End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems

Intel's network interface card for 10-Gigabit Ethernet (10GbE) allows individual computer systems to connect directly to 10GbE Ethernet infrastructures. Results from various evaluations suggest that 10GbE could serve in networks from LANs to WANs.

Justin (Gus) Hurwitz
Wu-chun Feng
Los Alamos National Laboratory

From its humble beginnings as shared Ethernet to its current success as switched Ethernet in local-area networks (LANs) and system-area networks and its anticipated success in metropolitan and wide area networks (MANs and WANs), Ethernet continues to evolve to meet the increasing demands of packet-switched networks. It does so at low implementation cost while maintaining high reliability and relatively simple (plug and play) installation, administration, and maintenance.

Although the recently ratified 10-Gigabit Ethernet standard differs from earlier Ethernet standards, primarily in that 10GbE operates only over fiber and only in full-duplex mode, the differences are largely superficial. More importantly, 10GbE does not make obsolete current investments in network infrastructure. The 10GbE standard ensures interoperability not only with existing Ethernet but also with other networking technologies such as Sonet, thus paving the way for Ethernet's expanded use in MANs and WANs.

Although the developers of 10GbE mainly intended it to allow easy migration to higher performance levels in backbone infrastructures, 10GbE-based devices can also deliver such performance to bandwidth-hungry host applications via Intel's new 10GbE network interface card (or adapter). We implemented optimizations to Linux, the Transmission Control Protocol (TCP), and the 10GbE adapter configurations and performed several evaluations. Results showed extraordinarily higher throughput with low latency, indicating that 10GbE is a viable interconnect for all network environments.

Architecture of a 10GbE adapter

The world's first host-based 10GbE adapter, officially known as the Intel PRO/10GbE LR server adapter, introduces the benefits of 10GbE connectivity into LAN and system-area network environments, thereby accommodating the growing number of large-scale cluster systems and bandwidth-intensive applications, such as imaging and data mirroring. This first-generation 10GbE adapter contains a 10GbE controller implemented in a single chip that contains functions for both the media access control and physical layers. The controller is optimized for servers that use the I/O bus backplanes of the Peripheral Component Interface (PCI) and its higher speed extension, PCI-X.

Published by the IEEE Computer Society

0272-1732/04/$20.00 © 2004 IEEE

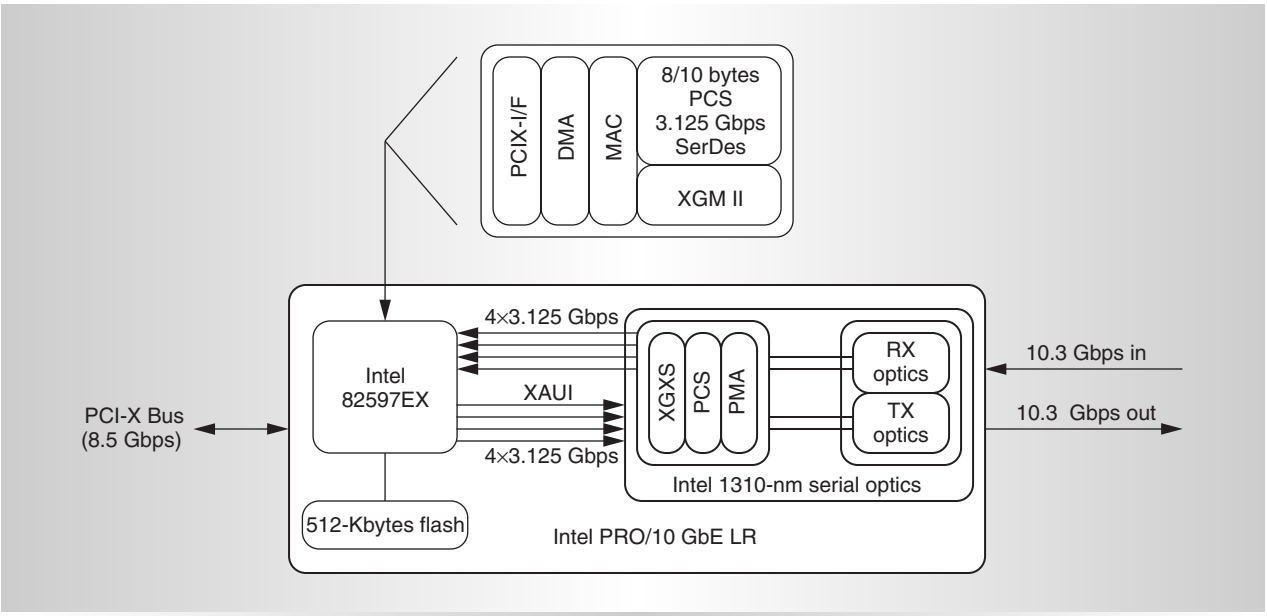

Figure 1. Architecture of the 10GbE adapter. The three main components are a 10GbE controller, flash memory, and serial optics.

Figure 1 gives an architectural overview of the 10GbE adapter, which consists of three main components: an 82597EX 10GbE controller, 512 Kbytes of flash memory, and 1,310-nm wavelength single-mode serial optics transmitter and receiver. The 10GbE controller includes an Ethernet interface that delivers high performance by providing direct access to all memory without using mapping registers, minimizing the interrupts and programmed I/O (PIO) read access required to manage the device, and offloading simple tasks from the host CPU.

As is common practice with high-performance adapters such as Myricom's Myrinet[1] and Quadrics' QsNet,[2] the 10GbE adapter frees up host CPU cycles by performing certain tasks (in silicon) on behalf of the host CPU. However, the 10GbE adapter focuses on host off-loading of certain tasks—specifically, TCP and Internet Protocol (IP) checksums and TCP segmentation—rather than relying on remote direct memory access (RDMA) and source routing. Consequently, unlike Myrinet and QsNet, the 10GbE adapter provides a general-purpose solution to applications that does not require modifying application code to achieve high performance. (The "Putting the 10GbE Numbers in Perspective" sidebar compares 10GbE's performance to these other adapters.)

Testing environment and tools

We evaluate the 10GbE adapter's performance in three different LAN or system-area network environments, as Figure 2 shows:

- direct single flow between two computers connected back-to-back via a crossover cable,
- indirect single flow between two computers through a Foundry FastIron 1500 switch, and
- multiple flows through the Foundry FastIron 1500 switch.

Host computers were either Dell PowerEdge 2650 (PE2650) or Dell PowerEdge 4600 (PE4600) servers.

Each PE2650 contains dual 2.2-GHz Intel Xeon CPUs running on a 400-MHz front-side bus (FSB) using a ServerWorks GC-LE chipset with 1 Gbyte of memory and a dedicated 133-MHz PCI-X bus for the 10GbE adapter. Theoretically, this architectural configuration provides 25.6-Gbps CPU bandwidth, up to 25.6-Gbps memory bandwidth, and 8.5-Gbps network bandwidth via the PCI-X bus.

Each PE4600 contains dual 2.4-GHz Intel

Putting the 10GbE Numbers in Perspective

To appreciate the 10GbE adapter's performance, we consider it in the context of similar network interconnects. In particular, we compare 10GbE's performance with that of its predecessor, 1GbE, and the most common interconnects for high-performance computing (HPC): Myrinet, QsNet, and InfiniBand.

1GbE

With our extensive experience with 1GbE chipsets, we can achieve near line-speed performance with a 1,500-byte maximum transmission unit (MTU) in a LAN or system-area network environment with most payload sizes. Additional optimizations in a WAN environment should let us achieve similar performance while still using a 1,500-byte MTU.

Myrinet

We compare 10GbE's performance to Myricom's published performance numbers for its Myrinet adapters (available at http://www.myri.com/myrinet/performance/ip.html and http://www.myri.com/myrinet/performance/index.html). Using Myrinet's GM API, sustained unidirectional bandwidth is 1.984 Gbps and bidirectional bandwidth is 3.912 Gbps. Both numbers are within 3 percent of the 2-Gbps unidirectional hardware limit. The GM API provides latencies of 6 to 7 μs.

Using the GM API, however, might require rewriting portions of legacy application code. Myrinet provides a TCP/IP emulation layer to avoid this extra work. This layer's performance, however, is notably less than that of the GM API: Bandwidth drops to 1.853 Gbps, and latencies skyrocket to more than 30 μs.

When running Message-Passing Interface (MPI) atop the GM API, bandwidth reaches 1.8 Gbps with latencies between 8 to 9 μs.

InfiniBand and QsNet II

InfiniBand and Quadrics' QsNet II offer MPI throughput performance that is comparable to 10GbE's socket-level throughput. Both InfiniBand and Quadrics have demonstrated sustained throughput of 7.2 Gbps. Using the Elan4 libraries, QsNet II achieves sub-2-μs latencies[1] while InfiniBand's latency is around 6 μs.[2]

Our experience with Quadrics' previous-generation QsNet (using the Elan3 API) produced unidirectional MPI bandwidth of 2.456 Gbps and a 4.9-μs latency. As with Myrinet's GM API, the Elan3 API may require application programmers to rewrite the application's network code, typically to move from a socket API to the Elan3 API. To address this issue, Quadrics also has a highly efficient TCP/IP implementation that produces 2.240 Gbps of bandwidth and less than 30-μs latency. Additional performance results are available elsewhere.[3]

We have yet to see performance analyses of TCP/IP over either InfiniBand or QsNet II. Similarly, we have yet to evaluate MPI over 10GbE. This makes direct comparison between these interconnects and 10GbE relatively meaningless at present. This lack of comparison might be for the best. 10GbE is a fundamentally different architecture, embodying a design philosophy that is fundamentally different from other interconnects for high-performance computing. Ethernet standards, including 10GbE, are meant to transfer data in any network environment; they value versatility over performance. Interconnects designed for high-performance computing environments (for example, supercomputing clusters) are specific to system-area networks and emphasize performance over versatility.

References

1. J. Beecroft et al., "Quadrics QsNet II: A Network for Supercomputing Applications," *presented at Hot Chips 14* (HotC 2003); available from http://www.quadrics.com.
2. J. Liu et al., "Micro-Benchmark Level Performance Comparison of High-Speed Cluster Interconnects," *Proc. Hot Interconnects 11* (HotI 2003), IEEE CS Press, 2003, pp. 60-65.
3. F. Petrini et al., "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, Jan.-Feb. 2002, pp. 46-57.

Xeon CPUs running on a 400-MHz FSB, using a ServerWorks GC-HE chipset with 1 GByte of memory and a dedicated 100-MHz PCI-X bus for the 10GbE adapter. This configuration provides theoretical bandwidths of 25.6-, 51.2-, and 6.4-Gbps for the CPU, memory, and PCI-X bus.

Since publishing our initial 10GbE results, we have run additional tests on Xeon processors with faster clocks and FSBs, as well as on 64-bit Intel Itanium2 and AMD Opteron systems, as we discuss later.

In addition to these hosts, we use a Foundry FastIron 1500 switch for both our indirect single flow and multiflow tests. In the latter case, the switch aggregates Gigabit Ethernet and 10GbE streams from (or to) many hosts into a 10GbE stream to (or from) a single host. The total backplane bandwidth (480 Gbps) in the switch far exceeds our test needs because both 10GbE ports are limited to 8.5 Gbps.

From a software perspective, all the hosts run current installations of Debian Linux with customized kernel builds and tuned TCP/IP stacks. We tested numerous kernels from the Linux 2.4 and 2.5 distributions. Because the performance differences between these kernel builds prove negligible, we do not report the kernel version in our results.

The experiments described in this article focus on bulk data transfer performance. We use two tools to measure network through-

put—NTTCP (New Test TCP, http://www.leo.org/~elmar/nttcp/) and Iperf (Internet Performance, http://dast.nlanr.net/Projects/Iperf/)—and note that the experimental results from these tools are generally confirmed by another oft-used tool called Netperf (Network Performance, http://www.netperf.org). We do not report our NetPerf results as they are too coarse for the purposes of this study.

NTTCP and IPerf measure the time required to send a stream of data. IPerf measures the amount of data sent over a consistent stream in a set time. NTTCP measures the time required to send a set number of fixed-size packets. In our tests, IPerf is well suited for measuring raw bandwidth while NTTCP is better suited for optimizing the performance between the application and the network. As our goal is to maximize application performance, NTTCP provides more valuable data in these tests. We therefore present primarily NTTCP data throughout the article. (Typically, the performance difference between the two is within 3 percent, and in no case does IPerf yield results significantly contrary to those of NTTCP.)

In addition to reporting throughput results for bulk data transfer, we provide preliminary results on end-to-end latency. To determine the latency between a pair of 10GbE-enabled hosts, we use NetPipe (http://www.scl.ameslab.gov/netpipe/). We compute latency values by halving the average round-trip time of a statistically representative number of single-byte ping-pong tests. Such tests indicate the responsiveness of the network to small packets.

We use the Stream benchmark (http://www.cs.virginia.edu/stream/) to measure memory bandwidth.

To estimate the CPU load across throughput tests, we sample /proc/loadavg at 5- to 10-second intervals. The CPU load is a unitless number that roughly corresponds to the percentage of the CPU processing capability in use at any time. A load less than 1 indicates that the CPU is completing more tasks than are being requested of it; a load greater than 1 indicates that more tasks are being asked of the CPU than it is capable of performing in a given time.

Finally, to facilitate data transfer analysis, we use two tools: tcpdump (http://www.tcpdump.org), a widely available tool for analyzing protocols at the wire level; and Magnet,[3] a publicly available tool developed by our research team at Los Alamos National Laboratory.

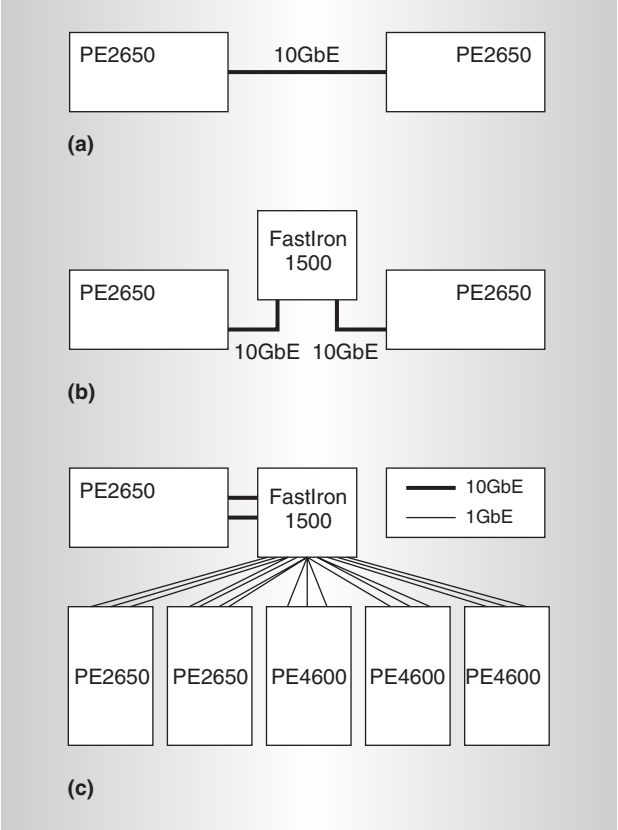

Figure 2. Local- and system area network testing environments: direct single flow (a), indirect single flow (b), and multiple flows (c) through the switch.

Experimental results

Our tests focus primarily on throughput performance. To facilitate analysis, we begin with a standard TCP configuration as our baseline and apply optimizations cumulatively. We also provide initial measurements of end-to-end latency over 10GbE and anecdotal results on recently available hardware, including AMD Opteron- and Intel Itanium2-based systems.

Throughput

We begin our experiments with a stock TCP stack, implementing optimizations one by one to improve network performance between two identical Dell PE2650s connected via 10GbE.

The more common network adaptor and TCP optimizations result in little to no per-

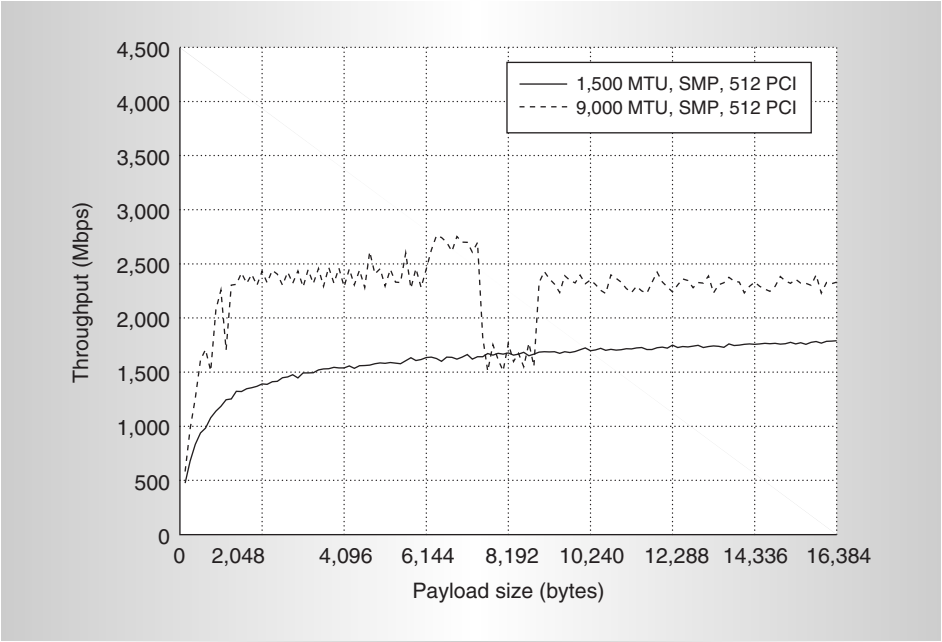

Figure 3.Results for stock TCP for 1,500– and 9,000–byte MTUs. Because smaller MTUs impose greater loads on the CPU, larger MTUs produce 40 to 60 percent better throughput.

formance gains. These optimizations include changing variables such as the device transmit queue and stack backlog lengths and using TCP time stamps. We then tune TCP by calculating the ideal bandwidth-delay product and setting the TCP window sizes accordingly. The product should be relatively small for a LAN, even at 10GbE speeds. We observe a latency of 19 µs running back-to-back and 25 µs running through the Foundry switch. At full 10GbE speed, this results in a maximum bandwidth-delay product of about 48 Kbytes, well below the default window setting of 64 Kbytes. At observed speeds, the maximum product is well under half of the default. In either case, these values are within the scope of the default maximum-window settings.

Baseline: Standard TCP with standard MTU sizes. We begin with single-flow experiments across a pair of unoptimized (stock) Dell PE2650s using a standard 1,500-byte maximum transfer unit and a 9,000-byte jumboframe MTU. In their stock configurations, the dual-processor PE2650s have a standard maximum PCI-X burst transfer size—controlled by the maximum memory read byte count (MMRBC) register—of 512 bytes and run a symmetric multiprocessing (SMP) kernel. In each single-flow experiment, NTTCP transfers 32,768 packets ranging in size from 128 bytes to 16 Kbytes at increments of 32 to 128 bytes.

Figure 3 shows that using a larger MTU size produces 40 to 60 percent better throughput than the standard 1,500-byte MTU. This result is directly attributable to the additional load that 1,500-byte MTUs impose on the CPU—for example, interrupt processing occurs every 1,500 bytes instead of every 9,000 bytes. Specifically, for 1,500-byte MTUs, the CPU load is approximately 0.9 on both the send and receive hosts; for 9,000-byte MTUs the CPU load is only 0.4.

We observe bandwidth peaks at 1.8 Gbps with a 1,500-byte MTU and 2.7 Gbps with a 9,000-byte MTU. In addition, we note that using a 9,000-byte MTU introduces high-frequency oscillations in throughput, irrespective of payload size, as well as a substantial low-frequency oscillation (that is, dip) for payloads between 7,436 and 8,948 bytes in size. This dip is preceded by a similar frequency peak, starting at about 6,272 bytes. These low-frequency oscillations in throughput occur because virtually all TCP implementations force the alignment of the congestion window with the MTU's payload size. Further explanation is provided in the "Results analysis" section.

Increasing the PCI-X burst transfer size. Although the default maximum PCI-X burst transfer size is 512 bytes, the 10GbE adapter supports a burst size as large as 4,096 bytes. Thus, we increase the PCI-X burst transfer size (that is, the MMRBC register) to 4,096 bytes. This simple optimization to the 10GbE hardware improves peak performance for a 9,000-byte MTU to over 3.6 Gbps, a 33-percent increase over the baseline case. With a 1,500-byte MTU, increasing the burst size only produces a marginal throughput increase, indicating that the burst size does not hinder performance for

smaller MTUs. The CPU load remains relatively unchanged from the baseline numbers reported in the previous section.

Running a uniprocessor kernel. Having optimized the PCI-X burst transfer size, our next counterintuitive optimization is to replace the SMP kernel with a uniprocessor kernel. Currently, the Pentium 4 Xeon SMP architecture assigns each interrupt to a single CPU instead of processing them in a round-robin manner among CPUs. Consequently, the interrupt context code of the 10GbE driver cannot take advantage of an SMP configuration. Coupled with the additional cost of kernel locking, this results in uniprocessor kernels running faster than SMP kernels.

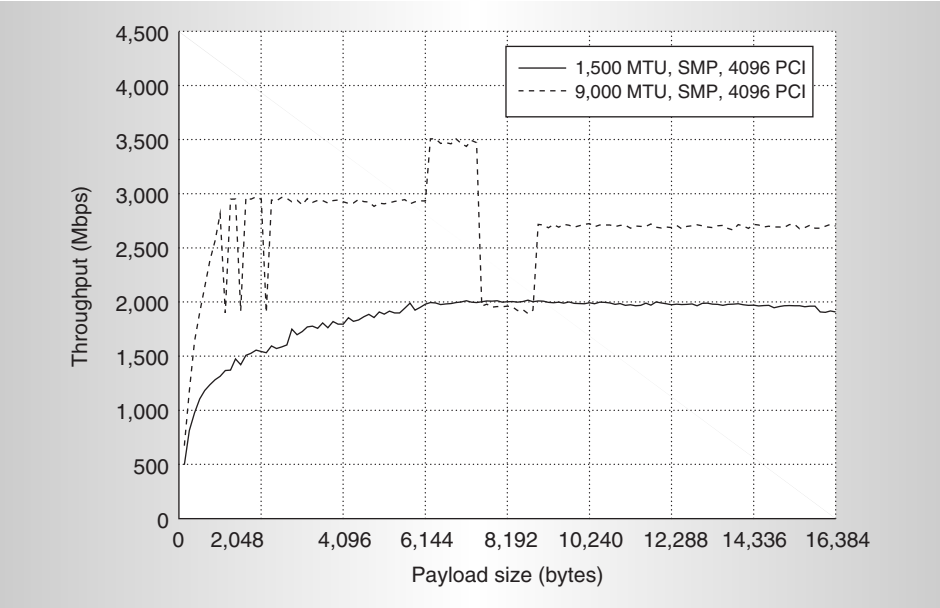

Figure 4. Results for a uniprocessor kernel. Average throughput improves 20 percent for a 9,000-byte MTU; both average and maximum throughout increase by 20 percent for a 1,500-byte MTU.

Figure 4 shows the results when we switch to a uniprocessor kernel. Using 9,000-byte MTUs improves the average throughput by an additional 20 percent. For 1,500-byte MTUs, both the average and maximum throughputs increase by nearly 20 percent. In these tests, the CPU load is uniformly lower than in the SMP tests, primarily because the CPU spends less time in a spin-locked state.

Tuning buffer sizes and MTU sizes. As stated previously, we observe sharp decreases and increases in the throughput for the baseline case of a 9,000-byte MTU. Using tcpdump, we trace the causes of this behavior to inefficient window use by both the sender and receiver. This problem manifests itself in low-latency environments where the bandwidth-delay product is relatively large.

We partially overcome the first limitation by greatly increasing the default socket buffer size. However, this is a poor band-aid solution. Setting the socket buffer at many times the ideal window size should not be necessary in any environment; in a WAN environment, setting the socket buffer too large can adversely affect performance. The low latencies and large MTU size in LAN and system-area network environments, however, undermine the conventional wisdom surrounding window settings, as discussed later. (More precisely, these problems are related to the large maximum segment size rather than the MTU. However, the maximum segment size is so closely tied to MTU that, for ease of readability, we use the MTU exclusively throughout our discussions. This does not affect the accuracy of our analyses.)

Figure 5 shows the effects of increasing the socket buffer sizes from 64 Kbytes to 256 Kbytes. First, all the sharp throughput peaks and valleys for the 9,000-byte MTU disappear. In addition, oversizing the socket buffer improves peak throughput by 15 percent for 1,500-byte MTUs (from 2.15 Gbps to 2.47 Gbps) and by 7 percent for 9,000-byte MTUs (from 3.64 Gbps to 3.9 Gbps). Average throughput increases by 7 percent for 1,500-byte MTUs and by 31 percent for 9,000-byte MTUs

We achieve even better performance with nonstandard MTU sizes. As Figure 6 shows, the peak observed bandwidth achieved with a 9,000-byte jumbo-frame compatible MTU is 4.11 Gbps with an 8,160-byte MTU (it is possible to use 8,160-byte MTUs in conjunction

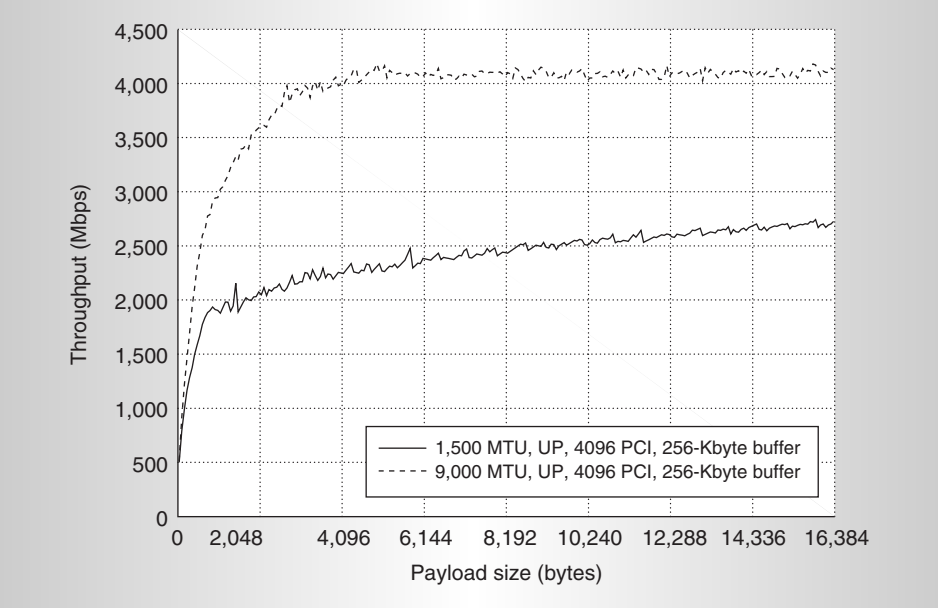

Figure 5. Performance with standard MTUs. Increasing the socket buffer size from 64 to 256 Kbytes removes the bandwidth peaks and valleys observed with standard MTU sizes and improves performance.

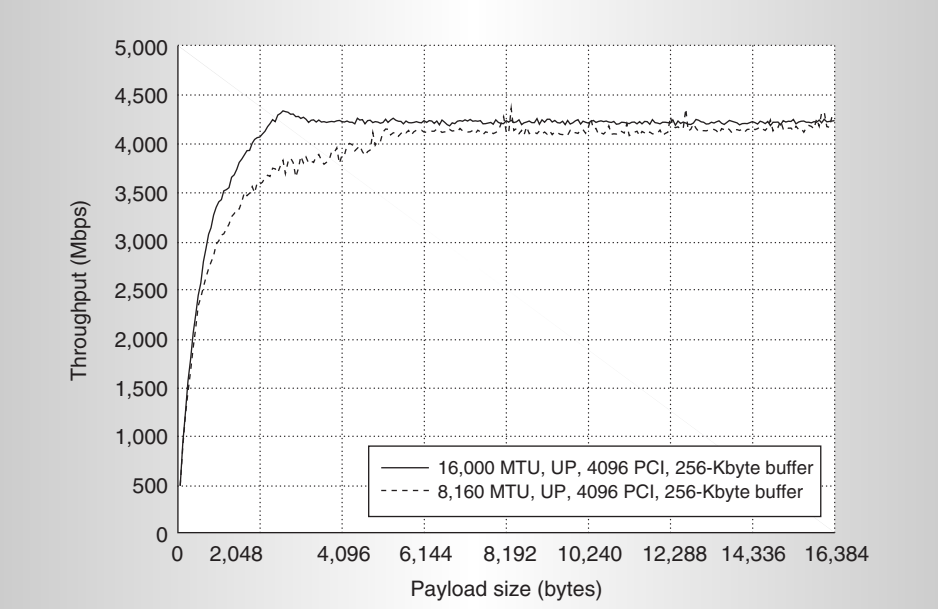

Figure 6. Performance with nonstandard MTU sizes. Peak performance is achieved with nonstandard MTUs that work better with the system's memory subsystem.

with any hardware that supports 9,000-byte MTUs). With a 16,000-byte MTU, we obtain a similar peak bandwidth (4.09 Gbps) but with a higher average bandwidth.

Performance is better with the 8,160-byte MTU than with the larger 9,000-byte MTU because the smaller size allows the entire payload to fit into 8,192-byte memory buffers. The Linux kernel allocates buffers in power-of-two-sized blocks of pages (for example, 32; 64; ... 8,192; or 16,384 bytes; and so on). As the buffer sizes increase, the kernel's memory management subsystem requires more time to allocate them. A 9,000 byte MTU requires 16,384-bytes of buffer space, wasting nearly half of the allocated memory and increasing stress on the kernel.

Interestingly, the curve for the larger MTU shows typical asymptotic growth up to a point, after which it falls and then levels off. Analysis reveals that the congestion window artificially caps the bandwidth. This occurs because the MTU is large relative to the bandwidth-delay product. That is, as the MTU approaches one half of the bandwidth-delay product, it becomes increasingly difficult for available bandwidth to be efficiently used. This problem is discussed at greater detail in a later section. In this case, the congestion window becomes "stuck" at two segments.

End-to-end latency

Although our experiments focus primarily on maximizing the bulk data throughput of the 10GbE adapters, we recognize the importance of latency in LAN and system-area network environments, especially for high-performance scientific applications running on clusters.

Using NetPipe, we measure a 19-µs end-to-end latency between PE2650 systems. Running the tests through the Foundry Fast-

Iron switch increases the latencies to between 24 and 25 µs. Latency increases linearly proportional to payload size, as Figure 7 shows. The latency for packets with a 1,024-byte payload is 23 µs in a back-to-back configuration.

This increase in latency does increase the expected bandwidth-delay product, and consequently, the size to which we should set the socket buffer. This larger socket buffer, however, is still within the scope of the default socket buffer settings. We therefore did not explicitly compensate for this increase in our discussion of the buffer size's effects on throughput, as no compensation should have been needed. However, it is still necessary to set the socket buffer to many times the bandwidth-delay product to eliminate the wide oscillations in throughput observed throughout the previous section.

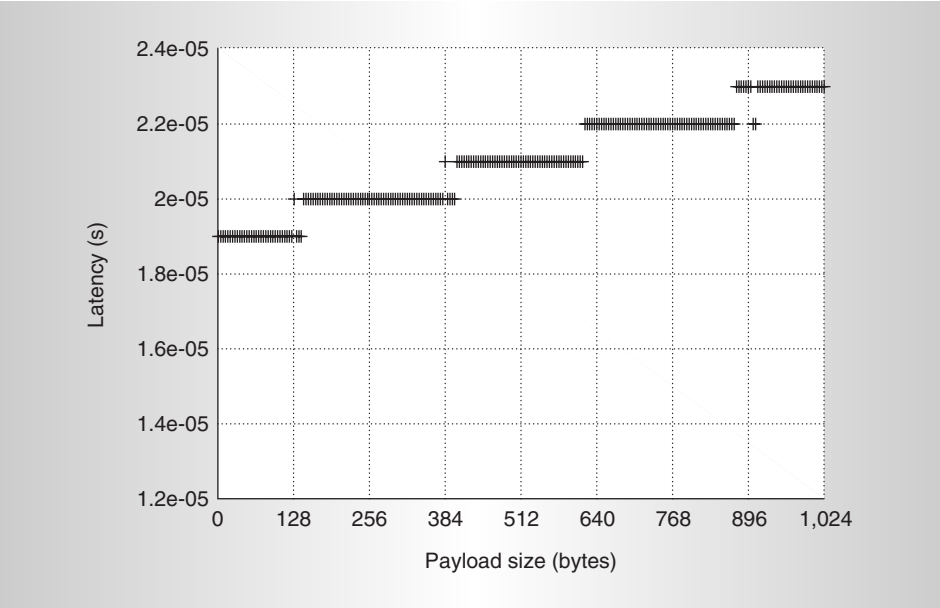

Figure 7. End-to-end latency. Latency increases linearly with payload size.

Eliminating interrupt coalescing can trivially reduce these numbers by about 5 µs. Specifically, we run all the 10GbE tests discussed here with the default 5-µs *receive interrupt delay*. This delay specifies the amount of time to wait between receiving a packet and raising an interrupt to the kernel to service packet reception. In high-performance Ethernet adapters, such delays allow a single interrupt to service multiple received packets (thereby reducing CPU load and the number of required context switches). Although it improves throughput for bandwidth-intensive applications, such an optimization is inappropriate for latency-sensitive applications.

Anecdotal results

Since the publication of our initial 10GbE results,[4] we have continued our testing on an expanded range of systems. These systems include both 32-bit Xeon systems (2.66 GHz and 3.02 GHz) from Intel and 64-bit systems from Intel and Angstrom Microsystems with Intel Itanium2 Deerfield and AMD Opteron 246 CPUs, respectively.

The system architecture of the Xeon is roughly the same as that of the PE2650s. Most notably, the Xeon systems have a 533-MHz FSB (compared to the PE2650s' 400-MHz FSB). The architectures of the 64-bit platforms are fundamentally different from their 32-bit counterparts as well as to each other.

As expected, both sets of faster Xeon-based systems perform better than the PE2650s. The 2.66-GHz nodes reach 4.64 Gbps whereas the 3.02-GHz nodes reach 4.80 Gbps. The disparity in observed performance is significantly less than the disparity in clock rates. The better performance results primarily from the faster FSB rather than the more computationally powerful processor. In addition, networks with the Xeon-based nodes consistently exhibit end-to-end latencies that are 2 µs shorter than those with PE2650s.

The 2.0-GHz AMD Opteron 246 platform achieves a consistent throughput of 6.0 Gbps with a 9,000-byte MTU. With a 16,000-byte MTU, the throughput exceeds 7.3 Gbps. End-to-end latency approaches the sub-10-µs level, holding at 10 µs.

At the November SC2003 conference, held in Phoenix, Arizona, these Opteron-based systems were used in an entry for the annual Bandwidth Challenge competition. The entry, a joint effort by the California Institute of Technology, the Stanford Linear Accelera-

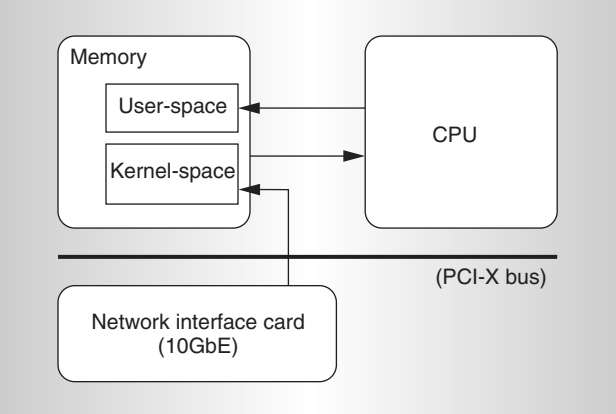

Figure 8. Packet path from the adapter to application. On receiving a packet, the 10GbE adapter copies the packet to kernel memory via the DMA controller. The CPU then copies the data into the user application's memory space.

tor Center, and Los Alamos National Laboratory, won the "Sustained Bandwidth Award" (also dubbed the "Moore's Law Move Over" award) for demonstrating the best vision and articulation of the need for high-performance networks to serve science. The team moved a total of 6,551.134 gigabits of data, reaching a peak of 23.23 gigabits per second. Members of this team from other institutions have also used other platforms, including Xeon and Itanium2-based systems.

As this article goes to press, we have also received a pair of 1.0-GHz Intel Itanium2 Deerfield systems for testing. At this time, we do not have access to the necessary tool chains to compile optimized versions of our testing applications, the kernel, or the 10GbE adapter drivers. Despite these handicaps, the unoptimized Itanium2 achieves an impressive 5.14 Gbps with a 9,000-byte MTU and better than 6.0 Gbps with a 16,000-byte MTU. (Because of the radically different architecture of the Itanium processor family, we expect dramatic performance increases to result from appropriately optimized code.)

Although code optimization is a current obstacle with the Itanium2, the system's Performance Monitoring Unit (PMU)[5] offers tremendous potential gains. The PMU is a processor-based, discrete-event monitor that measures the occurrences of nearly 500 CPU activities at both the system and process levels (for example, it includes nearly 200 individual counters for various cache activities). This PMU is a microbenchmarking tool with unprecedented precision and granularity. We expect it will let us optimize code for the Itanium2 in ways otherwise unimaginable.

Like the PE2650s, the Itanium2 systems perform better with an 8,160-byte MTU than with a 9,000-byte MTU (albeit by a margin of less than 2 percent). Although we do not observe this behavior for the Opterons or the faster Xeons, we do see a leveling off of bandwidth between 8,160- and 9,000-byte MTUs, indicating a performance penalty. In these systems, however, the penalty is less than the gain afforded by the larger MTUs. We attribute this behavior to the Opterons' and Xeons' faster FSB (as well as the Opterons' overall system architecture); we expect the Itanium2s to behave like the Opteron and faster Xeon-based systems once we have access to optimizing tool chains.

Results analysis

Given that the hardware-based bottleneck in the Dell PE2650s is the PCI-X bus at 8.5 Gbps, the peak throughput of 4.11 Gbps is only about half the expected rate. One or more bottlenecks must therefore exist.

During both transmission and reception, packets follow fairly well-defined paths. When transmitting a packet, the CPU copies the packet from a user-space memory buffer to a kernel-space memory buffer (requiring two transfers across the memory bus). The DMA controller then copies the packet across the PCI-X bus to the 10GbE adapter, which sends it to the remote system (requiring a third copy across the memory bus). The receiving adapter at the remote system performs the reverse operation to receive the packet. Figure 8 shows the path for receiving a packet.

This path has at least five possible bottlenecks: the adapter, the PCI-X bus, the memory bus, the FSB, and the CPU. In looking for the bottleneck, we first consider the most external elements of each computing node—the 10GbE adapter and PCI-X bus—and progress inward toward the CPU.

10GbE adapter and PCI-X bus

Because the 10GbE adapter card, rated at up to 10.3 Gbps, achieved over 10 Gbps of throughput in independent tests, we dismiss it as a bottleneck. The PE2650's PCI-X bus

is also not a bottleneck, as we confirm through two sets of tests.

For the first set of tests, we force the 133-MHz PCI-X slot to run at 100 MHz by installing the card on a PCI-X bus that it shared with another PCI-X card. This reduces the effective bandwidth of the PCI-X bus to at most 6.4 Gbps, but it did not affect our test results.

For the second set of tests, we install two 10GbE cards in a single PE2650 but on separate PCI-X buses. We multiplex multiple TCP streams over each card, and therefore each bus, thereby eliminating any bottleneck that a single bus imposed. Multiplexing GbE flows across both 10GbE adapters yields results that are statistically identical to those obtained when multiplexing GbE flows over a single 10GbE adapter, indicating that the bus is not limiting the throughput.

Memory bus and FSB

Using the Dell PE4600s, we determine that memory bandwidth is not a likely bottleneck either. The PE4600s use the GC-HE chipset, offering a theoretical memory bandwidth of 51.2 Gbps. The Stream memory benchmark reports 12.8-Gbps memory bandwidth on these systems, nearly 50 percent better than that of the Dell PE2650s. Despite this higher memory bandwidth, we observe no increase in network performance. These tests are inconclusive, however, because of the architectural differences between the PE2650s and PE4600s.

Tests run on the 2.66- and 3.02-GHz Intel Xeon machines confirm our conclusion that memory bandwidth is not a bottleneck. The Stream results for these machines are within a few percent of the PE2650's results. With a throughput of better than 4.6 Gbps, these machines produce more than 13 percent higher throughput than the PE2650s do, however. The difference in memory bandwidth cannot account for this disparity.

Furthermore, these systems saw roughly equal performance increases over the PE2650s despite their disparate clock rates. Because the performance increase cannot be attributed to the memory bandwidth or clock rate, the performance boost arises primarily from the systems' 533-MHz FSB as the FSB is the only other substantial architectural difference between these systems.

While the bandwidths of the FSB and memory bus are related, they are not the same. The FSB connects the CPU to the memory bus, and therefore, provides an upper bound on the transfer rate between the CPU and memory. The memory bus, however, is shared between the CPU and other bus master devices (including, for instance, the 10GbE adapter). It can therefore attenuate the transfer rate to less than what the FSB supports.

CPU

In all of our experiments, the CPU load remains low enough to suggest that the CPU is not a primary bottleneck. Moreover, disabling TCP time stamps yields no increase in throughput (disabling time stamps gives the CPU more time for TCP processing and should therefore yield greater throughput if the CPU were a bottleneck). Empirically, we ran applications with low to medium CPU load without losing significant throughput.

CPU load, however, might not be the appropriate metric for uncovering a CPU as a bottleneck. Multiple points along the Linux TCP's send and receive path could allow the kernel to omit TCP processing time from the load computation. Furthermore, bulk data transfers have a high context-switching rate, which can artificially deflate the computed average load.

We cannot determine that the CPU is the bottleneck until we can demonstrate that it is computationally saturated (which it is not). If the CPU causes TCP to stall, it is not due to computational tasks but rather inefficient CPU utilization—for example, it might be blocking on noncomputational tasks. Microbenchmark analyses can help determine where the CPU is spending time.

In the absence of such microbenchmark analyses, we rely on previous experiences with high-performance interconnects[1,6] and speculate that intercomponent latencies limit the throughput. Specifically, inefficient cache utilization in reading packets from main memory causes CPU blocking. The performance of the 2.66- and 3.02-GHz Xeon systems supports this conclusion.

Short fat network considerations

As we discussed earlier, throughput can dip sharply for large MTU bandwidth. Specifi-

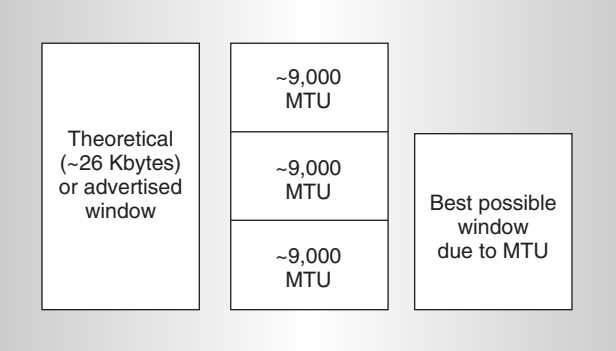

Figure 9. Small window versus large MTU. The Linux TCP implementation forces the congestion window to align with the MTU payload size.

cally, for a 9,000-byte MTU, we see a dip in throughput between payload sizes of 7,436 and 8,948 bytes, preceded by a jump at about 6,272 bytes. We eliminate these dips by setting the window size to be many times the size of the bandwidth-delay product. This solution, however, is only a workaround; it does not identify the problem's source, let alone fix it. Moreover, the solution is contrary to common wisdom about optimizing TCP/IP.

The TCP/IP community focuses more often on high-latency, high-bandwidth network connections—*long fat networks* (LFNs)—than on low-latency, high-bandwidth networks—*short fat networks* (SFNs). A common optimization for LFNs sets the window size to be at least as large as the bandwidth-delay product.[7,8] However, setting too large a window wastes memory and can severely affect throughput by allowing the sender to overrun the link capacity.

Comparatively little work on optimizing SFNs exists. In this study, we find that optimizing TCP for SFNs requires a semantically different heuristic than optimizing TCP for LFNs. Specifically, we find that using large MTUs in SFNs causes the erratic behavior observed with large MTUs and small (though theoretically appropriate) windows. Or more precisely, problems arise when the MTU is large relative to the window size. As the MTU grows, each packet consumes more of the window and receiver buffer, reducing the possible values for the sender's congestion window and creating modal behavior in the receiver's advertised window.

Figure 9 demonstrates how such a problem arises on the sender side. Given 4.1-Gbps throughput and 25-µs latency, we have a theoretical bandwidth-delay product of 26 Kbytes. Because most modern TCP implementations, including Linux's, force the alignment of the congestion window with the MTU's payload size, when we try to saturate an SFN with 9,000-byte packets, we can only fit two complete packets into the SFN pipe before we overrun the capacity. The result is a nearly 33 percent throughput loss. Although this example is extreme, it appears to be exactly the case (albeit with 16,000-byte MTUs) that limits bandwidth when running without "over-sizing" the TCP windows.

Even more bizarre is the receive side's behavior. The Linux kernel uses a window advertisement approach that is designed to limit the bandwidth when buffer space is at a premium. This gives the kernel time to allocate memory to buffers while under memory pressure. A large MTU reduces the possible amount of free memory, as with the congestion window, making it easy for the kernel to become stuck trying to free more buffer space than the bandwidth-delay product will allow.

In our tests, this behavior is intrinsic to the Linux kernel, especially given the buffer-autotuning functionality of the 2.4 kernel series. In addition, a source-code examination of the BSD (Berkeley Software Distribution) stack leaves us certain that analogous behavior would occur there. In fact, we expect every TCP stack to have similar provisions, and thus similar behavior.

Further research should determine the best heuristics for working with SFNs. We anticipate that possible solutions will involve sizing the MTU to maximize the number of segments that can fit in a window; such solutions should be compatible with TCP on both the sender and receiver.

The SFN problem originally manifested itself in our work with TCP over Quadrics QsNet. The problem is far more significant in this 10GbE study, and we expect it to worsen in SFNs as latency continues to decrease, bandwidth increases, and MTU size increases.

Ongoing work

10GbE fits seamlessly into existing infrastructures, working equally well in LAN, sys-

tem-area network, MAN, and WAN environments and not requiring modification to existing code or adopting proprietary software interfaces. Although 10GbE is currently the most expensive interconnect, it will not be for long: Prices will continue to decrease exponentially with the imminent arrival of next-generation 10GbE cards, which use less-expensive multimode optics, and soon, copper. (Ironically, the adoption of a copper interface could benefit InfiniBand at least as much as 10GbE. Because the two interconnects will share the same cabling standard, the increased production volume of 10GbE interfaces will probably lower the cost of InfiniBand far more than InfiniBand manufacturers could hope for on their own.) In addition, switch providers are now offering multiport 10GbE modules and the soon to be available next-generation switch architectures will bring inexpensive layer-2 switching to the 10GbE market.[9]

We anticipate that the remaining performance gaps among Ethernet, InfiniBand, and Quadrics will close significantly within the next year. From a throughput perspective, the IETF's Remote Direct Data Placement (RDDP) effort seeks to translate RDMA's success (over interconnects such as InfiniBand and Quadrics) to Ethernet, thus eliminating Ethernet's last obstacle—that is, the host-interface bottleneck, to high throughput and low processor utilization. From a latency perspective, all the high-speed interconnects will soon run into speed-of-light limitations.

We are currently continuing this work on several fronts. Principally, we are working on microbenchmarking TCP over 10GbE. Using Magnet, we are instrumenting the Linux TCP stack and performing per-packet profiling and tracing of the TCP stack's control path. Magnet lets us profile arbitrary sections of the stack with CPU clock accuracy while 10GbE stresses the stack with previously unfathomable loads. Analysis of this data will provide our research community with an extraordinarily high-resolution picture of the most expensive aspects of TCP processing.[10] In addition, we are also investigating hardware-level microbenchmarking using the PMU on Intel's Itanium2 processors.

Although a better understanding of current performance bottlenecks is essential, our experience with Myrinet and Quadrics (see the "Putting the 10GbE Numbers in Perspective" sidebar) leads us to believe that an operating-system-bypass (OS-bypass) protocol implemented over 10GbE would simultaneously result in throughput capable of saturating the bus bottleneck (that is, 7 to 8 Gbps) with end-to-end latencies well below 10 μs while keeping CPU utilization low. However, because high-performance OS-bypass protocols require an on-board (programmable) network processor on the adapter, Intel's 10GbE adapter currently cannot support such a protocol. MICRO

Acknowledgments

We thank the Intel team—Matthew Baker, Patrick Connor, Caroline Larson, Peter Molnar, Marc Rillema, and Travis Vigil of the LAN Access Division—for their support of this effort and Eric Weigle for his assistance throughout this project. This work was supported by the US DOE Office of Science through Los Alamos National Laboratory contract W-7405-ENG-36.

References

1. N. Boden et al., "Myrinet: A Gigabit-Per-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, Jan.-Feb. 1995, pp. 29-36.
2. F. Petrini et al., "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, Jan.-Feb. 2002, pp. 46-57.
3. M.K. Gardner et al., "Magnet: A Tool for Debugging, Analysis, and Reflection in Computing Systems," *Proc. 3rd IEEE/ACM Int'l Symp. Cluster Computing and the Grid* (CCGrid 2003), IEEE Press, 2003, pp. 310-317.
4. G. Hurwitz and W. Feng, "Initial End-to-End Performance Evaluation of 10-Gigabit Ethernet," *Proc. Hot Interconnects 11* (HotI 2003), IEEE CS Press, 2003, pp. 116-122.
5. *Introduction to Microarchitectural Optimization for Itanium2 Processors*, Intel Corp.; http://www.intel.com/software/products/vtune/techtopic/software_optimization.pdf.
6. D. Tolmie et al., "From HiPPI-800 to HiPPI-6400: A Changing of the Guard and Gateway to the Future," *Proc. 6th Int'l Conf. Parallel Interconnects* (PI 1999), IEEE CS Press, 1999, pp. 194-204.
7. A. Engelhart, M.K. Gardner, and W. Feng,

"Re-Architecting Flow-Control Adaptation for Grid Environments," to appear in *Proc. 18th Int'l Parallel and Distributed Processing Symp.* (IPDPS 04), IEEE CS Press, 2004.
8. W. Feng et al., "Automatic Flow-Control Adaptation for Enhancing Network Performance in Computational Grids," *J. Grid Computing*, vol. 1, no. 1, 2003, pp.63-74.
9. T. Shimizu et al., "A Single Chip Shared Memory Switch with Twelve 10Gb Ethernet Ports," presented at Hot Chips (HotC 2003), datasheet available at http://edevice.fujitsu.com/fj/MARCOM/find/21-4e/pdf/36.pdf.
10. D. Clark et al., "An Analysis of TCP Processing Overhead," *IEEE Comm.*, vol. 27, no. 6, June 1989, pp. 23-29.

Justin (Gus) Hurwitz is a member of the Research and Development in Advanced Network Technology (Radiant) team in the Computer and Computational Sciences Division at Los Alamos National Laboratory. His research interests include protocol and operating-system design for high-performance networking and general approaches to fair utilization and optimization of finite resources. Hurwitz has a bachelor's degree in liberal arts from St. John's College and will pursue graduate degrees in intellectual property and constitutional law and computer science starting this fall.

Wu-chun Feng is a technical staff member and team leader of the Research and Development in Advanced Network Technology (Radiant) team in the Computer and Computational Sciences Division at Los Alamos National Laboratory and a fellow of the Los Alamos Computer Science Institute. His research interests include high-performance networking and computing, network protocols and architecture, bioinformatics, and cybersecurity. Feng has a BS in electrical and computer engineering and music; and an MS in computer engineering from Pennsylvania State University. He has a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the ACM.

Direct questions and comments about this article to Gus Hurwitz, Computer and Computational Sciences Division, Los Alamos Nat'l Lab., PO Box 1663, MS D451, Los Alamos, NM 87545; ghurwitz@lanl.gov.

Get access

to individual IEEE Computer Society documents online.

More than 67,000 articles and conference papers available!

US$9 per article for members

US$19 for nonmembers

http://computer.org/publications/dlib/

IEEE COMPUTER SOCIETY