# Bridging the Ethernet-Ethernot Performance Gap

Ethernet seeks to reach the system-area-network performance of Ethernot networks by introducing 10-Gigabit Ethernet and adding the hardware-offloaded protocol stacks that give Ethernots their performance edge. As this study shows, such moves might indeed be closing the gap.

**Pavan Balaji**
Ohio State University

**Wu-chun Feng**
Virginia Tech

**Dhabaleswar K. Panda**
Ohio State University

In the past decade, a wide array of interconnect technologies have entered the system-area network (SAN) environment—notably InfiniBand (http://www.infinibandta.org), Myrinet,[1] and QsNet.[2] The success of these networks, dubbed Ether*nots,* depends primarily on their use of hardware-offloaded protocol stacks and feature-rich interfaces that are exposed to end-host applications. In a wide-area network (WAN), however, Ethernots are less successful because they are incompatible with existing infrastructure (switches and routers, for example). In this realm, Ethernet and Ethernet-compatible technologies such as Sonet/Synchronous Digital Hierarchy are ubiquitous. Coupled with the traditional IP-Ethernet infrastructure, which dates to the early 1970s, these technologies support not only the Internet but also the emerging environments that overlay it—computational grids and peer-to-peer networking, among others.

Ethernet's foothold in WANs will become even stronger as long-haul network providers move from the more expensive Sonet to the newer 10-Gigabit Ethernet (10GigE)[3,4] backbones. Indeed, in late 2004, the longest continuous 10GigE connection was established between Tokyo, Japan, and Switzerland via Canada and the US (http://www.gridtoday.com/04/1206/104373.html). This 18,500-km 10GigE connection used 10GigE WAN physical layer technology to set up a pseudo local-area network (LAN) at the University of Tokyo that appeared to include systems 17 time zones away. Ethernet—albeit 1GigE—also dominates the Top500 supercomputer list (http://www.top500.org).

Recent developments could also empower Ethernet to penetrate the SAN arena, where it has struggled against the Ethernots' serious performance advantage. Despite that, network providers have been reluctant to view it as a serious SAN interconnect because of Ethernet's formidable performance lag relative to mainstream SAN technologies.

With 10GigE, this could change. Ethernet's recent developments to move closer to Ethernot performance, and Ethernot's adoption of Ethernet-compatible technology might make this performance lag much less of an issue. Myricom, Myrinet's vendor, recently introduced network adapters that use Myrinet interconnect technology but implement Ethernet as the underlying wire protocol. Similarly, Quadrics introduced network switches that use the company's



Published by the IEEE Computer Society

technology but apply it to the Ethernet market.

For their part, Ethernet vendors have introduced 10GigE TCP-IP offload engines (TOEs)[5] as hard evidence of their desire to move toward the performance capabilities of Ethernot networks. The question is whether 10GigE can bridge the Ethernet-Ethernot performance gap, yet retain Ethernet's ease of deployment and low cost. For ease of deployment at least, the outlook is optimistic. The IEEE 802.3ae 10-Gbps standard, which the 10GigE Alliance supports, already ensures interoperability with existing IP-Ethernet infrastructure, and the manufacturing volume of 10GigE is driving costs down exponentially, just as it did for Fast Ethernet and Gigabit Ethernet. Indeed, per-port costs for 10GigE have dropped tenfold in the past two years.

Convergence in the performance of 10GigE and traditional Ethernot technologies—the focus of our study—also looks promising.

## Defining the performance gap

With so many networking technologies in the current high-speed network market, characterizing the performance gap among them is not straightforward. Each technology exposes its own communication interface, which affects both lower level performance characterization and application development. With each new technology, lack of portability threatened to become a huge issue, and application developers were quick to demand a common interface to rectify the problem. The message passing interface (MPI) and the sockets interface are two of the more popular solutions: MPI is the de facto standard for scientific applications, while sockets are prominent in traditional scientific applications as well as in grid and peer-to-peer computing; file and storage systems; and other commercial applications, including online transaction processing.

Because the solution of traditional sockets over host-based TCP-IP has not been able to cope with increasing network speed, Ethernot technologies, specifically InfiniBand (IBA) and Myrinet, proposed high-performance sockets implementations such as the Sockets Direct Protocol, or SDP (http://www.rdmaconsortium.org). SDP lets existing sockets-based applications transparently exploit the hardware-offloaded protocol stack that these networks provide. As a result, from the Ethernet side, Chelsio and other 10GigE vendors have recently released adapters that deliver hardware-offloaded TCP-IP protocol stacks to provide high-performance support for existing sockets-based applications. One such protocol stack is the TCP-IP offload engine (TOE).

Our study focused on the sockets interface. We first compared the performance of the host-based TCP-IP stack over 10GigE to that of 10GigE TOEs so that we could understand the performance gains achievable through the use of hardware-offloaded protocol stacks for 10GigE. We then compared the performance of 10GigE TOEs with that of other interconnects providing similar hardware-offloaded protocol stacks such as IBA and Myrinet. Although QsNet provides a similar hardware-offloaded protocol stack, there is no mechanism to use it transparently for sockets-based applications; that is, there is no SDP implementation. Consequently, we did not include this network in our evaluation.

We evaluated the performance of 10GigE, IBA, and Myrinet at both a detailed *microbenchmark* level and an *application* level with sample applications from multiple domains.

## Protocol offload engines

Traditionally, the job of processing protocols such as TCP-IP has fallen to the software running on the host CPU. Recently, network speeds have outpaced the CPU, which has become burdened with resource-intensive memory copies, checksum computation, interrupts, and reassembly of out-of-order packets—all part of protocol processing's heavy load. In high-speed networks, the CPU ends up dedicating more cycles to network traffic handling than to the applications it is running.

Protocol offload engines (POEs) are emerging as a solution to limit the processing that CPUs require for networking. The basic idea of a POE is to offload protocol processing from the host CPU to the network adapter. Providers can implement a POE with a network processor and firmware, specialized application-specific ICs, or a combination. High-performance networks such as IBA and Myrinet provide their own protocol stacks that are offloaded onto the network-adapter

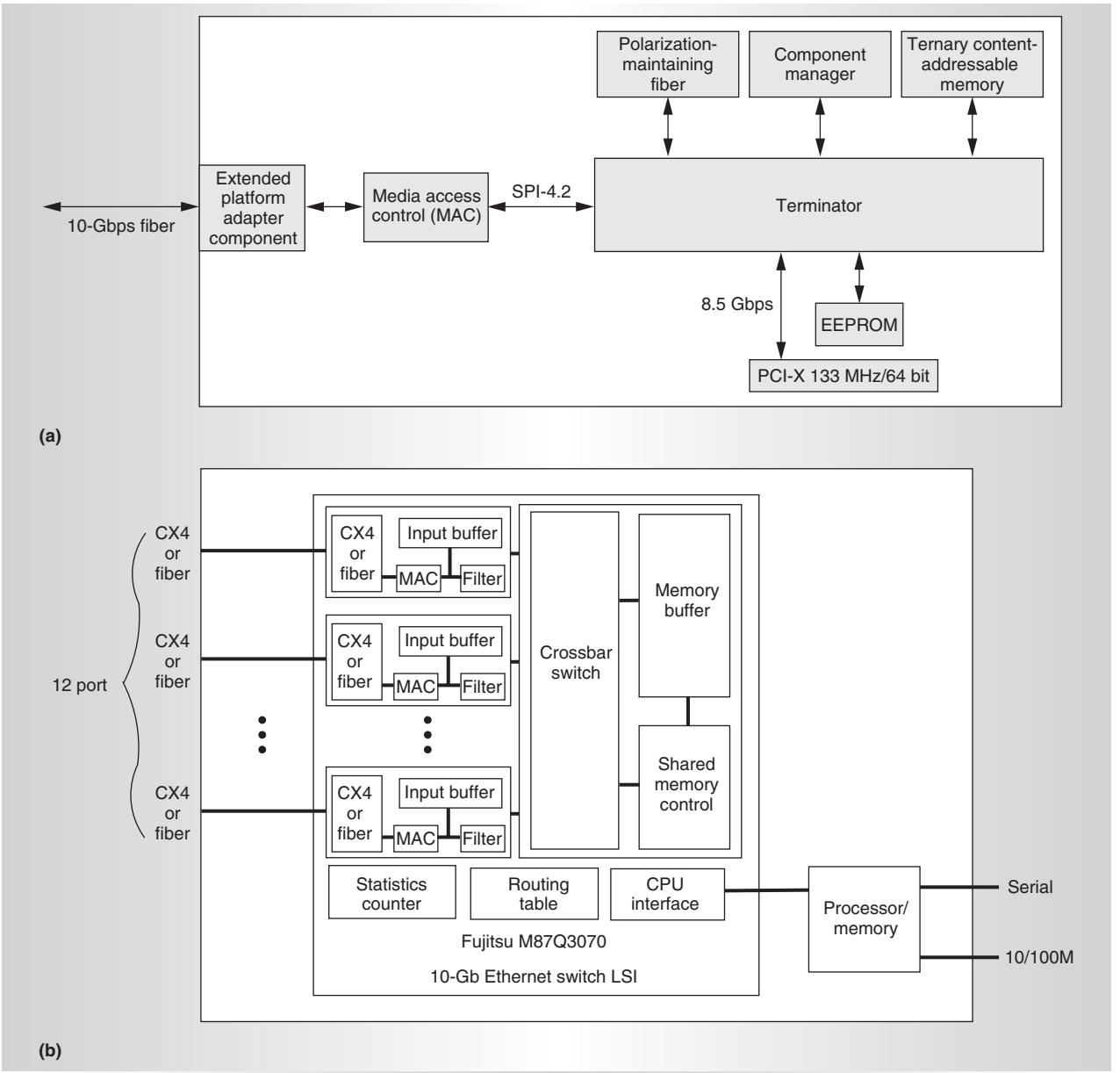


Figure 1. Two foundational hardware elements of the 10GigE Network: Chelsio T110 adapter architecture (a) and 12-port Fujitsu XG800 switch architecture (b).

hardware. Many 10GigE vendors, on the other hand, have chosen to offload the ubiquitous TCP-IP stack so as to maintain compatibility with the traditional IP-Ethernet infrastructure, particularly over the WAN. For this reason, a special case of POE has evolved—the TOE.

## 10GigE

The 10GigE infrastructure that we evaluated consists of two foundational hardware blocks: the Chelsio T110 TOE-based network adapter and the Fujitsu XG800 virtual cut-through switch.

As Figure 1a shows, the Chelsio T110 is a PCI-X network adapter that can support complete TCP-IP offloading from a host system at line speeds of 10 Gbps. The adapter consists of the terminator, which provides the basis for offloading; separate memory systems, each designed for holding particular types of data; media access control (MAC); and an extend-

ed platform adapter component (XPAC) optical transceiver for physically transferring data over the line.

The 10GigE infrastructure interconnects the Chelsio T110 network adapters using a Fujitsu XG800 virtual cut-through switch. Figure 1b shows a functional block diagram of the switch, which features nonblocking layer-2 switching for 12 10GigE ports with 450-ns flow-through latency. The XG800 switch is also unique in that it uses the Fujitsu MB87Q3070 switch-on-a-chip, which significantly reduces the switch footprint.

### InfiniBand

IBA defines a switched network fabric to interconnect processing and I/O nodes that provides the communication and management infrastructure for interprocessor communication and I/O. Host-channel adapters (HCAs) that reside on the processing or I/O nodes connect network nodes to the fabric.

In our study, we evaluated the InfiniScale switch from Mellanox Technologies, a full wire-speed 24-port switch that supports link-packet buffering, inbound and outbound partition checking, and automatic negotiation of link speed. The switch has an embedded reduced instruction-set computer (RISC) processor for exception handling, out-of-band data-management support, and support for counters to allow performance monitoring. The InfiniHost HCA connects to the host through the PCI-X bus and delivers bandwidth of up to 8 Gbps over its ports. The hardware implements both memory protection and address translation. The HCA supports on-board dual data rate (DDR) memory up to 1 Gbyte.

### Myrinet

Myrinet is a high-speed network that uses wormhole-routed crossbar switches to connect all the network adapters. MX and GM, two of Myrinet's low-level messaging layers, provide protected user-level access to the adapters and ensure reliable, in-order message delivery. The Myrinet network in our evaluation consists of a Myrinet-2000 switch—a 16-port crossbar—that connects Myrinet-2000 E cards, each of which has two ports, and each port has a 2-Gbps bandwidth. Thus, the network adapter can support an aggregate of 4 Gbps in each direction using both ports. The network adapter connects to a 133-MHz, 64-bit PCI-X interface on the host. It has a programmable Lanai-XP processor running at 333 MHz with a 2-Mbyte on-board synchronous RAM (SRAM). The Lanai processor can access host memory via the PCI-X bus through the direct memory access (DMA) controller.

## Interfacing with protocol offload engines

Because the Linux kernel does not currently support POEs, many researchers have studied ways of enabling applications to interface with POEs. The two predominant approaches are high-performance sockets implementations (such as SDP) and TCP stack override.

### High-performance sockets

High-performance sockets are pseudo sockets implementations built around two goals. The first is to provide a smooth transition in deploying existing sockets-based applications on clusters connected with networks using offloaded protocol stacks. The second is to use the offloaded stack for protocol processing, which lets applications tap into most of the raw network performance. As Figure 2a shows, these sockets layers override the existing kernel-based sockets and force transfer of the data directly to the offloaded stack. SDP is an industry-standard specification for such high-performance sockets implementations.

In the high-performance sockets approach, the TCP-IP stack in the kernel does not have to be touched at all because all the data communication calls (read, write, and so on) are trapped and directly mapped to the offloaded protocol stack.

However, this requires duplicating functionality that the sockets layer handles (such as buffer management for data retransmission and pinning of buffers) in the high-performance sockets implementation. IBA and Myrinet use this approach so that sockets-based applications can use the offloaded protocol stacks.

### TCP stack override

The second approach, which Figure 2b shows, retains the kernel-based sockets layer but bypasses the host TCP-IP stack and pushes the data directly to the offloaded protocol stack. The Chelsio T110 10GigE adapter takes this approach.

Because the Linux operating system lacks

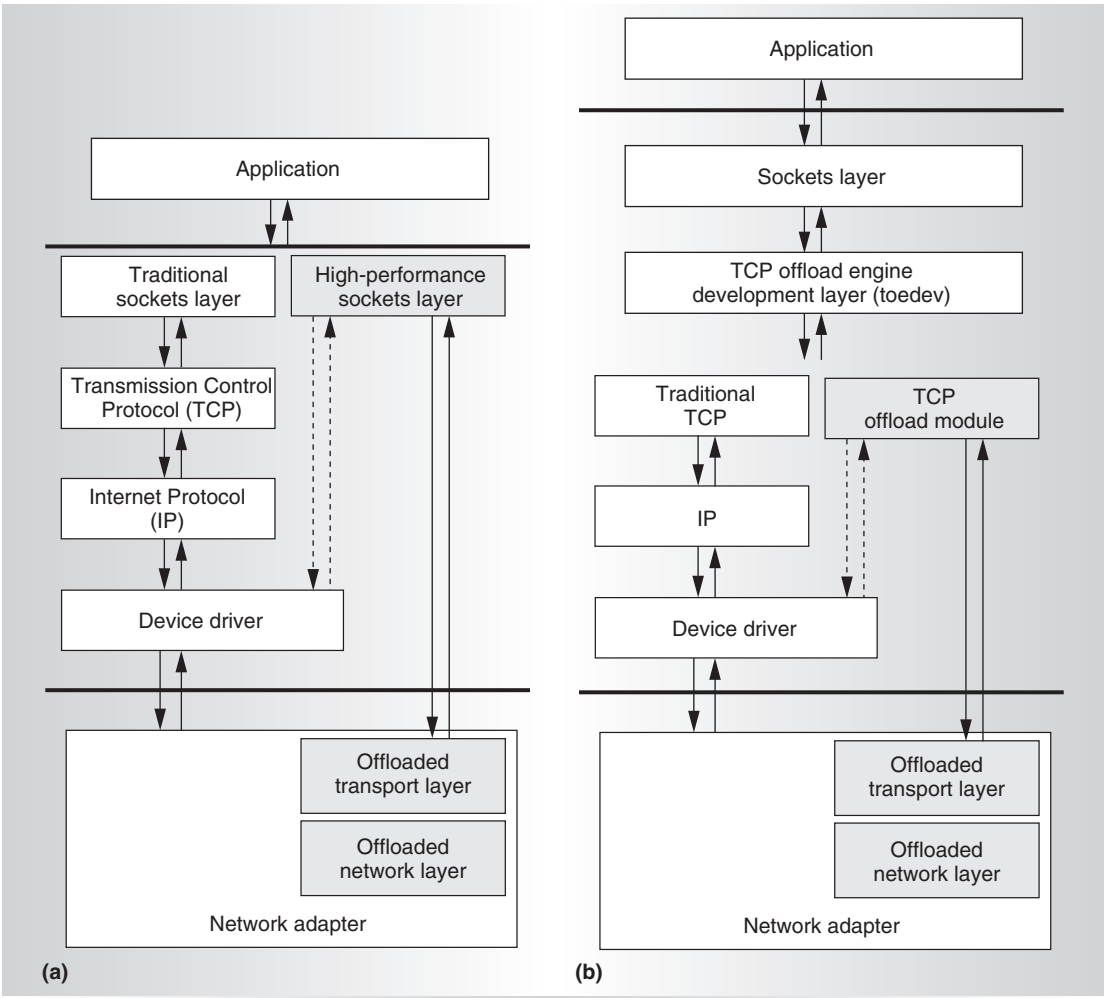


Figure 2. Interfacing with POEs: high-performance sockets (a) and TCP stack override (b).

support for TOE devices, Chelsio provides a TCP offload module (TOM) framework and toedev, a thin layer that decides if a connection goes to the TOM or the traditional host-based TCP-IP stack. The TOM is responsible for implementing the TCP processing that the TOE cannot do. It also maintains the state of all offloaded connections.

The advantage of this approach is that it does not require any duplication of the socket layer's functionality. The disadvantage is that an application cannot use this approach without modifications to the kernel.

## Experimental testbed

To evaluate the performance of these three networks, we ran the study on three experimental clusters.

*Cluster 1* consists of two Opteron 248 nodes, each with a 2.2-GHz CPU along with 1 Gbyte of a 400-MHz DDR synchronous, dynamic RAM (SDRAM) and 1 Mbyte of L2 cache. The nodes connect back-to-back with the Chelsio T110 10GigE adapters.

*Cluster 2* consists of four Opteron 846 nodes, each with four 2.0-GHz CPUs (quad systems) along with 4 Gbytes of 333-MHz DDR SDRAM and 1 Mbyte of L2 cache. Nodes connect via a 12-port Fujitsu XG800 10GigE switch with Chelsio T110 10GigE adapters at the end hosts.

We performed experiments on Clusters 1 and 2 with the SuSE Linux distribution installed with kernel 2.6.6 (patched with Chelsio modules). We used these clusters to compare the performance of the host-based TCP-IP stack on 10GigE with that of the 10GigE TOEs. For this comparison, in gen-

eral, we used Cluster 1 for all experiments requiring only two nodes and Cluster 2 for those requiring more.

*Cluster 3* consists of four nodes built around SuperMicro's Super X5DL8-GG motherboards with ServerWorks GC LE chipsets, which include 64-bit, 133-MHz PCI-X interfaces. Each node has two Intel Xeon 3.0-GHz processors with a 512-Kbyte L2 cache, 533-MHz front-side bus, and 2 Gbytes of 266-MHz DDR SDRAM. We used the RedHat 9.0 Linux distribution and the Linux-2.4.25smp kernel. Each node also had the 10GigE, IBA, and Myrinet networks. We used this cluster to compare 10GigE, IBA, and Myrinet.

The 10GigE network in Cluster 3 is based on Chelsio T110 10GigE adapters with TOEs connected to a 12-port Fujitsu XG800 switch. The driver version on the network adapters is 1.2.0. The IBA network is based on InfiniHost MT23108 dual-port 4x HCAs through an InfiniScale MT43132 24-port nonblocking switch. The adapter firmware version is fw-23108-rel-3 2 0-rc4-build-001, and the software stack is based on the Voltaire IBHost-3.0.0-16 stack. Research groups, including some at Mellanox Technologies[6] and Ohio State University,[7] have recently implemented research prototypes for zero-copy implementations of SDP over IBA, but these implementations tend to be less stable than the more widely available buffered-copy implementation of SDP that we used in our study.

Finally, the Myrinet network in Cluster 3 is based on Myrinet-2000 E (dual-port) adapters connected by a Myrinet-2000 wormhole-routed crossbar switch. Each adapter is capable of a 4-Gbps bandwidth in each direction. For SDP/Myrinet, we performed evaluations with two implementations. The first uses the GM/Myrinet drivers (SDP/GM v1.7.9 over GM v2.1.9). The second implementation runs over the newly released MX/Myrinet drivers (SDP/MX v1.0.2 over MX v1.0.0). The SDP/MX implementation achieves significantly better performance than the older SDP/GM, but being a very new implementation, SDP/MX comes with its share of stability issues. Because of this, we conducted only the ping-pong latency and unidirectional bandwidth tests for both SDP/MX and SDP/GM, but the rest of the tests were for SDP/GM alone. With Myricom's current effort on SDP/MX, we expect these stability issues to be resolved very soon; consequently, the Myrinet results should improve.

For all evaluations, we ran each experiment 10 times, dropped the highest and lowest values, and took the mean of only the remaining eight runs. For microbenchmark evaluations, each run consisted of 100,000 iterations.

## Host-based TCP-IP versus TCP offloading engine

To evaluate the performance of 10GigE with TOE as compared with that of the host-based TCP-IP stack over 10GigE—hereafter, TOE and non-TOE—we used a suite of microbenchmarks. We first performed evaluations on the basis of a single connection measuring the point-to-point latency and unidirectional bandwidth together with the CPU utilization. We then performed evaluations on the basis of multiple connections using the multistream, hot-spot, fan-in, and fan-out tests.

### Single-connection microbenchmarks

Figures 3 and 4 show the basic single-stream performance (point-to-point latency and unidirectional bandwidth) of the 10GigE TOE as compared to non-TOE. Point-to-point latency is the time a sockets application takes to transfer $X$ bytes of data. In this experiment, the sender application process sends $X$ bytes of data to the receiver process; the receiver process, upon receipt of this data, returns $X$ bytes of data to the sender process. We repeated this for $N$ iterations and calculated the average. We measured the one-way point-to-point latency as half the average value (across the $N$ iterations) and reported it.

For the unidirectional bandwidth experiment, the sender process continuously sends $N$ messages to the receiver process, each message containing $X$ bytes of data. The receiver process, on receipt of all $N \times X$ data bytes, sends an acknowledgment message to the sender process. The sender process calculates the total time measured from just before it started sending the data until it received the acknowledgment. Subtracting the one-way latency for the acknowledgment message from this value gives the total time taken to transfer

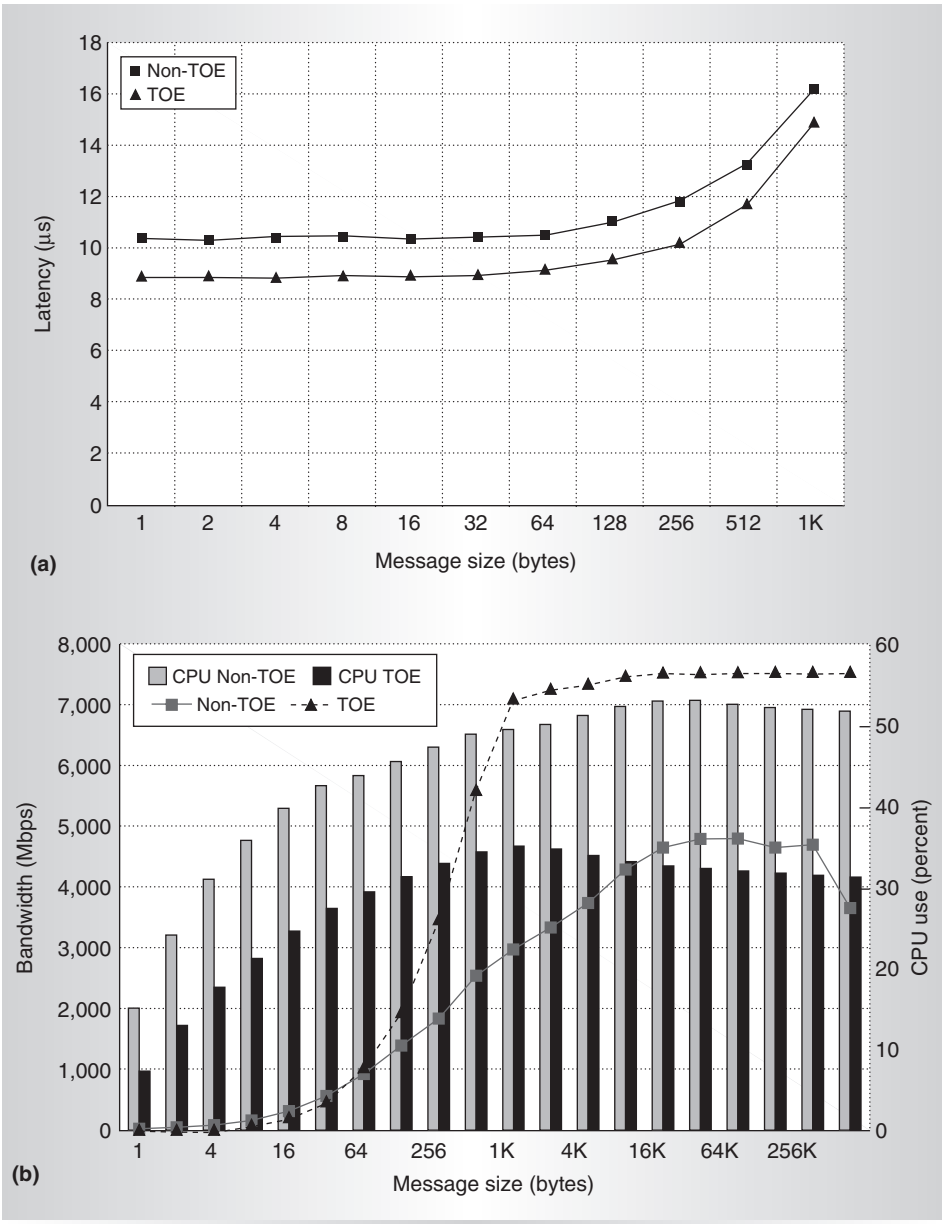


Figure 3. Sockets-level microbenchmarks with a maximum transmission unit (MTU) of 1.5 Kbytes: ping-pong latency (a) and unidirectional bandwidth (b). All bandwidth results refer to the application data transferred per second and exclude TCP, IP, and Ethernet headers.

the $N \times X$ bytes. We calculated the unidirectional bandwidth as the total amount of data transferred divided by the total time taken.

As Figure 3a shows, for a standard Ethernet frame size of 1.5 Kbytes, the 10GigE TOE can achieve a point-to-point latency of about 8.9 µs compared with the 10.37 µs achievable with a non-TOE—an improvement of about 14.2 percent. As Figure 3b shows, the TOE achieves a maximum bandwidth of 7.6 Gbps as compared with the 5 Gbps for a non-TOE—an improvement of about 52 percent.

Figure 4 shows the results of increasing the adapter's maximum transmission unit (MTU) size to 9 Kbytes (jumbo frames). The non-TOE bandwidth increases to 7.2 Gbps, but there is no additional improvement for the TOE because of the way it handles message transmission; the device driver hands over large message chunks (16 Kbytes) to the network adapter, which segments the chunks into frames sized for the MTU. This causes only a few interrupts, which the host processor receives regardless of MTU size. In other words, the TOE shields the host from the overhead of smaller MTU sizes; but for non-TOE, an MTU of 1.5 Kbytes results in more segments and correspondingly more interrupts that must be handled for every message, yielding lower performance relative to an MTU of 9 Kbytes.

Figures 3b and 4b also show the CPU utilization. For TOE, utilization remains close to 38 percent with an MTU of 1.5 Kbytes as well as 9 Kbytes. But for non-TOE, the CPU utilization increases slightly (52 to 58 percent) when moving from standard (1.5-Kbyte) to jumbo (9-Kbyte) frames. The stack implementation explains some of this trend. When the application calls a `write` call, the host CPU copies the data into the socket buffer. If there is no space in the socket buffer, the CPU waits for the network adapter to complete its send of the existing data and create space for the new data to be copied. Once the data is copied, the underlying TCP-IP stack handles the actual data transmission. If the network adapter sends the data out faster,

space is created in the socket buffer faster, and the host CPU spends a larger fraction of its time copying data to the socket buffer rather than waiting for space to be created. Thus, when performance improves, we expect the host CPU to spend more time copying data and using CPU cycles. But the use of jumbo frames also reduces the CPU overhead for non-TOE because there are fewer interrupts. Because of these two conditions, we found only a 6 percent increase in CPU utilization with jumbo frames.

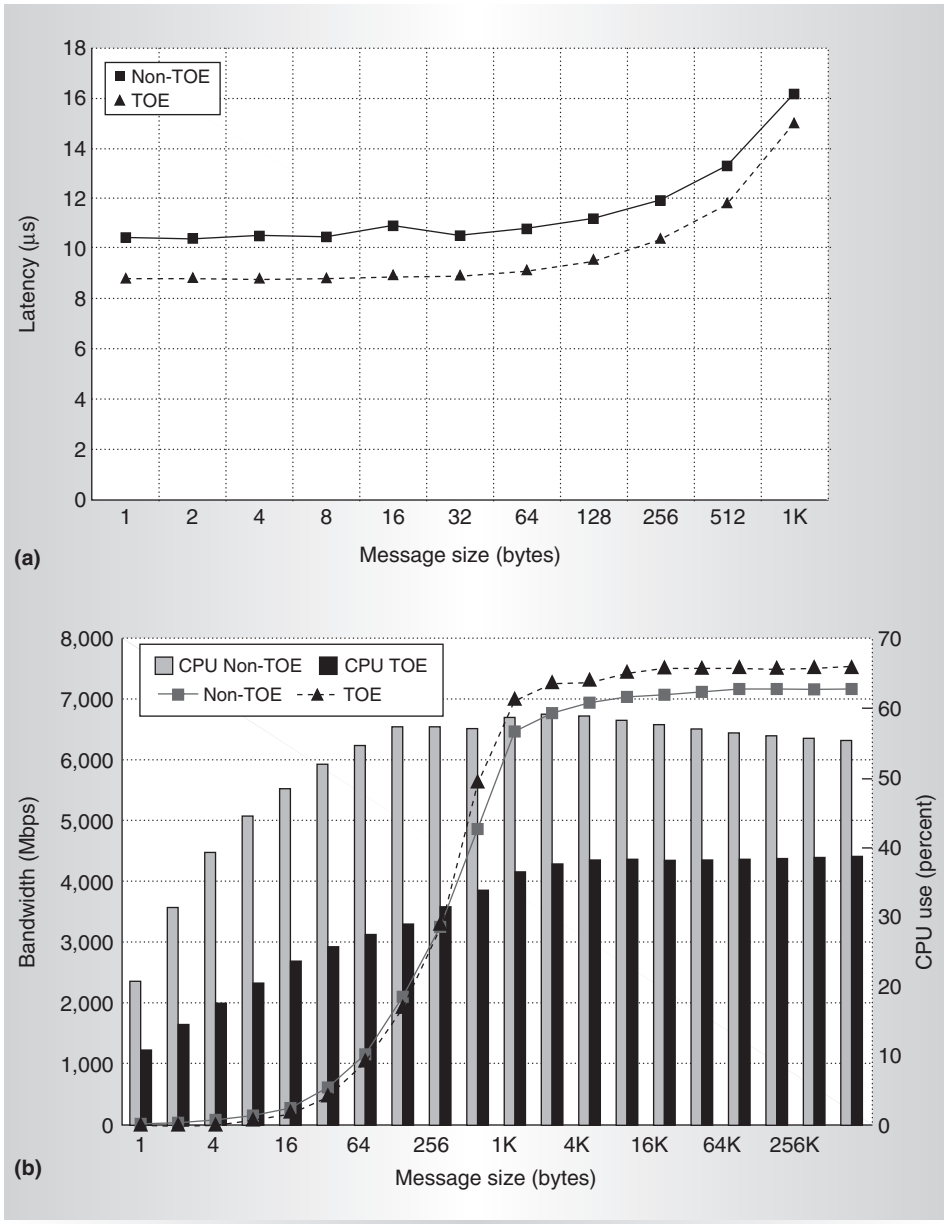


Figure 4. Sockets-level microbenchmarks (MTU of 9 Kbytes): ping-pong latency (a) and unidirectional bandwidth (b).

## Multiple-connection microbenchmarks

Our next evaluations were for TOE and non-TOE networks with microbenchmarks that use multiple simultaneous connections. For all these experiments, we used an MTU of 1.5 Kbytes to abide by the standard Ethernet frame size.

Figure 5a shows the aggregate bandwidth for two nodes in Cluster 1 simultaneously executing multiple instances of the unidirectional bandwidth test. The TOE network achieved 7.1 Gbps to 7.6 Gbps (equally divided between each thread). The non-TOE stack peaked at 4.9 Gbps (again, equally divided between each thread). These results are similar to the single-stream results; thus, using multiple simultaneous streams to transfer data does not make much difference.

Figure 5b shows the impact of multiple connections on small-message transactions. In this experiment, client nodes performed a point-to-point latency test with the same server, forming a hot spot. We performed this experiment on Cluster 2 with one node acting as a server node and each of the other three nodes hosting 12 client processes in all. We allotted the clients cyclically, so "three clients" refers to one client per node, "six clients" refers to two clients per node, and so on.

As Figure 5b shows, both the TOE and non-TOE networks show similar scalability as clients increase. We can thus deduce that the TOE performs lookup for connection-related data structures efficiently enough to avoid a significant bottleneck.

Although results from the hot-spot test show that the lookup time for connection-

related data structures is quite efficient on the TOE, the test did not stress the other resources on the network adapter, such as management of memory regions for buffering data during transmission and reception. To test these other resources, we performed the fan-in and fan-out tests. In both the fan-in and fan-out bandwidth tests, which we performed on Cluster 2, one server process carries out unidirectional bandwidth tests simultaneously with multiple client processes. In the fan-out test, the server sends data to the different clients, stressing the transmission path on the network adapter; and in the fan-in test, the clients send data to the server process, stressing the receive path on the network adapter. These tests differ from the multistream test, in which we performed bandwidth tests between multiple processes on the same two nodes. In the fan-in and fan-out tests, we performed bandwidth tests between one server process and multiple client processes on multiple physical nodes.

Figure 6 shows the TOE and non-TOE performance for the fan-in and fan-out bandwidth tests. The performance for both tests remains constant as the client number increases. This shows that if a server must stream data to or receive data streams from multiple clients simultaneously over a 10GigE TOE network, it does not suffer any noticeable performance degradation. This in turn suggests efficient transmit- and receive-path implementations for the TOE in the presence of multiple flows corresponding to different remote nodes.

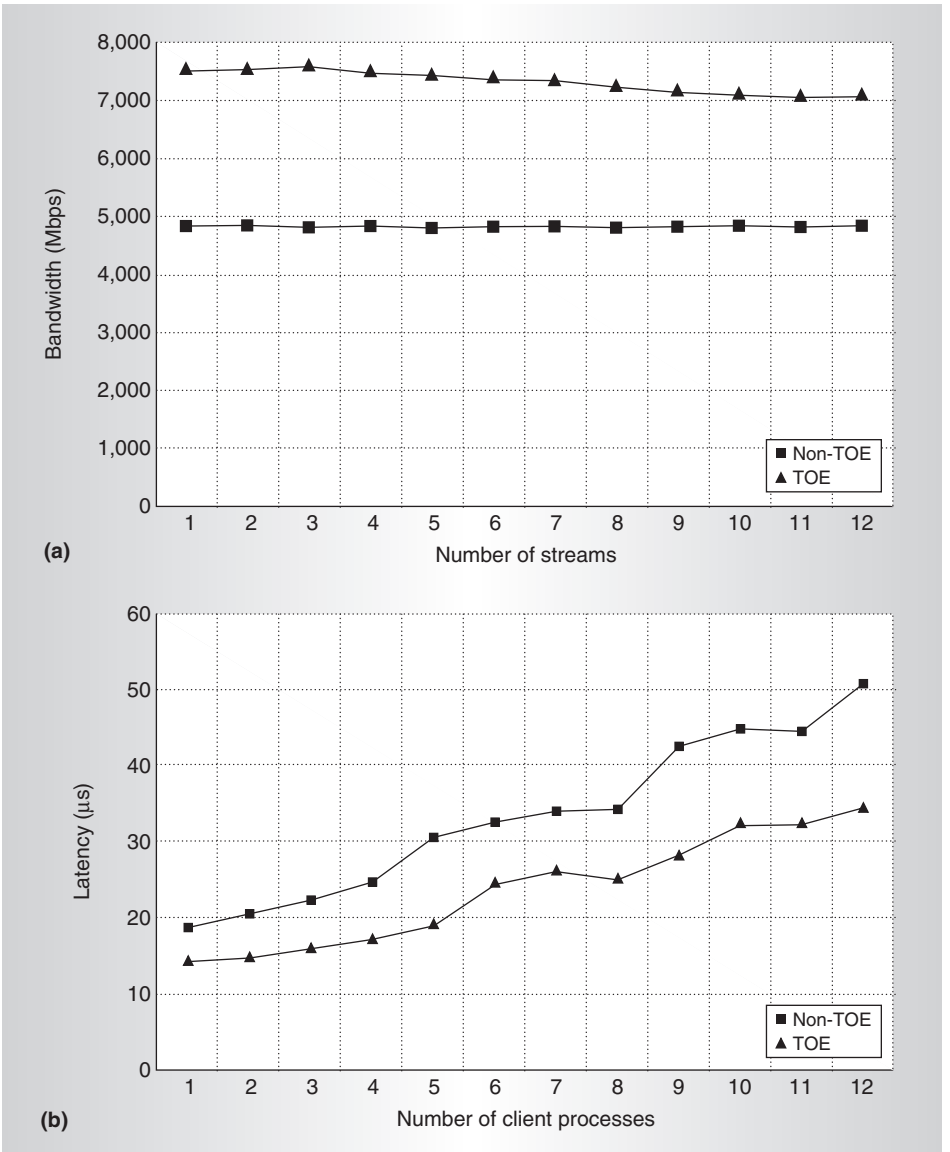


Figure 5. Multistream bandwidth (a) and hot-spot latency (b) for 1 byte. Both tests used only MTU 1500 (1.5 Kbytes) to abide by standard Ethernet frame size.

## 10GigE TCP offload engine versus IBA and Myrinet

We conducted these experiments to evaluate the performance of the Chelsio T110 10GigE adapter with TOE as compared to the SDP implementations on top of IBA and Myrinet. We performed all these experiments on Cluster 3.

### Microbenchmark comparison

The performance benefits of TOE over non-TOE networks hint at TOE's capabilities, but to get a more complete picture, we had to compare the performance of TOE networks with that of the traditional Ethernot networks.

Figure 7 shows the basic microbenchmark perfor-

mance of the 10GigE TOE as compared to SDP/IBA and SDP/Myrinet (SDP/MX/Myrinet and SDP/GM/Myrinet).

Figures 7a and 7b compare ping-pong latency for the network stacks. IBA and Myrinet provide both polling- and event-based mechanisms to inform the user about the completion of data transmission or reception; 10GigE provides only an event-based mechanism. In the polling approach, the sockets implementation must continuously poll on a predefined location to check if the data transmission or reception has completed. This approach delivers high bandwidth and low latency but the continuous monitoring requires a large percentage of CPU resources. In the event-based approach, the sockets implementation requests the network adapter to inform it of a completion and then sleeps. On a completion event, the network adapter wakes this process up through an interrupt.

Although the event-based approach requires a lower percentage of CPU resources (because the application does not have to continuously monitor the data-transfer completions), it incurs the additional cost of an interrupt. In general, for single-threaded applications, the polling approach is more efficient; for most multithreaded applications, the event-based approach performs better.

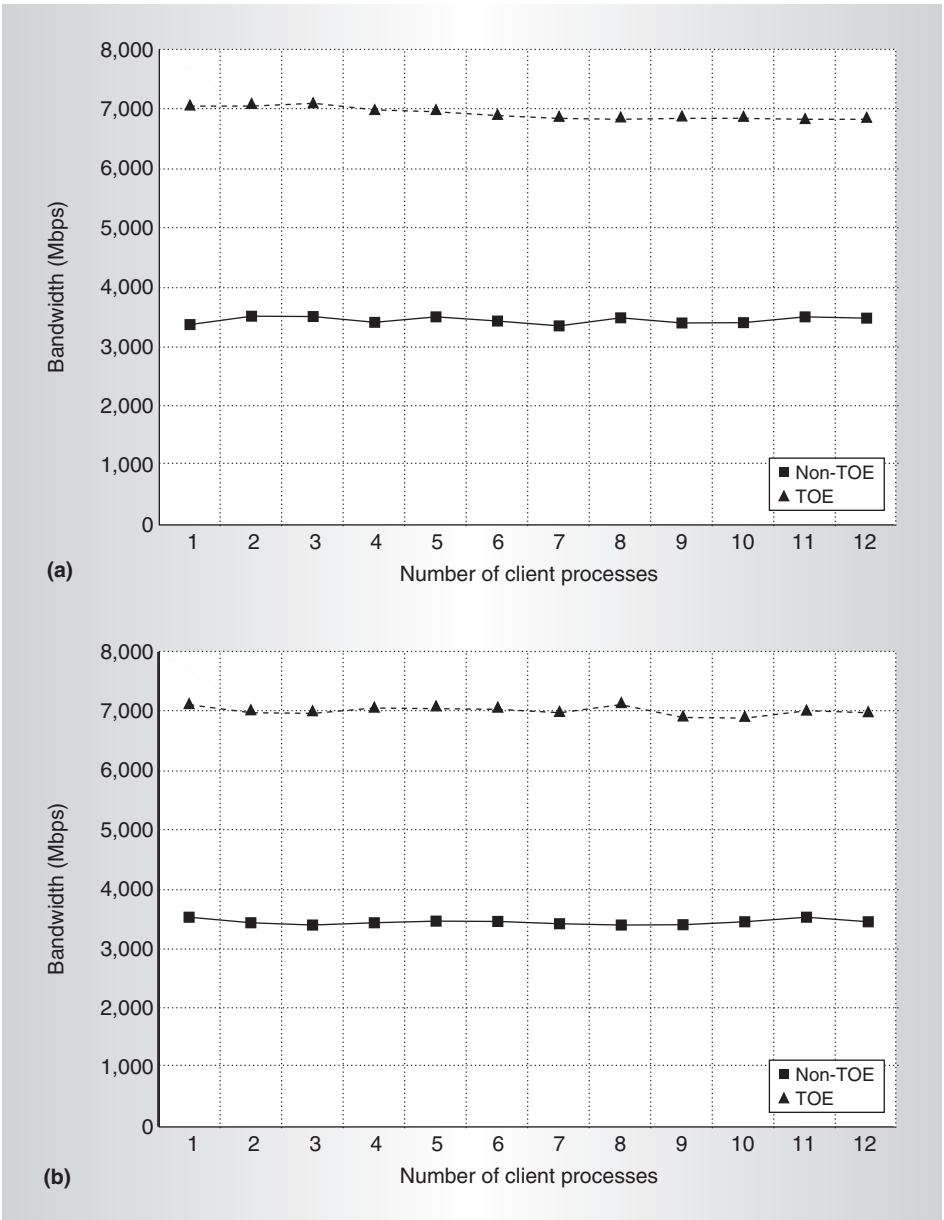


Figure 6. Fan-out (a) and fan-in (b) bandwidth tests.

As Figures 7a and 7b show, SDP/Myrinet generally achieves the lowest small-message latency for both approaches. For the polling-based models, SDP/MX/Myrinet and SDP/GM/Myrinet achieve latencies of 4.64 µs and 6.68 µs, compared with the 8.25 µs that SDP/IBA achieves. For the event-based models, SDP/MX/Myrinet and SDP/GM/Myrinet achieve latencies of 14.47 µs and 11.33 µs, compared with the 14.3 µs and 24.4 µs that 10GigE TOE and SDP/IBA achieve.

The figure also shows, however, that for messages larger than 2 Kbytes for event-based and 4 Kbytes for polling-based communication, SDP/Myrinet performance deteriorates. For messages in this range, SDP/IBA performs best, followed by 10GigE TOE and then the two SDP/Myrinet implementations. For the ping-pong latencies, the 10GigE (Fujitsu), IBA (Mel-

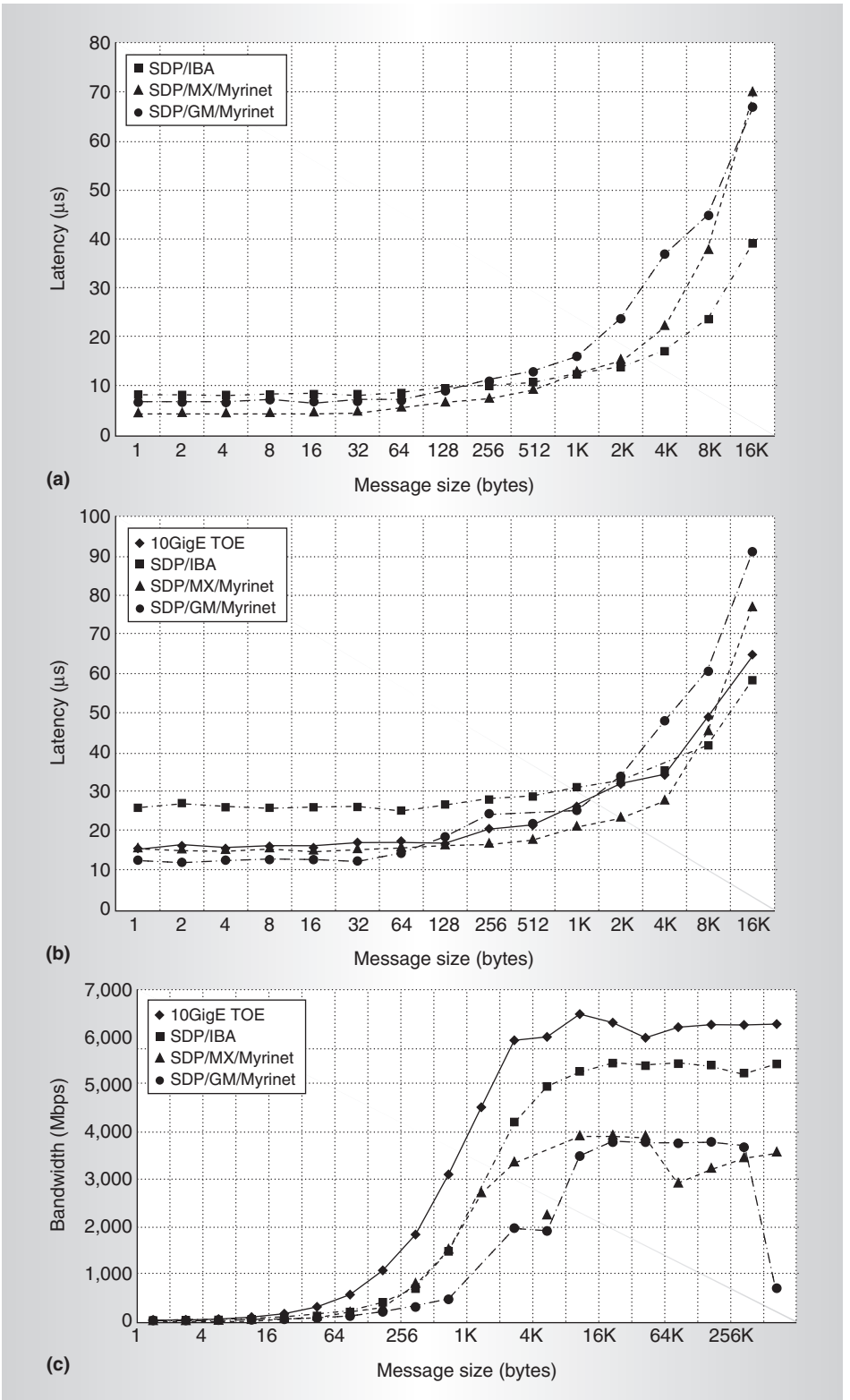


Figure 7. Single-connection microbenchmarks: latency versus message size, polling-based (a); latency versus message size, event-based (b); and unidirectional bandwidth versus message size, event-based (c).

lanox), and Myrinet (Myricom) switches contribute approximately 1000 ns, 300 ns, and 60 ns, respectively.

Figure 7c shows the results of the unidirectional bandwidth test. The 10GigE TOE achieves the highest bandwidth at close to 6.4 Gbps, compared with the 5.4 Gbps and 3.9 Gbps that SDP/IBA and SDP/Myrinet achieve. (The theoretical peak for Myrinet is 4 Gbps.)

In the event-based results, the bandwidth drop for SDP/GM/Myrinet for messages of 512 Kbytes is likely due to this implementation's high dependency on L2 cache activity. Even 10GigE TOE shows a slight drop in performance for very large messages, but not one as drastic as SDP/GM/Myrinet exhibits. Our systems use a 512-Kbyte L2 cache and relatively slow memory (266-MHz DDR SDRAM), which causes the drop to be significant. Systems with larger L2 caches, L3 caches, faster memory, or better memory architectures (nonuniform memory access, for example), will likely experience smaller drops. Further, bandwidth for all networks is the same whether or not they use a switch, so switches do not appear to be a bottleneck for this test.

## Application-level comparison

In this series of tests, we evaluated the performance of four applications across the three network technologies (IBA, Myrinet, and 10GigE TOE):

- Virtual Microscope (VM),[8] a biomedical image-visualization tool;

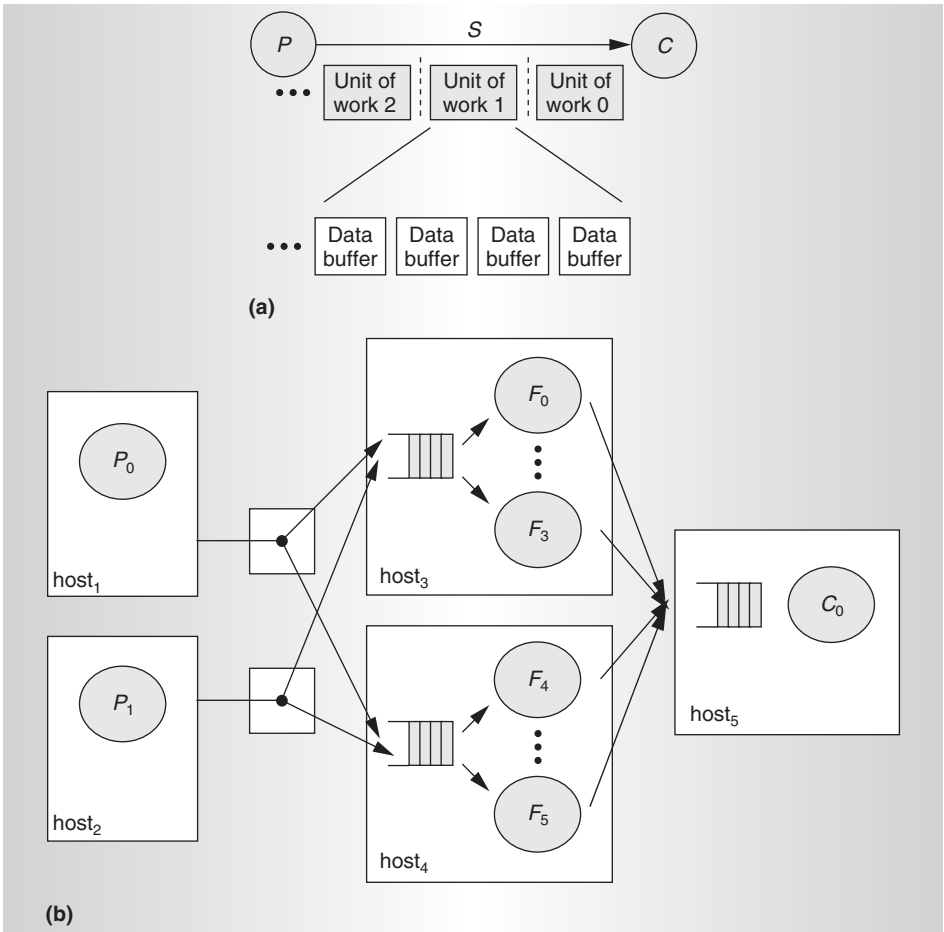


Figure 8. Data-Cutter stream abstraction and support for copies includes data buffers and end-of-work markers on a stream (a) and a filter group (b)—consisting of producer (P), filter (F), and consumer (C)—instantiated using transparent copies.

- Iso-Surface Oil-Reservoir Simulation (ISO);[9]
- Parallel Virtual File System (PVFS),[10] a cluster file-system; and
- Ganglia,[11] a popular cluster-management tool.

The first two applications run on Data-Cutter, a component-based framework developed at the University of Maryland to provide a flexible and efficient runtime environment for data-intensive applications on distributed platforms.

## Data-Cutter

Data-Cutter implements a filter-stream programming model, in which the application processing structure is a set of components, or *filters*, that exchange data through a stream abstraction. Filters connect via logical streams, each of which denotes a unidirectional data flow from one filter (the producer) to another (the consumer). A filter reads data from its input streams and writes data to its output streams. The logical-stream implementation uses the sockets interface for point-to-point stream communication. A filter group—a set of filters connected through logical streams—realizes the application's overall processing structure. When a filter group is instantiated to process an application query, the runtime system establishes socket connections between filters placed on different hosts before starting the execution of the application query.

The filter group handles an application query as a unit of work. The processing of a UOW can be done in a pipelined fashion; different filters can work on different data ele-

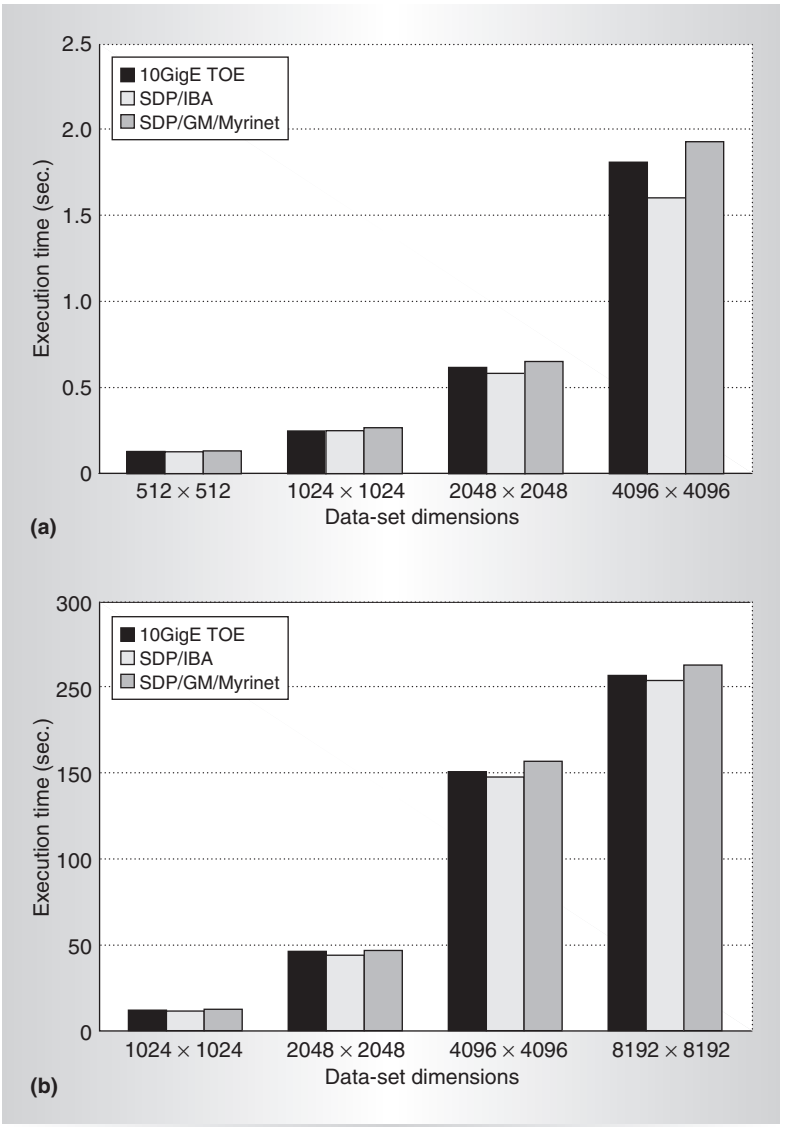


Figure 9. Comparison of three networks running two applications—Virtual Microscope (a) and Iso-Surface Oil-Reservoir Simulation (b)—developed using the Data-Cutter component environment.

ments simultaneously, as Figure 8 illustrates.

*Virtual Microscope.* VM is a data-intensive digitized microscopy application. The software support required to store, retrieve, and process digitized slides to provide interactive response times for the standard behavior of a physical microscope is a challenging issue. The main difficulty stems from handling large volumes of image data, which can range from a few hundred megabytes to several gigabytes per image. At a basic level, the software system should emulate the use of a physical microscope, including continuously moving the stage and changing magnification. The processing of client queries requires projecting high-resolution data onto a grid of suitable resolution and appropriately composing pixels mapping onto a single grid point.

*Iso-Surface Oil-Reservoir Simulation.* Iso is the result of computational modeling for the seismic analysis of oil reservoirs that examines a reservoir's seismic properties by using output from oil reservoir simulations. The main objective of oil reservoir modeling is to understand the reservoir properties and predict oil production. This in turn lets companies optimize return on investment from a given reservoir, while minimizing environmental effects. Iso demonstrates a dynamic, data-driven approach to solve optimization problems in oil reservoir management. Analysts evaluate the output from seismic simulations to investigate the change in the reservoirs' geological characteristics and to guide future oil reservoir simulations. Seismic simulations produce output that represents the traces of sound waves generated by sound sources and recorded by receivers on a 3D grid over many time steps. One analysis of seismic data sets involves mapping and aggregating traces onto a 3D volume through seismic imaging. The resulting 3D volume is suitable for visualization or for generating input for further reservoir simulations.

*Performance comparison.* Figure 9 compares the performance of VM and Iso over the three networks. As Figure 9a shows, SDP/IBA outperforms the other two networks for VM, primarily because IBA delivers lower latency than TOE and SDP/GM/Myrinet for medium-sized messages (see Figure 7a). Although VM deals with large data sets (each image was about 16 Mbytes), the data set is broken into small UOW segments that the network processes in a pipeline. This makes the application sensitive to the latency of medium-sized messages.

Figure 9b compares the performance of the Iso application for the three networks with a dataset of about 64 Mbytes. Again, although the performance of the three networks is much closer than it was for VM, SDP/IBA slightly outperforms the other two networks.

## Parallel Virtual File System

Figure 10 shows the results of running PVFS on the three networks. PVFS, which

Clemson University and Argonne National Laboratory jointly developed to meet the increasing I/O demands of parallel applications in cluster systems, is one of the leading parallel file systems for Linux cluster systems. As Figure 10 shows, a typical PVFS environment comprises several nodes configured as I/O servers with one node (either an I/O server or a different node) configured as a metadata manager.

PVFS stripes files across a set of I/O server nodes, which allows parallel access to the data. It uses the native file system on the I/O servers to store individual file stripes. An I/O daemon runs on each I/O node and services requests from the computational nodes, primarily read and write requests. Thus, data transfers directly between the I/O servers and the computational nodes. A manager daemon running on a metadata manager node handles metadata operations involving file permissions, truncation, file stripe characteristics, and so on. Metadata is also stored on the local file system. The metadata manager provides a clusterwide, consistent name space to applications but does not participate in read or write operations.

PVFS also supports a set of feature-rich interfaces, including support for both contiguous and noncontiguous accesses to both memory and files. It is available with a range of application programming interfaces (APIs): native, Unix/Posix, MPI-I/O, multidimensional block, and array I/O. This API flexibility is a key factor in the popularity of PVFS.

To evaluate the performance of concurrent read or write operations in PVFS, we used the *pvfs-test* program from standard PVFS releases. This test uses an MPI program to parallelize file write or read access of contiguous 2-Mbyte data buffers from each computational node. We performed two types of tests for both read and write: three clients simultaneously read a file from or write a file to the server (1S/3C) and a single client reads the stripes from or writes the stripes to all three servers simultaneously (3S/1C). Figure 11 shows that the 10GigE TOE network outperforms the other two in both tests for read and write. This follows the same trend as that in Figure 7c.

Figure 12 shows the performance of MPI-Tile-I/O,[12] a tile-reading MPI-I/O application that tests the performance of tiled access to a

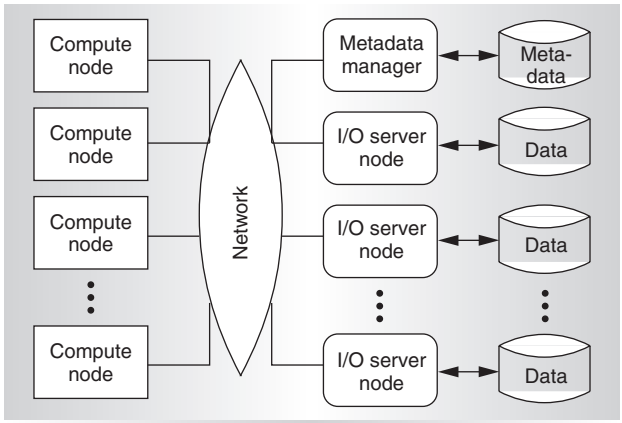


Figure 10. Typical PVFS environment.

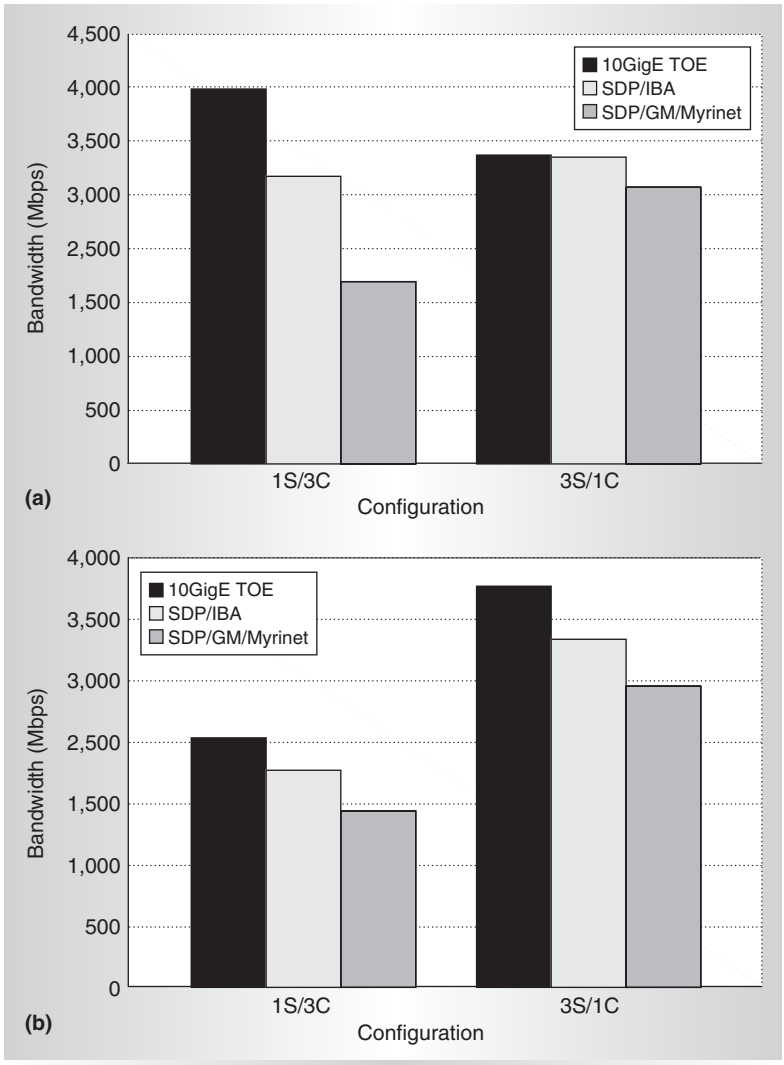


Figure 11. Performance of PVFS concurrent read (a) and write (b) operations with two configurations: 1S/3C is one server, three clients; and 3S/1C is three servers, one client.

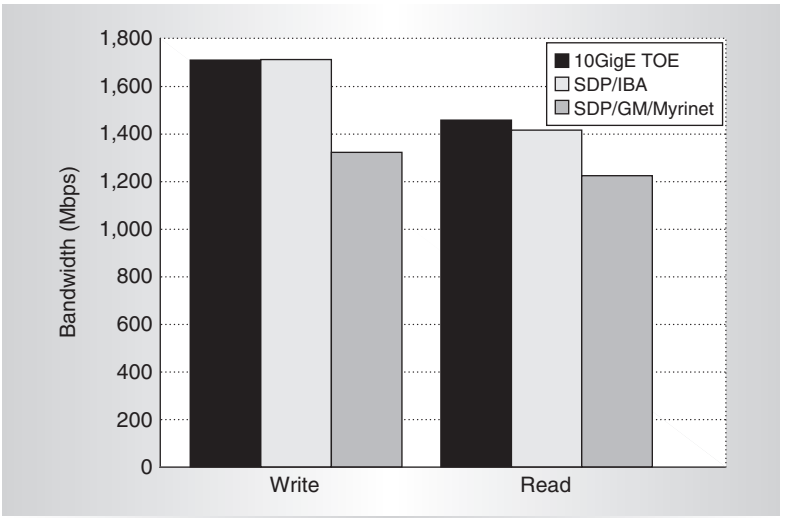


Figure 12. Performance of MPI-Tile-I/O over PVFS.

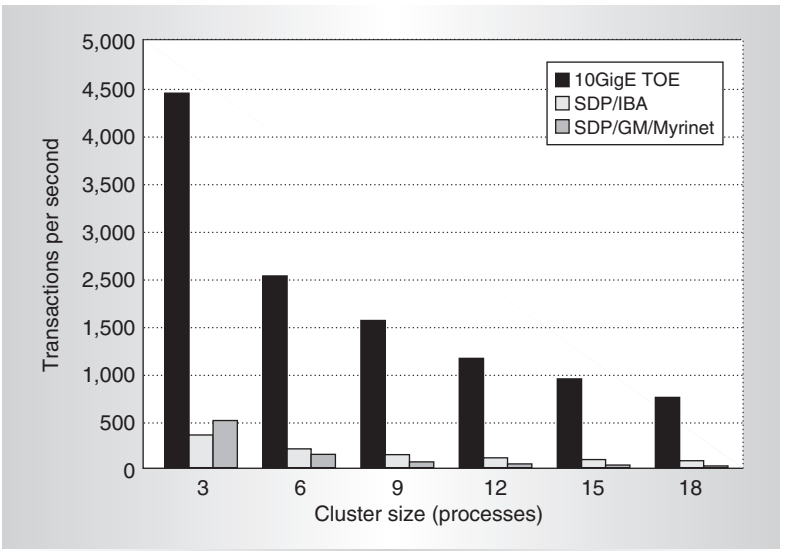


Figure 13. Ganglia cluster management tool.

2D dense data set, simulating the workload in some visualization and numerical applications. In our experiments, we used two nodes as servers and the other two as clients running MPI-Tile-I/O processes. Each process renders a $1 \times 2$ array of displays, each with $1024 \times 768$ pixels. The size of each element is 32 bytes, leading to a file size of 48 Mbytes.

We evaluated both the read and write performance of MPI-Tile-I/O over PVFS. As the figure shows, the 10GigE TOE network provides better performance than the other two in terms of both read and write bandwidth. All the networks performed considerably worse in this test than in the concurrent file I/O test. This is likely due to the MPI-Tile-I/O benchmark's noncontiguous data-access pattern, which adds significant overhead.

### Ganglia

Figure 13 presents the results of running Ganglia, an open-source project that grew out of the University of California, Berkeley, Millennium Project. It is a scalable distributed monitoring system for high-performance computing systems (such as clusters and grids) that is based on a hierarchical design targeted at federations of clusters. Ganglia leverages widely used technologies such as Extensible Markup Language (XML) for data representation, the External Data Representation standard for compact and portable data transport, and the open source tool, RRDtool, for data storage and visualization. Ganglia uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency.

The Ganglia system contains a server-monitoring daemon that runs on each cluster node and occasionally monitors the various system parameters including CPU load, disk space, and memory use. Ganglia also contains a client tool that contacts the servers in the clusters and collects relevant information.

Ganglia supports two forms of global data collection for the cluster. In the first method, the servers can communicate with each other to share their respective state information, and the client can communicate with any one server to collect the global information. In the second method, the servers just collect their local information without communicating with other server nodes, while the client communicates with each server node to obtain the global cluster information. In our experiments, we used the second approach.

As Figure 13 shows, the 10GigE TOE network outperforms the other two by up to a factor of 11 in some cases. This performance difference stems from Ganglia's work pattern. The client node is an end node that gathers information about all servers in the cluster and displays it to the user. To collect this information, the client opens a connection with each node in the cluster and obtains the relevant information (from 2 to 10 Kbytes) from the nodes. Thus, Ganglia is quite sen-

sitive to connection time and to medium-sized message latency.

Although Figures 7a and 7b show that 10GigE TOE and SDP/GM/Myrinet do not perform very well for medium-sized messages, the connection time for 10GigE is only about 60 µs, whereas the connection times for SDP/GM/Myrinet and SDP/IBA are in the *milli*second range. During connection setup, SDP/GM/Myrinet and SDP/IBA must preregister a set of buffers to carry out the required communication; this operation is quite expensive for Myrinet and IBA because it involves informing the network adapters about each of these buffers and the corresponding protection information. This coupled with other overheads, such as the state transitions (Init-RTR-RTS) that IBA requires during connection setup, increase the connection time tremendously for SDP/IBA and SDP/GM/Myrinet. All in all, the connection setup time dominates the performance of Ganglia in our experiments, resulting in better performance for the 10GigE TOE network.

Our results demonstrate that TCP offloading not only provides 10GigE a significant push in the performance it can achieve, but also enables a performance comparable to that of traditional Ethernot networks, such as IBA and Myrinet, for sockets-based applications.

With the advent of TOEs for 10GigE, Ethernet has largely bridged the performance gap with IBA and Myrinet via the sockets interface—a successful first step on the part of Ethernet toward a network infrastructure that delivers high performance in a SAN while maintaining WAN compatibility.

Although the sockets interface is the most widely used interface for grids, file systems, storage, and other commercial applications, MPI is considered the de facto standard for scientific applications. Thus, a feasibility study of 10GigE as a SAN is incomplete without comparing MPI over the various networks. This will be our focus in future work. MICRO

## Acknowledgments

We thank Joel Saltz, Tahsin Kurc, and Umit Catalyurek from the Department of Biomedical Informatics at Ohio State University for providing us access to the Data-Cutter runtime library and the application data sets; Felix Marti, Ásgeir Eiriksson, and Kianoosh Naghshineh from Chelsio Communications, and Takeshi Horie and Vic Herring from Fujitsu for providing us with support on the Chelsio 10GigE TOE adapters and the Fujitsu XG800 10GigE switch, respectively; Markus Fischer from Myricom Inc. and Gali Zisman, Andy Hillaker, Erez Strauss, Yaron Haviv, and Yaron Segev from Voltaire Corp. for providing us access to the SDP/Myrinet and SDP/IBA stacks, respectively.

This work was supported by Los Alamos National Laboratory contract W-7405-ENG-36, Department of Energy grant DE-FC02-01ER25506, National Science Foundation grants CNS-0403342 and CNS-0509452, and technical and equipment support from Chelsio Communications and Fujitsu.

## References

1. N.J. Boden et al., "Myrinet: A Gigabitper-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, 1995, pp. 29-36.
2. F. Petrini et al., "The Quadrics Network (QsNet): High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, Jan.-Feb. 2002, pp. 46-57.
3. P. Balaji et al., "Head-to-TOE Evaluation of High Performance Sockets over Protocol Offload Engines," *Proc. IEEE Int'l Conf. Cluster Computing*, IEEE CS Press, 2005; http://nowlab.cse.ohio-state.edu/publications/conf-papers/2005/balaji-cluster05.pdf.
4. W. Feng et al., "Performance Characterization of a 10-Gigabit Ethernet TOE," *Proc. IEEE Int'l Symp. High-Performance Interconnects* (Hotl 05), IEEE CS Press, 2005, pp. 58-63.
5. E. Yeh et al., "Introduction to TCP/IP Offload Engine (TOE)," May 2002; http://www.ethernetalliance.org/technology/white_papers/10gea_toe.pdf.
6. D. Goldenberg et al., "Zero Copy Sockets Direct Protocol over InfiniBand—Preliminary Implementation and Performance Analysis," *Proc. IEEE Int'l Symp. High-Performance Interconnects* (Hotl 05), IEEE CS Press, 2005, pp. 128-137.
7. P. Balaji et al., "Asynchronous Zero-copy Communication for Synchronous Sockets in the Sockets Direct Protocol (SDP) over InfiniBand," *Proc. Workshop Comm.*

*Architecture for Clusters* (CAC 06), IEEE CS Press, 2006; http://nowlab.cse.ohio-state.edu/publications/conf-papers/2006/balaji-cac06.pdf.

8. A. Afework et al., "Digital Dynamic Telepathology—The Virtual Microscope," *Proc. 1998 Amer. Medical Informatics Assoc. Annual Fall Symp.*, AMIA, 1998, pp. 912-916.
9. M.D. Beynon et al., *A Component-based Implementation of Iso-surface Rendering for Visualizing Large Datasets*, Tech. Report CS-TR-4249 and UMIACS-TR-2001-34, Univ. of Maryland, 2001.
10. P.H. Carns et al., "PVFS: A Parallel File System For Linux Clusters," *Proc. 4th Ann. Linux Showcase and Conf.*, 2000, pp. 317-327.
11. M.L. Massie, B.N. Chun, and D.E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, no. 7, July 2004, pp. 817-840.
12. R.B. Ross, "The Parallel Virtual File-System (PVFS)," Parallel I/O Benchmarking Consortium; http://www-unix. mcs.anl.gov/~rross/pio-benchmark.

**Pavan Balaji** is a doctoral student in computer science and engineering at Ohio State University. His research interests include high-speed networks, efficient IP-based protocols (TCP, UDP, iWARP) and the sockets programming interface for cluster and grid computing. Balaji has a BTech in computer science and engineering from the Indian Institute of Technology at Madras. He is a student member of the IEEE.

**Wu-chun Feng** is an associate professor of computer science with a joint appointment in electrical and computer engineering at Virginia Tech. His research interests include high-performance networking and computing. He has a BS and an MS in computer engineering and a BS in music—all from Pennsylvania State University—and a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a senior member of the IEEE.

**Dhabaleswar K. Panda** is a professor of computer science and engineering at Ohio State University. His research interests include parallel computer architecture, high-performance networking, and network-based computing. Panda has a PhD in computer engineering from the University of Southern California at Los Angeles. He is a senior member of the IEEE Computer Society and a member of ACM.

For questions about this article contact Pavan Balaji, 2015 Neil Ave., 395 Dreese Laboratory, Columbus, OH 43210; balaji@cse.ohio-state.edu.