Chapter 42

An Introduction to the InfiniBand™ Architecture

GREGORY F. PFISTER

IBM Enterprise Server Group
Server Technology and Architecture
Austin, Texas, 78758, USA

Abstract

The InfiniBand Architecture (IBA) is a new industry-standard architecture for server I/O and inter-server communication. It was developed by the InfiniBand[SM] Trade Association (IBTA) to provide the levels of reliability, availability, performance, and scalability necessary for present and future server systems, levels significantly better than can be achieved with bus-oriented I/O structures. This chapter provides a description of the reason IBA was developed, an brief overview of the architecture as a whole, more detailed information about several selected IBA topics, and discussion of industry implications of this architecture.

42.1 The IBTA and the Specification

The IBTA is a group of 180 or more companies founded in August 1999 to develop IBA. Membership is also open to Universities, research laboratories, and others. The IBTA is lead by a Steering Committee whose members come from Dell, Compaq, HP, IBM, Intel, Microsoft, and Sun, co-chaired by IBM and Intel. Sponsor companies are 3Com, Cisco Systems, Fujitsu-Siemens, Hitachi, Adaptec, Lucent Technologies, NEC, and Nortel Networks.

Approximately 100 individuals from the IBTA member companies worked for approximately 14 months to define and describe IBA, and the result is both deep and broad: It is deep in the sense that IBA extends from physical interconnects and form factors up to high-level management functions; and it is broad in the sense that IBA provides a very wide range of function from simple unreliable communication to partitioning, with many options. The resulting specification [1, 2] is large—approximately 1500 pages long. Clearly this article can only introduce the concepts and features involved; the reader is referred to the specification itself for details.

The size of the specification is also partly the result of two goals of the development process: First, the result had to scale down to cost-effective small server systems as well as scaling up to large, highly robust, enterprise-class facilities. Second, it should provide as much scope as possible for new invention and vendor differentiation.

The first item, scale down and up, resulted in a large number of options and a great deal of parameterization. There are multiple link widths, multiple MTU (minimum transfer unit) sizes, very small architectural minima on all properties, and major features that are optional. To simplify software support, *profiles* have been defined. While referred to simply as “Profile A” and “Profile B,” the intent is that one defines features and sizes that software can expect on smaller systems; and the other similarly defines large systems.

The second item, scope for new invention, produced some seemingly odd results. For example, while connectors for copper interconnect are fully defined (as required for interoperability), cable length and wire gauge is not. Instead, there is an attenuation budget defined (15dB), and any cable implementation meeting that budget is allowed. (Initially this budget is expected to allow cables of 10m to 17m, depending on cost and implementation.) In a similar vein, the specification of how host software can control IBA contains no APIs, defined registers, etc. Instead it is specified as a collection of *verbs*—abstract representations of the function that must be present, but may be implemented with any combination and organization of hardware, firmware, and software. This is not expected to cause portability problems because application programs will likely never see IBA directly, instead using standard APIs and OS-defined interfaces for I/O and communication.

42.2 Reasons for the InfiniBand Architecture

The primary reason why development of IBA was initiated is that processing power is substantially outstripping the capabilities of industry-standard I/O systems using busses.

While busses have the major advantage of simplicity, and have served the industry well up to this point, bus-based I/O systems do not use their underlying electrical technology well to provide data transfer (bandwidth) out of a system to devices. There are several reasons for this.

First, buses are inherently shared, requiring arbitration protocols on each use, a "tax" that increases with the number of devices or hosts. As the clock speed of a bus increases, and its long burst bandwidth increases, the negative effects of arbitration overhead are magnified.

A second reason why busses are inefficient users of electrical technology actually is really a function of how they are used. Since common industry standard busses are memory-mapped, short sequences, such as command transmission and the reading status, are performed using processor *load* and *store* operations. For *store* operations, this is not a major problem since modern processors allowing out-of-order completion can overlap substantial additional processing with *store* processing. Unfortunately, most devices require *load* instructions for operations like reading status and retrieving small amounts of data. *Loads*, unlike *stores*, usually cannot proceed very far without making the processor stop to wait for the requested data. This can be a very serious efficiency problem. Internal analysis by the author of a competitor's system executing a commercial benchmark requiring many fairly small I/O operations—the TPC-C benchmark—indicated that nearly 30% of processor execution time was spent waiting on I/O *loads* in this fashion.

In addition to the above problems, busses also do not provide the level of reliability and availability now being required of server systems. A single device failure can inhibit the correct operation of the bus itself, causing all the devices on the bus to become unavailable, including those attached by bridges. Finding out which device is at fault, or whether there is a failure of the bus itself, often becomes an aggravating and time-consuming exhaustive search.

Several external connection techniques, such as Fibre Channel, have been used to overcome come of these difficulties. However, they must still enter the processing complex through an industry-standard bus, making it impossible to avoid the bottlenecks and low availability characteristic of standard I/O busses.

These difficulties appear at a time when there are trends to use ever more I/O bandwidth and function. For example: Increasing use of data mining techniques requires scans of amounts of data now regularly exceeding terabytes; closer coupling between web front ends, intermediate transaction-oriented applications, and back-end data mining facilities are increasingly desired; and the expected function is increasing from simple pages of data, through streaming audio to streaming video, with vastly increased data transfer requirements.

While the above are key reasons why the development of IBA was initiated, the result is much more than just a replacement for busses. Indeed, the author's opinion is that IBA has significant long-term implications for the way large systems will be structured in the future. These are discussed under <Industry Implications> later in this chapter.

42.3 An InfiniBand Architecture Overview

These problems mentioned above have been solved before by individual server vendors in a number of ways. However, all such solutions have been wholly or partially proprietary, thereby incurring significant costs. IBA is, instead, an industry-standard architecture, which should achieve significant volumes and hence much lower costs.

It avoids the problems with busses discussed above through two basic characteristics:

- point-to-point connections: All data transfer is point-to-point, not bussed. This avoids arbitration issues, provides fault isolation, and allows scaling to large size by the use of switched networks.
- channel (message) semantics: commands and data are transferred between hosts and devices not as memory operations but as messages.

In other words, IBA explicitly treats I/O as communication. Its purpose is then to provide very low-overhead, high bandwidth communication between devices and hosts, as well as among hosts themselves. In doing so, it makes explicit and deliberate use of many concepts originating in the realm of networking.

Like any modern communication system, IBA is a stack divided into physical, link, network, and transport layers. The discussion which follows will overview the main elements of IBA in approximately that order, followed by management. Unfortunately, no linear ordering of a description based on transport layers is ideal, since there are features at higher levels of the communication stack control and use features implemented at lower levels, and there are features at the lower levels which cannot be justified without describing the higher-level facilities they support.

Therefore, the rest of this chapter will consist first of a general overview of IBA, and then more detailed discussion of several selected topics.

42.3.1 The IBA Subnet

The smallest complete IBA unit is a subnet, illustrated in Figure 1. Multiple subnets can be joined by routers (not shown) to create large IBA networks.

The elements of a subnet, as shown in the figure, are endnodes, switches, links, and a subnet manager. Endn-

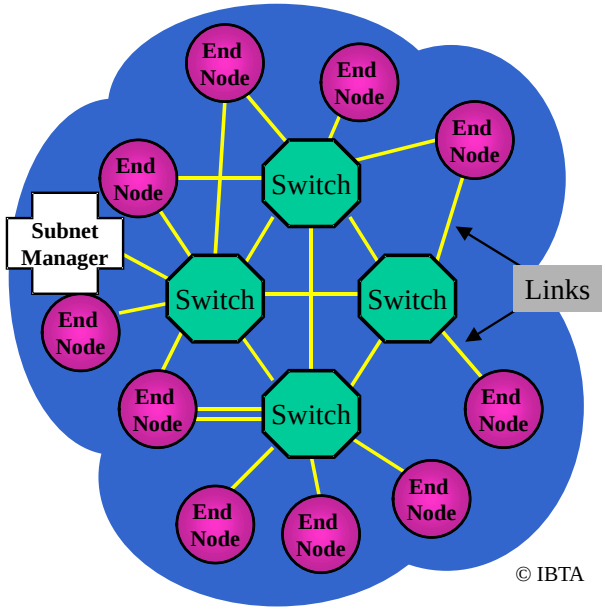


Figure 42.1: An InfiniBand Architecture Subnet

odes, such as hosts and devices, send messages over links to other endnodes; the messages are routed by switches. Routing is defined, and subnet discovery performed, by the Subnet Manager. Channel Adapters (CAs) (not shown) connect endnodes to links. Links may also incorporate retiming repeaters, but since these are architecturally invisible they will not be mentioned further.

42.3.2 Links

IBA links are bidirectional point-to-point communication channels, and may be either copper and optical fibre. The signalling rate on all links is 2.5 Gbaud in the 1.0 release; later releases will undoubtedly be faster. Automatic training sequences are defined in the architecture that will allow compatibility with later faster speeds.

The physical links may be used in parallel to achieve greater bandwidth, as shown in Table 1 below. The different link widths are referred to as 1X, 4X, and 12X.

Link Width	Bi-Directional Bandwidth
1X	500 MBytes/second
4X	2 GBytes/second
12X	6 GBytes/second

Table 42.1 Link Widths

The basic 1X copper link has four wires, comprising a differential signalling pair for each direction. Similarly, the 1X fibre link has two optical fibres, one for each direction. Wider widths increase the number of signal paths as implied. Figure 2 below illustrates the copper and optical cable plugs for all copper and optical widths There is also a copper backplane connection allowing dense structures of modules to be constructed; unfortunately, an illustration of that which reproduces adequately in black and white were not available at the time of publication. The 1X size allows up to six ports on the faceplate of the standard (smallest) size IBA module.

"Short reach" (multimode) optical fibre links are provided in all three widths; while distances are not specified (as explained earlier), it is expected that they will reach 250m for 1X and 125m for 4X and 12X. "Long reach" (single mode) fiber is defined in the 1.0 IBA specification only for 1X widths, with an anticipated reach of up to 10Km. Other width long reach links may be defined later.

42.3.3 Switches

IBA switches route messages from their source to their destination based on routing tables that are programmed with forwarding information during initialization and network modification. The routing tables may be linear, specifying an output port for each possible destination address up to a switch-specific limit, indexed by that address; or random, initialized by giving them {destination, output port} pairs. The exact format, content, and organization of these tables in the switch hardware is vendor-specific; all that is defined is the format of the data sent by the subnet manager to load them.

Messages are segmented into packets for transmission on links and through switches. The packet size is such that after headers are counted, the Maximum Transfer Unit of data, MTU, may be 256 bytes, 1KB, 2KB, or 4KB. In systems with mixed MTUs, subnet management provides endnodes with the Path MTU appropriate to reach a given destination. For most communication service types (see Section 42.3.6, below), segmentation of messages into packets on transmission and reassembly on receipt are provided by channel adapters at the endnodes.

Switch size—the number of ports—is vendor-specific, as is the link width supported. It is anticipated that a wide variety of switch implementations will be available with very different capabilities and price points. The maximum switch size supported is one with 256 ports, and switches can be cascaded to form large networks.

Switches may also optionally support multicast. Packets sent to a multicast address (a subset of the available addresses are dedicated to this) are then replicated in the switch, sent out multiple ports, as defined by separate multicast forwarding tables.

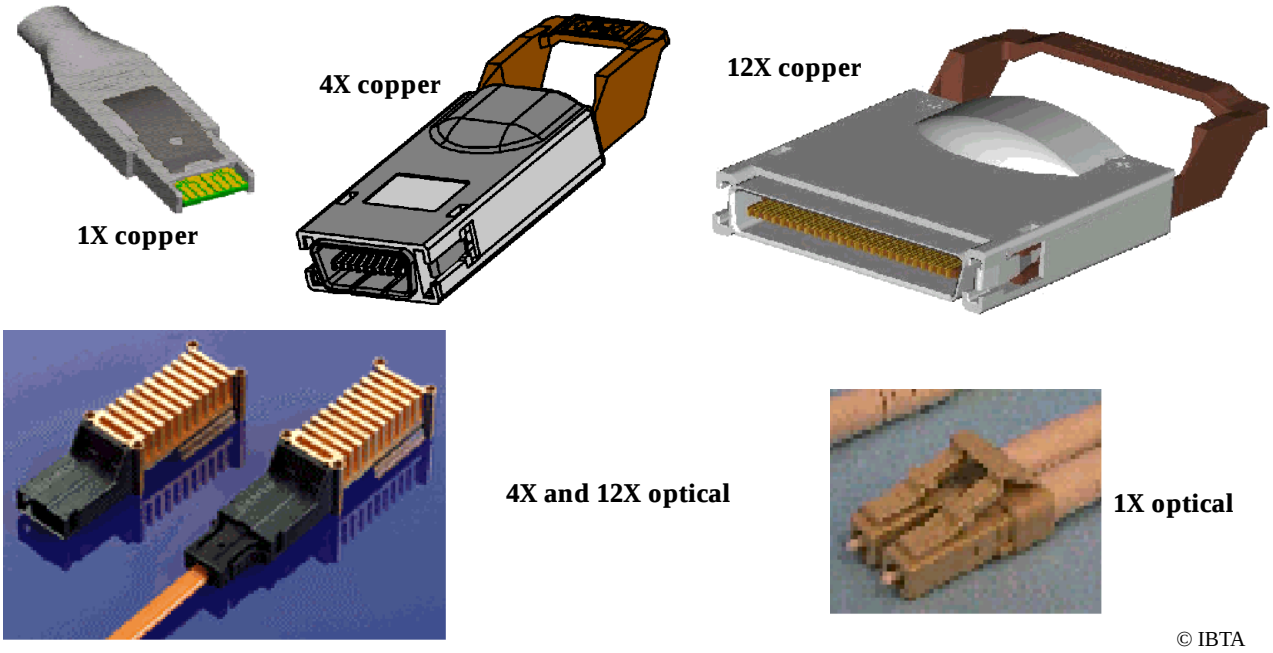


Figure 42.2: Physical Plugs and Connectors (not to same scale)

Switches also support multiple virtual lanes through a mechanism called service levels; this is discussed in the "selected topics" sections later.

The addressing used by switched (Local Identifiers, or LIDs) allows 48K endnodes on a single subnet; the remainder of the 64K LID address space is reserved for multicast addresses. Routing between different subnets is done on the basis of a Global Identifier (GID) 128 bits long, modelled after IPv6 addresses. This allows for essentially unlimited expansion capability.

42.3.4 Endnodes

IBA endnodes are the ultimate sources and sinks of communication in IBA. They may be host systems or devices (network adapters, storage subsystems, etc.). It is also possible that endnodes will be developed that are bridges to legacy I/O busses such as PCI, but whether and how that is done is vendor-specific; it is not part of the InfiniBand architecture. Note that as a communication service, IBA makes no distinction between these types; an endnode is simply an endnode. So all IBA facilities may be used equally to communicate between hosts and devices; or between hosts and other hosts like "normal" networking; or even directly between devices, e.g., direct disk-to-tape backup without any load imposed on a host.

IBA defines several standard form factors for devices used as endnodes, illustrated in Figure 3: standard, wide, tall, and tall wide. The standard form factor is approximately 20x100x220 mm. Wide doubles the width, tall doubles the height, and tall wide doubles both dimensions. Power dissipation ratings range from 25W for standard to 100W for tall wide. In addition to link hot plug/unplug, there are standardized baseboard management functions for power control.

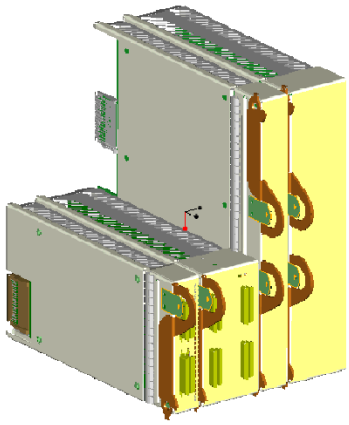


Figure 42.3: Standard, Standard Wide, Tall, and Tall Wide Form Factors

42.3.5 Channel Adapters

The interface between an endnode and a link is a Channel Adapter (CA). Host CAs (HCA) differ from the CA for a device (called a Target CA, TCA) in that the HCA has a collection of features that are defined to be available to host

programs, defined by *verbs*; a TCA has no defined software interface.

The position of an HCA in a host system is vendor-specific. It is expected that initial implementations will provide HCAs as cards attached to a standard I/O bus in order to quickly provide platforms for software development and evaluation. Ultimately, however, HCA implementations will undoubtedly attach directly to, or become a part of, the host memory control subsystem and partly or completely replace current I/O busses.

CAs source the several communication service types of IBA, using queues to hold requests for work to be done and completions, as discussed in a later section. HCAs also contain a specific memory model, discussed in the "selected topics" sections later.

42.3.6 Communication Service Types

IBA provides several different types of communication services between endnodes:

- Reliable Connection (RC): a connection is established between endnodes, and messages are reliably sent between them. This is optional for TCAs (devices), but mandatory for HCAs (hosts).
- (Unreliable) Datagram (UD): a single packet message can be sent to an endnode without first establishing a connection; transmission is not guaranteed.
- Unreliable Connection (UC): a connection is established between endnodes, and messages are sent, but transmission is not guaranteed. This is optional.
- Reliable Datagram (RD): a single packet message can be reliably sent to any endnode without a one-to-one connection. This is optional.
- Raw IPv6 Datagram & Raw Ethertype Datagram (optional) (Raw): single-packet unreliable datagram service with all but local transport header information stripped off; this allows packets using non-IBA transport layers to traverse an IBA network, e.g., for use by routers and network interfaces to transfer packets to other media with minimal modification. It will not be further discussed in this article.

In the above, "reliably send" means the data is, barring catastrophic failure, guaranteed to arrive in order, checked for correctness, with its receipt acknowledged. Each packet, even those for unreliable datagrams, contains two separate CRCs, one covering data that cannot change (Constant CRC) and one that must be recomputed (V-CRC) since it covers data that change; such change can occur only when a packet moves from one IBA subnet to another, however.

The classes of service above, with the exception of the oxymoronic Reliable Datagram class and the Raw classes, are strongly reminiscent of similarly-named networking communication services, such as those provided by IP. This is intentional, since they provide essentially the same services. However, these are designed for hardware implementation, as required by a high-performance I/O system. In addition, the host-side functions have been designed to allow all service types to be used completely in user mode, without necessarily using any operating system services; and without any required copying of data ("zero copy").

The RC, UC, and RD classes also support remote DMA (RDMA)[1]: moving data directly into or out of the memory of an endnode. This and user mode operation implies that virtual addressing must be supported by the channel adapters, since real addresses are unavailable in user mode. The memory mapping paradigms used by IBA are discussed in the "selected topics" sections later.

In addition to RDMA, the reliable communication classes also optionally support atomic operations directly against endnode's memory. The atomic operations supported are Fetch-and-Add and Compare-and-Swap, both on 64-bit data. Atomics are effectively a variation on RDMA: a combined write and read RDMA, carrying the data involved as immediate data. Two different levels of atomicity are optionally supported: atomic with respect to other operations on a target CA; and atomic with respect to all memory operation of the target host and all CAs on that host.

The seemingly oxymoronic Reliable Datagram service is discussed in the "selected topics" sections later.

42.3.7 Queue Pairs and CA Operation

For all the service types, CAs communicate using Work Queues of three types: Send, Receive, and Completion. Send and Receive Queues are always used as *Queue Pairs* (QPs), as illustrated in Figure 4. The architecture is similar to that of the VI Architecture [3], but contains a number of extensions. A particular QP in a CA is the destination or source of all messages: Connected services connect QPs, and unconnected services target QPs on other CAs, specifying a QP number along with the Local Identifier (LID) of the CA port they target. The type of service used and QP's size, the maximum number of requests that can be queued, are specified when a QP is created.

Each QP also has an associated port, also specified when the QP is created; this is an abstraction of the connection of a CA to a link. CAs may have multiple ports, as also illustrated.

The number of QPs is at least three: Two for management (described later), and one for operation. The smallest maximum size of a QP (number of queued requests) is ven-

1. UC supports only RDMA write operations; the others support both write and read.

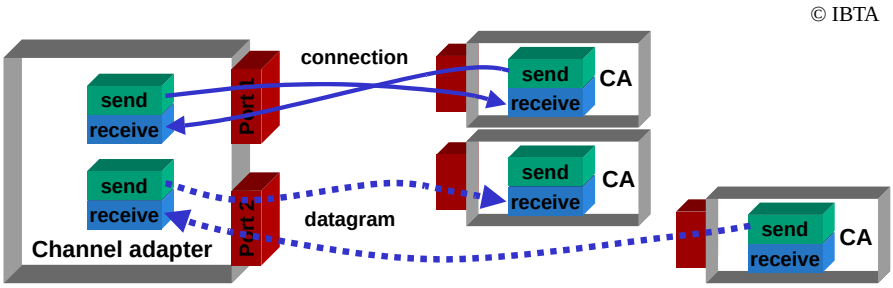


Figure 42.4: Queue Pairs, Ports, and their **Use**

dor-specific; Profile A sets the minimum of 16 entries, and Profile B sets a minimum of 128 for send and 256 for receive.

For HCAs, verbs define how software places work on queues for processing. The overall flow is illustrated in Figure 5: Work is placed on a send or receive queue, is processed by HCA hardware and consumed, and when processing is completed an entry is optionally placed on a completion queue associated with the work queue. The user may request that a completion notification routine be invoked when a new entry is added to a completion queue. The format of an actual Work Queue Entry is vendor-specific; the verbs specify how abstract Work Requests are placed on queues.

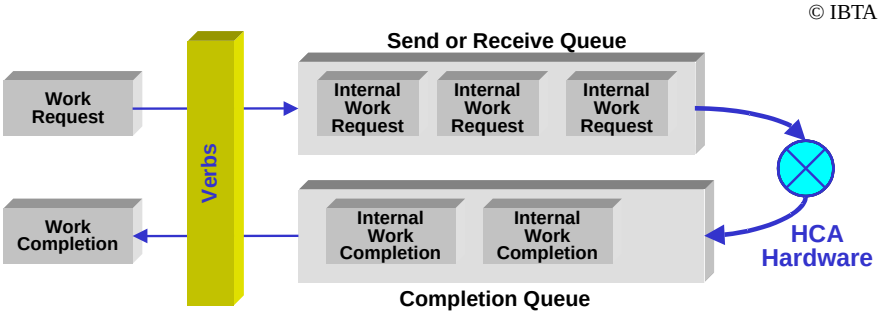


Figure 42.5: Work Request Processing

The types of WRs supported are:

- send and receive a message of the type supported by the QP
- perform a remote direct memory access (RDMA) operation
- bind a memory window (described later)
- atomic operation (optional).

Send, receive, and RDMA operations may also specify a gather/scatter list of data segments in user space to be sent or received into. The maximum number of entries available is vendor-specific; Profile A has a minimum of 3 for most cases, and Profile B has a minimum of 8.

42.3.8 Subnet Management

IBA management is defined in terms of managers and agents. Managers are active entities; agents are passive entities that respond to messages from managers. The only exception to agent passivity is that they may optionally emit traps targeting managers.

Every IBA subnet must contain a single master subnet manager., residing on an endnode or a switch. It discovers and initializes the network, assigning Local IDs (LIDs) to all elements, determining path MTUs, and loading the switch routing tables that determine the paths from endnode to endnode. The master subnet manager does this by communicating with Subnet Management Agents that must exist on all nodes; they respond with information about the node, such as whether it is a switch or a CA, whether it can be a manager, if so what it's priority is, etc. The subnet management agent is also in charge of receiving settings from the master manager such as a node's LID, the location of the master subnet manager, where to send traps, etc.

Following initialization, the master subnet manager provides information to endnodes on request through a closely related entity referred to as Subnet Administration (SA); this is effectively an "agent personality" of the master subnet manager. For example, SA is the source of information to an OS that informs it at boot time about what devices it may access; or provides detailed path information when connecting to another endnode, such as the path MTU, LID, available service levels, etc. The master subnet man-

ager also regularly scans the subnet to detect additions (hot plug) or deletions (hot unplug); optionally, traps can be provided that inform the subnet manager of this without explicit scanning.

These functions have been referred to as performed by the *master* subnet manager because additional *standby* subnet managers may also be present on the network for higher availability. These will typically perform some form of polling to ensure the master is operational, and failover to one of them when it is not. The details of this during normal operation are vendor specific; failing over between subnet managers from different vendors is in general not supported, since subnet management is strongly tied to routing and routing is vendor specific.

During subnet initialization, however, a specific polling algorithm is used, along with a specified initialization state machine, illustrated in Figure 6. This is necessary for endnodes (hosts, particularly) from different vendors to correctly collaborate on choosing a unique single master with the highest priority.

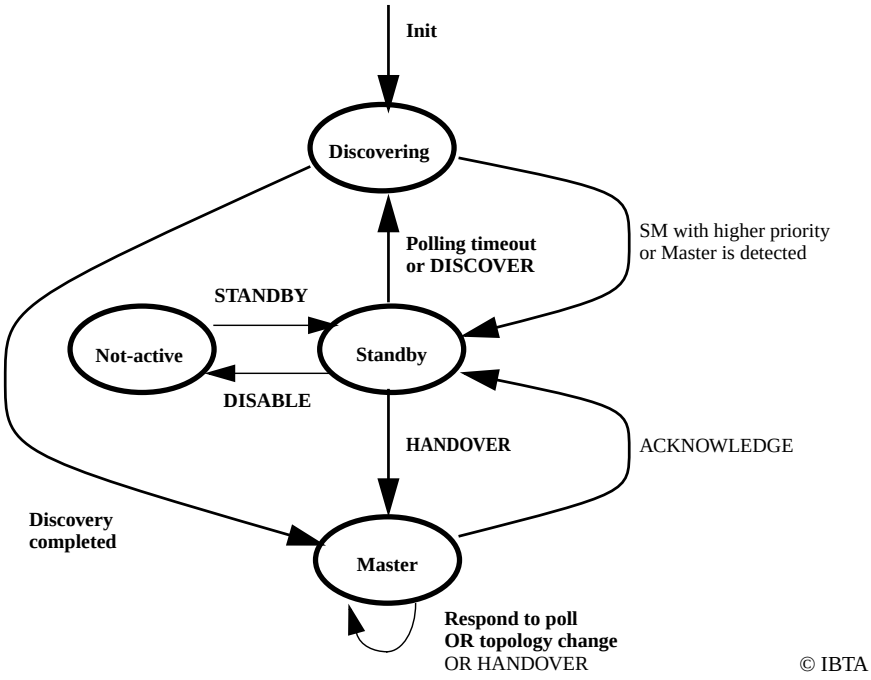


Figure 42.6: Subnet Management Initialization State Machine

The operation of the initialization state machine is as follows: Initially any node that can be a subnet manager enters the state machine at the "discovering" state and begins scanning the network. Once it discovers another subnet-management-enabled node of highest priority, or one already elected master, it goes into the standby state where it polls the master. Should the master fail to respond to a poll, it goes back to discovery. If no other node has higher priority, it has been elected master, enters the master state, and begins initializing the subnet. Unneeded standby managers can be put in a not-active state, where they cease to poll the master, using a command message (such messages are indicated in capital letters in the diagram). A master must, as mentioned, regularly scan the network; if it discovers a standby manager with a higher priority, it initiates a hand over of mastership with the handover message, which is acknowledged.

The protocol implemented by this state machine has been formally verified using an automated protocol verification system at IBM's Research Laboratory in Haifa, Israel.

IBA also defines several other managers, e.g.:

- Baseboard management, which controls power, LED settings, etc., of managed IBA modules.
- Performance management, which provides means for sampling a variety of required and optional performance counters. All endnodes and switches must have counters for octets received and transmitted, packets received and transmitted, and wait states. A variety of optional values to count are also defined including queue depth, octets and packets send/received per virtual lane, packets discarded, etc.

All management is performed in-band, using Management Datagrams (MADs). MADs specifically use least-common-denominator communication: They are unreliable datagrams with 256 bytes of data (minimum MTU). Their format, and the actions taken on receipt, are all specified in detail.

MADs specifically for subnet management, a subset of MADs called SMPs (Subnet Management Packets), are unique in several ways: They are the only packets allowed on virtual lane 15 (VL15); they are always sent and received on Queue Pair 0 of each port; and they can use directed routing. Directed routing is sometimes referred to in other areas a source routing. When using it, the SMP explicitly indicates which port it exits from when going from switch to switch; when using normal, LID (destination), routing, just the destination address is provided and the switch chooses the port using its routing tables. Directed routing is needed because, clearly, LID routing cannot be used before the routing tables are initialized.

MADs for other management purposes, referred to as General Service Packets (GSPs), travel on any VL except 15 and can be sourced from any QP. A node's agents, however, are always found by sending a message to QP1, which is reserved for this function. Having been found there, they can be redirected to another QP, or even another node, by providing a redirection response to any GSP. (Hence QP0 and QP1 are the two special QPs for management mentioned earlier that must always be present.)

42.4 Selected Topics

Now that the general outline of IBA has been covered, a few topics of particular interest will be described further: addressing of elements on the network, packet formats used, the memory model used to access host memory, partitioning, reliable datagram, and virtual lanes and service levels. Those descriptions follow.

42.4.1 Addressing

Three types of quantities are important to IBA addressing: LIDs, GUIDs, and GIDs.

- LIDs, Local IDentifiers, are subnet-unique 16-bit identifiers used within a network by switches for routing.
- GUIDs, Global Unique IDs, are 64-bit EUI-64 IEEE-defined identifiers for elements in a subnet.
- GIDs, Global IDs, are 128-bit global identifiers used for routing across subnets.

LIDs are assigned by the Subnet Manager to each port on an endnode and to each switch (switch ports are not assigned LIDs). Destination and source LIDs (DLIDs and SLIDs) are carried by each packet in its local routing header. The space of LIDS is divided into:

- preassigned values: there is exactly one of these, the "permissive LID" accepted by any destination; it is used only as a destination in directed route management packets so endnodes can be probed by the subnet manager before LIDs are assigned.
- unicast values: approximately 48K entries in the entire LID range. These map to a single destination.
- multicast values: the remaining approximately 16K entries in the LID range. These map to multiple destinations; a port may be the target of zero, 1, or more multicast LIDs.

For additional throughput and high availability, multiple paths through a subnet to the same port are required. These are provided by a feature of each port called the LID Mask Count (LMC), which indicates how many low-order bits of a LID are "don't cares" in routing to that port; sequential values within an LMC range must be assigned by the subnet manager, which may then program the switch routing tables to cause each of the LIDs in an LMC-defined range to take a different path, or paths with different qualities such as MTU, to the destination port. The LMC can have a value from 0 to 3, accommodating up to 128 paths to a port. It is possible that some vendors may also use the LMC mechanism to provide addressing to different partitions (also called Dynamic Domains, or LPARs (Logical Partitions)) of a host or I/O device, but this use is not specified by the InfiniBand Architecture.

GUIDs are global scope IEEE EUI-64 identifiers assigned to a device, created by concatenating 24-bit company_id, assigned by the IEEE Registration Authority [4], to a 40-bit extension identifier. Companies assign GUIDs to chassis, channel adapter (CA), switch, CA port, and router port. The SM may assign additional local-scope EUI-64s to a CA port or a router port.

GIDs are 128-bit identifiers used to identify an endnode port, switch, or multicast group. They are valid IPv6 identifiers with restrictions (see chapter 4 of [1] for details). GIDs are independent of LIDs, so they are immune to subnet reconfiguration.

GIDs are constructed by prepending a 64-bit GID prefix onto a GUID, as shown in Figure 7 below. The GID prefix has a detailed format (see chapter 4 of [1]) including a 16-bit subnet prefix as its low order 16 bits.)

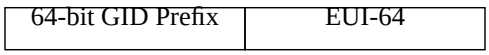


Figure 42.7: GID Format

Unicast GIDs can be created by:

[1]Concatenating of the default GID prefix (0xFE80::0) and an EUI-64 identifier. Packets with a destination using this form are never forwarded by a router, i.e. it has a local subnet scope.

2. Concatenating an SM assigned subnet prefix with the manufacturer's assigned EUI-64 identifier.

3. Assignment of a GID by the subnet manager. The subnet manager creates a GID by concatenating the subnet prefix with a set of locally assigned EUI-64 values (at GID infdex 1 or above). This allows for well-known assigned identifiers for services.

All CA and router ports and switches must be assigned at least one unicast GID using either of the first two methods. CA and router ports may also be assigned additional unicast GIDs using the third method.

Multicast GIDs are distinguished by having their eight initial bits be 11111111. Those are followed by a set of four flag bits, the first three of which are reserved and must be 0. The fourth bit equalling 0 indicates that this is a permanently assigned (e.g., well-known) IPv6 address, which must be constructed according to RFC 2373 and RFC 2375. If the fourth bit is 1, it indicates that this is a transient multicast address.

42.4.2 Packet Formats

Several examples of IBA packets are illustrated in Figure 8. There are packets that use IBA transport for local (intra-subnet) or for local (inter-subnet) traffic; and provision for "raw" packets that don't use the IBA transport layer for both of those cases. The raw format is intended to simplify processing in routers that do not target other IBA subnets, allowing hosts to directly provide them with packets already in the format needed. Which of those four formats is present in a packet is indicated by the Link Next Header (LNH) field in the initial Local Routing Header that is always present, as discussed below.

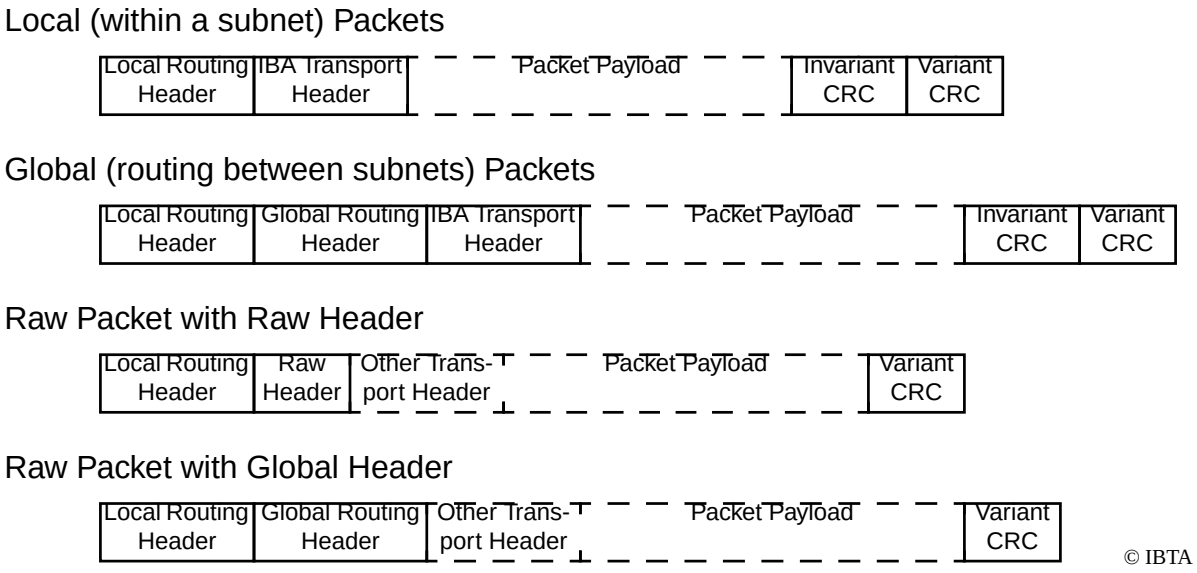


Figure 42.8: IBA Packet Format Examples

The complete format of IBA Packets is shown in Figure 9, with field sizes. Fields that are optional or may not be present are indicated by dashed lines (the Base Transport Header is not optional when IBA transport is used). The intimate details of all the bits in the headers can be found in the specification [1]. The general purpose and approximate contents of the fields (not all elements are described) are:

- Local Routing Header (LRH): This is used by switches to move a packet across a subnet to its destination endnode network port. It contains the source and destination LIDs, link protocol version ID, service level (see Section 42.4.6), virtual lane, packet length, and a "next header" field used to indicate which optional fields are present.

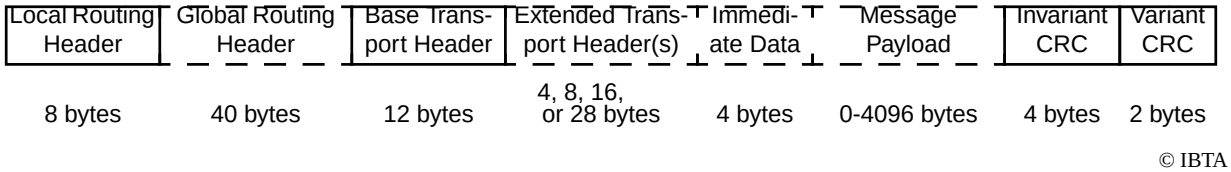


Figure 42.9: Complete IBA Packet Format

- Global Routing Header (GRH): This is used by routers to move packets between subnets. In addition to source and destination GIDs and global routing version ID, it contains payload length, IP version, traffic class, and hop limit. The LRH of a globally routed packet moves it to a router, which uses the GRH to determine where it goes; the destination router constructs a new LRH to move the packet to its destination port over the target subnet.
- Base Transport Header (BTH): This tells endnodes what to do with packets. In addition to yet another version ID, it has an opcode, destination Queue Pair, packet sequence number for reliable transports, partition key (see Section 42.4.4) and a number of other fields. Whether extended headers are present and what kind they are is indicated by the opcode.
- Extended Transport Header (ETH): This holds additional information depending on the BTH operation or LRH next header. For example, for RDMA operations it has a virtual address, length, and R_Key (see Section 42.4.3); for atomic operations, it has an opcode, virtual address, data to be swapped or compared, and an R_Key again; for ACKnowledgement packets it contains a syndrome and a sequence number; etc.
- Message Payload: This is the point of the whole exercise in most cases.
- Invariant CRC: This is a CRC over all the fields of the packet that cannot change in transit. It is not present in raw packets.
- Variant CRC: This is a CRC over all fields of the packet, including the Invariant CRC. It is present because, for example, the LRH must be replaced in globally routed packets.

42.4.3 Memory Model

Control of memory access by and through an HCA is provided by three primary objects: memory regions, memory windows, and protection domains. The relationship between regions, windows, and the underlying virtual memory system is illustrated in Figure 10 and explained below.

Memory regions provide the basic mapping required to operate with virtual addresses. A memory region is created by a verb that registers a segment of memory with the HCA. Doing so causes the operating system to provide the HCA with the virtual-to-physical mapping of that region and pin the memory (prohibit swapping it out in virtual memory operations). Registration creates an object called an *L_Key*, which must be used with each access to that memory region; it authenticates the use of that region by a

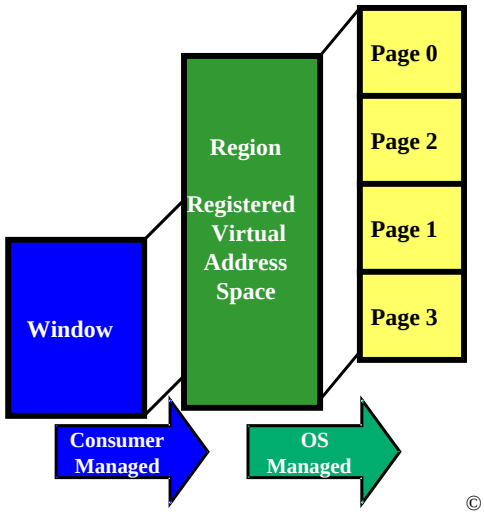


Figure 42.10: Window - Region - VM Relationship

QP, and specifies access rights. If remote access to memory by another endnode is desired for RDMA, a different object called an *R_Key* must be created and used; it must be sent to the endnode performing the memory access, and then passed back to the local endnode by the remote one as part of an RDMA request. It is an opaque object that indicates to the HCA which virtual address map to use, and authenticates the requestor.

Memory registration is necessarily a time-consuming operation; it is likely to be done just once for large segments of memory at a time, probably with a granularity of an operating system page (the actual granularity is vendor specific). Finer-grain and much lower overhead protection is provided by memory windows, so that individual operations can be restricted to refer only to the specific data they should access.

A memory window is created by a verb within an already-registered region; it specifies a contiguous virtual memory segment with byte granularity. The bounds of a window's segment of memory are re-registered using a work request. When that is done, subsequent work requests on that work queue must adhere to the memory bounds specified by the window.

Protection domains effectively glue QPs to memory regions and windows. They are opaque objects, but are expected to correspond roughly to operating system processes or groups of process which for example, might share a memory segment. A protection domain is created first, and then used when creating a QP, region or window. When applying or using a window or region, it the protection domain of the window (region) and the QP using it must match, or an error is reported.

42.4.4 Partitioning

An IBA system is intended to be used as a shared fabric to which multiple host systems and their I/O are attached. Ultimately, it is intended that an IBA fabric be able to be the only I/O fabric for all of those hosts. This poses a significant problem. To see it, first consider how clustered systems are typically constructed today.

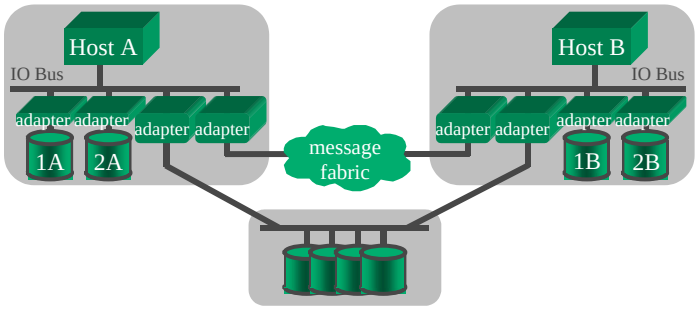


Figure 42.11: Typical Cluster Organization Today

This is shown in Figure 11. Typically there are multiple hosts, each with its own I/O bus, to which is attached adapters for private storage, communication adapters connecting the hosts through some fabric, and perhaps adapters that allow direct connection to shared storage devices (although many do not have that). The shared storage devices, if they exist, are known to be shared and are typically programmed in a special way within the hosts, using lock messages passed between the hosts across the fabric or other coordination.

Now, consider what would happen if you took the system of Figure 11 and made the change illustrated in Figure 12 below.

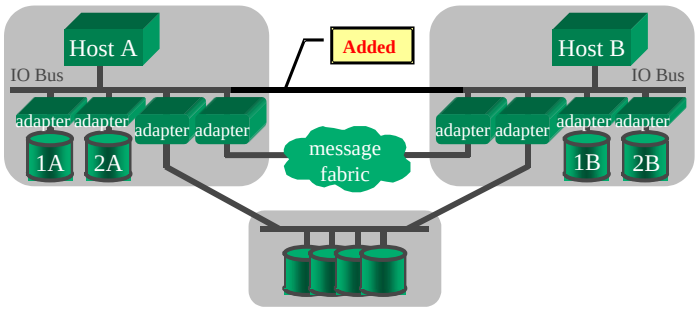


Figure 42.12: Mutilated Cluster System

Without massive changes to the operating system running on the hosts, and possibly to firmware also, merging the I/O busses into one will create a system that will not run. When the operating systems on each host do their usual “bus walk” to discover devices, they will both see all the adapters, including those that logically should be private to each of them—and each will think all the adapters are their own private property, accessing them without any synchronization with the other host. This simply does not work.

However, as presented so far, that is exactly the situation created when a single IBA subnet has more than one host attached. It is effectively what is shown in Figure 13, since InfiniBand is the I/O “bus.”

A mechanism must be put in place that gives host systems enforceable, private access to private devices, and allows shared resources to be controllably shared. That is partitioning. Applying it to the system of Figure 13 might, for example, produce a situation like that shown in Figure 14: There is a collection of endnodes that is private to A, another that is private to B, a third that is shared by both A and B (the shared disks), and finally, since the IBA fabric is also the communication medium between the hosts, a fourth that lets each host know the other exists so they can communicate. Host A should not even know that the devices private to Host B exist.

This is accomplished in IBA by the use of Partition Keys (P_Keys) and Partition Key Tables (P_Key Tables). Every endnode must have a P_Key Table. Furthermore, every QP

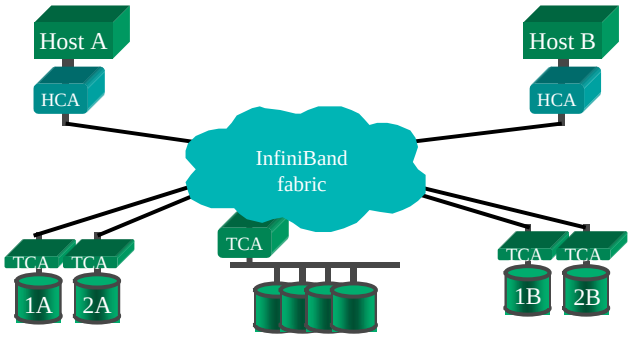


Figure 42.13: IBA Fabric as a Shared I/O Bus

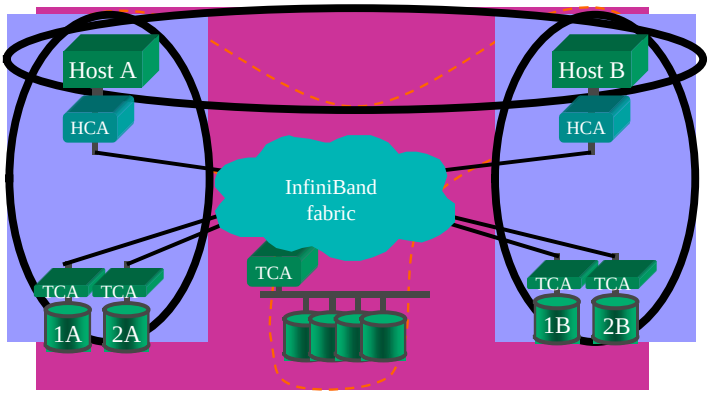


Figure 42.14: Partitions in an IBA Fabric

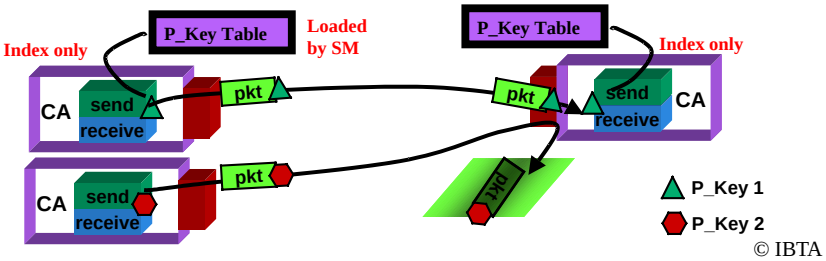


Figure 42.15: Partitioning Implementation

has associated with it an index into that table. Whenever a packet is sent from the QP's send queue, the indexed P_Key is attached; whenever a packet is received on the QP's receive queue, the indexed P_Key is compared with that of the incoming packet. If it does not match, the packet is silently discarded: The receiving CA does not know it arrived, and the sending CA gets no acknowledgement, negative or otherwise. From the sending side, it is as if it sent a packet to a nonexistent device; it was just lost. This is illustrated in Figure 15. One difference from literally sending a packet to a nonexistent device is that mismatches are recorded, and optionally a trap sent to the subnet manager indicating that a packet was sent somewhere it did not belong.

The security of partitioning is provided by the fact that there is deliberately no verb interface defined that allows a host to alter the contents of its own P_Key Table. All it can do is specify the index into the P_Key Table that a QP must use. A secure implementation of an IBA HCA will ensure that there is no hardware allowing the host to alter the P_Key Table, but that is up to the implementation.

The content of the P_Key table is instead loaded by SMPs: Management datagrams sourced from the master subnet manager. This loading is authenticated by a 64-bit

management key that all SMPs may[1] use, and must be matched at the recipient for the SMP to be processed. Thanks to this, and the inability of hosts to modify the P_Key table, the P_Keys themselves can be relative short (16 bits) and passed around from program to program in the clear, even in user mode. Setup of the partitions is a function of the subnet manager called partition management; it is done in response to customer-administrator input specifying how he/she wishes to configure his/her system.

IBA partitioning has an additional feature that is believed to be extremely useful: Partial partition membership. The high-order bit of each P_Key is reserved to indicate whether membership is full (0) or partial (1). If a partial membership P_Key arrives at a QP which itself has only partial membership, otherwise matching, the packet is rejected. Any combination other than partial-partial is accepted. What this allows is the creation of "server" resources (devices, hosts) in which the server is known, meaning in the partitions of, many clients; but all of the clients remain ignorant of each others' existent. To make that happen, the server has full membership, and the clients partial membership, in some designated server partition.

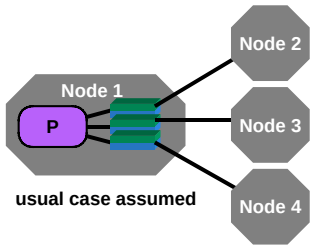
In practice, initial versions of InfiniBand will tend to confuse the situation with partitioning because, as mentioned earlier, they will implement HCAs as cards that plug into a standard I/O bus. Clusters using this will look exactly like Figure 11—a conventional cluster—perhaps with shared storage attached to the IBA fabric rather than a separate medium. Since the devices in this arrangement are, in effect, physically partitioned, little of IBA's partitioning function is necessary. However, as higher performance implementations that connect more directly to host memory systems are deployed, the partitioning functions described here will become a necessity.

42.4.5 Reliable Datagrams

Reliable datagrams are a solution to an otherwise difficult scaling problem that occurs when creating parallel programs that run across multiple systems, each with multiple processors.

Each of the N systems on which such a parallel program runs must communicate with every other one of those systems, and that communication must be reliable or else significant overhead must result from inducing reliability from the unreliable communication. This is usually conceptualized with a diagram such as Figure 16 below: Each system has N-1 QPs, one for communication with each other system. For IBA, this would typically be done with RC (Reliable Connected) service on each QP.

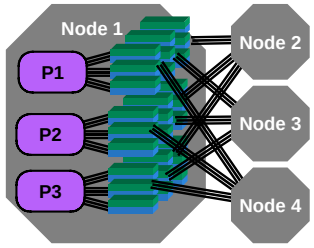
1.It is possible to turn off SM Key checking as an option.



© IBTA

Figure 42.16: Usual Parallel Communication Assumed

Unfortunately, if each of the endnodes is a multiprocessor, this actually does not represent the situation. Instead, each node hosts M processes. Communication within the node is usually accomplished by other means, such as shared memory. But across nodes, each of the M processes must communicate with each of the M processes on the other nodes. Thus each process on each node requires M*(N-1) queue pairs, as illustrated in Figure 17 below. Since each process requires that many QPs, and each node holds M processes, there are (N-1)*M^2 QPs required per node. This does not scale.



© IBTA

Figure 42.17: Reality of Communicating Among Multiprocessor Nodes

For example, parallel database systems are one important example of parallel programs of this type. For a typical, by no means largest, multiprocessor with 16 processors, a parallel database will typically have several hundred to a thousand processes on each node. With 1,000 processes per node, a mere four node system would require four million QPs per node. This is obviously an untenable result.

There are two possibilities that can be used to fix this, neither satisfactory:

- Create a separate communication process on each node, and multiplex all communication to and from that node through that process. This works, and is in fact what commercial databases often do today. Unfortunately, it involves significant overhead in the communication process, which must both send and receive data, perform scheduling of communication, and generally become fairly complex.
- Use an unreliable datagram transport. This allows just one QP to be used for each sending process, since the

same QP can target any QP in any node and thus target any process. Unfortunately, this is unreliable transport, and the overhead of inducing reliability may be even larger than the overhead of the communication process of the other alternative.

In the development of InfiniBand, something was realized: The problem isn't the QPs as such. The problem is that the *reliability context* is tied to the QP. The reliability context is the collection of state information that is used to provide a reliable connection, such as sequence counters, retry counters, etc.

The reliable datagram (RD) transport service breaks that connection. It instead places the reliability context in a separate entity, called the End-to-End context (EE context), and allocates one of those for each node (not process) being communicated with. Each RD message then specifies, along with its data and other possible parameters, an EE context (implicitly indicating the target node); and the target QP on the target node. When that message is processed by the CA, it uses the reliability context in the specified EE context, not in the QP as would be done for RC service.

The result is illustrated in Figure 18 below, which shows each QP effectively communicating through an EE context to any of the other nodes of the system. Hardware-provided reliability in communication is attained, and the cost is M QPs plus N EE contexts per node, which scales nicely with both the number of nodes and the number of processes.

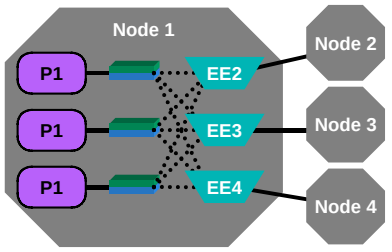


Figure 42.18: Use of Reliable Datagram

It is, however, the case that before beginning communication with any specific node, an EE context must be created for that node. This is a kind of initial connection setup—per node, not per QP—and so "reliable datagram" in some sense lacks the complete connectionless characteristics of the usual (unreliable) datagram service. It could justifiably have been called "multiconnected" service instead.

42.4.6 Virtual Lanes and Service Levels

IBA switches support a minimum of two, and a maximum of 16 virtual lanes (VLs). They are intended to be used for traffic prioritization, deadlock avoidance, and segregation of traffic classes. The required two are VL15, used exclusively for subnet management traffic; and VL0, used for normal data traffic.

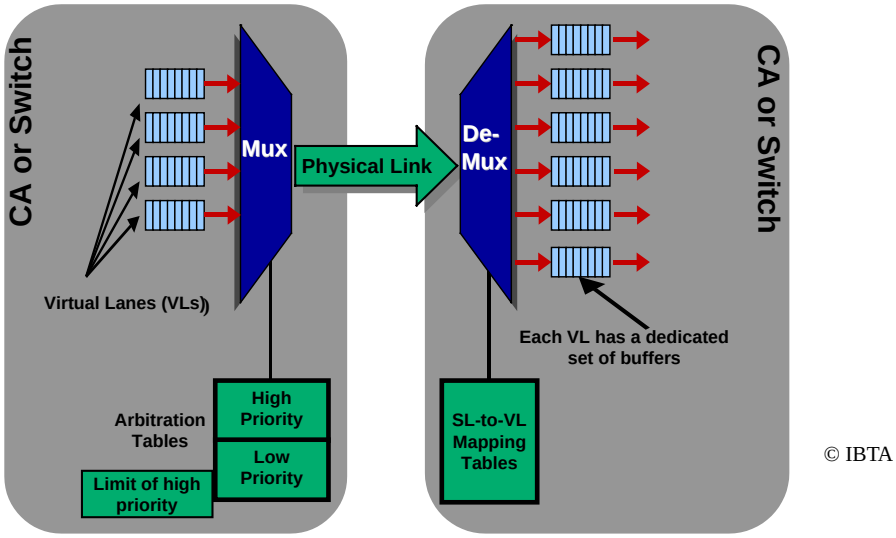


Figure 42.19: Operation of Virtual Lanes and SL-to-VL Mapping

Each virtual lane must be an independent resource for flow control purposes. When more than two lanes are implemented, the priorities of the data lanes are defined by VL arbitration tables, of which there are two: High priority, and low priority; a *Limit of High Priority* value specifies the maximum number of high priority packets that can be sent before a low priority packet is sent. The arbitration tables implement weighted round-robin arbitration within each priority level. Up to 64 table entries are cycled through, each specifying a VL and the number of packets (0 to 255) to be sent from that VL.

Since systems can be constructed with switches supporting different numbers of VLs, messages are actually marked with a more abstract Service Level (SL), and SL-to-VL mapping tables are used. This allows messages to be routed from a switch with many VLs, through one with few VLs, and back into another with many VLs without losing track of which VL should be used at any point.

SL-to-VL mapping, and arbitration of multiple data VLs, must also appear in the endnode Channel Adapters that connect to links.

The operation of VLs across a link is illustrated in Figure 19; note that the number of VLs in the receiving switch (or CA) is deliberately illustrated as larger than the number in the sender. The sender's VLs will, of course, determine the traffic pattern across the link.

42.4.7 Path Migration

If the levels of availability now required of servers are to be provided in IBA, methods of avoiding failures in the fabric must be provided. IBA provides two of these: Automatic Path Migration, and Send Queue Drain. Both are aimed at restoring or altering network operations without disrupting existing connections between QPs and, in the case of Send Queue Drain, EE contexts.

Automatic Path Migration is a fast hardware failover mechanism that can be used to bypass a large class of failures. It is optional. It provides for associating with a QP two independent paths to the destination, created when a connection is initially created with a QP on another endnode. QPs so initialized are initially placed in an "armed" state, in which packets flow across the first path normally. If the first path suffers errors and one side exhausts its retries, rather than producing an event indicating failure it switches to the other path. Packets sent on the other path contain a trigger bit in the BTH which, when received on the other side, cause it to also flip to the other path for sending its packets. When both sides have flipped, they are in a "migrated" state, and after repair can be changed by software to a "rearm" state to enable operation on the original path again.

Send Queue Drain is an enabler for software-driven recovery actions, and is applicable both to unplanned failures and to avoidance of planned outages. It is not optional. It allows software to put a QP into the "Send Queue Drain" state, in which it continues receiving packets normally, but after completing current send queue operations does not initiate any new ones. When both sides of a connection have drained their queues, meaning there are no possible acknowledgements left outstanding, the connection is quiescent and the state of the QPs is maintained statically. At this point, software can make modifications to the QPs and to the network, and when finished restart QP operation where it left off. This operation is particularly useful when changes need to be made that must in effect be atomic to QP operation, such as changing LID addresses of endnodes or changing P_Keys. Such operations may be necessary when, for example, it is necessary to join two subnets that have been operating independently and so have overlapping LID assignments.

42.5 Industry Implications and Conclusions

A question that is regularly asked about InfiniBand Architecture in general is "Why do we need yet another network architecture? Why not use Ethernet and IP?" These are, indeed, pervasive, familiar to an enormous number of people, and generally incumbent. The extension of Ethernet to 10Gbit/second speeds satisfies nearly all requirements, and with developments like Network-Attached storage and SCSI over IP (iSCSI), it is being extended into device attachment areas that appear to compete directly with IBA. These technologies are here now, being developed, and will be developed further in the years it will take InfiniBand to build its market presence.

This is an issue that will ultimately be answered in the market, and indeed it cannot be said that InfiniBand Architecture will be the "winner" with 100% certainty within its domain of server communication; certainly Ethernet and IP will continue to dominate areas that IBA does not address, such as connection to clients. However, there are two reasons why Ethernet and IP very well may not adequately address server I/O and communication requirements.

The first is IP software overhead. Serious commercial and technical server I/O cannot afford the software overhead associated with IP stacks. To be successful in this area, the majority of the processing will have to be offloaded. This is difficult. Partial offload of IP processing, such as checksum generation and gathering of data, has been done successfully for quite a while, but near complete offload has, while often attempted, never yet been commercially successful. Rather than shuffling the processing to another place, IBA eliminates it. Furthermore, to compete with IBA efficiency, the offloading performed must, in effect, re-invent some of the capabilities of IBA like such as zero-copy data transmission and user mode operation. This is more than just offloading processing. In addition, if IBA is the I/O system (as seems likely), connection to Ethernet NICs must be through IBA. It's obviously impossible for that to be more efficient than just using IBA as the native connection.

The second is simple presence. If IBA comes out of all the systems of interest in a native fashion, as the collection of steering committee and sponsor members of IBTA would suggest; and IB switches reach the cost per port of comparable function Ethernet switches (a volume issue

more than anything); then there is simply no reason not to use IBA. It is already there, with nothing else required to use it.

Assuming that IBA is successful, there are three very important things it will provide to the industry:

1. It will be a standard, high-volume enterprise-class server fabric, providing the reliability, availability, serviceability, manageability, performance, and scalability that implies. Such capabilities have heretofore only been available in proprietary systems. The market in this area will deepen and broaden, with significantly better facilities available to many customers at lower prices.
2. It will, for the first time, provide non-proprietary low-overhead inter-host communication. (VI Architecture, since it lacks hardware standardization, does only half that job.) This will enable functions in "open" systems that are now available only on proprietary systems such as Compaq (Tandem) Himalaya and IBM Parallel Sysplex. This will result in new cluster multi-tier server solutions/markets that have previously been impossible, such as high-function storage subsystems at commodity storage prices.
3. It will allow complete separation of I/O devices from processing elements. This, along with the standardized backplane wiring of IBA, will enable new form factors with much higher density of packaging than is now possible. It will also encourage new ways of looking at systems as a whole, e.g., data-centric views where the processing is considered peripheral to the data, which is central.

Separately, each of those three elements would be a very significant factor that could change the landscape of computing in the large. Together, they may well presage the widespread adoption of new, previously untried hardware and software structures for server computing.

References

[1] *InfiniBand Architecture Specification Volume 1, Release 1.0,* October 24, 2000, available from the InfiniBand Trade Association, http://www.infinibandta.org

[2] *InfiniBand Architecture Specification Volume 2, Release 1.0,* October 24, 2000, available from the InfiniBand Trade Association, http://www.infinibandta.org

[3] *Virtual Interface Architecture Specification Version 1.0,* Dec. 16, 1997, available from http://www.viarch.org/

[4] *Guidelines For 64-bit Global Identifier (EUI-64) Registration Authority,* available at http://www.standards.ieee.org/regauth/oui/tutorials/EUI64.html