High-Performance Ethernet-Based Communications for Future Multi-Core Processors

Michael Schlansker[†], Nagabhushan Chitlur[*], Erwin Oertli[#], Paul M. Stillwell Jr[*], Linda Rankin[*], Dennis Bradford[*], Richard J. Carter, Jayaram Mudigonda[†], Nathan Binkert[†], Norman P. Jouppi[†]

[†] Hewlett-Packard Labs/Advanced Architecture Lab, [*] Intel Corporation/Corporate Technology Group, [#] VMware

{mike.schlansker, jayaram.mudigonda, binkert, norm.jouppi}@hp.com,

{bhushan.chitlur, paul.m.stillwell.jr, linda.rankin, dennis.bradford}@intel.com, eoertli@vmware.com

ABSTRACT

Data centers and HPC clusters often incorporate specialized networking fabrics to satisfy system requirements. However, Ethernet's low cost and high performance are causing a shift from specialized fabrics toward standard Ethernet. Although Ethernet's low-level performance approaches that of specialized fabrics, the features that these fabrics provide such as reliable in-order delivery and flow control are implemented, in the case of Ethernet, by endpoint hardware and software. Unfortunately, current Ethernet endpoints are either slow (commodity NICs with generic TCP/IP stacks) or costly (offload engines). To address these issues, the JNIC project developed a novel Ethernet endpoint. JNIC's hardware and software were specifically designed for the requirements of high-performance communications within future data-centers and compute clusters. The architecture combines capabilities already seen in advanced network architectures with new innovations to create a comprehensive solution for scalable and high-performance Ethernet. We envision a JNIC architecture that is suitable for most in-data-center communication needs.

1. INTRODUCTION

Ethernet is nearly ubiquitous in today's data center. Its commodity pricing and high performance is a strong motivator for replacing specialized fabrics such as InfiniBand [9], Quadrics [14], Myrinet [3], and Fibre Channel [4]. Gigabit Ethernet (GigE) is used in over 40% of the top 500 supercomputers as of November 2006. As low-latency 10 Gigabit Ethernet solutions become widely available, Ethernet will dominate in-data-center communications. However, Ethernet's simple architecture requires complex computation in the endpoints to deal with its unreliable communications, out-of-order delivery, and lack of useful flow control. These limitations are typically addressed by the TCP/IP protocol suite. Current implementations of these protocols do not satisfy the needs of high-performance scalable in-data-center communications.

SC07 November 10-16, 2007, Reno, Nevada, USA

The Joint Network Interface Controller (JNIC) project is a collaborative research project between HP and Intel to explore high-performance in-data-center communications over Ethernet. We are investigating advances required to allow Ethernet to become the unifying fabric for high-performance in-data-center communications. This requires increased Ethernet performance, as delivered to the user, while reducing the endpoint costs. Future Ethernet-based data center fabrics can leverage the commodity pricing for all components including endpoints, switches and cabling. The project demonstrates that Network Interface Controller (NIC) hardware closely coupled into the CPU/memory complex can be combined with on-load software running on (i.e. on-loaded to) a multi-core CPU to achieve these goals.

The primary contribution of this work is the design, implementation, and evaluation of a network architecture for Ethernet-based communications in future data centers. The hardware and software architecture is designed with a number of key assumptions. Hardware must be simple for close integration in future multi-core processors. Inexpensive hardware minimizes cost for users with modest communication needs. The architecture uses front-side bus attachment to provide low latency, high bandwidth and efficient hardware/software interaction. The architecture must allow efficient scalable communication within very large data centers.

Complex control must be implemented using on-load software that exploits general-purpose processors and memory. Additional general-purpose resources are used to provide increased performance and more complex communication services. Functions such as congestion management, quality of service, and a variety of complex client communication services are incorporated as flexible on-load software to allow the incorporation of new functionality.

While most of JNIC's components are similar to components that have been explored within prior work, we believe that JNIC provides a unique and innovative system architecture that improves our understanding of communications architectures for future data centers

To understand future system performance, a hardware and software prototype was developed instead of using simulation. This carries benefits and limitations. We integrated and tested all components within a fully-functional system and our complex prototype demonstrates many real-system behaviors. A key limitation of our prototype is modest performance. When work began, we were aware that our 1 Gb Ethernet prototype

could not match the performance of existing products. Instead, our goals were to use this platform to identify sound architectures and to extrapolate realistic performance, functionality, and cost goals for future systems. The JNIC prototype serves this purpose well. Any future product would incorporate faster processors and network hardware as well as better-tuned and more complete software. With our existing prototype, we extrapolate across these limitations into the future.

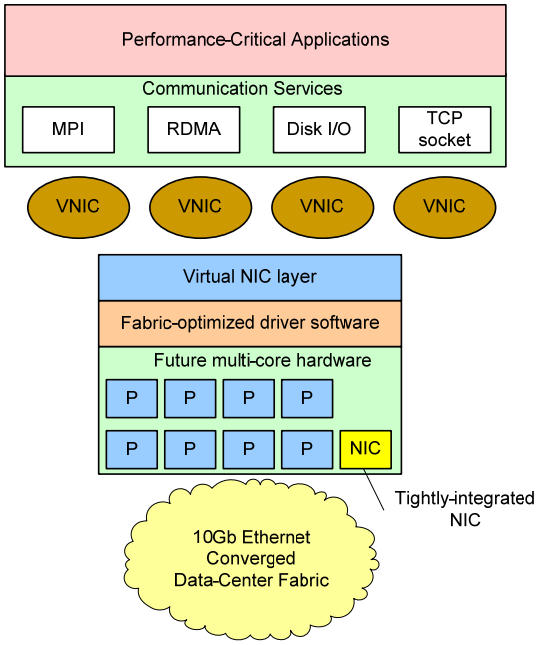

Figure 1: Future JNIC Systems—Future JNIC systems will use multi-core processors, integrated on-chip NIC, and on-load software.

The high-level, forward-looking vision of the JNIC system is depicted in Figure 1. At the bottom, we show a 10 Gigabit (or faster) Ethernet-based converged data center fabric. To communicate with that fabric, a multi-core processor incorporates tightly-integrated NIC hardware, with most of the complex communications functionality implemented in software running on the multi-core processor. This on-load software is fabric-optimized to manage congestion and ensure the efficient yet reliable transport of data. A software virtual NIC (VNIC) service layer offers virtualized NIC devices that are efficiently accessed by user- and kernel-mode clients with performance-critical applications layering a variety of communications services on the VNICs.

To achieve this vision, JNIC's architecture combines a number of key architectural elements. Coherent-attached NIC hardware maps transmit and receive apertures into a multi-processor address space as described in Section 3. JNIC's on-load helper software exploits the power of future chip multiprocessors to provide reliable data transport and virtualized NIC services as described in Section 4. JNIC Bulk Message Transport provides copy-free autonomous data delivery as described in Section 5. MPI is used to demonstrate JNIC as described in Section 6.

A central component of our prototype software is a driver that provides reliable and low latency system-to-system communication. This software runs in a helper task on a processor dedicated to communications processing. While ensuring reliability, this low-level software also regulates the messaging rate between peers to manage congestion. The Virtual NIC (VNIC) layer, which sits on top of this reliable communications substrate, presents a reliable datagram service to kernel and user clients. The device driver multiplexes message requests from the various VNIC clients to the reliable communications layer. Another component of our prototype software, the JNIC Bulk Message Transport (JBMT) layer, implements a remote DMA facility to support large message transfers. JBMT uses VNICs to implement efficient, copy-free, high-bandwidth data transport. MPI [16] has been ported to the JNIC testbed to demonstrate high performance short and long message transfers. We believe that other communication services, including remote disk access, can be efficiently supported by the VNIC and JBMT layers.

2. THE JNIC PROTOTYPE SYSTEM

We have developed a functional prototype (Figure 2) consisting of optimized NIC hardware and software for communications-intensive applications. Our hardware uses an FPGA-based GigE NIC that plugs into an Intel® Xeon® processor socket, allowing direct communication over the processor front side bus. With this hardware, we can explore software for future JNIC-inspired products, e.g. those based on 10GigE and next-generation processor interconnects. The JNIC itself is implemented using off-the-shelf FPGA, MAC, and PHY components.

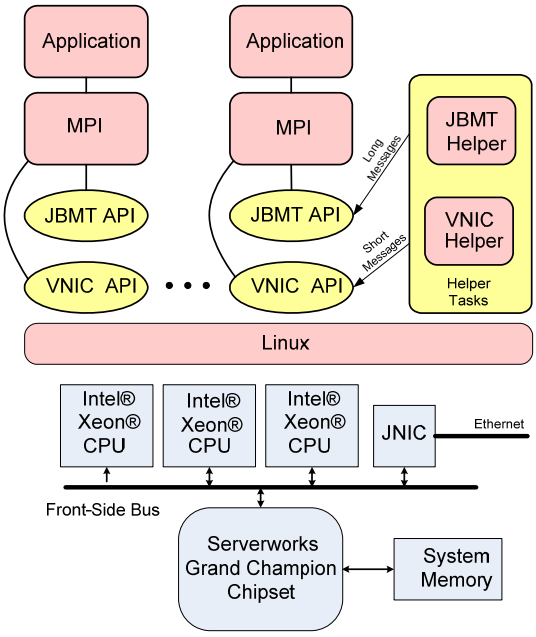

Figure 2: JNIC Testbed—The testbed combines a Xeon Multiprocessor, an onload driver, and FPGA-based NIC Hardware.

3. THE PROTOTYPE HARDWARE

The hardware testbed uses a 4-way multiprocessor server host system with our prototype NIC replacing one of the 4

processors. The host system can be configured with the prototype NIC, and one, two, or three CPUs. The prototype NIC is a fully compliant symmetric cache-coherent agent on the FSB (Front Side Bus) so the NIC can be accessed using coherent memory as opposed to PCI transactions. Residing on the FSB not only enables faster processor access to NIC registers, but also faster NIC access to system memory when it DMAs packet data to and from the network.

The prototype NIC is implemented in an FPGA. The NIC is constructed as three stacked circuit boards, the NIC Baseboard, external power board, and the IO mezzanine (Figure 3). The NIC Baseboard contains the FPGA, an optional onboard memory (currently unused), and a USB controller. The I/O mezzanine contains 4 Gigabit Ethernet ports implemented using discrete PHY and MAC (Media Access Controller) chips.

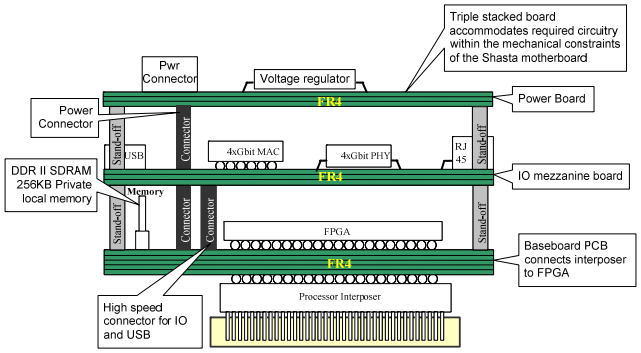

Figure 3: Physical hardware prototype—JNIC's triple-stacked board (with FPGA, MAC and PHY) plugs into a processor socket.

A unique feature of the NIC is that it provides a coherent network interface that is directly accessible by host CPUs. Major hardware resources on the NIC (i.e. buffers) are made visible as cacheable memory regions (apertures) that provide direct read and write access by CPUs using coherent memory transactions. On the transmit side, a CPU can directly push packet data into the NIC, while on the receive side the CPU can gain access to a packet even before it reaches system memory. Coherent access provides a powerful tool for lowering latency, reducing copy overhead, and increasing the efficiency of data transfer between the host CPUs and the NIC device.

Four main hardware resources are exposed via the memory apertures: the transmit command queue, the receive command queue, the receive data buffer, and the control status registers (Figure 4). These resources are updated and read via coherent memory transactions. The NIC hardware sees all system bus transactions and performs appropriate actions based on the transaction's type and address.

The NIC's Control and Status Registers (CSRs) are mapped to a single region of memory, with each CSR mapped to a single cache line. The CSRs are written and read via the use of coherent memory transactions. This is important since it allows efficient software polling of CSRs. For example, when software polls the receive buffer tail pointer to detect the arrival of data, the tail pointer value is cached within the processor for efficient repeated access that generates no front-side bus traffic. After the NIC receives a new message and updates the tail pointer, the pointer is invalidated by hardware permitting software to see the updated value.

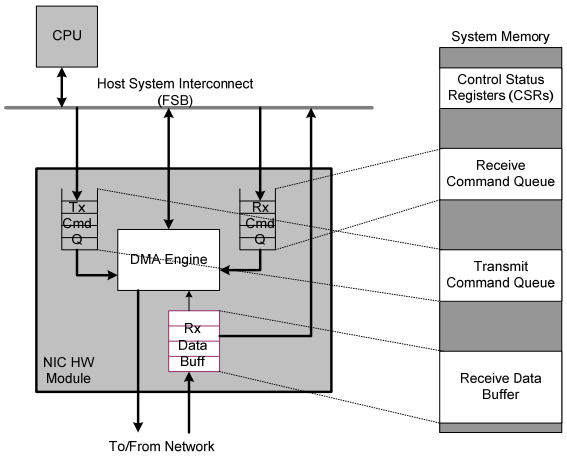

Figure 4: NIC System Memory Apertures—Apertures allow the efficient exchange of data, control, and status, between application and JNIC hardware.

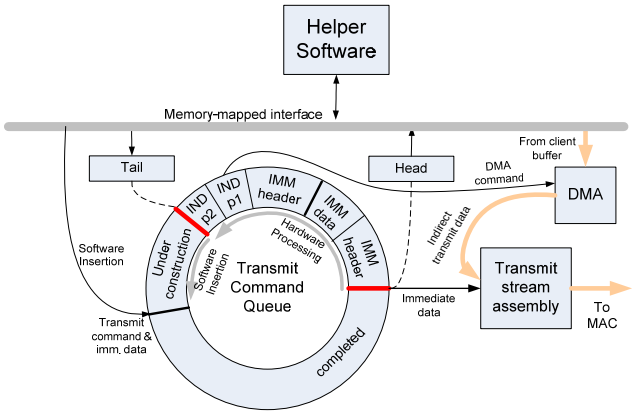

Figure 5: NIC Transmitter—The transmitter supports immediate commands for short messages and indirect commands for long message DMA.

3.1 Transmit Operation

The transmit command queue (Figure 5) is a circular queue that holds the commands for JNIC's transmit DMA engine. Head and tail pointers indicate the region of valid queue entries. Information is pushed onto the transmit command queue using coherent memory transactions. When the NIC observes a bus transaction that writes to the transmit command queue aperture, it captures the data and updates its local (in-NIC) hardware buffer. This unique ability allows the CPU to directly push data into the NIC hardware via coherent memory. The NIC transmit engine supports two transmit command types immediate and indirect transfer. For immediate transfers, which are used for small messages, the command and the associated data are both pushed directly into the transmit command queue. This saves the time that would otherwise be necessary to invoke a DMA. Indirect transfers are used for larger messages and require that only command information and no data be written to the command queue. Indirect commands direct the NIC to copy data from system memory into the transmit stream using hardware

DMA. Each DMA is from a contiguous physical address region, so multiple indirect commands are used to transfer a data region that is contiguous only in virtual address space.

The hardware flow of packet transmission can be divided into four parts: command transfer, command processing, data packet transmission, and completion status update. During command transfer, helper software writes a sequence of one or more transmit commands into the transmit command queue (Figure 5) by writing to the JNIC memory aperture. When a complete packet has been described, software updates the tail pointer, and launches hardware command processing of the region between the head and tail pointers. An Ethernet packet is then assembled in the NIC's local transmit buffer based on the sequence of commands. The first and last commands of each packet are marked so hardware knows when packet assembly begins and completes. Once the complete packet is assembled, data packet transmission sends data to the network. Finally, during the completion status update phase, the transmit command queue head pointer is moved to signal hardware completion.

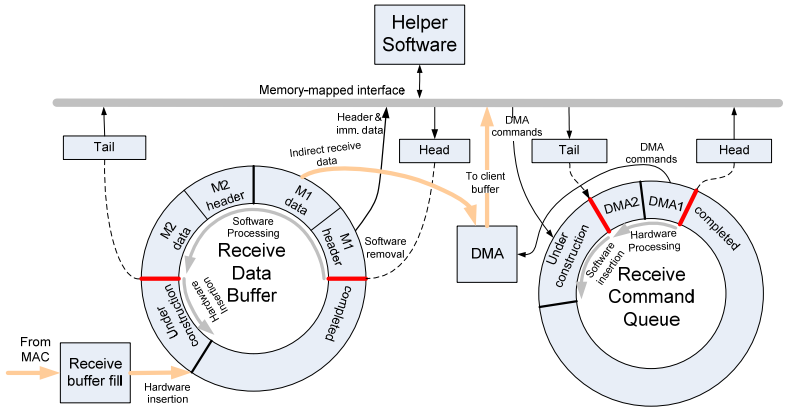

Figure 6: NIC Receiver—The receiver supports software-based header processing in memory-mapped receive data buffer. The receive command queue processes DMAs for copy-free delivery.

3.2 Receive Operation

The receive data buffer (Figure 6) holds data arriving from the network. Processors have direct access to this buffer and software can move packet data from the buffer directly to system memory without requiring multiple copies. Software can access received data in any manner it chooses, hence the received packets can be accessed and processed in arbitrary order. This feature can be used to reorder the delivery of received packets to reduce memory copy costs.

The receiver provides a receive command queue to control receive-side DMA processing. The receive command queue is similar in construction to the transmit command queue and holds commands that are executed by the receiver's DMA engine to process packets that are received by the NIC. The flow of a received packet can be divided into the following steps: receive data buffering, receive header processing, receive command transfer and receive data transfer. During receive data buffering, data received from the network is buffered within the NIC. This buffer is mapped to a system memory address range thereby making it visible to host software (Figure 4). Head and tail pointers track the region of valid entries. For each arriving packet, hardware delivers data into the receive data buffer and updates the tail pointer. This action invalidates the cache line containing the tail pointer within all processors caches and notifies any processor that is polling the tail pointer that new data is available. If a new packet arrives and the receive data buffer is full, the packet is dropped. This could occur if receiver software fails to keep up with the arriving packet stream. The result would be equal to that of a dropped packet in the network and might require software retransmission.

During receive header processing, software polls the receive data buffer tail pointer. When the tail pointer indicates a new packet is available, software reads only required header portions of the receive data buffer. Software interprets headers to determine whether to deliver data directly via software-copy or via hardware DMA. Delivery in software eliminates DMA overhead when the processor reads from the receive data buffer and writes data directly into the receiving client's address space. For larger amounts of data, DMA is invoked.

During receive command transfer, software invokes DMA by transferring one or more receive commands into the memory aperture for the receive command queue and advancing its tail. Each command invokes a DMA with a source address that identifies data in the receive data buffer and a physically addressed target in system memory. The NIC reads all data written to this aperture and updates its local (in-NIC) receive command queue. Head and tail pointers indicate valid commands in this queue. When a newly written command is ready, software updates the tail pointer, thereby launching receive command processing of the region between the head and tail pointers and initiating the data transfer phase.

During receive data transfer, hardware processes a command by initiating DMA from the receive data buffer to system memory. The receive command provides all information needed to move the data, including physical address pointers. As with the transmit-side DMA, multiple physically contiguous DMAs may be required to transfer data to a logically contiguous virtual address space. Once a DMA has competed, hardware indicates the completion status by updating the receive command queue head pointer. Software polls the head pointer to learn that the command has been processed by hardware.

4. JNIC SOFTWARE ARCHITECTURE

The JNIC software architecture exploits the increasingly abundant processing power from future multicore architectures to provide high levels of performance. Performance bottlenecks encountered by traditional architectures include the system call and interrupt processing overhead associated with conventional Ethernet device drivers. In addition, the traditional sockets API has copy semantics, requiring that messages be copied by software from user buffers to kernel buffers or vice versa. These copies are expensive for long messages and can account for a significant fraction of the time spent in processing packets. The JNIC architecture addresses these and other communications-related difficulties using a variety of techniques.

A kernel task running on a dedicated processing core is used to service hardware and user requests. System call overhead is eliminated using memory mapped VNIC queue structures that are shared by kernel user tasks. A user program uses simple loads and stores to enqueue and dequeue elements. The kernel task not only monitors these queues, but it also monitors the JNIC hardware's transmit command and receive data and command queues. All queues are polled by dedicated cores so interrupt overhead is not incurred.

The VNIC queues also (much like hardware) allow both copy-based communications for short messages and copy-free communications for long message. On top of this low level support, we provide an RDMA-like bulk transport capability that can be used by applications, such as MPI, to efficiently move large messages.

4.1 Helper Thread Scheduler

A kernel thread, which we call the helper, is central to JNIC's software architecture. This thread is bound to its own dedicated core and both VNIC-layer and JBMT-layer processing are performed within a single helper thread. Both the VNIC and JBMT implementations are decomposed into multiple subtasks. Because of the importance of helper processing, the JNIC software includes its own scheduler to manage helper subtasks in an efficient, modular, expandable and OS-independent way. Currently, the helper thread services all subtasks in round-robin fashion although other policies could incorporate more sophisticated quality-of-service mechanisms.

4.2 Virtual NIC Interface

Our architecture defines a VNIC interface that provides communication services to a potentially large number of clients with diverse needs. The interface uses shared memory queues that are polled by helper software. This facilitates very low latency communications as user mode clients do not need system calls to transmit or receive data.

VNICs support a simple command interface that minimizes overhead for simple communications. VNICs are also used to implement more complex services such as the RDMA-like bulk messaging described below. VNICs directly support reliable communications. We believe that the majority of in-data-center communications requires reliable communication and that optimizing reliable communications within the lowest software layer is most efficient.

VNICs support both immediate- and indirect-mode communications. Immediate mode provides the fastest way to communicate short messages especially when used in conjunction with JNIC's hardware immediate mode. This allows rapid multiplexing (within onload software) of short messages between multiple VNICs and memory-mapped JNIC transmit and receive buffers. However, immediate mode requires costly copying for large messages.

Efficient bulk data transfer is supported with the VNIC indirect mode. VNIC indirect mode provides the utility of JNIC's hardware DMA to VNIC clients. Data is passed, by reference, to lower layers and, when used in conjunction with JNIC's hardware indirect mode, copy-free transport can be implemented. Since JNIC hardware is restricted to transfer data between pinned pages, VNIC source and destination buffers are also pinned. Indirect mode may require overhead including: overhead for buffer registration, overhead to process separate command and data streams, and overhead for indirect data address lookup. Thus, while indirect mode is the most efficient way to move large amounts of data, indirect data specification may increase latency for short messages.

A client inserts commands into a VNIC command queue with each command providing information needed to move a single Ethernet packet of data from a sender to a receiver. These commands describe how to collect data at the local end, where to send the data, and how to distribute the data at the remote end. After building a command in the queue, the client moves a tail pointer across that command to notify the helper. This tail pointer is isolated to its own cache line to permit responsive polling of VNICs in a manner that generates no extra front-side bus traffic.

A VNIC command is composed of several fields including a header, a list of destination fields, and a list of source fields. The header gives general information about the message including information identifying the destination VNIC and the total message length. The destination field list tells the destination how to deliver the message when it is received. Each field in the destination list specifies either immediate or indirect. A destination immediate field instructs the receiving JNIC's helper thread to copy the next N bytes from the message directly into the destination VNIC where it would be available to the client. In contrast, a destination indirect field instructs the receiving JNIC's helper thread to copy the next N bytes from the message to a location in the destination user's address space represented by a handle and an offset. The JNIC helper thread can choose to copy the data directly via software or through use of the JNIC hardware's DMA engine.

The fields in the source list are also of the immediate and indirect forms. They notify the VNIC as to how to acquire the data to form the transmitted packet. By using combinations of these primitive destination and source immediate and indirect fields, more complex interactions can be built. For example, for MPI, an unbounded length MPI message is decomposed into Ethernet-frame-sized VNIC messages with MPI headers that support features such as MPI's message matching. MPI headers are inserted and retrieved from the VNIC message using destination and source immediate fields. MPI also uses our bulk messaging protocol which uses destination and source indirect fields for efficient long message transfer.

4.3 The VNIC Implementation

The VNIC helper task multiplexes all VNICs onto the physical NIC hardware. It also provides both end-to-end reliability and congestion control. Software running on a dedicated core close to NIC hardware supports highly responsive and flexible packet processing. The actions of the VNIC helper are partitioned into four subtasks as shown in Figure 7. These helper subtasks use simple producer-consumer relationships to facilitate lock-free communications that can be seen as single-input single-output queues that are used between: helper and application (blue), helper and helper (pink), and helper and hardware (yellow).

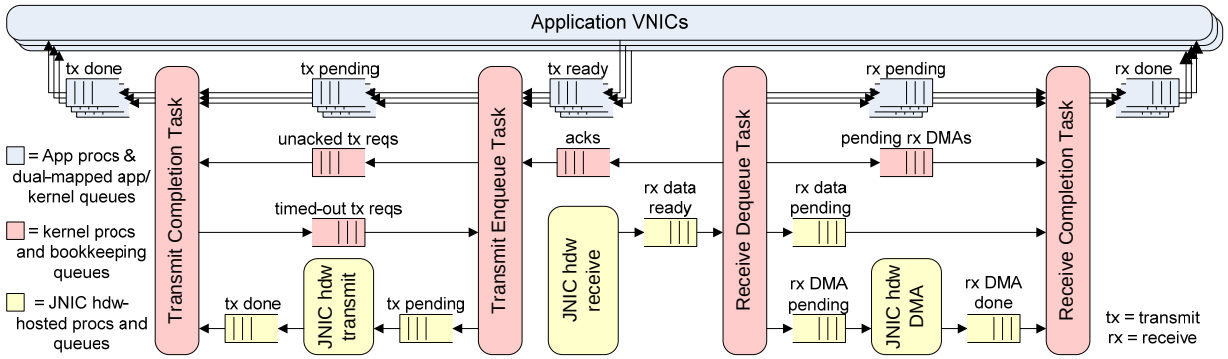

Figure 7: VNIC Helper Tasks and Queues—The VNIC helper is decomposed into multiple sub-tasks needed to service user tasks and JNIC hardware.

The low-level interface between the VNIC helper task software and JNIC hardware can be seen in Figure 7 where hardware is shown in yellow. To send data, the transmit enqueue task writes into the JNIC hardware's "tx pending" queue. This corresponds to appending commands at the tail of the transmit command queue of Figure 5. After hardware completes the transmit operation, the transmit completion task senses completions in the "tx done" queue by observing the motion of the transmit queue head pointer caused by hardware. The receive dequeue task observes received data in the JNIC hardware's "rx data ready" queue which corresponds observing the tail of the receive data buffer in Figure 6. After header data has been processed, messages are delivered either by copying data in software, or by delivering data using DMA. To invoke DMA, the receive dequeue task inserts a command into the "rx DMA pending", queue which corresponds to appending at the tail of the JNIC hardware's receive command queue. The receive completion task senses DMA completions by watching the "rx DMA done" queue which corresponds to watching the advancement of the head of the receive command queue as hardware completes each DMA.

The VNIC helper implements reliable communications and ensures both reliable transmission and efficient use of network resources. Because congested Ethernet switches drop packets, reliability and congestion control are closely coupled. JNIC's software provides reliability using a TCP/IP like protocol. Our communication layer uses highly responsive software to provide low latency and high bandwidth. Typical TCP/IP implementations are designed to accommodate both wide-area and in-data-center communications and, as a result, are not designed for low-latency. By minimizing round-trip latency, we can respond quickly to, and minimize, fabric congestion.

4.4 The VNIC Reliable Datagram Service

While communication interfaces such as a TCP socket and InfiniBand's Queue Pair are reliable and connection-based, JNIC's VNIC is reliable and datagram based. Connection-based interfaces support one-to-one communications between a local and a single remote interface. High performance compute clusters use hundreds or thousands of processing nodes (computer systems) to run compute-intensive parallel applications. Each node may consist of multiple CPUs or CPU cores, and may concurrently run multiple application processes (shown as "Users" in Figure 8 and Figure 9). Each user communicates with many other users and may do so in an unpredictable manner. To support such communication, each user needs distinct connection-based interfaces to send and receive messages from all other users. Thus, in an N-user cluster, each user requires N-1 communication interfaces.

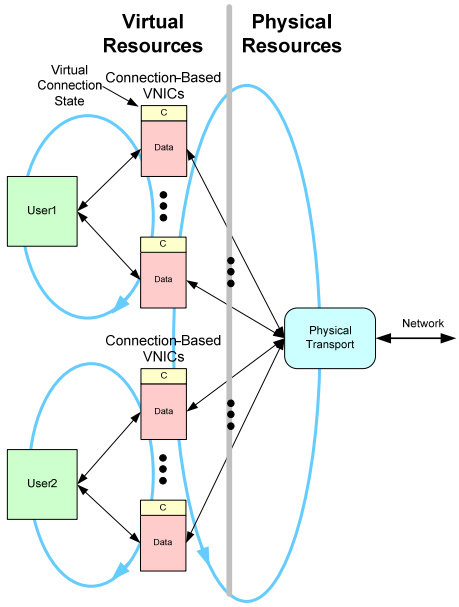

Figure 8: Connection-based Interface—Connection based interfaces require a distinct receive buffer for each remote interface.

For large-scale clusters, connection-based receive buffers cause inefficient memory usage. Within a single MPI application, each source user may send data to all destinations many times. Sometimes, a large amount of data is exchanged over a single connection and high bandwidth requires a substantial receive buffer for that connection. If we assume that a buffer of M bytes is needed to provide high bandwidth for a single connection, then (N-1)M bytes are needed to support all connection-based receive buffers for that user. Every user requires a similar amount of space and thus the total space required is N(N-1)M bytes. These buffers are used repeatedly and are often resident in expensive main memory. When buffers are swapped to disk, performance degrades severely as disk access is required to sustain communications performance. However, the utilization of these receive buffers is necessarily poor. It is rarely the case

that all users send messages to a single user and fill all receive buffers for connections to that user. Hence, most receive buffers are empty. For very large clusters, this problem is so severe that complex techniques have been developed to reduce the number of active TCP buffers [7].

Connection-based user interfaces require multiplexing and de-multiplexing data from many virtual interfaces to a single physical interface. This can be done with polling (shown using blue ovals in (Figure 8), but polling is not scalable. The use of doorbells improves scalability but introduces interrupts or other complexity in VNIC access. Cache performance degrades as data moves through memory addresses associated with a large number of communication interfaces. Connection-based interfaces may also incorporate connection state. For example, TCP connections maintain state including a TCP window to track data exchange with a single remote connection. For each VNIC, connection state is virtualized and unaware of the demands of any other connection.

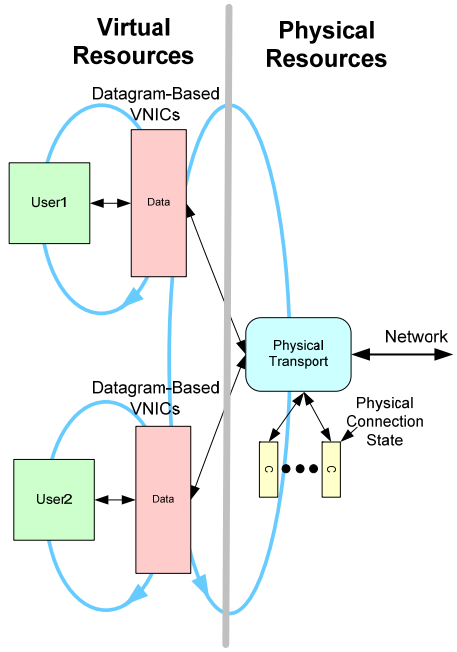

Figure 9: JNIC's Datagram-based Interface—Datagram based interfaces share receive buffers across many remote interfaces.

Unlike connection-based interfaces, datagram-based user interfaces, such as the VNIC, support one-to-many and many-to-one communications. Each datagram-based interface sends to, and receives from, many remote interfaces providing buffer sharing opportunities. Each user now uses only a single datagram-based interface (Figure 9). Therefore the physical transport supports far fewer interfaces because the number of datagram-based interfaces is independent of the number of connections. A single, shared, receive buffer holds messages from many senders allowing communications to scale efficiently without requiring dedicated per-connection buffers. However, this key advantage is not without penalty. Prior techniques for connection-based flow control must be extended to support buffer space management for shared receive buffers. This is discussed in greater detail in Section 4.5.

4.5 Flow Control for Reliable Datagrams

VNIC receive queues are shared for enhanced scalability. To share queues, VNIC clients must guarantee adequate space within fixed-size memory-pinned buffers. JNIC's flow control uses the notion of a buffer space availability guarantee, or credit. Credits can be distributed to remote hosts as remote credits or held locally as local credits. JNIC's enhanced credit-based flow control provides the dual benefits of low latency and scalable use of shared buffers. Low latency is provided when unpredicted data is sent using remote credits without waiting for a round trip credit request. However, remote credits consume valuable unused pinned memory when many users hold remote credits to guarantee the right to send an unexpected message to a common recipient. For large messages, a round-trip request gathers local credits to transmit data without wasting unused buffer space. JNIC's flow control overlaps a request for additional local credits with sending useful data. Flow control has been implemented for the MPI and JBMT VNIC clients.

5. BULK MESSAGING

The JNIC bulk message transport (JBMT) provides an RDMA-like facility that is implemented as a kernel mode client to the VNIC interface. JBMT supports copy-free and autonomous delivery of bulk data. JBMT uses the VNIC indirect mode for both source and destination to eliminate software data copies and, instead, copies are performed by JNIC's hardware DMA.

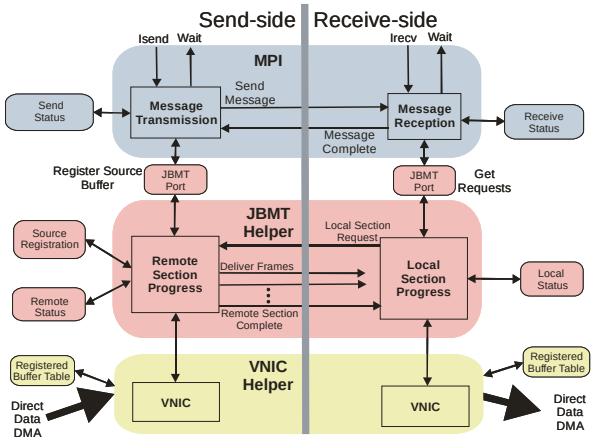

Figure 10: MPI and JBMT Overview—JBMT provides autonomous and copy-free bulk data transport. MPI uses JBMT for long messages.

JBMT is implemented (Figure 10) with helper software that receives requests and returns responses through virtualized command and completion ports. Communications is implemented by accessing the VNIC service with a VNIC dedicated for JBMT. JBMT implements a GET operation similar to that of RDMA. First, a memory region is registered by the sender and the operation returns a registration token that provides access control. JBMT registration does not pin the region and consumes only a small kernel table entry. The registration token is sent to a receiver using conventional VNIC messaging. Then, a JBMT GET is submitted to JBMT by the receiver. The GET specifies the registered source region's token, a source offset, a target region pointer, and a transfer

length. Legal transfers are limited to remote source regions for which access was granted in prior source registrations. After transfer, a completion is deposited in the JBMT port's completion queue.

DMA hardware access is restricted to pinned data to prevent page relocation during DMA. Before initiating low-level VNIC transfers, JBMT guarantees that source and target buffers are pinned for DMA. Pinning is kernel-managed to eliminate user control over critical page resources. JBMT decomposes large messages into smaller, dynamically pinned, sections. First, a receive-side local section is pinned. The local section can be any subset of the GET's target region as dictated by memory availability. A request for a local section is then sent to the remote (send-side) interface. A remote section is then pinned. The remote section is again a subset of the requested source region. The pinned remote section is fragmented into Ethernet-packet-size frames and transmitted using the VNIC's indirect message feature.

After a remote section is transferred, it is unpinned and additional remote sections are pinned, transmitted, and unpinned until the entire requested local section is received. After the local section is received it is unpinned and the next local section is pinned, requested, and unpinned until the entire data transfer, requested by the GET, has been processed. This procedure transfers regions of unbounded size while ensuring forward progress with limited memory resources.

6. MPI MESSAGE PASSING USING JNIC

In order to demonstrate message passing, the MVAPICH2 [10] version of MPI has been ported to JNIC. Our MPI implementation uses JBMT for long messages and a raw VNIC interface for short messages. When MPI is initialized, it opens VNIC and JBMT communication ports on each node in a cluster. Connections are opened so that each VNIC messaging port (for short messages), and JBMT messaging port (for long messages) can communicate with all nodes in the cluster. MPI allows out-of-order message matching between senders and receivers. Each receive operation provides a match key to indicate whether an arriving message should be delivered as the matching received message.

Different approaches are used for short and long messages. The decision to treat a message as short or long is made in software. Message length thresholds are carefully set when JNIC is tuned for a specific hardware and software approach. Short messages are sent directly through a VNIC. After a matching receive is found, the short message is copied, by MPI, to a user program variable. Received messages may be matched and delivered out of order. For example, if a first receive does not match the head-of-queue message, but does match the second entry, the second entry can be delivered before the message at the head of the queue. When the head-of-queue message is matched and delivered for a subsequent read, both entries are deallocated. When a VNIC receive queue becomes congested with messages for which there are no matching receives, messages are copied into MPI's receive status for later matching and deallocated from the receive queue.

MPI transmits long messages using JNIC's JBMT. MPI supports asynchronous messaging which overlaps long message transfers with computation. An MPI client executes an asynchronous ISend to initiate message transfer. For a long message, an envelope (a message with no data) carrying MPI rank and tag information is treated much like a short message and sent to the receiving interface where it is held until it matches a receive. The receiving MPI client executes an asynchronous IRecv. After successful receive-side matching, a bulk transfer is invoked by submitting a GET command through the JBMT interface. Performance is enhanced when application compute resumes while JBMT autonomously transports data. When JBMT signals the completion of the GET, MPI is informed that the transfer has completed, first on the receiving side, and then the sending side. The GET completion allows both receiver and sender to proceed past MPI Wait commands needed for correct asynchronous execution. At this time, the receiver may utilize received data and the sender may overwrite sent data

7. EXPERIMENTAL RESULTS

We have working prototype hardware and preliminary software capable of demonstrating our architecture's functionality. Our prototype provides a deep understanding of complex system performance but also suffers prototype limitations. The prototype uses dated 3GHz Intel® Xeon® processors, 1 Gb Ethernet, and an FPGA-based NIC. The quad data rate front-side bus provides a 64-bit wide data path with a 100MHz clock. FPGA-based prototyping suffers performance limitations for timing-critical front-side-bus circuits. These limitations reduce performance for JNIC's hardware command path and for JNIC's DMA. Similarly, JNIC software is first generation software whose performance is neither fully characterized nor tuned. With these limitations, we report JNIC's performance for a number of micro-benchmarks that characterize the technology.

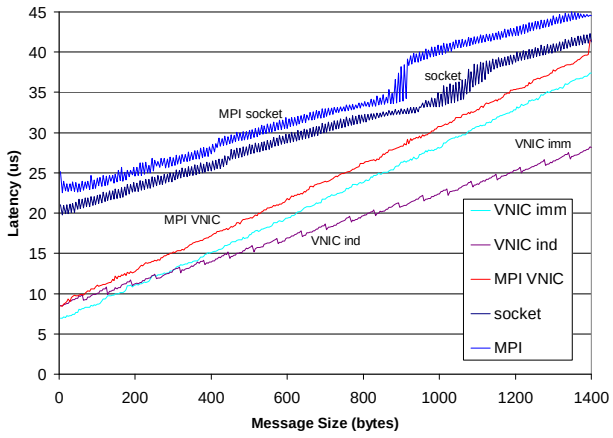

Figure 11: Latency versus message size—Latency for immediate, indirect and MPI over VNIC immediate modes, plus message latency for TCP socket and MPI over TCP socket.

The VNIC-to-VNIC message latency is plotted in Figure 11. This experiment was performed using a simple two-node application that creates a VNIC on each node. Nodes were directly connected with no switch latency. A message of given length is sent from a node A to B while B waits. After the message is received, then the message is sent from B back to A while A waits. This process is repeated many (N) times and the duration of the experiment is divided by 2N. This measures the time needed to send a message of given length.

The experiment was first performed using VNIC immediate (VNIC imm) mode. For immediate mode, all data was copied by the client application through the VNIC interface. In addition, data was copied by helper software to and from JNIC hardware for transmit and receive. The experiment was also performed using the VNIC's indirect (VNIC ind) mode. For indirect mode, software does not copy data. Instead, hardware DMA copies transmitted and received data directly from and to user buffers. While the immediate mode shows better performance for messages less than 128 bytes, the indirect mode shows improved performance for longer messages. A single byte immediate mode message takes about 8 microseconds. This result can be compared to the commonly reported ping-pong latency. To demonstrate baseline results for a mature I/O attached solution against JNIC, a similar experiment was repeated, on the same system, using TCP socket software (socket) and an Intel® PRO/1000 NIC. Latencies are also shown for MPI over VNIC (MPI VNIC) and MPI over TCP (MPI socket).

Latency will improve with a number of enhancements. Better hardware including faster processors, faster bus interfaces, improved cut-through data transfer, and a 10-fold Ethernet rate increase will all contribute to improved latency. Software performance will improve substantially with modern processor technology. Modern processors are twice as fast and NIC hardware improvements should also provide at least a factor of two latency improvement. Software improvements will result from profiling and tuning. Software improvements will allow JNIC's DMA to be used for longer VNIC immediate messages. Using DMA to copy long immediates from and to VNICs will improve performance for long immediate messages. We currently estimate that planned architecture improvements can reduce the latency to about 2us.

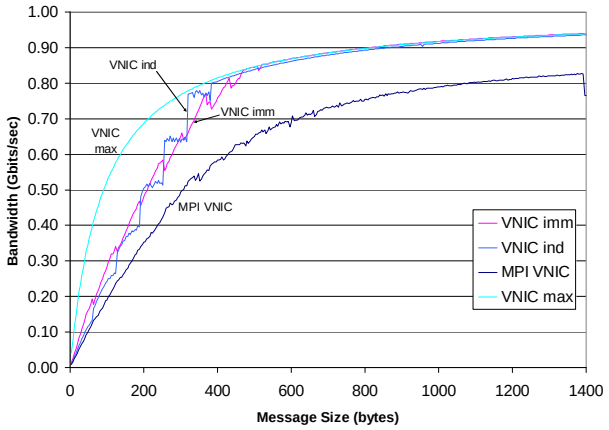

Figure 12: Bandwidth versus message size—Bandwidth for VNIC immediate, VNIC indirect and MPI over VNIC immediate. A theoretical upper bound VNIC max is also included.

Bandwidth is plotted in Figure 12. Each calculation measures payload bandwidth for a reliable stream of messages from source to destination client. Tests measure VNIC immediate (VNIC imm), VNIC indirect (VNIC ind), and MPI over VNIC (MPI VNIC) modes. A VNIC max plot is introduced to show a theoretical maximum payload bandwidth for an optimal stream of VNIC messages that fully saturates 1Gb Ethernet. The max bandwidth diminishes for small messages as the overhead for Ethernet headers, VNIC headers and inter-frame gap dominate the payload. This experiment shows a modest indirect mode bandwidth improvement at medium message lengths of about 300 bytes. Messages longer than about 500 bytes saturate the 1Gb network. Note that for both the immediate and indirect modes, the 1GigE link approaches saturation. Effects due to 64 byte cache lines are clearly visible for medium sized data. The achieved bandwidth for MPI is not yet as high as expected.

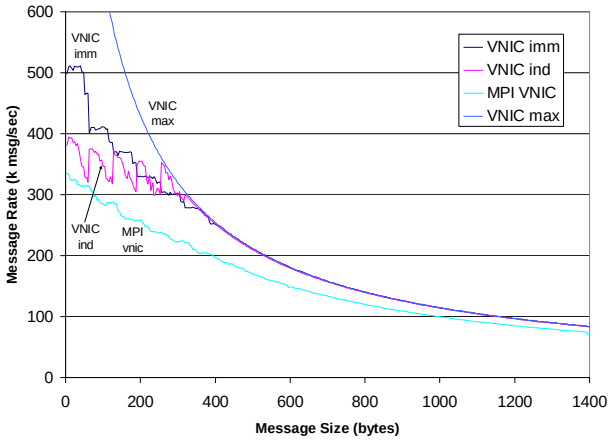

Figure 13: Message rate versus message size—Rate for VNIC immediate, VNIC indirect, and MPI over VNIC immediate. A theoretical upper bound VNIC max is also included.

The VNIC message rate is plotted in Figure 13. Each calculation measures the number of messages per second that can be sent for messages of a given size for VNIC immediate (VNIC imm), VNIC indirect (VNIC ind), and MPI over VNIC (MPI VNIC). The maximum rate, achieved in immediate mode, is about 420k messages per second. For short messages, immediate mode achieves a higher rate than indirect mode. For medium size messages, indirect mode is superior and cache line effects are especially visible. For long messages, rates are identical and limited by 1Gb Ethernet. Again, max plots the maximal VNIC message rate assuming that headers plus data saturate 1Gb Ethernet.

A number of factors will combine to improve future VNIC message rates. FPGA prototype hardware retains key limitations. Front-side-bus-interactions are slow and CPU-to-NIC and NIC-to-CPU data rates are not equal to that of custom circuits. Many future improvements will be found that increase message rates for our first generation software. The use of a modern CPU will increase rates substantially. We can also use additional processing power to exploit task-level parallelism. One strategy uses one processor for transmit and another for receive. This would tap additional compute power when cores are plentiful.

One of the strengths of the JNIC's system is high bandwidth. While at 1Gb, indirect mode communications is not very important, indirect mode will be important for 10Gb where software copy costs are relatively much higher. For indirect mode, software overhead is per message and the payload is processed by hardware DMA. With adequate frame sizes, and with suitable DMA hardware, our architecture scales to very high data rates. For example, at our currently achieved indirect-

mode message rates of 330k msgs/second, we could saturate 10Gb Ethernet with 3.8kB packets. If the message rate is increased by 2.5 times, 10Gb Ethernet will saturate with standard 1500 byte packets. This is a realistic goal with NIC improvements, software improvements, and up-to-date processor technology. Although much higher message rates may be possible, this would have to be demonstrated in future research and, with current techniques, longer frames are probably needed to saturate future 100Gb Ethernet.

For systems consisting of thousands of nodes, TCP socket connection-oriented clusters suffer scaling inefficiencies that are so costly that alternative fabrics are currently required. Compared to this, JNIC has clear scalability advantages. For example, JNIC provides the inherently scalable JBMT for long message transport. Since JBMT copies directly between user buffers, no significant system buffers are needed for copy-free transport. While JBMT is suitable for long messages, its performance for medium-sized messages may be dominated by buffer registration overhead, and round-trip messaging overhead for buffer handles. Here immediate-mode communications through a VNIC interface may be preferred.

When scaling VNIC immediate-mode communications, two key factors are considered: connection bookkeeping data and VNIC receive buffer space. At each local node, JNIC requires connection-oriented bookkeeping for all remote nodes. Inspection of key data structures indicates that for an example 4000 node cluster, less than 1MB of memory is required, at each local node, to represent control information for all remote communications.

VNIC receive buffer space is needed for immediate mode data. We estimate VNIC buffers as follows. A credit request round trip time is estimated at 40µs or about 2 times the MPI-to-MPI round trip time. This estimates time needed for MPI to ask for and receive remote credits. Assuming a 10Gb network, 50KB of data can be sent from a given source before a credit reply is returned. Assuming 4000 nodes, a pool of 200MB of buffer space is needed to buffer unexpected sends arriving simultaneously from all senders. While buffer sharing techniques will improve, 200MB receive buffers per node in a 4000 node cluster is acceptable.

8. RELATED WORK

Prior work on Ethernet studied close-attach NIC architectures and on-load software. Afterburner [5] investigated the use of an I/O-attached NIC with on-load software to provide high-performance communications. A prototype NIC was developed and demonstrated.

Work by Mukherjee et. al [12] used simulation to explore coherent attached network interfaces for fine-grained communication. This work investigated microarchitectures for attaching NICs to a processor cache. Work by Binkert et. al. [2] investigated simple on-chip NIC hardware with on-load software to provide high performance network communications. This work was explored in conjunction with a simulation model that runs a complete system including user code, kernel code and hardware model.

Credit-based flow control has been studied in many contexts. An early use for credit-based flow control was to manage available buffer space across transmission links within switched ATM networks [8]. Credit-based flow control has also been enhanced to incorporate speculation [11] in order to improve buffer usage efficiency. Both of these efforts focused on managing credits between a single sender and a single receiver whereas we focus on managing credits between many senders and a single receiver. Work on scalable InfiniBand clusters [17] extends credit management to treat shared receive buffers.

A number of previous research and product efforts have developed network interfaces for high-performance communications. The Virtual Interface Architecture [6] defined an OS-bypass interface for low latency and high bandwidth communications that allowed the elimination of unnecessary data copying.

Users needing high-performance communications performance typically rely on specialized Myrinet [3], Quadrics [14] and InfiniBand [9] networks to construct scalable compute clusters. Myrinet, Quadrics, and InfiniBand incorporate high-performance communications interfaces into their architectures. These solutions utilize expensive switch fabric hardware providing features such as reliable delivery and link-level flow control. Nodes use optimized NIC hardware and driver software to deliver needed performance. Connection-based interfaces such as InfiniBand's Queue Pair suffer from scalability limitations. Research by Sur et. al. [18] explores scalable approaches for cluster based communications over InfiniBand.

The InfiniBand architecture provides powerful hardware capabilities to help manage congestion and quality of service. Service levels and virtual lanes allow for prioritized treatment of packets. Each service level is mapped to a virtual lane that supports lossless transmission with link-level flow control. A weighting scheme controls the relative rate for service among lanes. Qlogic (previously PathScale) offers Hypertransport-attached InfiniBand NICs that utilize onload software to achieve impressive performance.

Many specialized fabric vendors are now moving toward Ethernet. Myrinet provides PCI express connected Myri-10G NICs that communicate over both Myrinet's proprietary fabric and standard 10GigE. Myrinet's NIC combines offload processing and host driver software to offer high bandwidth and low latency for demanding scalable applications. Similarly, Quadrics is also investing in Ethernet. Specialized fabric solutions are at risk of being overtaken by the rapid pace of Ethernet development. Many experts predict that Ethernet will dominate competing approaches as greater investment drives Ethernet's progress.

Many existing Ethernet solutions do not focus on scalable high-performance in-data-center communications. Ethernet-based systems use a TCP/IP software stack that is not well suited for high performance communications. When TCP sockets are used for cluster communications latency is high and TCP-socket-based communications scales poorly for large clusters. Prior work explored the use of TCP onload [15] to exploit multi-processor architecures for TCP acceleration. However this work was primarily for commercial applications and did not address issues of low latency, copy-free delivery, and scalability needed for cluster computing. Prior work also explored splintering TCP [7] which improves TCP's performance and improves the scalability of current TCP implementions that require a system buffer for each TCP connection.

Ethernet has been extended to provide RDMA with iWARP. Vendors such as Chelsio and NetEffect support 10Gb adapters that support iWARP. TCP Offload Engines or TOEs [1] can improve Ethernet performance by offloading TCP from a host to an intelligent NIC. However, many of these solutions do not focus on low latency or the scalability needed for many in-data-center applications. Research by Yoon et. al. [19] explored the use of the VIA communication interface with Ethernet. Research reported by Park et. al.[13] describes the architecture of a VIA-based network adaptor for Gigabit Ethernet. In this work an FPGA-based TOE was developed and benchmarked for latency and bandwidth.

Until recently, Ethernet switches suffered from low bandwidth and high latency and were not competitive with more specialized hardware solutions. We now see vendors like Fulcrum and Fujitsu who offer modestly priced Ethernet switches that provide 10Gbps bandwidth and less than ½ microsecond latency. Fulcrum's switch also supports fat-tree-based fabrics needed to provide very large bandwidth across scalable data centers. Woven systems incorporates fast Ethernet switches within a data center fabric that provides high bisection bandwidth, congestion-based load balancing, and fabric partitioning.

We believe that using JNIC-like solutions with future Ethernet components will alleviate many of Ethernet's handicaps and provide an architectural approach that achieves high communications performance using Ethernet.

9. FUTURE WORK

We hope to advance JNIC research in a number of directions. We will improve our understanding of JNIC's current performance, identify bottlenecks, and optimize performance for common usage models. A better understanding is needed for JNIC's performance in realistic future product and application settings. Of interest are 10GigE (and beyond), congestion in data center networks, and large applications requiring virtual memory. JNIC performance should be extrapolated to future product-relevant environments to better understand JNIC's applicability.

We plan to continue to explore architectures for tightly coupled NIC integration and improve our understanding of the best attachment architectures for future multiprocessors. On-load and off-load approaches should be compared to characterize differences in ease of use, performance, cost, and power. We hope to define on-load architectures that scale to 100GigE and many cores. Also needed are architectures that combine the responsiveness of polling and yet relinquish CPU resources during idle periods for added performance or reduced power. Can we dynamically scale on-load compute for needed communications performance?

As diverse network traffic competes for resources within converged fabrics, improved architectures are needed to prevent troublesome system failures. Currently data-center administrators are unwise to combine mission critical traffic with general traffic on a common Ethernet fabric. Next-generation converged fabrics must provide improved congestion management and quality of service. JNIC provides an architecture that facilitates improvements in congestion management and quality of service and this is a key area for future JNIC research.

System architectures are needed that incorporate JNIC benefits while supporting both traditional and innovative application interfaces. Future systems must support communications between tiers in a multi-tiered server and storage communication. While JNIC software currently demonstrates high-performance MPI, we hope to develop software for common communications APIs such as TCP or UDP. We will also consider developing architectures that support important interfaces such as iSCSI, RDMA, and DAPL.

10. CONCLUSIONS

The JNIC project has developed a prototype testbed to explore future Ethernet architectures for high-performance in-data-center communications. JNIC hardware and software architectures demonstrate that high-performance communications can be achieved within future low-cost data centers. The hardware models inexpensive closely attached NICs for future multi-core processors. The software demonstrates that general-purpose cores can be coupled with innovative software to deliver low latency, high message rate, and high bandwidth.

While our prototype does not yet deliver performance that is competitive with more specialized and higher-cost products, projections indicate that future, low-cost JNIC solutions can deliver the low latency, high bandiwdth, and scalability needed for the vast majority of in-data-center communicains.

We believe that inexpensive NICs integrated into future chip multiprocessor systems will provide a flexible platform that supports most communications needs. Such architectures will use flexible software for a broad spectrum of complex communications requirements. Low-level network functionality will be developed as on-load software to support network, messaging, disk and other communication needs.

11. REFRENCES

[1] P. Balaj et. al., "Head-to-TOE Evaluation of High-Performance Sockets over Protocol Offload Engines" Proc IEEE International Conference on Cluster Computing, Boston MA, Sept. 2005.

[2] N. Binkert, et. al. "Integrated Network Interfaces for High-Bandwidth TCP/IP", ASPLOS, 2006, pp. 315-324.

[3] N. J. Boden, et. Al.. "Myrinet: A gigabit-per-second local area network." IEEE Micro, vol. 15, no. 1, pp. 29-36.

[4] L. Cherkasova, et. al. "Fibre Channel Fabrics: Evaluation and Design." 29th Hawaii International Conference on System Sciences (HICSS'96), Volume 1: Software Technology and Architecture, pp. 53-62.

[5] C. Dalton, et. al. "Afterburner [network-independent card for protocols]" IEEE Network, July 1993, pp. 36-43.

[6] D. Dunning; et. al. "The Virtual Interface Architecture", IEEE Micro, Volume 18, Issue 2, March-April 1998, pp. 66–76.

[7] P Gilfeather et. al., "Making TCP Viable as a High Performance Computing Protocol", Proceedings of the Third LACSI Symposium, Oct 2002.

[8] H. T. Kung and R. Morris. “Credit-Based Flow Control for ATM Networks”, IEEE Network Magazine, pp. 40-48, March, 1995.

[9] J. Liu, et. al. “High Performance RDMA-Based MPI Implementation over Infiniband”, Proceedings of the 17th Annual Conference on Supercomputing, June 2004, pp. 295-304.

[10] J. Liu, et. al., “Micro-Benchmark Performance Comparison of High-Speed Cluster Interconnects”, IEEE Micro, January/February 2004, pp. 42-51.

[11] C. Minkenberg and Mitchell Gusat, “Speculative Flow Control for High-Radix Datacenter Interconnect Routers”, IPDPS, 2007, pp. 1-10.

[12] F. S. Muckherjee et. al., “Coherent Network Interfaces for Fine-Grained Communication”, ISCA, 1996, pp. 247-258.

[13] S. Park, et. al., “Implementation and performance study of a hardware-VIA-based network adapter on Gigabit Ethernet” , Journal of Systems Architecture, Vol 51, Issues 10-11, Oct. Nov 2005, pp. 606-616.

[14] F. Petrini, et. al. “The Quadrics Network: High Performance Clustering Technology”, IEEE Micro, Feb. 2002, pp. 46-57.

[15] Regnier et. al., “TCP Onloading for Data Center Servers”, IEEE Computer, November 2004, pp. 48-58.

[16] M. Snir et. al., *MPI: The Complete Reference*, MIT press (Vols 1 and 2), 1998.

[17] S. Sur, et. al. “Shared Receive Queue based Scalable MPI Design for InfiniBand Clusters”, IPDPS '06, April 2006.

[18] S. Sur, et. al., “High-performance and scalable MPI over InfiniBand with reduced memory usage: an in-depth performance analysis”, Proc 2006 ACM/IEEE conference on Supercomputing.

[19] I. Yoon, et. al., “Implementation and Performance Evaluation of M-VIA on AceNIC Gigabit Ethernet Card”, Proc of Euro-Par 2003, August, 2003, pp. 995-1000.