Performance Evaluation of Gigabit Ethernet and Myrinet for System-Area-Networks

Mustafa Imran Ali, *ID# 230203*

***Abstract*— Low latency and high bandwidth networking is essential for cluster computing and System-Area-Networks (SAN). The performance of a SAN optimized interconnect, Myrinet, is compared with gigabit Ethernet running TCP/IP. Though Myrinet has lower latencies and higher throughput than gigabit Ethernet, it is found that an efficient implementation of message passing interface library over TCP/IP achieves performance very close to Myrinet. These observations leads to the fact that gigabit and the upcoming 10 gigabit Ethernet can serve as a cost-effective alternative to specialized interconnects if the efficiency of protocol implementations can be improved. This paper also surveys some recent work being done on increasing the efficiency of TCP/IP and other efficient protocol implementations that benefit all applications using the ubiquitous sockets interface over Ethernet without using specialized libraries such as GM over Myrinet.**

I. Introduction

CLUSTERS are increasingly being used for parallel processing in a wide variety of applications such as database processing, data centers, web servers and scientific and engineering problem solving [1]. System-Area-Network (SAN) is used to describe computers (servers, workstations or PCs) connected through a high-speed network, cooperating to handle processing demands of parallel applications. In cluster supercomputing domain, Beowulf style parallel machines [2] have became popular due to cost-performance benefit it offers by using commodity-off-the-shelf (COTS) hardware to build cheap parallel machines. Ethernet is the most widely used LAN interconnect and as such has been used as a COTS component for interconnecting computers in clusters. Ethernet has evolved over past 20 years: 10 Mbps to Fast Ethernet (100 Mbps) and currently Gigabit Ethernet (1 Gbps) is gaining widespread adoption in place of Fast Ethernet. Furthermore, 10 Gbps Ethernet products have been released, leading to adoption of Ethernet beyond LANs up to MANs/WANs [3].

Neither the Ethernet nor TCP/IP protocol that runs over it were originally designed for the very high-speed communication requirements of cluster computing and as such their shortcomings have become evident under the demands of increasing line speeds [4]. In the last 10 years, a variety of high speed interconnect technologies have emerged for clusters. These include Myricom Inc. Myrinet [5], Quadrics QsNetII [6], Scalable Coherent Interface (SCI) [7], Virtual Interface Architecture (VIA) [8], and more recently InfiniBand [9]. Compared to TCP/IP over Ethernet, these technologies use highly optimized protocols and hardware implementations to realize low-latency and high-bandwidth communications. Ethernet is trying to catch up with the release of Gigabit and 10-Gbps Ethernet. The edge Ethernet has over these specialized interconnects is that it is backward compatible with a huge installed base of network applications and has the cost benefits of COTS, while the higher cost specialized interconnects may only be good for those parallel applications that can take advantage by rewriting code to use efficient libraries/APIs.

This paper evaluates the performance of Gigabit Ethernet and Myrinet-2000 by using microbenchmarking applications, Message Passing Interface (MPI) library benchmarks and parallel application benchmarks to measure respectively the raw communication performance, message passing performance and the overall effect on runtime of parallel applications. It is revealed that TCP/IP over Ethernet is not the highest performing cluster networking solution, but better libraries can allow full potential of gigabit Ethernet to be available to applications so that performance gap between Myrinet and Ethernet can be smaller. This is shown by results obtained with an event-driven TCP/IP based MPI implementation on NAS parallel benchmarks [10]. The paper briefly points out causes of poor performance, associated bottlenecks and surveys the proposed solutions for improving the performance of TCP/IP and Ethernet.

The paper is organized as follows. In the next section a brief overview of technologies is presented. In Section III the experimental methodology and benchmarks are explained. Results obtained are discussed in Section IV. A brief survey of ongoing TCP/IP and Ethernet research is given in Section V, followed by conclusion in Section VI.

II. Overview of Interconnects and Protocols

A. Gigabit Ethernet

Gigabit (Gbps) Ethernet [12]is an extension of the 10 Mbps Ethernet and 100 Mbps Fast Ethernet standards for network connectivity. It was developed to provide a high-capacity Ethernet-based network backbone that allowed for the aggregation of lower-speed (10 and 100 Mb/s) Ethernet LAN traffic. It was also developed to provide significantly greater throughput capacity to large network servers, increasing their ability to service ever larger numbers of clients in a timely manner. The Gigabit Ethernet standard, IEEE 802.3z [11], was officially approved by the IEEE standards board in June 1998. Gigabit Ethernet employs the same Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol, same frame format and same frame size as its predecessors, 10

Manuscript submitted May 28, 2005.

This work was done as part of course work requirements for Computer Networks (COE-540) Spring 2005, under the supervision of Dr. Tarek Sheltami.

Mustafa Imran Ali is a Research Assistant with the Computer Engineering Department, College of Computer Sciences & Engineering, King Fahad University of Petroleum & Minerals, Dhahran, Saudi Arabia.

Mbps Ethernet and Fast Ethernet. A number of changes were required to Fast Ethernet including the physical layer and much of the data exchange protocols.

B. TCP/IP

In building clusters from COTS components, researchers have used TCP/IP communication protocol over Ethernet. This general purpose protocol has certain characteristics that make it unsuitable for high-performance SAN. The first problem is due to the kernel intervention in all the network operations (in-kernel networking) [13]. In particular, it has been found that the standard implementation over UNIX which realizes a copy of the data from user memory space to a kernel buffer penalizes performance greatly when the network line speed is of the same order of magnitude as the system-bus to memory bandwidth. This also represents an unsurmountable limit to the minimum possible latency achievable during a transmission. To get over these bottlenecks, several systems has been developed where the Network Interface Card (NIC) performs read/write operations from network to memory space (and vice versa) directly accessible to the user processes (user-level networking) [14].

The other source of problem is that TCP/IP was originally devised to guarantee reliability in wide area communications and thus includes overheads which become the limiting factor of potential performance of physical network in cluster environments. It is difficult to optimize this protocol for the SAN and WAN environments simultaneously. Researchers have developed kernel patches that allows the dynamic activation and deactivation of the delayed ACK algorithm. Also, different possible variations of the slow start, congestion avoidance, fast retransmit and fast recovery algorithms to obtain optimal behavior in the SAN environment are being investigated [15]

C. TCP/IP Transmission over Ethernet

This section explains the TCP/IP transmission over Ethernet [16]. The explanation is presented so that in Section V the improvements proposed for TCP/IP will be presented in this context. TCP/IP processing can be classified into three major categories: connection processing, receive side processing, and transmit-side processing.

1) Connection processing: Connection processing refers to the establishment and tear down of TCP connections between communicating end systems. Once the systems establish a connection, data transfer occurs over the connection through receive and transmit operations.

2) Receive-side processing: Receive-side processing begins when the NIC hardware receives an Ethernet frame from the network. To extract the packet embedded inside the frame, the NIC removes the frame delineation bits and updates the descriptor data structure with the packet information. The NIC driver software supplies these descriptors, which are typically organized in circular rings, to the NIC. Through these descriptors, the NIC driver informs the NIC of information such as the memory buffer address in which to store the incoming packet data. The NIC uses a DMA operation to copy the incoming data into this memory buffer. Once it places the packet in

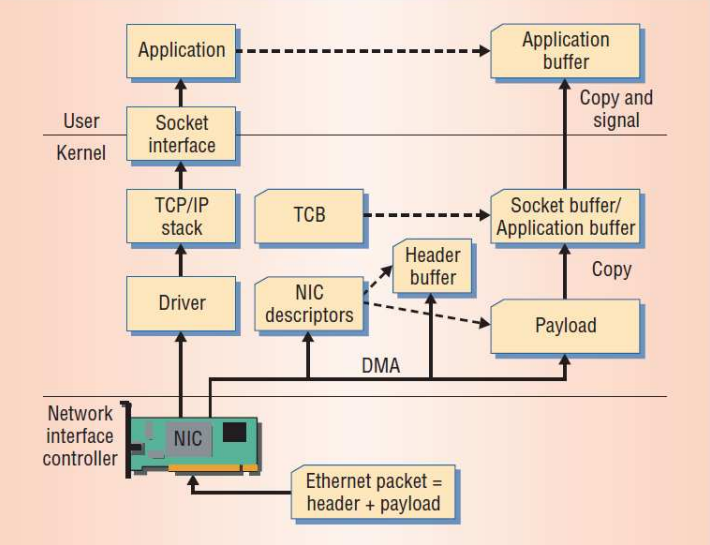

Fig. 1. Data flow in receive-side processing. Receive-side processing starts with the NIC device driver reading the descriptor and ends with the TCP/IP stack notifying the application that the data has been placed in its buffer

memory, the NIC updates a status field inside the descriptor to indicate to the driver that this descriptor holds a valid packet. It then generates an interrupt to initiate processing of the received packet. Figure 1 shows the overall receive-side processing flow. The NIC device driver reads the descriptor and passes the header and payload memory locations to the TCP/IP stack. The next step is to identify the connection that this packet belongs to. The TCP/IP software stores each connections state information in the TCP/IP control block (TCB) data structure. Since there can be many open connections and thus many TCBs, the TCP/IP stack uses a hashing mechanism to perform a fast lookup of the right TCB. The hash value is calculated from the IP address and port number of both the source and destination machines. When the TCP/IP stack receives a new packet, it updates several fields in the TCB such as the sequence numbers for received and acknowledged bytes and checks whether the application has preposted buffers. If the application has already posted a buffer to receive the incoming data, the TCP/IP stack copies the incoming data directly from the NIC buffer to the application buffer. Otherwise the stack stores the data in a temporary buffer for later delivery to the application. Performing a memory copy to move data from the NIC buffer to the application buffer is one of the most time-consuming operations in receive-side processing.

3) Transmit-side processing: Transmit-side processing starts when an application passes a data buffer to the TCP/IP stack. The application passes a socket ID along with the data buffer, and the TCP/IP stack uses this socket ID to locate the connections TCB. The TCP/IP stack can then copy the applications data into an internal buffer. Many TCP/IP stacks employ optimizations to avoid this copy. When the receivers window size indicates it is time to transmit data, the TCP/IP stack divides the accumulated data into maximum transfer unit segments. An MTU is typically 1,460 bytes on Ethernet LANs. The stack then computes the header: 20 bytes for TCP, assuming no options, and 20 bytes for IPv4. The stack then

appends these segments with the IP headers and passes them down to the NIC driver. The driver sets up the DMA to transfer the headers and application data to the NIC.

D. Ethernet Enhancements

A number of enhancements in NICs have been implemented recently to keep pace with increased ethernet line speeds of Gbps and 10 Gbps.

1) Interrupt Coalescing: Also referred to as interrupt moderation, this mechanism helps reduce the overhead of using the interrupt mechanism by grouping together multiple interrupts using one host interrupt for multiple events. Traditionally, network interfaces would interrupt the host processor after completing a send (to indicate that the state information for that packet may be freed) or after a receive (to indicate that the driver should process the new data). However, most of the newer interfaces improve performance by interrupting the CPU only after a certain number of frames have arrived or been sent, a certain time has elapsed, or some resource has become exhausted (such as DMA descriptors).

2) Jumbo Frames: Jumbo frames refers to the use of larger Maximum Transmission Unit (MTU), that is the maximum frame size that can be transmitted, of up to 9000 bytes compared to 1500 bytes used with previous generations of Ethernet. To preserve compatibility with 10 Mbps and 100 Mbps Ethernet, the Gigabit Ethernet standard still limits the MTU to 1500 bytes. It is to be expected that high speed networks such as Gigabit Ethernet would benefit from an MTU larger than 1500. MTU size in the order of few Kbytes indeed reduces the fragmentation overhead of whatever Ethernet-based protocol, and the interrupt overhead as well by using fewer number of exchanged packets. The impact of Jumbo frames under TCP/IP has been documented in many studies [4]. As a result of these factors many NICs and switches now support Jumbo Frames.

3) Offloading TCP operations: **Checksum offload**: Checksum calculation on a general-purpose processor is an expensive operation, but TCP/IP stacks can offload this feature to the NIC if it supports checksum calculation. Since implementing a checksum in hardware is relatively simple, offloading it to the NIC hardware does not add much complexity or cost.

Large segment offload (LSO): Segmenting large chunks of data into smaller segments and computing TCP and IP header information for each segment is expensive if done in software. Most current-generation NICs support this feature because doing this in hardware does not add much complexity. This feature benefits only transmit-side processing and is only beneficial when the application wants to send data larger than the maximum segment size (MSS), which the connections two end points negotiated during the TCP establishment phase. When transmitting a 64-Kbyte application payload, LSO can improve performance by up to 50 percent.

E. Myrinet-2000

Myrinet-2000 [17], developed by Myricom, is a proprietary network technology and is compliant to the Physical and Data Link layer defined in the ANSI/VITA 26-1998 standard.

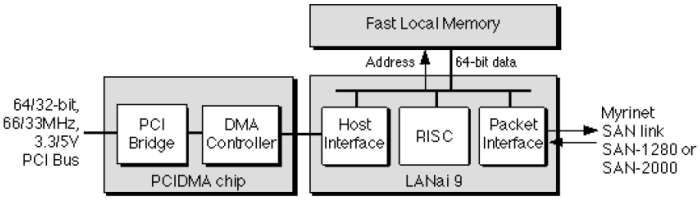

Fig. 2. Architecture of LANai9 1.28 Gbps Hardware

Myrinet is a switched, Gigabit per second network that is widely used in Beowulf clusters and embedded system. A Myrinet-2000 network is composed of crossbar switches and network adaptors. Network adaptors are connected to the switches through point-to-point duplex links and the crossbar switches can be interconnected in an arbitrary topology.

The basic building block for the Myrinet-2000 switche is a 16-port crossbar (XBar16). Data packets are wormhole routed from one network adaptor to another through a series of crossbar switches enabling a latency of approximately half a microsecond. Flow control is achieved by inserting STOP and GO control bytes into the opposite channel of a link by the receiver side to stop or restart data transmission on the sender side. As a result, Myrinet would not normally drop packets unless the receiver fails to drain the network. Error control is accomplished by computing an 8-bit cyclic-redundancy check (CRC-8) on the entire packet including packet header. The CRC-8 is then carried in the packet trailer and recomputed in each network stages. Thus, data packet entering a host interface with a non-zero CRC-8 indicates transmission errors.

The Myrinet-2000 network adaptor is a programmable communication device that provides an interface to the SAN. It consists of three major components (see Figure 2):

1) A custom VLSI (LANai) chip,
2) A synchronous static RAM memory,
3) A PCI Bridge and a DMA controller.

Myrinet adapters have a programmable network interface processor known as LANai9. LANai9 is a 32-bit RISC processor that operates at up to 133MHz for the PCI64B interfaces, or at up to 200MHz for the PCI64C interfaces. Using the Myrinet Control Program (MCP), which is stored in the on-board static RAM (SRAM), LANai9 controls the data transfer between the host and the network (through the host and packet interface), performs data buffer management (through memory interface), and maintains network mapping and monitoring. The benefit of a programmable network processor is that it enables researchers to explore many protocol design options.

To increase the data transfer rate, a Myrinet-2000 network adaptor is equipped with three DMA engines. Two DMA engines are associated with the packet interface: one for receiving packets and one for sending packets. The third DMA engine is used for data transfer between the SRAM and the host system memory through the host interface. Like most systems that support DMA, the on board memory can be mapped into user space and is thus accessible directly to user processes.

This mapping is performed in two steps: the OS first maps the NIC address space into kernel space and then, on a request

by user, the kernel region is mapped into user space. This memory mapping technique is commonly known as ”memory pinning”. In order to support zero-copy APIs efficiently, the DMA operations can be performed with arbitrary byte counts and byte alignments. Additionally, the DMA engine also computes the IP checksum for each transfer and provides a ”doorbell” signaling mechanism that allows the host to write anywhere within the doorbell region, and have the address and data stored in a FIFO queue in the local memory.

The host processor can also access the Myrinet-2000 SRAM through the programmable input/output (PIO) interfaces. With PIO, the host processor reads the data from the host memory and writes it into the Myrinet-2000 SRAM. This mode of data transfer typically results in many PCI I/O bus transactions. Although Myrinet-2000 PCI64 interfaces are capable of sustained PCI data rates approaching the limits of the PCI bus, the network performance greatly depends on the data transfer rate of the hosts memory and PCI-bus implementation.

All Myricom software for the PCI64 family of interfaces is based on the GM Myrinet Control Program (MCP) and the GM API.

F. GM Protocol/API

GM [17] is an open-source lightweight and user-level communication system for Myrinet that supports reliable ordered delivery and protected access, provided directly by Myricom. It requires the use of DMAable memory (registered with the system or allocated through the system). It supports messages up to $2^{31}-1$ bytes if the OS allows such amount of DMAable memory. It has 2 levels of message priority to help avoiding deadlocks in an efficient way . In the GM programming model a reliable connection is established between hosts, while communication endpoints do not need any communication establishment to communicate (connectionless). It supports up to 10000 nodes. Sends and receives are regulated by implicit tokens representing space allocated to the client by the system in its queues. GM achieves its exceptional performance with a technique known as ”Operating-System bypass” (OS-bypass). After initial operating-system calls to allocate and register memory for communication, application programs can send and receive messages without system calls. Instead, the GM API functions communicate through common memory with the Myrinet Control Program that is executing continuously on the processor in the Myrinet NIC.

G. MPI Library Implementation

Most MPI libraries provide messaging support on a broad range of interconnect technologies, such as Ethernet, Myrinet, Quadrics etc. This paper compares the performance of the Ethernet and Myrinet messaging layers in the Los Alamos MPI (LA-MPI) library. The LA-MPI library is a high-performance, end-to-end, failure-tolerant MPI library developed at the Los Alamos National Laboratory (LANL)[18]. The LA-MPI version 1.4.5 library implementation consists of three almost independent layers: the MPI Interface Layer, the Memory and Message Layer (MML), and the Send and Receive Layer (SRL). The interface layer provides the API for the MPI standard version 1.2 specification. The MML provides memory management, storage of message status information, and support for concurrent and heterogeneous network interfaces. Finally, the SRL interfaces with the specific network hardware in the system, and is responsible for sending and receiving messages over the network. LA-MPI supports each type of network, including Myrinet and Ethernet, using independent path modules within the SRL for each type of network. LA-MPI provides a TCP path (LA-MPI-TCP) in order to communicate over Ethernet networks and a GM path (LA-MPI-GM) in order to communicate over Myrinet networks.

An optimized, event-driven, TCP path (LA-MPI-TCP-ED) splits the library into two threads - a main thread to provide the core functionality through the functional interface of the library, and an event thread to handle message communication using TCP sockets over Ethernet [19]. LA-MPI-TCP-ED enables more effective overlap of computation and communication of an MPI application, leading to better performance of the application. However, the event-driven version of the LA-MPI library relies strongly on operating system support for detecting network events. A similar approach cannot be applied to Myrinet messaging since it uses a user-level GM communication library, which does not have the necessary facilities for detecting and delivering events.

1) TCP Path: The TCP path supports TCP message communication over Ethernet using the socket interface to the operating systems network protocol stack. For communication in the TCP path, the library uses at least one bidirectional TCP connection between each pair of communicating nodes. These connections are not established during initialization. Rather, they are established only when there is an actual send request to a node that does not already have an open connection to the sending node. Once a connection is established, it remains open for future communication, unless an error on the socket causes the operating system to destroy it. In that case, a new connection would be established for future messages between those nodes. The TCP path relies on the operating system for its memory buffering requirements. The socket buffers provide the necessary buffering for outgoing and incoming messages. The LA-MPI library copies data out of and into the socket buffer with the read and write system calls. However, the socket interface only allows for sequential access of data from the head of its buffer by the user application. Thus, a message has to be copied out of the socket buffer before the next message can be accessed. As a result, in the case of a message receive, which has not been posted in advance by the user application, the LA-MPI library needs to allocate temporary buffer space to hold this message after reading it out of the socket buffer. The only other instance when the library needs to provides its own message buffering is for buffered MPI send requests.

2) GM Path: The GM path of the LA-MPI library supports MPI communication over Myrinet. For this purpose the path utilizes the GM communication library. GM allows memory-protected user-level OS-bypass network interface access to applications, thereby achieving zero-copy message sends and receives at the application level. However, to support this feature, GM requires the presence of DMA accessible memory

on the host to send messages from, or receive messages into.

The GM communication library provides reliable, ordered delivery between communication endpoints, called ports. The communication model is connectionless: the LA-MPI library simply builds a message and sends it to any port in the network. The sends and receives in GM are regulated by implicit tokens, representing space allocated to the library in various internal GM queues. Unlike the TCP path, the GM path of LA-MPI does not use the operating system to provide memory buffers. Furthermore, the GM communication library does not provide any memory buffer management routines, and holds the GM path of LA-MPI responsible for providing all buffer management. However, being a user-level communication library, GM allows the library to allocate buffers for message sends and receives directly on the network interface card. This helps to eliminate copies between user-space and kernel-space. The GM path also does not have to provide temporary buffering for unexpected messages since there is no restriction on access of data from the network interface buffers. These buffers on the network interface are managed through the use of the send and receive tokens by the GM path of the LA-MPI library.

III. Methodology

The performance measurement of networks is best done by running actual user workloads. However, this hides the micro-level details of the hardware and protocol performance and their interaction with application requirements. For these reasons performance measurements are performed by using three different benchmarking approaches. Micro-benchmarks measure the raw metrics, mainly bandwidth and latency to give an idea about the intrinsic limits of underlying hardware and associated protocol implementations. Message Passing Interface (MPI) library performance is an indicator of the performance of parallel programs that rely on it. MPI benchmarks measure the bandwidth and latency of using MPI implementations on top of protocols that run on the respective network interfaces. Finally application benchmarks are important since synthetic benchmarks may indicate the ideal performance and not the actual achievable performance to the end user. The following sub-sections briefly describe specifics of each type of benchmark used in some detail.

3) Micro-benchmarks: Many open-source benchmarking packages are available for TCP/IP based performance measurement. Some measure only bandwidth while others offer latency measurement tests as well. The most well known are NTTCP [20] and Iperf [21] in the former category, while Netperf [22] and NetPIPE [23] belong to the latter. NetPIPE stands for NETwork Protocol Independent Performance Evaluator. Since in this work besides TCP/IP over Gigabit Ethernet measurements, the performance of GM protocol over Myrinet is also required, NetPIPE is used to serve as a common tool which also enables a fair comparison.

NetPIPE visually represents the network performance under a variety of conditions. It performs simple ping-pong tests, bouncing messages of increasing size between two processes across a network. Message sizes are chosen at regular intervals, and with slight perturbations, to provide a complete test of the communication system. Each data point involves many ping-pong tests to provide an accurate timing. Just as a computer's performance cannot be accurately described using a single sized computation, neither can the performance of a network be described using a single sized communication transfer. NetPIPE increases the transfer block size k from a single byte until transmission time exceeds 1 second. Hence, NetPIPE is a variable time benchmark and will scale to all network speeds. Latencies are calculated by dividing the round trip time in half for small messages (less than 64 Bytes). NetPIPE consists of two parts: a protocol independent driver, and a protocol specific communication section. The driver is based on the principles presented by the HINT [25] computer performance metric. The communication section contains the necessary functions to establish a connection, send and receive data, and close a connection. This part is different for each protocol. However, the interface between the driver and protocol module remains the same. Therefore, the driver does not have to be altered in order to change communication protocols.

NetPIPE produces a file that contains the transfer time, throughput, block size, and transfer time variance for each data point and is easily plotted by any graphing package.

4) MPI Benchmarks: While the raw latency and bandwidth of the network is important, the MPI library can add additional overhead. Many implementations of MPI exist such as MPICH [26], LA-MPI [18], LAM/MPI [27], MPI/PRO [28], MP_Lite [29], among others. Since the performance of MPI varies among implementations, it is only fair to compare performance of one implementation on both TCP/IP and GM. For this paper, the Los Alamos MPI (LA-MPI) library was used as it provides two different paths over TCP/IP, one of which is an optimized event-driven path as described in Section II. The reason for using two different TCP paths allows observing the effect of computation-communication overlap feature when the performance on NAS Parallel Benchmarks is measured, as described in Section IV.

Messaging latency can be measured by a simple microbenchmark with two communicating nodes. The first node simply sends a message to the second node, which returns the message back to the first node. The message latency is then half of the total time elapsed in this two-way message transfer. In this microbenchmark, the default blocking versions of send and receive are used for all MPI communication. Messaging bandwidth can also be measured by another simple microbenchmark with two communicating nodes. The first node repeatedly sends a fixed-size message to the second node as fast as it can. The second node simply receives these messages. The bandwidth is then the total number of bytes sent divided by the time elapsed to transfer all of the messages. Again, the default blocking versions of send and receive are used for all MPI communication.

A variety of suites of MPI benchmarks are available including Intel MPI Benchmarks (formerly Pallas MPI Benchmarks) [30], MPBench [31], perftest [32] and SKaMPI [33]. These differ in the number of different MPI routines tested and the approach taken to measure. The first two are relatively simple benchmark implementations compared to SKaMPI

suite, which takes a sophisticated approach to measuring performance of MPI routines and is used for this paper. It has comprehensive coverage of all MPI routines but for the purpose of this paper, only point-to-point throughput and latency measurements were useful.

5) Parallel Application Benchmarks: The simple microbenchmarks do not necessarily translate into overall application performance since they simply measure communication performance in isolation. In any real MPI application, communication occurs in parallel with computation, and most applications are written using non-blocking library calls to maximize the overlap between communication and computation.

The Numerical Aerodynmics Simulation (NAS) parallel benchmarks (NPB) are a more realistic set of benchmarks that include both computation and communication [10]. NPB is a set of benchmarks, comprised of both application kernels and simulated computation fluid dynamics (CFD) applications. This paper uses five benchmarks from the NPB version 2.2 suite BT, SP, LU, IS and MGto provide a thorough comparison of the various LA-MPI versions using both Gigabit Ethernet and Myrinet networks. Among the benchmarks, the first three are simulated CFD applications and the latter two are smaller application kernels. The NPB version 2.2 suite supports up to three precompiled data-sets, A, B, and C (in increasing order of size), for each benchmark.

A. Test Setup

The experimental configuration consisted of a 9 nodes cluster, with one node dedicated as an NFS server. Each node was a Dell PowerEdge 2650 server having a 2.2 GHz Intel Xeon CPU running on a 400-MHz front-side bus using a ServerWorks GC-LE chipset with 1 Gbyte of memory and a dedicated 133-MHz PCI-X bus. From a software perspective, all hosts ran Red Hat Linux version 8.0 (kernel version 2.4.18-27.8.0). Each host carried both a 3Com 3C996B PCI Gigabit (copper) Ethernet adaptor and a Myrinet 1.28 Gbps LANai9 one-port 64-bit PCI-X host adaptors (SAN-1280). In addition to hosts, a Cisco Catalyst 3550 12T Gigabit Ethernet Switch was used as well a 16-port Myrinet-2000 switch with Myrinet-Fiber ports and monitoring capability (M3F-SW16M).

IV. Experiments & Results

A. Raw Bandwidth & Latency

Figure 3 shows the raw network latency comparison between Ethernet and Myrinet across a range of message sizes. The topmost line corresponds to the latency of Gigabit Ethernet NIC on both the receiver and the sender. The middle line shows the same data, but in this case all interrupt coalescing in the NIC driver is turned off. These two lines show that interrupt coalescing causes a significant increase in the latency of Ethernet communication, by as much as 30 microsec for 1 byte messages. Finally, the lowest line gives the Myrinet communication latency between two Myrinet NIC equipped nodes. The figure shows that Myrinet has a significantly lower latency than Gigabit Ethernet without interrupt coalescing. Thus, Myrinet has around 40 microsec lower latency than Gigabit Ethernet, and around 10 micosec lower latency than

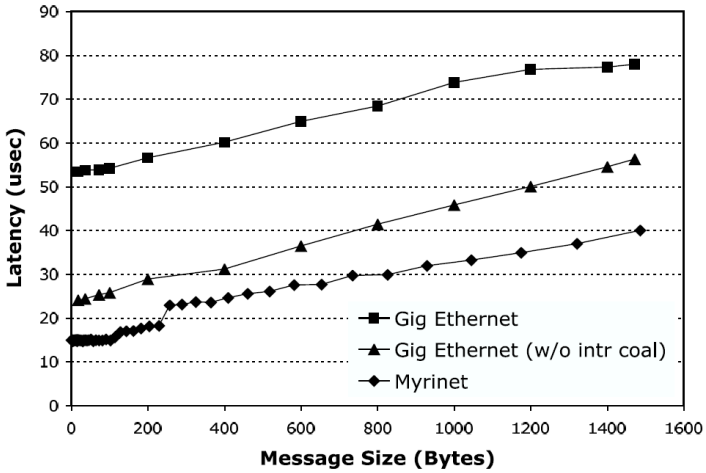

Fig. 3. Comparison of raw network ping latencies for Myrinet and Ethernet

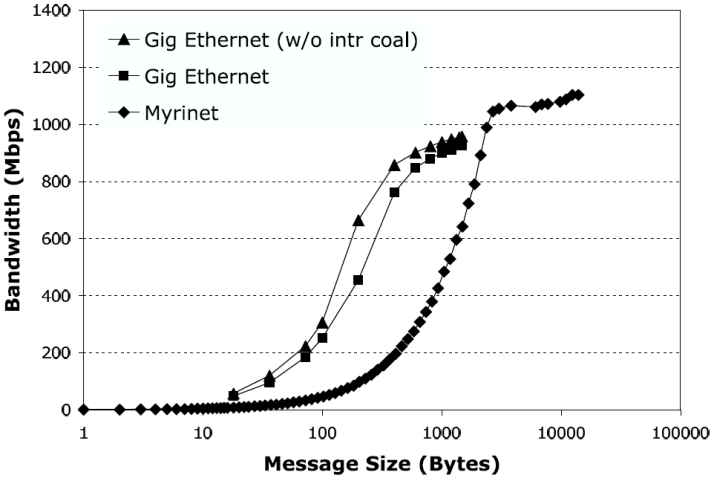

Fig. 4. Comparison of raw unidirectional message bandwidths for Myrinet and Ethernet

Gigabit Ethernet without interrupt coalescing, in the range of message sizes shown.

Figure 4 shows the raw network unidirectional bandwidth comparison between Ethernet and Myrinet across a range of message sizes. The figure shows that the bandwidth obtained on Gigabit Ethernet with interrupt coalescing is remarkably similar in overall profile to that without interrupt coalescing. However, the obtained unidirectional bandwidth is slightly higher when the NIC driver does not coalesce any interrupts. For Gigabit Ethernet without interrupt coalescing, the unidirectional bandwidth saturates around 960 Mbps. When interrupts are coalesced, the bandwidth saturates about 30 Mbps lower. On the other hand, Myrinet bandwidth is substantially lower than Gigabit Ethernet for the same message size up to 1500-byte messages. This result is surprising since the version of Myrinet used here has a higher peak bandwidth (1.2 Gbps) than Gigabit Ethernet (1 Gbps). As a result, Myrinet bandwidth saturates at a higher 1100 Mbps, but at much larger message sizes than Ethernet.

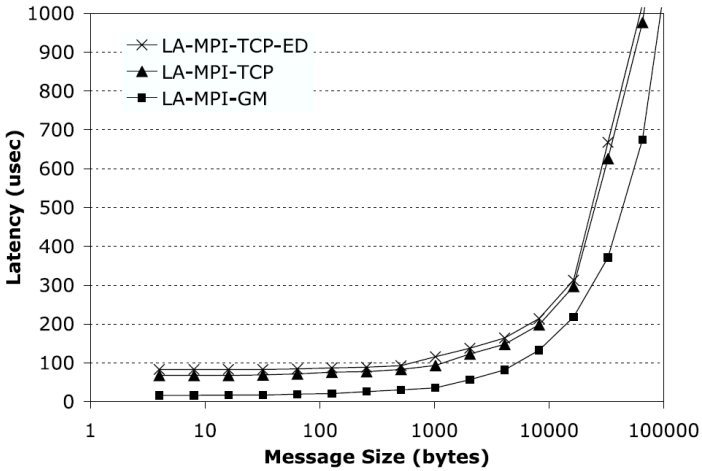

Fig. 5. Comparison of message ping latencies for Myrinet and TCP over Ethernet

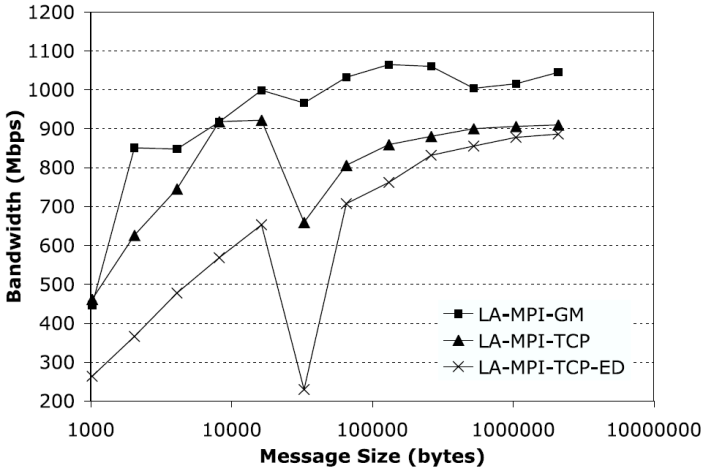

Fig. 6. Comparison of unidirectional message bandwidths for Myrinet and TCP over Ethernet

B. MPI Bandwidth & Latency

Figure 5 shows message latency as a function of message size for LA-MPI using both GM over Myrinet and TCP over Ethernet as a communication medium. This plot shows that the ping latency of messages over Myrinet is substantially lower than that over Ethernet. LA-MPI using TCP, LA-MPI-TCP, consistently has a significantly higher latency; about 50 microsec higher than LA-MPI-GM for 4 Byte messages and increasing steadily to about 420 microsec higher than LA-MPI-GM for 64 KB messages. When TCP is used in an event-driven fashion, LA-MPI-TCP-ED, there is a further latency increase of 15 microsec above LA-MPI-TCP. This latency increase is the result of thread switching overhead between the main thread and the event thread within the library.

All of the library versions use the *eager* message transfer protocol for messages up to 16 KB, and then switch to the *rendezvous* protocol for larger messages. This results in as much as a two-fold increase in message latency for 32 KB messages compared to 16 KB messages, as shown in Figure 5. In the *eager* protocol, messages are sent immediately to the receiver, requiring the receiver to be able to receive these messages whether they are expected or not. In the *rendezvous* protocol, the sender only sends a small fragment of the message at first, which serves as a request-to-send (RTS) message. The receiver must then respond with a clear-to-send (CTS) message before the sender can send the rest of the message. Typically, the receiver does not send the CTS until the corresponding receive gets posted by the application. This minimizes the required buffering on the receiver node, and prevents long messages from delaying other messages for which the receiver might be specifically waiting.

Figure 6 shows unidirectional messaging bandwidth as a function of message size for LA-MPI using both GM over Myrinet and TCP over Ethernet as a communication medium. The figure shows that the achieved messaging bandwidth for 1 KB messages is low for all of the library versions. For such small messages, the per-message overhead of both the MPI library and the communication substrate limit the achievable performance. As the message size increases beyond 1 KB, however, Myrinet enables the messaging bandwidth to increase faster than TCP over Ethernet. For *eager* messages (less than or equal to 16KB), increasing message size increases achievable bandwidth until the network saturates: the TCP messaging bandwidth saturates at around 930 Mbps, whereas the higher theoretical peak bandwidth of Myrinet enables it to attain up to 1100 Mbps. There is a sharp drop in messaging bandwidth in all cases as the message size increases from 16 KB to 32 KB, because of the switch from the *eager* protocol to the *rendezvous* protocol. For messages larger than 32 KB, all library versions again show a consistent increase in bandwidth with message size. For *rendezvous* messages, LA-MPI-GM achieves a peak unidirectional bandwidth of about 1050 Mbps (with 2 MB messages). With the same message size, the TCP versions of LA-MPI, achieve slightly lower maximum bandwidths of around 900 Mbps, with LA-MPI-TCP-ED about 20 Mbps lower than LA-MPI-TCP. LA-MPI-TCP-ED consistently sustains less bandwidth than LA-MPI-TCP because of thread switching overhead, and because the increased responsiveness of LA-MPI-TCP-ED leads to 100% unexpected receives on this simple benchmark. However, as the figure shows, the difference in performance between the two consistently decreases with increasing message size.

C. NAS Parallel Benchmarks Performance

The NAS benchmarks are run on a 4, 8, or 9 nodes depending on whether the benchmark requires a squared or power-of-2 number of nodes. The 4-node experiments are run with the mid-size data-set (B class) and the 8-node (or 9-node) experiments are run with both the B data-set and the larger C data-set.

Figure 7 compares the performance of different execution configurations of the five NAS benchmarks, achieved with LA-MPI-TCP, LA-MPI-TCP-ED, and LA-MPI-GM. Each set of three bars of the graph corresponds to one particular configuration of the benchmark and is named accordingly. The first part of name gives the benchmark name, the middle part conveys the data-set size used for that particular experiment, and the last part conveys the number of participating MPI

nodes for the experiment. Each bar of the graph depicts the normalized execution time of the corresponding benchmark using a particular library version relative to LA-MPI-GM. Thus, a bar of length less than 1.0 indicates a performance gain over the LA-MPI-GM library. Each data point for the graph is generated by averaging the execution times over 10 runs of each experiment.

Figure 7 shows that LA-MPI using TCP over Ethernet (LA-MPI-TCP) consistently performs worse than LA-MPI using GM over Myrinet (LA-MPI-GM) on all of the 5 NAS benchmarks. Overall, across the 15 benchmark configurations, LA-MPI-TCP is 5.2% slower than LA-MPI-GM on average. However, when TCP is used in a more efficient manner (LA-MPI-TCP-ED), it reduces the overall performance advantage of LA-MPI-GM across the 15 benchmark configurations to just over 0.3% on average. More interestingly however, LA-MPI-TCP-ED is able to match or beat LA-MPI-GM on 6 out of the 15 benchmark configurations, with a peak speedup of 7% on IS.C.8. Also, moving to a bigger cluster, or a larger data-set improves LA-MPI-TCP-ED with respect to LA-MPI-GM. Thus, as the communication component of a MPI application increases, a properly designed application is able to extract greater benefits from the increased concurrency between communication and computation, that the event-driven library provides. To ensure that the results are not biased towards any particular Myrinet implementation of the MPI library, these same experiments were also run with the Myrinet port of the MPICH MPI library (MPICH-GM). On average, the performance of these benchmarks with MPICH-GM differs from that with LA-MPI-GM by only 0.25%.

These results are especially interesting because they show that the LA-MPI-TCP-ED library using TCP messaging is able to match or outperform the Myrinet library version on several of the benchmark configurations, in spite of the significant latency and bandwidth advantage that Myrinet has over Gigabit Ethernet. LA-MPI-TCP-ED is able to outperform the other libraries because it is able to effectively overlap communication and computation. Longer messages using non-blocking MPI communication provide greater scope for an effective overlap, and thus the applications which use more of these messages show performance improvements with LA-MPI-TCP-ED.

V. Discussion & Analysis

The goal of this work was to gauge the position of COTS technology for high-performance SAN compared to specialized interconnects such as Myrinet and trying to answer the question that how can gigabit Ethernet and emerging 10 GbE be a viable alternative to specialized interconnect technologies. In this section, a brief survey is presented of the ongoing research efforts in achieving high-performance networking relying on Ethernet and/or by improving TCP/IP performance.

The Gigabit Ethernet results presented in Section IV are indicative of the problems associated with TCP/IP that become performance bottlenecks at beyond gigabit transmission rates. Initially, the primary source of overhead in TCP/IP processing was unclear, and developers assumed that the base protocol processing was the culprit. However, landmark research analyzing the associated overheads showed that TCP/IP protocol-specific processing by itself is not the main overhead [34]. The source for most of the overhead was the environment - interrupts, OS scheduling, buffering and data movement - in which the TCP/IP protocol operates. While this dissuaded researchers from replacing the TCP/IP protocol, overall processing continued to be expensive. The situation with emerging 10 Gigabit Ethernet is even more dire as indicated by studies that have shown to achieve maximum throughput of 4.11 Gbps under highly optimized conditions [4] and only 1.3 and 2.6 Gbps in another study [35].

Since the major overheads identified with TCP/IP are environment - interrupts, OS scheduling, buffering and data movement, many research efforts in high-performance networking have taken the Myrinet like protocol approach and concentrated on developing custom alternatives to the TCP/IP protocol stack for communication over Ethernet. These approaches are classified under user-level network protocol [14] that aim to benefit from OS-bypass and zero-copy communication. Shivam et al. proposed a message layer called Ethernet Message Passing for enabling high-performance MPI [36]. EMP implements custom messaging protocols directly in the network interface, and bypasses the entire protocol stack of the operating system. EMP supports reliable messaging and provides zero-copy I/O to user MPI applications for pre-posted receives. Experimental results show that EMP provides significantly lower message latency than TCP over Ethernet. In a later work, they also extended EMP to parallelize the receive processing to take advantage of a multiple CPU network interface and, showed further gains in MPI performance [37]. A similar work reporting much better performance than TCP/IP is GAMMA (the Genoa Active Message MAchine) by [38] which is a lightweight messaging system for Fast and Gigabit Ethernet. GAMMA also supports MPI based parallel applications through MPI/GAMMA interface. To benefit applications using the sockets interface, sockets over EMP [40] and GAMMA [39] have also been implemented and show better performance than TCP/IP sockets. Another work that aims for zero-copy solution in software by modifying the TCP/IP is called speculative defragmentation [41]. This work implement a defragmenting driver based on the same speculative techniques that are common to improve microprocessor performance with instruction-level parallelism.

TCP Offload Engines (TOE) [42] are also an being investigated as a means to enable highest possible TCP performance by offloading all or part of TCP processing onto an Ethernet network interface to reduce the processing load on the host processor, and reducing acknowledgment-related interrupts as a side-effect [44], [45]. Offloading TCP segmentation tasks can also result in significant performance benefits [46], [47]. The use of zero-copy sockets can improve TCP performance further by avoiding extraneous copies for message sends and posted message receives [34], [48], [49]. Since these techniques utilize the ubiquitous TCP/IP networking support present in all modern day operating systems, they can be easily deployed on a wide range of hardware and software platforms. Some promising results have been obtained by Intel [16] by

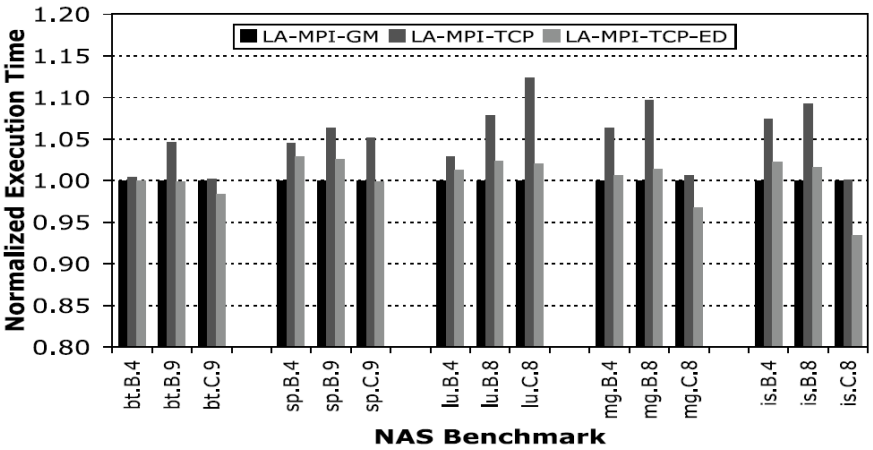

Fig. 7. Execution times of NAS benchmarks for the different MPI library versions normalized to LA-MPI- GM (bars shorter than 1 represent a speedup over LA-MPI-GM).

on-loading TCP processing on one of the cores of a chip multiprocessor (CMP) architecture. An example of TCP/IP full offload products is Chelsio Communication's 10 gigabit Ethernet adaptor T210 Protocol Engine [43] which achieves 7.9 Gbps performance.

TCP splintering is another research work aimed at improving the performance of TCP messaging over Ethernet [50]. This work focuses on the limitations of TCPs congestion-control and flow-control policies in a cluster environment, and proposes offloading parts of the protocol stack to the network interface to provide improved MPI performance. Specifically, it targets congestion control and acknowledgment generation for offloading to the network interface in order to reduce the CPU overhead of message-passing, and improve message latency.

VI. Conclusion

Comparative performance of gigabit Ethernet and Myrinet is studied in this paper. It is found that with efficient implementations of protocols running over COTS Ethernet hardware, performance close to Myrinet can be achieved at a lower cost-performance ratio. As the NAS benchmarks show, the library's ability to allow overlapping communication and computation is equally important as raw latency and bandwidth. With comparable networking technologies, 1 Gbps Ethernet and 1.2 Gbps Myrinet, several (6 out of 15 configurations) of the NAS benchmarks run faster (by as much as 7%) when the MPI library uses TCP over Ethernet for communication rather than GM over Myrinet. More importantly, the efficient use of TCP communication by the LA-MPI library reduces the performance advantage of GM over Ethernet from above 5% to around 0.3%. While the most recent Myrinet networks provide higher bandwidth, multiple Gigabit Ethernet links should provide competitive performance. Furthermore, 10 Gbps Ethernet should extend Ethernet's performance advantage. In addition, there are several other optimizations that could further improve the performance of TCP over Ethernet as a communication substrate for MPI messaging. The latency gap between Ethernet and Myrinet is largely due to the memory copies required to move data between the application and the kernel, and the overhead of interrupt processing. The user-level communication protocols employed by specialized networks, including Myrinet and Quadrics, avoid these copies by directly transferring data between the network interface and application buffers, resulting in lower communication latencies. However, these same techniques can be integrated into commodity protocols like TCP. Furthermore, techniques such as the event-driven LA-MPI library merely harness the capabilities of the TCP/IP protocol stack more efficiently, enabling complete independence of these techniques to the underlying TCP protocol stack implementation. The use of copy avoidance, interrupt avoidance, and TCP offload together with an efficient TCP path in the MPI library can make TCP/IP over Ethernet a low-cost, portable, and reliable alternative to specialized networks. Moreover, these TCP enhancements would simultaneously benefit all networking applications that rely on TCP for communication.

References

[1] M. Baker and R. Buyya and D. Hide, *Cluster Computing: A High Performance Contender*, IEEE Computer, 32(7):79-83, July 1999.

[2] D. J. Baker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, C. V. Packer, *Beowulf: A Parallel Workstation for Scientific Computation*, Proceedings of International Conference on Parallel Processing, CRC Press, Boca Raton, FL, USA, August 1995.

[3] Steven J. Vaughan-Nichols, *Will 10-Gigabit Ethernet Have a Bright Future?*, IEEE Computer, 22-24, June 2002.

[4] Justin (Gus) Hurwitz, Wu-chun Feng, *End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems*, IEEE Micro, 10-22, January 2004.

[5] N. J. Boden et al., *Myrinet: A Gigabit-per-Second Local Area Network*, IEEE Micro, 15(1):29-36, January 1995.

[6] F. Petrini et al., *The Quadrics Network: High-Performance Clustering Technology*, IEEE Micro, 22(1):46-57, January 2002.

[7] H. Hallwagner, *The SCI Standard and Applications of SCI*, SCI: Scalable Coherent Interface, LNCS vol. 1291, Springer Verlag, pp. 95-116, 1999.

[8] D. Dunning et al., *The Virtual Interface Architecture*, IEEE Micro, pp. 66-76, March 1998.

[9] D. Cassiday, *InfiniBand Architecture Tutorial*, Hot Chips 12, August 2000.

[10] D. Bailey et al., *The NAS Parallel Benchmarks 2.0*, Technical Report NAS-95020, 1995.

[11] IEEE Std 802.3z-1998, *Media Access Control (MAC) Parameters, Physical Layer, Repeater and Management Parameters for 1000 Mbps Operation*, IEEE Press, Piscataway, N.J., 1998.

[12] Howard Frazier, Howard Johnson, *Gigabit Ethernet: From 100 to 1000 Mbps*, IEEE Internet Computing, pp. 24-31, January 1999.
[13] J. Chase et al., *End-System Optimizations for High-Speed TCP*, IEEE Communications, Special Issue on High-Speed TCP, June 2000.
[14] Raoul A. F.Bhoedjang et al., *User-Level Network Interface Protocols*, IEEE Computer, pp. 53-59, November 1998.
[15] W. Ligon, W. McMillan, R. Ross, *Tuning TCP performance in beowulf computers*, Parallel Architecture Research Laboratory, Clemson University, 1999, citeseer.ist.psu.edu/ligon99tuning.html.
[16] Greg Regnier et al., *TCP Onloading for Data Center Servers*, IEEE Computer, pp. 48-58, November 2004.
[17] Myricom Inc., *www.myricom.com*.
[18] R. L. Graham et al., *A Network-Failure-Tolerant Message- Passing System for Terascacle Clusters*, In Proceedings of the 16th Annual ACM International Conference on Supercomputing, June 2002.
[19] S. Majumder, S. Rixner, and V. S. Pai, *An Event- driven Architecture for MPI Libraries.* In Proceedings of the Los ALamos Computer Science Institute Sympo- sium (LACSI'04), October 2004.
[20] *NTTCP: New TTCP program*, http://www.leo.org/ẽlmar/nttcp/.
[21] *Iperf 1.6 - The TCP/UDP Bandwidth Measurement Tool*, http://dast.nlanr.net/Projects/Iperf/.
[22] *Netperf: Public Netperf Homepage*, http://www.netperf.org/.
[23] *NetPIPE*, http://www.scl.ameslab.gov/netpipe/.
[24] Quinn O. Snell, Armin R. Mikler, John L. Gustafson, *NetPIPE:A Network Protocol Independent Performance Evaluator*, IASTED Conference, http://www.scl.ameslab.gov/netpipe/.
[25] Gustafson, J. and Snell, Q., *HINT: A New Way to Measure Computer Performance*, Proceedings of the 28th Annual Hawaii International Conference on Systems Sciences, IEEE Computer Society Press, Vol. 2, pp. 392-401.
[26] *MPICH Home Page*, http://www-unix.mcs.anl.gov/mpi/mpich/.
[27] *LAM/MPI*, http://www.lam-mpi.org/.
[28] *MPI/PRO*, http://www.mpi-softtech.com/.
[29] *MPI_Lite*, http://www.scl.ameslab.gov/Projects/MP_Lite/.
[30] *Intel MPI Benchmarks 2.3*, http://www.intel.com/software/products/cluster/clustertoolkit/.
[31] *MPIbench Home Page*, http://icl.cs.utk.edu/projects/llcbench/mpbench.html.
[32] *Performance Tests*, http://www-unix.mcs.anl.gov/mpi/mpich/download.html.
[33] *SKaMPI Home Page*, http://liinwww.ira.uka.de/s̃kampi/.
[34] Clark, D. D., Jacobson, V., Romkey, J., Salwen, H., *An Analysis of TCP Processing Overheads*, IEEE Communication Magazine, Vol. 27, No. 2, June 1989, pp. 23-29.
[35] Sven Ubik, *Field trial with Intel 10 Gigabit Ethernet adapters for PC*, CESNET technical report number 10/2003, CESNET, Prague, Czech Republic, October 2003.
[36] P. Shivam, P. Wyckoff, and D. Panda. *EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing*. In Proceedings of SC2001, Nov 2001.
[37] P. Shivam, P. Wyckoff, and D. Panda. *Can User Level Protocols Take Advantage of Multi-CPU NICs?*, In Proceedings of IPDPS2002, Apr 2002.
[38] G. Ciaccio, M. Ehlert, B. Schnor, *Exploiting Gigabit Ethernet Capacity for Cluster Applications*, 27th Annual IEEE Conference on Local Computer Networks (LCN'02).
[39] Schneidenbach, L.; Schnor, B.; Petri, S., *Architecture and Implementation of the Socket Interface on Top of GAMMA*, Local Computer Networks, 2003. LCN'03. Proceedings. 28th Annual IEEE International Conference, Oct. 2003 pp. 528-536
[40] Pavan Balaji et al., *High Performance User Level Sockets over Gigabit Ethernet*, IEEE International Conference on Cluster Computing (CLUSTER'02).
[41] Christian Kurmann, Felix Rauch, Thomas M. Stricker, *Speculative Defragmentation - Leading Gigabit Ethernet to True Zero-Copy Communication*, Cluster Computing Journal, 4(1):7-18, Kluwer Academic Publishers, March 2001.
[42] Andy Currid, *TCP Offload to the Rescue*, ACM Queue, pp. 59-65, May 2004.
[43] *Chelsio Communications*, http://www.chelsio.com/.
[44] H. Bilic, Y. Birk, I. Chirashnya, and Z. Machulsky. *Deferred Segmentation for Wire-Speed Transmission of Large TCP Frames over Standard GbE Networks*. In Hot Interconnects IX, pages 8185, Aug. 2001.
[45] Y. Hoskote, B. A. Bloechel, G. E. Dermer, V. Erra- guntla, D. Finan, J. Howard, D. Klowden, S. G. Naren- dra, G. Ruhl, J. W. Tschanz, S. Vangal, V. Veera-machaneni, H. Wilson, J. Xu, and N. Borkar. *A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS. IEEE Journal of Solid-State Circuits*, 38(11):18661875, Nov. 2003.
[46] S. Makineni and R. Iyer. *Architectural characterization of TCP/IP packet processing on the Pentium M microprocessor*. In International Symposium on High-Performance Computer Architecture, pages 152162, Feb. 2004.
[47] D. Minturn, G. Regnier, J. Krueger, R. Iyer, and S. Makineni. *Addressing TCP/IP processing challenges using the IA and IXP processors*. Intel Technology Journal, 7(4), 2003.
[48] A. Gallatin, J. Chase, and K. Yocum. *Trapeze/IP: TCP/IP at near-gigabit speeds*. In Proceedings of 1999 USENIX Technical Conference, pages 109120, June 1999.
[49] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. *High-Performance Local Area Communication With Fast Sockets*. In Proceedings of the 1997 USENIX Technical Conference, pages 257274, Jan. 1997.
[50] P. Gilfeather and A. B. Macabe. *Making TCP Viable as a High Performance Computing Protocol*. In Proceedings of the Third LACSI Symposium, Oct 2002.