01h	Read the *ClassPortInfo* attribute. A *ComMgtGet-Resp(ClassPortInfo)* is sent in response.
ComMgtSet(ClassPort-Info)	02h	Write to the *ClassPortInfo* attribute. A *ComMgtGet Resp(ClassPortInfo)* is sent in response.
ComMgtGetResp (message)	81h	Response to a *ComMgtGet(ClassPortInfo)* or a *ComMgtSet(ClassPortInfo)*.
ComMgtSend(message)	03h	Send a CM message (i.e., a CM attribute other than *ClassPortInfo*). This is how CM connection-related MADs are sent.

CM Attributes

Refer to Table 36-3 on page 1072. With the exception of the *ClassPortInfo* attribute, the remainder of the attributes are actually the various types of CM messages (e.g., REQ, REP, RTU, etc.). A CA's CM sends a CM MAD by transmitting a *ComMgtSend(attribute)* MAD, where the message is the attribute contained in the attribute data area (dwords 6 through 63 in Table 36-1 on page 1070).

Table 36-3: CM Attributes

Attribute	Attribute ID	Description
None of the CM attributes use a modifier.		
ClassPortInfo	0001h	Presence of *ClassPortInfo* for a management class (see Table 28-7 on page 794) confirms availability of that management class on a CA, switch, or router. *ClassPortInfo* provides: • Indication of port's ability to generate traps and notices. • Port's RespTimeValue. • If port's CM requires redirection of CM MADs, address to be redirected to is supplied (GID, LID, QP, etc.). • If port's CM supports traps, address to which traps are delivered is supplied (GID, LID, etc.). • Whether multicast, RC, RD, or Raw QPs are supported. The following CM-specific *ClassPortInfo.CapabilityMask* bits are implemented: • Bit 8 = IsMulticastCapable. 1 = Multicast is supported. • Bit 9 = IsReliableConnectionCapable. 1 = RC QPs are supported. • Bit 10 = IsReliableDatagramCapable. 1 = RD QPs are supported. • Bit 11 = IsRawDatagramCapable. 1 = Raw QPs are supported.
ConnectRequest	0010h	A *ComMgtSend(ConnectRequest)* indicates this is a connection request (REQ) MAD. See "REQ (Request) MAD" on page 1074.
MsgRcptAck	0011h	A *ComMgtSend(MsgRcptAck)* indicates this is an MRA MAD. See "MRA (Message Receipt Acknowledgment) MAD" on page 1096.

Table 36-3: CM Attributes (Continued)

Attribute	Attribute ID	Description
ConnectReject	0012h	A *ComMgtSend(ConnectReject)* indicates this is a connection reject (REJ) MAD. See "REJ (Reject) MAD" on page 1098.
ConnectReply	0013h	A *ComMgtSend(ConnectReply)* indicates this is a connection reply (REP) MAD. See "REP (Reply) MAD" on page 1088.
ReadyToUse	0014h	A *ComMgtSend(ReadyToUse)* indicates this is a Ready To Use (RTU) MAD. See "RTU (Ready to Use) MAD" on page 1093.
DisconnectRequest	0015h	A *ComMgtSend(DisconnectRequest)* indicates this is a Disconnect Request (DREQ) MAD. See "DREQ (Disconnect Request) MAD" on page 1094.
DisconnectReply	0016h	A *ComMgtSend(DisconnectReply)* indicates this is a Disconnect Reply (DREP) MAD. See "DREP (Disconnect Reply) MAD" on page 1095.
ServiceIDResReq	0017h	A *ComMgtSend(ServiceIDResReq)* indicates this is Service ID Resolution Request (SIDR_REQ) MAD. See "SIDR_REQ (ServiceID Resolution Request) MAD" on page 1108.
ServiceIDResReqResp	0018h	A *ComMgtSend(ServiceIDResReqResp)* indicates this is Service ID Resolution Reply (SIDR_REP) MAD. See "SIDR_REP (ServiceID Resolution Reply) MAD" on page 1109.
LoadAlternatePath	0019h	A *ComMgtSend(LoadAlternatePath)* indicates this is a Load Alternate Path (LAP) MAD. See "LAP (Load Alternate Path) MAD" on page 1104.
AlternatePathResponse	001Ah	A *ComMgtSend(AlternatePathResponse)* indicates this is a Alternate Path Response (APR) MAD. See "APR (Alternate Path Response) MAD" on page 1107.

37 Device Management

The Previous Chapter

The previous chapter provided a detailed description of the CM MADs, Methods, Attributes, etc. Specifically, it covered:

- CM MAD format.
- CM methods.
- CM attributes.
- CM MADs.
- Definition of *Client* and *Server*.
- Definition of *Active* and *Passive CMs*.
- The three supported models.
- Definition of a stale communications channel.

This Chapter

This chapter provides a detailed description of the Device Manager (DM), Device Management Agents (DMAs), DM Methods, DM Attributes, and so forth. Specifically, it covers:

- Definition of IOU and IOCs.
- Each IOU contains a DMA.
- DM MAD format.
- DM methods.
- DM attributes.

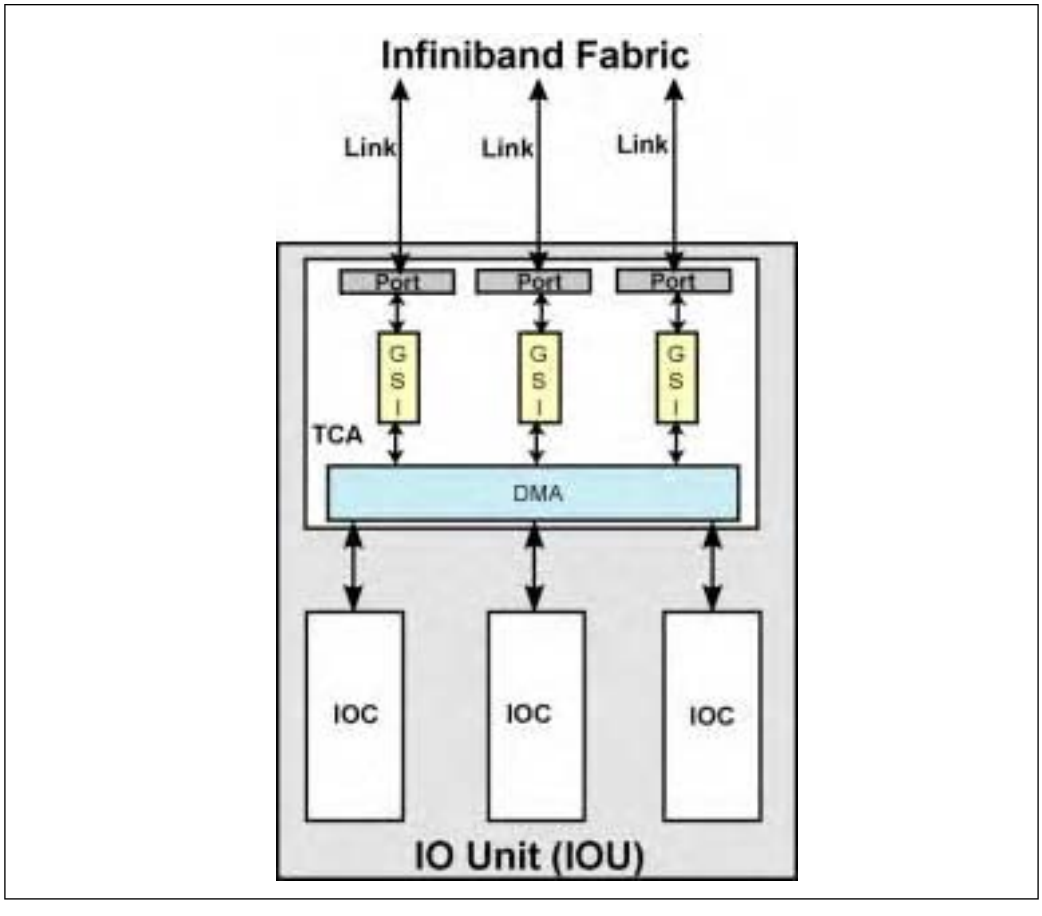
Definition of IOU and IOCs

Refer to Figure 37-1 on page 1118. The terms IOU and IOC were defined very early in the book (see "Some Preliminary Terminology" on page 10).

IOU Contains a DMA

Each IOU contains a DMA (Device Management Agent) that handles inbound attribute access requests issued by the Device Manager (DM) and, if and when an internal DM-related event occurs, sends a *DevMgtTrap(Notice)* MAD to the DM (assuming that traps have been enabled by the DM).

Figure 37-1: IOU, IOCs, and TCA



DM MAD Format

DM MADs have the format shown in Table 37-1 on this page.

Table 37-1: DM MAD Format

Dword		Byte 0	Byte 1	Byte 2	Byte 3	
0	Base MAD Header	Base Version = 01h	Management Class = DevMgt	Class Version = 01h	R	DM Method (see "DM Methods" on page 1120)
1		Status is only meaningful in response packets. See Table 27-2 on page 776. In addition, the following DM-specific status bits are defined: • Bit 8 = NoResponse 1 = IOC Not responding • Bit 9 = NoServiceEntries 1 = Service Entries are not supported • Bit 15 = GeneralFailure 1 = IOC General Failure		Not used		
2		Transaction ID				
3						
4		DM AttributeID (see "DM Attributes" on page 1120)		Reserved		
5		AttributeModifier (see "DM Attributes" on page 1120).				
6–15	DM-specific fields	40 bytes (reserved)				
16–63		192-byte DM MAD payload. Structure and content depends upon Method, Attribute, and Attribute Modifier fields in the header.				

DM Methods

Table 37-2 on this page defines the methods implemented for the DM class. Also see Table 37-4 on page 1125.

Table 37-2: Device Management Methods

Method	ID	Description
DevMgtGet()	01h	Read an attribute. The DMA responds with a *DevMgtGetResp()* MAD.
DevMgtSet()	02h	Write to an attribute. The DMA responds with a *DevMgtGetResp()* MAD.
DevMgtGetResp()	81h	DMA response to *DevMgtGet()* or *DevMgtSet()*.
DevMgtTrap(Notice)	05h	A device's DMA uses this method to deliver an event notification (i.e., the *Notice* attribute) to the DM. See "Traps" on page 790.
DevMgtTrap Repress(Notice)	07h	Sent to a device's DMA by the DM instructing the DMA to stop repeatedly issuing the same event notification [i.e., *DevMgtTrap(Notice)*].

DM Attributes

General

Table 37-3 on page 1121 lists the DM class attributes. Also see Table 37-4 on page 1125.

Table 37-3: Device Management Attributes

Attribute	ID	Attribute Modifier	Description
ClassPortInfo	0001h	00000000h	See Table 28-7 on page 794. There are no class-specific *CapabilityMask* bits defined.
Notice	0002h	00000000h–FFFFFFFF	DM-related event notifications are logged in the *Notice* attribute and, if traps have been enabled by the DM, an event notification is automatically sent to the DM when a DM-related event occurs within the IOU. The event is delivered to the DM in a *DevMgtTrap(Notice)*. See Table 28-9 on page 798 and "Notice Attribute" on page 1125.
IOUnitInfo	0010h	00000000h	Read-only. List of all IOCs present in a given IOU. Each IOU may support up to FFh (255) controllers. See "IOUnitInfo Attribute" on page 1126.
IOControllerProfile	0011h	00000001h–000000FFh	Read-only. IOC Profile Information (see "IOControllerProfile Attribute" on page 1127). AttributeModifier identifies the IOC.
DiagCode	0024h	00000001h–FFFFFFFFh	Vendor-specific diagnostic information for device (IOC) specified by AttributeModifier (same author's note as in description of *PrepareToTest* in this table). Results are reported in a format consistent with the 16-bit *PortInfo.DiagCode*. For a detailed description, refer to "VendorDiag and PortInfo.DiagCode Attributes" on page 843.

Glossary

Term	Description
3U module	The 3U form factor is a well-established open standard for small passive-backplane systems. It was developed for use in VMEbus systems. Its size, 100mm x 160mm, was defined in the original VMEbus specification, IEEE 1014-1987 and complies with the IEEE 1101.1 mechanical specification.
6U module	The 6U form factor is a well-established open standard for small passive-backplane systems. It was developed for use in VMEbus systems. Its size, 160mm x 233mm was defined in the original VMEbus specification, IEEE 1014-1987 and complies with the IEEE 1101.1 mechanical specification.
8B/10B encoding	See “8-bit/10-bit Encoder per Lane” on page 706.
::	See “IPv6 Address Documentation Convention” on page 154.
Abandon operation	See “Resync Operation” on page 499.
ABR	Adjusted Block Received. See “Terminology” on page 640.
Ack coalescing	See “Responder May Coalesce Acks for Sends or RDMA Writes” on page 385.
Ack, ghost	“Detecting Ghost Acks” on page 210.
Ack, window	See description of Window in Table 33-10 on page 949.
Acknowledge CP	Acknowledge Control Packet. See Table 30-1 on page 855.
Acknowledge Extended Transport Header (AETH)	See Figure 17-6 on page 369 and Table 17-3 on page 370.
AckReq bit	Acknowledge Request bit (BTH:AckReq). See “Ack Packet Scheduling” on page 435.

Term	Description
ActCount	Master SM's Activity Counter. See "Are You Alive?" on page 854.
ActDefer state	Link Layer Active Defer state. See "LinkActDefer State" on page 609.
Action	The specification uses the term Action one time and offers no definition. It appears to be referring to the mysterious Action method that a device's SMA acts upon.
Active state	Link Layer Active state. See "LinkActive State" on page 608.
Active/Active Model	See "Active Client to Active Client" on page 1112.
Active/Passive Model	See "Active Client to Passive Server" on page 1112.
actively managed chassis	See the description of chassis, actively managed in the glossary.
Activity Counter	Master SM's Activity Counter. See "Are You Alive?" on page 854.
Address Handle	See "Address Handles" on page 526.
Address vector	See: • Address Vector in Table 19-3 on page 508 for RC and UC. • Address Vector in Table 12-1 on page 223 for RD EEC. • "Create Address Handle Verb" on page 526 for UD.
Adjusted Blocks Received (ABR)	See the definition of ABR in the glossary.
Admin Data Area	See Table 33-1 on page 919 and Table 33-2 on page 920.
AETH	See the definition of Acknowledge Extended Transport Header in the glossary.
Affiliated Asynchronous Error	See "Affiliated Asynchronous Errors" on page 293.

Term	Description
Affiliated Asynchronous Event	See “Affiliated Asynchronous Events” on page 292.
Aggregatable Global Unicast Addresses	See “Unlimited Global Unicast Addresses (Travel the Globe!)” on page 148.
Alternate Path Response (APR)	See “APR (Alternate Path Response) MAD” on page 1107.
Alternate path	This is the alternate path between two QPs or EECs that is switched to when Automatic Path Migration is triggered. Refer to: • Table 23-1 on page 578 for RC and UC QPs. • Table 23-2 on page 580 for RD EECs. • “Automatic Path Migration” on page 575.
APM	See “Automatic Path Migration” on page 575.
Application-specific Manager	See “General Services Managers” on page 167.
APR	See the definition of Alternate Path Response in the glossary.
Armed state	“Automatic Path Migration” on page 575.
Asynchronous Event Handler	See “Registering a Handler” on page 292.
Asynchronous event	See the definition of Affiliated Asynchronous Event in the glossary.
Atomic Ack packet	See “Atomic Operation Consists of a Request and Ack Packet” on page 96.
Atomic Compare and Swap If Equal operation	See “Atomic Compare and Swap If Equal Operation” on page 98.
Atomic Fetch and Add operation	See “Atomic Fetch and Add Operation” on page 97.
Atomic operation	See “Atomic RMW Operations” on page 94.

Term	Description
Atomic Request packet	See “Atomic Operation Consists of a Request and Ack Packet” on page 96.
AtomicAckETH	See “Atomic Acknowledge Packet Contents” on page 96 and Figure 17-11 on page 381.
AtomicETH	See “Request Packet Contains AtomicETH Field” on page 96 and Figure 17-10 on page 380.
Attribute ID	See Table 28-1 on page 783 and Table 28-2 on page 783.
Attribute	See “Definition of an Attribute” on page 26.
Automatic Path Migration (APM)	See “Automatic Path Migration” on page 575.
Auxiliary Power signal group	See “Chassis and Module” on page 989.
Base LID	See “Source Port’s LID Address” on page 45 and “Assigning Port’s Base LID Address” on page 137.
Base Transport Header (BTH)	See “Every Packet Contains a BTH” on page 15. Also see Figure 11-1 on page 206.
Baseboard Management Agent (BMA)	See Table 9-1 on page 170 and “Baseboard Management” on page 987.
Baseboard Management Key (B_Key)	See “Baseboard Management Key (B_Key)” on page 340.
Baseboard Manager (BM)	See “General Services Managers” on page 167 and “Baseboard Management” on page 987.
Beacon Sequence	Also referred to as TS1 (i.e., Training Sequence 1). Refer to “Link Training” on page 732 and Figure 26-30 on page 739.
Big-endian	The most-significant byte of a multi-byte data object is stored in the start memory location, and the successive lower bytes of the object are stored in successively higher memory locations.

Term	Description
Bind operation	See “Binding a Window To a Region” on page 310.
BTH	See the definition of Base Transport Header in the glossary.
Bulk-Power signal group	See “Chassis and Module” on page 989.
Byte Striping	See “Byte Striping” on page 703.
B_Key	See the definition of Baseboard Management Key in the glossary.
B_KeyLeasePeriod	See “Device Logic Detects Death of BM” on page 346.
B_KeyProtectBit	See “The Key Comparison” on page 343.
B_KeyViolations	See “Starting the Countdown and Handling a Timeout” on page 346.
CA	Channel Adapter. See “Some Preliminary Terminology” on page 10.
CapabilityMask	This is a 16-bit attribute element of the *ClassPortInfo* attribute. It indicates the capabilities of the general services management agent associated with this *ClassPortInfo* attribute. • Bit 0. If = 1, the MA is capable of generating *Trap(Notice)* MADs. • Bit 1. If = 1, the MA supports execution of the *Get(Notice)* and *Set(Notice)* methods for accessing the device’s Notice Queue to obtain event notifications. • Bits 7:2. Reserved. • Bit 15:8 are class-specific capability bits.
Channel Adapter (CA)	See the definition of CA in the glossary.
Channel Interface (CI)	The verb layer, the HCA driver, and the HCA interface (and, if present, the device ROM) are collectively referred to as the CI in the specification.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X