Analysis of an Infiniband Channel Adapter

Philip Buonadonna

Compaq Computer Corporation, Cupertino, CA[1]

Abstract

The Infiniband Architecture is a work-in-progress design specification for a high performance network intended to become the next generation interconnect for I/O. This paper presents an implementation and analysis of a prototype Infiniband channel adapter for the Myrinet System Area Network. We discuss the inherent costs and performance potential of the Infiniband Architecture in terms of both the upper level verbs (i.e. API) and lower level packet formats. We also discuss the implications of channel adapter design requirements for I/O applications.

1. Introduction

In order to enable the widespread deployment of high performance, scalable systems, there has been a concerted effort to develop a standardized cluster communication architecture for system area networks (SAN). This effort yielded the Virtual Interface (VI) Architecture [5] in 1998, and is now focused on the emerging Infiniband [2] architecture which also seeks to encompass network based I/O. The Infiniband architecture specification defines an interconnect technology to link processing nodes and I/O devices over a high-speed switched network fabric. At its core is a set of design principles for how to implement operating system independent (i.e. user-level) communication in a manner that virtualizes resources among an arbitrary number of processes. It outlines both a low-level link format and a high-level software interface, called verbs, upon which communication abstractions are implemented. Infiniband incorporates much of the VI Architecture, but is larger in scope, and represents the intellectual merger of many industry efforts in high performance networked I/O. While it does introduce some new concepts and components, its core is strongly based on the VI Architecture primitives.

In Infiniband, network elements attach to the fabric through a channel adapter (CA). The channel adapter is a "device that terminates an Infiniband link and executes transport-level functions." There are two categories of channel adapter types, host and target, that are used in processing nodes and I/O devices, respectively. The specification suggests that the target channel adapter support a subset of the functionality of its host counterpart. However, the exact design is left to the implementer.

[1] Research sponsored by Compaq Computer Corp. The information presented here does not necessarily reflect the position of Compaq Computer Corp. and no official endorsement should be inferred.

This paper investigates two related aspects of the Infiniband Architecture. The first is what are the inherent performance characteristics of an Infiniband channel adapter; both in terms of its upper level application interface and lower level packet definitions. Second is, how do these performance traits coupled with distributed I/O architectures affect the design traits of channel adapters. In the next section, we briefly discuss the Infiniband architecture and the prototype implementation on Myrinet. Sections 3 & 4 presents a performance analysis and a retrospective of lesson learned from the prototype. Section 5 discusses the impact of distributed I/O architectures on channel adapter design.

2. Infiniband Prototype

Infiniband is the logical merger of several industry efforts (i.e., Next Generation I/O and Future I/O) in network based I/O architectures. Here, the I/O devices are effectively separated from the host CPU(s) by a switched network fabric (Figure 1). Different classes of devices connect to the network through one of two types of interfaces called channel adapters. The host channel adapter (HCA) is used to connect processing nodes to an Infiniband network. A principal characteristic of an HCA is that it exports Infiniband 'verbs': a collection of methods with which applications conduct communication transactions. The target channel adapter (TCA) is the network interface for the individual I/O devices (e.g. disks and WAN adapters). The TCA is similar to the HCA, but can be simplified according to the requirements of the attached device(s). It need not export verbs and may include specialized hardware as required by the I/O device.

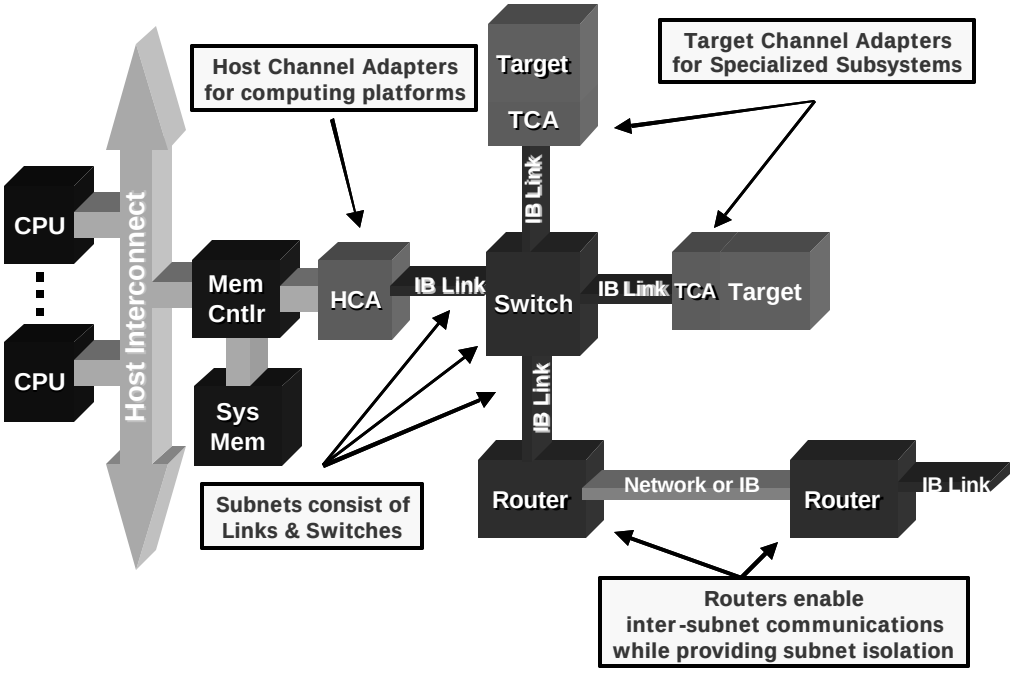

Figure 1: Infiniband architecture overview.

The fundamental transport abstraction supported by the HCA/TCA is the queue pair (QP). Each QP consists of two queues: send and receive. Each queue contains a FIFO list of work requests that describe a communication transaction to take place. Work requests are the Infiniband equivalent of VI architecture descriptors, but the queuing mechanisms are not expose. Data exchange between QPs is sourced/sinked to registered memory regions established by the application. Infiniband provides packet and message level flow control schemes based on receive credits and NAK's. To provide differentiated service and robust network management, data traffic is multiplexed onto multiple independent streams called Virtual Lanes (VLs). Infiniband supports 16 VLs -- 15 for data and one for management functions.

The Infiniband prototype in this study was developed on the Myrinet system area network with the M3M programmable network interface. The programmable nature of this system allowed rapid prototyping on a flexible, instrumentable system. The Myrinet network consists of 1.2 Gbs full-duplex links arranged in an arbitrary topology. The switches are full-crossbar with some amount of internal buffering. Packets are forwarded through the network using source-based oblivious cut-through routing. The network interface is a 33/66 MHz, 64-Bit PCI card with a general-purpose processor, 2MB of SRAM and PCI-DMA bridge (Figure 2). The processor is a 134 MHz 32-bit LANai-9 RISC processor with a 4-stage instruction pipeline and no instruction or data caches. The absence of caching is offset by an aggressive instruction pre-fetch policy with priority for not-taken branches. The local memory is 64-bit, pipelined, zero-bus turnaround SRAM. There are 6 DMA engines on the NIC: 2 network (transmit and receive) and 4 host. The DMA engines, processor and SRAM are linked with an 64-bit proprietary bus. Additionally, there is direct hardware support for a 'doorbell' mechanism. Values written to a region of mapped PCI space are directly written to a FIFO queue in local memory.

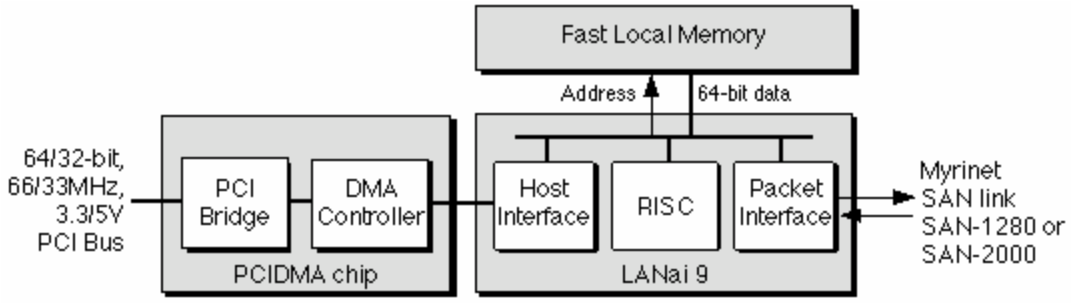

Figure 2: The Myrinet programmable network interface.

The prototype work started with a set of software and firmware that emulated a VI Architecture interface. Features of this VI emulation included:

- 1024 VI's, VI descriptor processing, Registered Memory, Host DRAM based Completion Queues
- Unreliable, Reliable Delivery and Reliable Reception modes
- Send-Receive, RDMA Write and RDMA Read messaging primitives

- Round-robin VI and internal message queue transmit scheduling. Incoming packets are always given scheduling priority.

The internal message queue is used to generate acknowledgements and responses to RDMA read requests. It is scheduled with the same priority as a VI transmission.

The network interface firmware implements two separate message pipelines: transmit and receive. Figure 3 illustrates the operation of the pipelines as they would appear for a send/receive descriptor. The pipeline behaves similarly for RDMA read responses and acknowledgements, but the descriptor download and data download stages are combined. Each pipeline has two staging buffers (Tx-1/2 and Rx-1/2) which permit the simultaneous operation of the host and network DMA engines. The scheduling gap in the transmit pipeline is variable and depends on the amount of incoming network traffic. Since the onboard processor can begin only one pipe at a time, the transmit pipe may stall until the network is drained.

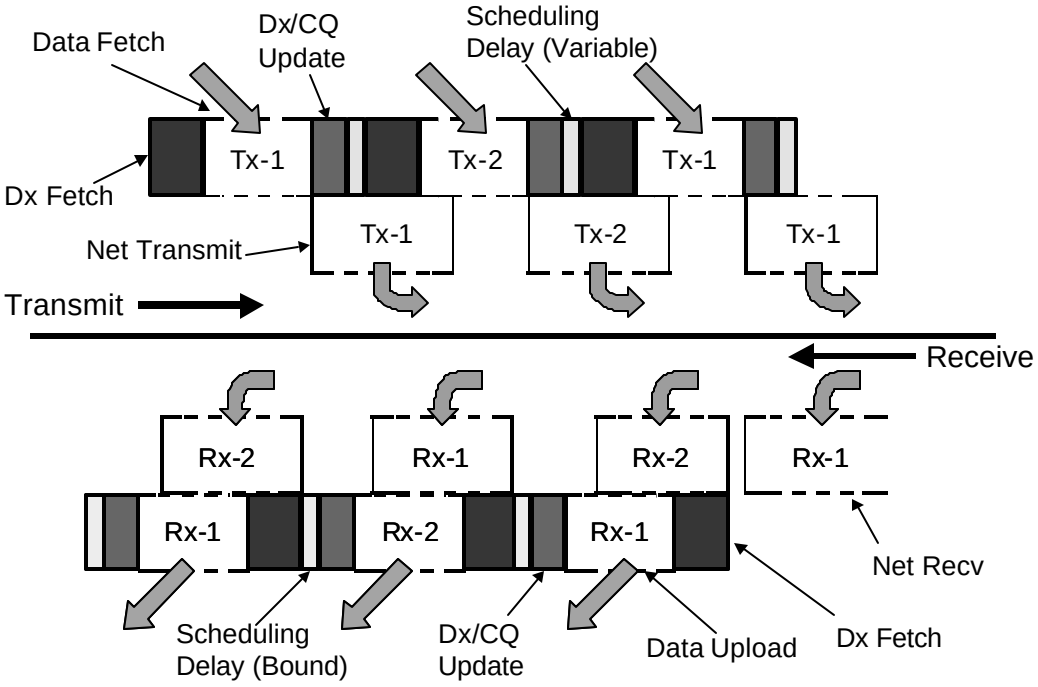

Figure 3: VI/Infiniband message pipelines implemented by the network interface firmware. (Dx = Descriptor, CQ = Completion Queue)

Implementation of the Infiniband prototype was accomplished in essentially two stages. The first stage of the build involved modifying the firmware to support the Infiniband packet format. The prototype supports most of the components except global routing, data-gram and atomic headers. Other exceptions include: lack of the variant CRC, inclusion of the VL identifier in computing the invariant CRC and the addition of Myrinet specific routing flits at the head of the packet. Additionally, the firmware only supports a single virtual lane per connection.

The second stage added Infiniband software verbs to the prototype. The verbs layer supports reliable and unreliable QP connections and the message operations allowed on each. While the verbs prototype implements memory registration, it does not support memory windows or protection domains. Connections between QPs are handled through a separate mechanism available in the software driver. Due to the similarity of the Infiniband and VI Architecture primitives, no changes were made to the existing NIC firmware. The software methods implement Infiniband verbs over the VI-like operations supported by the NIC. For example, invoking the *post send* verb with a work request creates a VI send descriptor and an associated doorbell operation. The *poll for completion* verb monitors a VI completion queue for entries. Completed descriptors are translated to Infiniband work completions that are handed back to the application.

3. Performance Analysis

The Infiniband prototype was evaluated using round-trip time (RTT) benchmarks. The benchmarks measured the RTT for unreliable send-receive, reliable send-receive and RDMA read communication modes. Details of these benchmarks are illustrated in Figures 4 & 5.

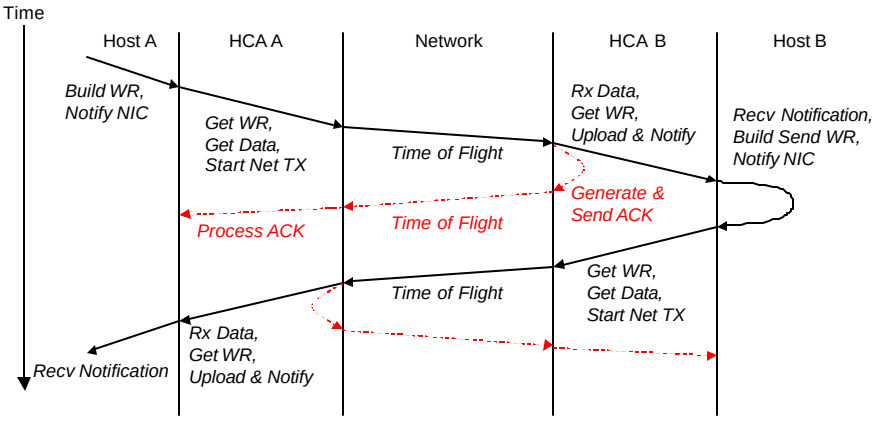

Figure 5: RTT benchmark for unreliable and reliable send-receive. The dotted line indicate the ACK path for a one-way message.

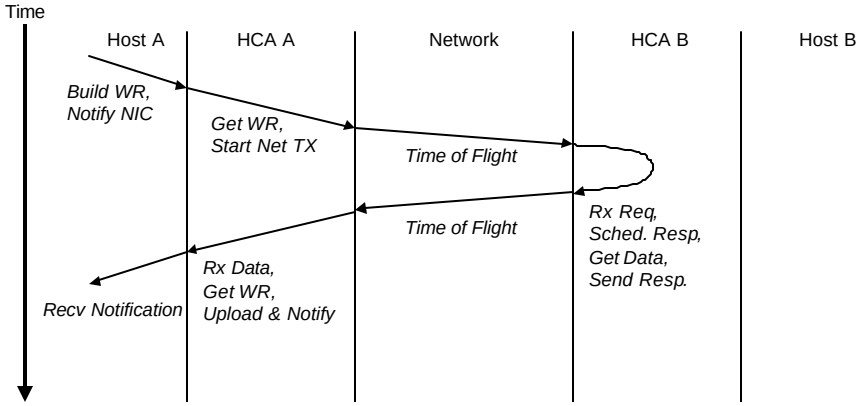

Figure 4: RTT benchmark using RDMA Read.

To understand the performance characteristics of various Infiniband components, the benchmark set was run for three different scenarios. The first executed the benchmarks on the original VI implementation. The second was executed on the first stage prototype with the standard VIPL on top but used the Infiniband packet formats. The intent here was to understand any inherent costs resulting from using the specified formats. The final scenario involved the final prototype which included the verbs layer. All measurements were taken on a pair of Compaq SP700s, each with a 550 MHz Pentium III processor and a 33 MHz, 32-Bit PCI bus. The message size was fixed at 4 bytes. The results are presented in Figure 6.

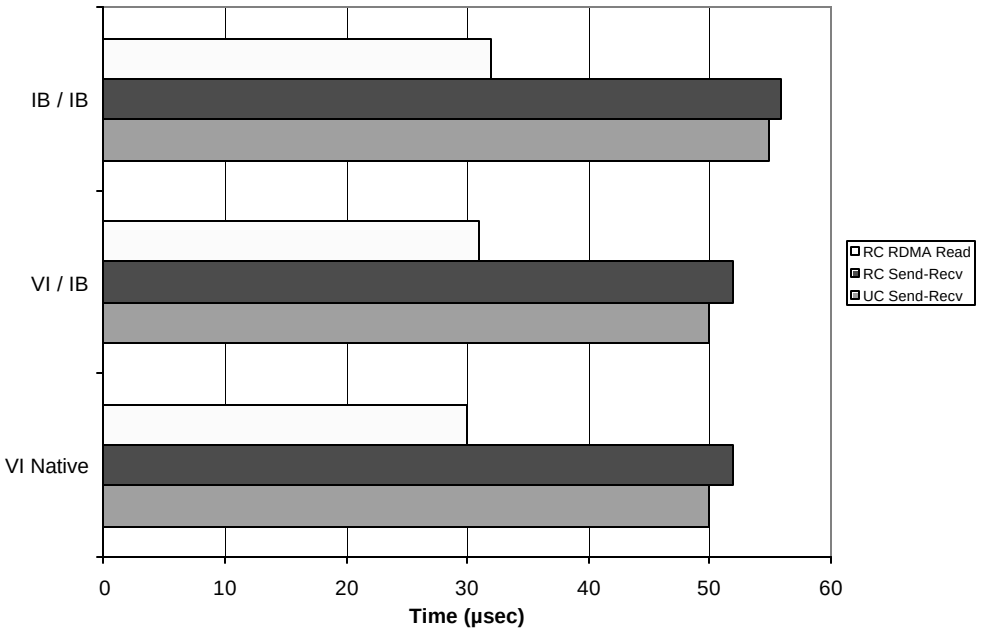

Figure 6: RTT Benchmark results for 4-byte messages. 'VI Native' refers to a native VI implementation on Myrinet. 'VI / IB' refers to the VI implementation using Infiniband packet formats. 'IB / IB' refers to the final prototype with an Infiniband verbs layer.

Both the VI native and VI over Infiniband packet formats exhibit 50,52 and 30 µsec RTT for the unreliable send-receive, reliable send-receive and RDMA read modes respectively. This suggests that the Infiniband formats do not incur excessive penalty over a native format.

The full Infiniband prototype, which includes the verbs component, adds 4-5 µsec to the send-receive RTT and 2 µsec to the RDMA read case. There are several reasons for this increase performance cost. First, Infiniband requires the use of completion queues; there is no notion of polling an individual QP as there is in the VI Architecture. This introduces a layer of indirection that consumes processor cycles. In the VI Architecture, this indirection was explicit in that the application would invoke one method to wait on a completion queue (i.e. `VipCQDone`) and then de-queue the descriptor with another method (i.e. `VipSendDone` or `VipRecvDone`). In Infiniband, this is handled by a single verb. This is not to imply that the completion queue concept is wrong. From a software engineering viewpoint, it is a scalable means of monitoring several I/O operations at once. Given this, the completion mechanisms must be carefully engineered and streamlined to ensure minimal impact on the host CPU.

Another source of performance loss occurs from manipulating Infiniband work requests. The work request format used for this implementation was derived directly from the

specification and has a minimum size of 108 bytes, not including data segments. This compares to 64 bytes for a VI descriptor that includes 2 data segments. While an application might not have to touch every field of a work request for every network transaction, the verbs layer necessarily must process/queue the arguments to the NIC. The size of the request might result in multiple cache misses and larger I/O bus transactions to transfer data to the NIC. While some optimizations could be made in the prototype firmware to make it Infiniband aware, the author conjectures that the work request size will always incur a host CPU cost.

4. Prototype Retrospectives

This section presents an overall discussion of lessons learned from the prototype. Emphasis is placed on design points that might be incorporated in production systems.

Small Messages. One of the many things that Infiniband inherits from its VI predecessor is a lack of small message support. Sending an 8-byte value requires building a work request more than 13 times the message size. Work Requests can include a 32-bit immediate data value, but there are two issues in using it. First, it is not clear that 4-bytes is sufficient to do a wide range of meaningful operations. The author suggests that its width should be at least the precision of an address value (i.e. 8 bytes). This would permit a wide range of operations such as passing address space pointers. The send issue lies in the fact that the immediate value is considered a special case, thus requiring extra processing by both the software verbs and the NIC. For send-receive message transactions, the value could always be considered valid; whatever the sender inserts into the data field is reflected in the receivers work completion. However, the 'always valid' method may pose problems for RDMA writes which only optionally consume a receive work request if the immediate is present. There are arguments that suggest the RDMA write with an optional immediate is best for signaling remote I/O completions. It is not clear, however, that this outweighs potential performance gains and that there doesn't exist alternative means.

Hardware/Software Boundary. As I/O devices become more complex, the addition of a general-purpose processor opens a wide space of design options. The question becomes where to draw the hardware/software support boundary. The fully programmable nature of the Myrinet network allows a great deal of flexibility and agility for development and testing. However, involving the processor in all aspects of packet processing limits the overall capabilities of the interface. Previous work illustrated the importance of hardware doorbell support for VI NICs [1] and the addition of such mechanisms from earlier Myrinet products improved performance. For Infiniband, parsing the packet formats and interpreting bit-field options could be handled by logic optimized for doing so. Higher-level functions such as the QP transmit scheduler, error handling or management functions could be implemented in software. This would simplify development and permit future changes and/or upgrades.

Channel Adapter Integration. Future implementations of Infiniband channel adapters might be connected directly to the system memory bus or even built into a processor. Such integration yields new design issues over I/O bus interfaces. At first, performance might be expected to improve as the network is moved closer to the data. However, adverse effects could occur if care is not taken. From this study, one feature of a memory bus-based channel adapter that would be important is cache coherence. Previous efforts with coherent network interfaces that enable I/O to be cached illustrate performance gains by allowing direct reads and writes of registers to be cached [3]. For an Infiniband adapter, allowing completion queue state to be cached could significantly reduce wait costs for the processor. An earlier version of the Infiniband prototype used NIC based completion queues that were monitored through un-cached PIO reads. Moving the queues into the host DRAM allowed the CPU to spin on an empty completion queue in its cache. Completion updates would invoke the coherence mechanisms of the memory system to notify the processor. The impact of host-based queues was an immediate 5-6 µsec improvement in completion queue performance. As processors move into multi gigahertz speeds, touching the memory bus becomes a more expensive operation, thus emphasizing the importance of caching.

5. TCA Architecture

The Infiniband specification vaguely defines the TCA as an interface for I/O devices that need not support a verbs software layer. Present efforts view the TCA as a scaled-down version of an HCA: fewer QP resources (tens instead of tens of thousands) and a limited verbs implementation. However, the actual design of a TCA will depend heavily on the overall distributed I/O architecture. Figure 7 illustrates the components of a basic I/O stack. Network based I/O effectively breaks this stack at one (or more) layers. This section discusses TCA requirements with respect to where the stack is split.

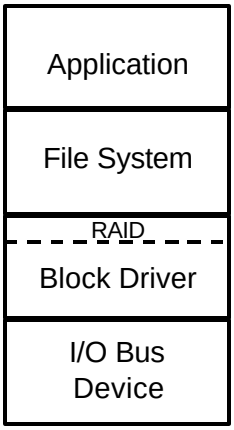

Figure 7: A simplified I/O Stack

Application & File System Boundary. Separating the application node from the file-systems leads to a TCA that directly exports or helps support a user-mode file abstraction. An example is the Direct Access File System (DAFS) protocol. Here, applications on the same or different nodes may want to access the same remote storage facility. This would require the TCA to support many QPs and or end-to-end contexts (EEC) to support several connections. Additionally, it is conceivable that the I/O device would be ‘intelligent’, i.e. it would have some form of onboard computing ability. Here,

the addition of a verbs layer to the TCA would prevent having to reinvent a new interface between the I/O processor and the network.

File System & Block Device Boundary. Another approach to network I/O is to abstract the remote device as a logical block server. A file system resident on one or more servers connects to block server which may consist of a single disk or a RAID-like subsystem. The Network Attached Secure Disk (NASD) effort implements this model of storage. A TCA supporting a block server would also have to support several connections to multiple hosts and/or applications, thus requiring support for as many QPs or EECs as there are hosts in the storage network. The need for a verbs interface on this form of TCA is not clear. A programmable I/O device could make use of a verbs layer, but the device may be simple enough to have the TCA implement all necessary functionality.

Block Driver & I/O Bus. Splitting the I/O stack at or below the device driver boundary effectively creates a remote bus with a message passing network in between. One application of this is a remote PCI bus. PCI operations are bridged by an HCA and a remote TCA. The advantage here is that bus-based legacy I/O devices and drivers can continue to be used without modification. In this I/O architecture, there would only be one or two hosts accessing the remote device, thus only a few QPs need be supported. Devices would interact using PCI semantics and would probably not require a verbs interface. However, the TCA and HCA would both require some specialized hardware support to convert PCI commands to and transmit then over the network.

Across all three general I/O architectures, there are some common requirements for the TCA:

RDMA. Previous work suggested the advantages of RDMA read and write in network attached storage devices [4]. RDMA enables an I/O device to schedule data transfers, providing a level of implicit flow control and minimizing the amount of on-device buffering. For remote file system or block devices, RDMA allows an I/O device to make use of optimized disk scheduling techniques. In the remote I/O bus case, RDMA semantically parallels the DMA operations of the bus.

Address Translation. Another common requirement might be the need for an address translation mechanism. On an HCA, a method of translating user-addresses to physical addresses is used when processing work requests. A TCA might use a similar mechanism to translate file system block requests to logical disk block requests. Alternatively, a remote I/O bus scenario might use a translation mechanism between I/O Virtual addresses and physical I/O addresses.

Single User Image. Unlike an HCA, a TCA would not need to support multiple, un-trusted applications running on the I/O device. Thus, strong protection mechanisms and multi-user memory registration would not be needed. Note, though, that the TCA would still need mechanisms to authenticate remote users and protect against malformed remote requests.

As network I/O develops it is conceivable that the TCA is equivalent to an HCA. Instead of "computers talking to disks", the model becomes "computers talking to computers" and the message protocols are semantically equivalent to IPC. Such communication is not limited to between processing nodes and devices. Intelligent devices would communicate with each other for availability or performance reasons. Such a shift would require a more generalized channel adapter with verbs support for onboard processors and the ability to established several connections (QPs).

6. Conclusion

The network-based I/O concept in Infiniband represents a significant architectural revolution for today's systems. This paper has detailed an implementation and analysis of an Infiniband prototype for the Myrinet SAN. The results provide proof-of-concept of Infiniband semantics and initial performance results. The results are not intended to be absolute, but rather insight into inherent performance costs and advantages of the Infiniband architecture. Also presented was a comparison of TCA design requirement vs. distributed I/O architectures. By approaching the problem from stacked I/O architecture model, questions such as number of QPs and need for a verbs interface were discussed. It is, perhaps, in the realm of distributed I/O that much of the Infiniband work remains. Although Infiniband provides fundamental communication primitives, these are of little use if their integration into a scalable distributed I/O model is not understood.

References

[1] Buonadonna, P., Geweke A., and Culler, D.E., "An Implementation and Analysis of the Virtual Interface Architecture", SC '98, November 1998

[2] Infiniband Trade Association, "Infiniband Architecture Specification, Vol 1", Infiniband Trade Association, March 2000, http://www.infinibandta.org

[3] Mukherjee, S.S., Falsafi, B., Hill M.D., and Wood, D.A., "Coherent Network Interfaces for Fine-Grain Communication", 23rd ISCA, May 1996

[4] Nagle, D.F., Ganger, G.R., Butler, J., Goodson, and G., Sabol, C. "Network Support for Network-Attached Storage", Hot Interconnects 1999, August 1999

[5] "Virtual Interface Architecture Specification. Version 1.0", Dec 16 1997, http://www.viarch.org