Proceedings of IEEE Hot Interconnects: 11th Symposium on High-Performance Interconnects; Palo Alto, CA, USA; August 2003.

Initial End-to-End Performance Evaluation of 10-Gigabit Ethernet*

Justin (Gus) Hurwitz, Wu-chun Feng
{ghurwitz,feng}@lanl.gov

Research & Development in Advanced Network Technology (RADIANT)
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract

We present an initial end-to-end performance evaluation of Intel's® 10-Gigabit Ethernet (10GbE) network interface card (or adapter). With appropriate optimizations to the configurations of Linux, TCP, and the 10GbE adapter, we achieve over 4-Gb/s throughput and 21-µs end-to-end latency between applications in a local-area network despite using less capable, lower-end PCs. These results indicate that 10GbE may also be a cost-effective solution for system-area networks in commodity clusters, data centers, and web-server farms as well as wide-area networks in support of computational and data grids.

1. Introduction

From its humble beginnings as shared Ethernet to its current success as switched Ethernet in local- and system-area networks (LANs and SANs) 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. Furthermore, it does so at low implementation cost while maintaining high reliability and relative simplicity in installation (i.e., "plug-n-play"), administration, and maintenance.

Although the recently ratified standard for 10-Gigabit Ethernet (10GbE) differs from earlier Ethernet standards, primarily with respect to operating only over fiber and only in full-duplex mode, it remains Ethernet, and more importantly, does not obsolete current investments in network infrastructure. The 10GbE standard ensures interoperability not only with respect to existing Ethernet but also other networking technologies such as SONET, thus paving the way for Ethernet's expanded use in MANs and WANs.

While 10GbE was mainly intended to allow for easy migration to higher performance levels in backbone infrastructures, we show in this paper that such performance can also be delivered to bandwidth-hungry host applications via the new 10GbE network interface card (or adapter) from Intel®. We first begin with an architectural overview of the adapter in Section 2. In Sections 3 and 4, we present the testing environments and experiments for the 10GbE adapters, respectively. Section 5 provides results and analysis. In Section 6, we examine the bottlenecks that currently impede achieving greater performance. Section 7 compares the 10GbE results with other high-speed interconnects. Finally, we make a few concluding remarks in Section 8.

*This work was supported by the US DOE Office of Science through LANL contract W-7405-ENG-36. This paper is also available as the following technical report: LA-UR 03-2713, April 2003.

2. Architecture of a 10GbE Adapter

The world's first host-based 10GbE adapter, officially known as the Intel® PRO/10GbE LR server adapter, introduces benefits of 10GbE connectivity into LAN and SAN environments, thereby accommodating the growing number of large-scale systems and bandwidth-intensive applications such as imaging and data mirroring. This first-generation 10GbE adapter contains a 10GbE controller that is implemented in a single chip and contains both the MAC and PHY layer functions. The controller, in turn, is optimized for servers that use the PCI and PCI-X I/O bus backplanes.

Figure 1 provides an architectural overview of the 10GbE adapter, which consists of three main components: 82597EX 10GbE controller, 512-KB of flash memory, and 1310-nm serial optics. The 10GbE controller provides an Ethernet interface that delivers high performance by providing direct access to all memory without using mapping registers, minimizing interrupts and programmed I/O (PIO) read access that are required to manage the device, and offloading the host CPU of simple tasks such as TCP checksum calculations.

As is common practice with high-performance adapters such as Myricom's Myrinet [2] and Quadrics' QsNet [12], the 10GbE adapter frees up host-CPU cycles by performing certain tasks (in silicon) on behalf of the host CPU. In contrast to Myrinet and QsNet adapters, however, the 10GbE

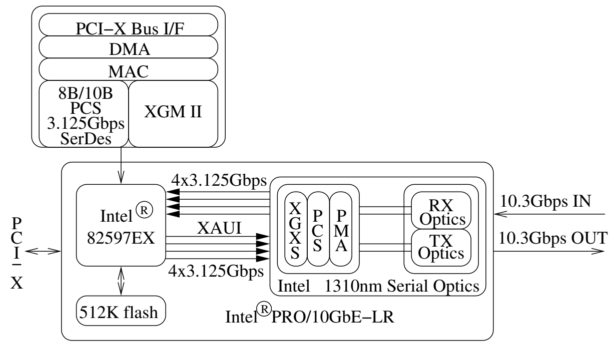

Figure 1. Architecture of the 10GbE Adapter

adapter focuses on host off-loading of certain TCP/IP tasks[1] rather than on RDMA and source routing. As a result, unlike Myrinet and QsNet, the 10GbE adapter provides a general-purpose solution to applications, a solution that does *not* require any modification to applications code to achieve high performance, e.g., over 4 Gb/s from application to application with an end-to-end latency near 20 μs.

3. Testing Environments

We evaluate the performance of the Intel 10GbE adapter in three different LAN/SAN environments:

- 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,
- Multiple flows through the FastIron 1500 switch,

where the computers that host the 10GbE adapters are either Dell® PowerEdge™ 2650 (PE2650) servers 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 GB of memory and a dedicated 133-MHz PCI-X bus for the 10GbE adapter. Theoretically, this architectural configuration provides 25.6-Gb/s CPU bandwidth, up to 25.6-Gb/s memory bandwidth, and 8.5-Gb/s network bandwidth via the PCI-X bus.

Each PE4600 contains dual 2.4-GHz Intel Xeon CPUs running on a 400-MHz FSB, using a ServerWorks GC-HE chipset with 1 GB of memory and a dedicated 100-MHz PCI-X bus for the 10GbE adapter. This particular configuration provides theoretical bandwidths of 25.6-Gb/s, 51.2-Gb/s, and 6.4-Gb/s for the CPU, memory, and PCI-X bus, respectively.

In addition to the above hosts, we use a Foundry FastIron 1500 switch for both our indirect single-flow and multi-flow tests. In the latter case, the switch aggregates GbE and 10GbE streams from (or to) many hosts into a 10GbE stream to (or from) a single host. The total backplane bandwidth (480 Gb/s) in the switch far exceeds the needs of our tests as each of the two 10GbE ports is limited to 8.5 Gb/s.

From a software perspective, all the above hosts run current installations of Debian Linux with customized kernel builds and tuned TCP/IP stacks. Specific kernels that we used include 2.4.19, 2.4.19-ac4, 2.4.19-rmap15b, 2.4.20, and 2.5.44. Because the performance differences between these various kernel builds prove negligible, we do not report the running kernel version in any of the results.

[1] Specifically, TCP & IP checksums and TCP segmentation.

4. Experiments

In this paper, our experiments focus on the performance of bulk data transfer. We use two tools to measure network throughput — NTTCP [11] and IPerf [6] — and note that the experimental results from these two tools correspond to another oft-used tool called `netperf` [9].

NTTCP and IPerf work by measuring 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, a `ttcp` variant, 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 performance to the application, NTTCP provides more valuable data in these tests. We therefore present primarily NTTCP data throughout the paper. (Typically, the performance difference between the two is within 2-3%. In no case does IPerf yield results significantly contrary to those of NTTCP.)

To estimate the end-to-end latency between a pair of 10GbE adapters, we use NetPipe [10] to obtain an averaged round-trip time over several single-byte, ping-pong tests and then divide by two.

To measure the memory bandwidth of our Dell PowerEdge systems, we use STREAM [13].

To estimate the CPU load across our throughput tests, we sample `/proc/loadavg` at five- to ten-second intervals.

And finally, to better facilitate the analysis of data transfers, we make use of two tools, `tcpdump` [14] and MAGNET [5]. `tcpdump` is commonly available and used for analyzing protocols at the wire level. MAGNET is a publicly available tool developed by our research team at Los Alamos National Laboratory.

5. Experimental Results and Analysis

We begin our experiments with a stock TCP stack. From this starting point, we implement optimizations one by one to improve network performance between two identical Dell PE2650s connected via 10GbE.

The more common device and TCP optimizations result in little to no performance gains. These optimizations include changing variables such as the device transmit queue lengths and the use of TCP timestamps.

We then tune TCP by calculating the ideal bandwidth-delay product and setting the TCP window sizes accordingly. Running in a LAN, we expect this product to be relatively small, even at 10GbE speeds. The latency is observed to be 21 μs running back-to-back and 27 μs running through the Foundry switch. At full 10GbE speed, this results in a maximum bandwidth-delay product of about 52 KB, well below the default window setting of 64 KB. 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. Minor performance increases may be had by increasing the default window settings to be equal to the maximum window settings.

As our immediate goal is to demonstrate the maximum possible performance across the 10GbE adapters so that it may be reproduced, we present our optimizations in a cumulative manner. Further analysis of the specific benefits of each optimization will be addressed in future work.

5.1. Baseline: Stock TCP with Standard MTU Sizes

We begin with single-flow experiments across a pair of unoptimized (stock) Dell PE2650s using a standard 1500-byte MTU as well as a 9000-byte jumbo-frame 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 KB at increments ranging in size from 32 to 128 bytes.

Figure 2 shows that using a larger MTU size produces 40-60% better throughput than the standard 1500-byte MTU. This result can be directly attributed to the additional load that 1500-byte MTUs impose on the CPU, e.g., interrupts every 1500 bytes instead of every 9000 bytes. Specifically, for 1500-byte MTUs, the CPU load is approximately 0.9 on both the send and receive hosts while the CPU load is only 0.4 for 9000-byte MTUs.

We observe bandwidth peaks at 1.8 Gb/s with a 1500-byte MTU and 2.7 Gb/s with a 9000-byte MTU. The sharp drops and jumps in bandwidth are addressed in Section 5.4.

5.2. 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 4096 bytes. Thus, in these experiments, we increase the PCI-X burst transfer size (i.e., MMRBC register) to 4096 bytes. As shown in Figure 3, this simple optimization to the 10GbE hardware improves peak performance for a 9000-byte MTU to over 3.6 Gb/s, a throughput increase of 33% over the baseline case. With a 1500-byte MTU, increasing the burst size only produces a marginal increase in throughput, indicating that the burst size does not hinder performance with respect to smaller MTUs.

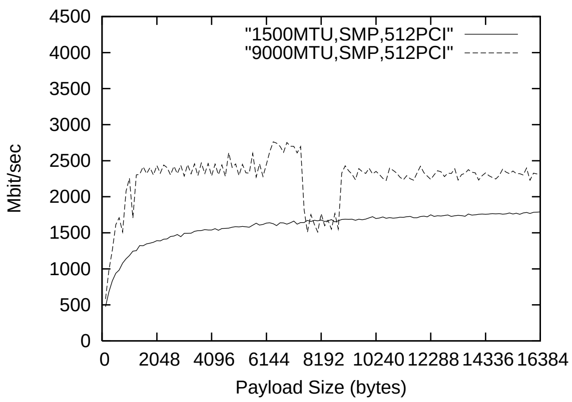

Figure 2. Stock TCP: 1500- vs. 9000-byte MTU

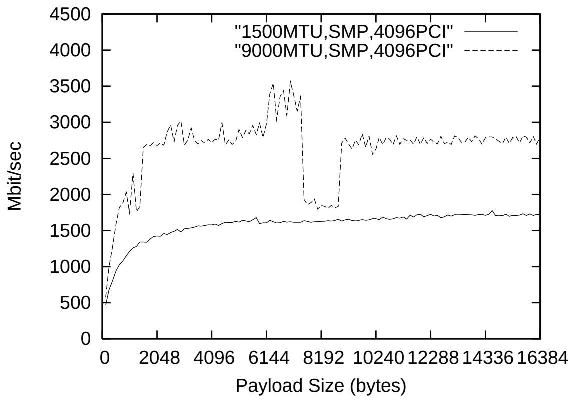

Figure 3. 4096-byte PCI-X Burst Transfer Size

The CPU load remains relatively unchanged from the baseline numbers reported above.

5.3. Running a Uniprocessor Kernel

With the optimization to the PCI-X burst transfer size in place, our next *counterintuitive* optimization is to replace the SMP kernel with a uniprocessor (UP) kernel. At the present time, the P4 Xeon SMP architecture assigns each interrupt to a single CPU instead of processing them in a round-robin manner between CPUs. As a result, the interrupt context code of the 10GbE driver cannot take advantage of a SMP configuration. Coupled with the additional cost of kernel locking, this currently results in UP kernels running faster than SMP kernels.

When switching from an SMP to UP kernel, Figure 4 shows that the average throughput for 9000-byte MTUs improves by approximately 20% (compared to Figure 3). For 1500-byte MTUs, both the average and maximum throughputs increase by nearly 20%.

In these tests, we observe that the CPU load was uniformly lower than in the SMP tests. This is primarily due to less time being spent by the CPU in a spin-locked state.

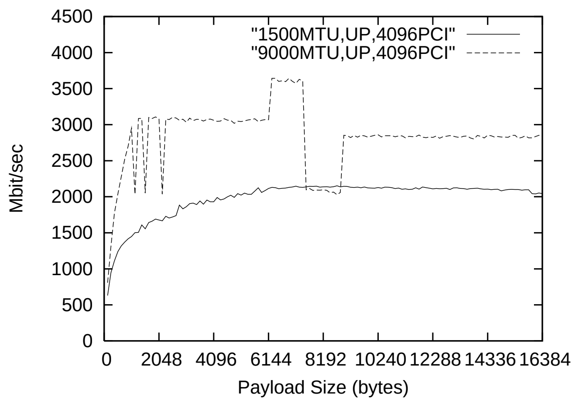

Figure 4. Uniprocessor Kernel

5.4. Memory Tuning

In Section 5.1, we observed sharp decreases and increases in the throughput for the baseline case of a 9000-byte MTU. And although the frequency of the throughput changes decreased with each optimization that we made, the magnitude of the throughput changes increased sharply.

Using `tcpdump`, we trace the causes of this behavior to ineffecient window use by both the sender and receiver. To fully explain these causes requires a technical discussion that is beyond the scope of this paper. Briefly, the causes are a result of (1) a large MSS[2] relative to the ideal window size and (2) Linux's TCP stack keeping both the advertised and congestion windows MSS-aligned.[3]

On the receive side, the actual advertised window is significantly below the expected values from Section 5, e.g., 52 KB. This behavior is a consequence of Linux's implementation of the Silly Window Syndrome (SWS) avoidance algorithm [3]. Because the advertised window is kept aligned with the MSS, it cannot be increased by small amounts.[4] The larger that the MSS is relative to the advertised window, the harder it becomes to increase the advertised window.

On the sender side, performance is similarly limited because the congestion window is kept aligned with the MSS [1]. For instance, with a 21-μs latency, the theoretical ideal window size for 10GbE is about 52 KB, as noted in Section 5. With a 9000-byte MTU (8948-byte MSS, with options), this translates to just under 6 packets per window. Neither the sender nor the receiver can transfer 6 complete packets; both can do at best 5 packets. This attenuates the ideal data rate by nearly 15%. The effect can be even more severe at lower data rates.

We partially overcome the first limitation by simply increasing the default socket buffer size. (Performance with the larger default socket buffers is discussed in the next section.) However, this is a poor "band-aid" solution. There should be no need to set the socket buffer to many times the ideal window size in any environment; in a WAN environment, setting the socket buffer too large can severely hurt performance. The low latencies and large MSS in LAN/SAN environments, however, undermine the conventional wisdom surrounding window settings.

Furthermore, this solution does not prevent the sender from being artificially limited by the congestion window. A better solution might take the form of modifications to the SWS avoidance and congestion-window algorithms to allow for fractional MSS increments when the number of segments per window is small.

We first ran into this problem when working with IP over the Quadrics interconnect. The problem manifested itself to a lesser extent due to the lower data rates of Quadrics QsNet (3.2 Gb/s). However, as latency decreases, bandwidth increases, and perhaps most importantly, MTU size increases, this problem will only exacerbate itself further.

5.5. Peak Performance

Figure 5 shows the performance for 1500-byte and 9000-byte MTUs with 256-KB socket buffer sizes. The peak bandwidth is 2.47 Gb/s with a 1500-byte MTU and 3.9 Gb/s with a 9000-byte MTU.

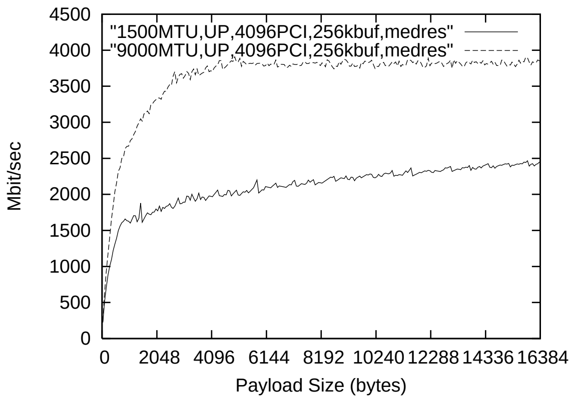

Figure 5. Peak Performance, Standard MTUs

We achieve better performance with non-standard MTU sizes. The peak observed bandwidth achieved with a 9000-byte jumbo-frame compatible MTU (Figure 6) is 4.11 Gb/s with an 8160-byte MTU.[5] With a larger MTU of 16000 bytes, we get similar peak bandwidth (4.09 Gb/s) but with a higher average.

Interestingly, the curve for the larger MTU shows typical asymptotic growth up to a point, at which time it falls, and then levels off. The analysis of our tests reveal that, as discussed in the previous section, the congestion window is artificially capping the bandwidth. In this particular case,

[2] Loosely speaking, MSS = MTU – packet headers.

[3] Linux is not unique in this behavior; it is shared by most modern TCPs.

[4] The window is aligned by $adv\_wnd = (\frac{avail\_wnd}{MSS}) * MSS$. This rounds the window *down* to the nearest increment of MSS bytes.

[5] 8160-byte MTUs can be used in conjunction with any hardware that supports 9000-byte MTUs.

the congestion window gets "stuck" at 2 segments. (As a point of reference, Figure 6 also labels the theoretical maximum bandwidths for Gigabit Ethernet, Myrinet [7, 8], and QsNet [12].)

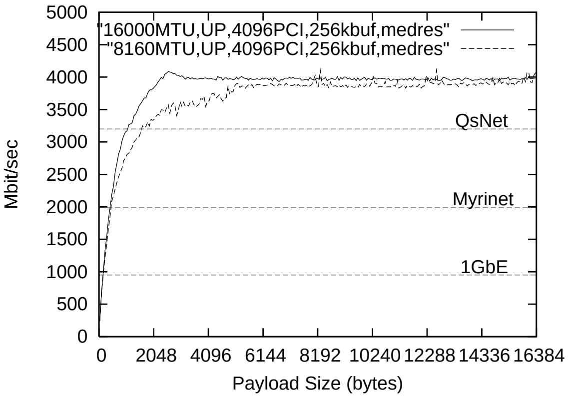

Figure 6. Peak Performance, Other MTUs

6. Analysis of Performance Limits

Given that the hardware based bottleneck in the Dell PE2650s is the PCI-X bus at 8.5 Gb/s, the peak throughput of 4.11 Gb/s is only about half the rate that we expect.

In all of our experiments, the CPU load remains low enough for us to believe that the CPU is not a primary bottleneck. This is supported by the fact that disabling TCP timestamps yields no increase in throughput: disabling timestamps gives the CPU more time for TCP processing and should therefore yield greater throughput if the CPU were a bottleneck.

It is possible that the inherent complexity of the TCP receive path (relative to the transmit path) results in a receive path bottleneck. In addition, while we have anecdotal evidence that the ServerWorks GC-LE chipset is capable of sustaining better than 90% of the PCI-X bandwidth it has not been confirmed. In short, both the TCP receive path and the actual PCI-X bus performance are potential bottlenecks. To evaluate both, we conduct multi-flow testing of the 10GbE adapters through our Foundry FastIron 1500 switch. These tests allow us to aggregate nearly 16 Gb/s from multiple 1GbE-enabled hosts to one or two 10GbE-enabled hosts (or vice versa).

In the first set of tests, we transmit to (or from) a single 10GbE adapter. These tests identify bottlenecks in the receive path, relative to the transmit path, by multiplexing the processing required for one path across several machines while keeping the aggregated path to (or from) a single 10GbE-enabled Dell PE2650 constant. These results unexpectedly show that the transmit and receive paths are of statistically equal performance. Given the relative complexity of the receive path compared to the transmit path, we initially expect to see better performance when the 10GbE adapter is transmitting to multiple hosts than when receiving from multiple hosts. Previous experience provides a likely explanation for this behavior. Packets from multiple hosts are more likely to be received in frequent bursts than are packets from a single host, allowing the receive path to benefit from interrupt coalescing, thereby increasing the receive-side bandwidth relative to transmit bandwidth.[6].

Multiplexing GbE flows across both 10GbE adapters yields results statistically identical to those obtained using a single 10GbE adapter. We can therefore rule out the PCI-X bus as a primary bottleneck. In addition, this test also eliminates the 10GbE adapter as a primary bottleneck.

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 Gb/s; the STREAM [13] memory benchmark reports 12.8-Gb/s memory bandwidth on these systems, nearly 50% better than that of the Dell PE2650s. Despite this higher memory bandwidth, we observe no increase in network performance. There are, unfortunately, enough architectural differences between the PE2650 and PE4600 that further investigation is required.

Thus, this (obviously) leaves the host software's ability to move data between every component in the system as the likely bottleneck. Given that the Linux kernel's packet generator reports a maximum total bandwidth of approximately 5.5 Gb/s (8160-bytes packets at approximately 88,400 packets/sec), the host software itself attenuates throughput by 3 Gb/s (i.e., 8.5 Gb/s - 5.5 Gb/s) and is the primary bottleneck toward achieving higher performance.

7. Putting the 10GbE Numbers in Perspective

In this section, we discuss the actual performance that one can expect out of Gigabit Ethernet, Myrinet, and even QsNet (rather than the theoretical maximums shown in Figure 6) in order to provide a better reference point for the 10GbE results.

Our extensive experience with 1GbE chipsets[7] allows us to achieve near line-speed performance with a 1500-byte MTU in a LAN/SAN environment with most payload sizes. With additional optimizations in a WAN environment, similar performance can be achieved while still using a 1500-byte MTU.[8]

For comparison to Myrinet, we report Myricom's published performance numbers for their adapters [7, 8]. Using their proprietary GM API, sustained unidirectional bandwidth is 1.984 Gb/s and bidirectional bandwidth is 3.912 Gb/s. Both of these numbers are within 3% of the 2-Gb/s unidirectional hardware limit. The GM API provide latencies on the order of 6-7 μs. To use this API, however, requires rewriting portions of legacy applications' code.

[6] This confirms results in [4], albeit by very different means.

[7] e.g., Intel's e1000 line and Broadcom's Tigon3

[8] Internet2 Land Speed Record set on November 19, 2002: single-stream TCP/IP of 923 Mb/s over a distance of 10,978 km.

Myrinet provides a TCP/IP emulation layer to avoid this problem. The performance of this layer, however is notably less than that of the GM API. Bandwidth drops to 1.853 Gb/s, and latencies skyrocket to over 30 μs.

Our experiences with Quadrics' QsNet produced unidirectional bandwidth and latency numbers of 2.456 Gb/s and 4.9 μs, respectively, using QsNet's Elan3 API. As with Myrinet's GM API, the Elan3 API requires application codes to rewrite their network code, typically from a sockets API to Elan3 API. To address this issue, Quadrics also has a highly efficient implementation of TCP/IP that produces 2.240 Gb/s of bandwidth and under 30-μs latency. For additional performance results, see [12].

In summary, when comparing TCP/IP performance across all interconnect technologies, our initial 10GbE bandwidth number (4.11 Gb/s) is over 300% better than GbE, over 120% better than Myrinet, and over 80% than QsNet while our 10GbE latency number (21 μs) is roughly 400% better than GbE and 50% better than Myrinet and QsNet. Finally, even when comparing our 10GbE TCP/IP performance numbers with the numbers from other interconnects' specialized network software (e.g., GM and Elan3), we find the 10GbE performance to be highly competitive.

8. Conclusion

With the current generation of SAN interconnects such as Myrinet and QsNet being theoretically hardware-capped at 2 Gb/s and 3.2 Gb/s, respectively, achieving 4 Gb/s of end-to-end throughput with 10GbE makes it a viable *commodity* interconnect for SANs in addition to LANs. However, its Achilles' heel is its 21-μs end-to-end latency, which is about three times slower than Myrinet/GM (but 1.5 times better than Myrinet/IP) and four times slower than QsNet/Elan3 (but 1.5 times better than QsNet/IP). This difference can be attributed mainly to the host software.

In recent tests on 533-MHz FSB Intel E7505-based systems running Linux, we have achieved 4.64 Gb/s throughput. The greatest difference between these systems and the PE2650s is the FSB, which indicates that the CPU's ability to move — but not process — data, might be an important bottleneck. These tests have not yet been fully analyzed.

To continue this work, we are currently instrumenting the Linux TCP stack with MAGNET to perform per-packet profiling and tracing of the stack's control path. MAGNET allows us to profile arbitrary sections of the stack with CPU-clock accuracy, while 10GbE stresses the stack with previously impossible loads. Analysis of this data is giving us an unprecedentedly high-resolution picture of the most expensive aspects of TCP processing overhead [4].

While a better understanding of current performance bottlenecks is essential, the authors' past experience with Myrinet and Quadrics leads them to believe that an OS-bypass protocol implemented over 10GbE would result in throughput approaching 7-8 Gb/s and end-to-end latencies on the order of 10 μs. However, because high-performance OS-bypass protocols require an on-board (programmable) network processor on the adapter, the 10GbE adapter from Intel currently cannot support an OS-bypass protocol.

The availability of 10-Gigabit Ethernet provides a remarkable opportunity for network researchers in LANs, SANS, MANs, and even WANs. The unprecedented (commodity) performance offered by the Intel PRO/10GbE server adapter has also enabled us[9] to smash the Internet2 Land Speed Record (http://lsr.internet2.edu) on February 27, 2003, by sustaining 2.38 Gb/s across 10,037 km between Sunnyvale, California and Geneva, Switzerland, i.e., 23,888,060,000,000,000 meters-bits/sec.

Acknowledgements

We would like to thank the Intel team — Patrick Connor, Caroline Larson and Peter Molnar of the LAN Access Division — for their support of this effort and Eric Weigle for his assistance throughout this project.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *RFC-2581*, April 1999

[2] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, "Myrinet: A Gigabit-Per-Second Local Area Network," *IEEE Micro*, Vol. 15, No. 1, January/February 1995.

[3] D. Clark, "Window and Acknowledgment Strategy in TCP," *RFC-813*, July 1982.

[4] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications*, Vol. 27, No. 6, Jume, 1989, pp. 23-29.

[5] M. K. Gardner, W. Feng, M. Broxton, A Engelhart, and G. Hurwitz, "MAGNET: A Tool for Debugging, Analysis and Reflection in Computing Systems," *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'2003)*, May 2003.

[6] "Iperf 1.6 - The TCP/UDP Bandwidth Measurement Tool," http://dast.nlanr.net/Projects/ Iperf/.

[7] "Myrinet Ethernet Emulation (TCP/IP & UDP/IP) Performance," http://www.myri.com/myrinet/performance/ip.html.

[8] "Myrinet Performance Measurements," http://www.myri.com/myrinet/performance/index.html.

[9] "Netperf: Public Netperf Homepage," http://www.netperf.org/.

[10] "NetPIPE," http://www.scl.ameslab.gov/netpipe/.

[11] "NTTCP: New TTCP program," http://www.leo.org/~ elmar/nttcp/.

[12] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, Vol. 22, No. 1, January/Feburary 2002.

[13] "STREAM," http://www.cs.virginia.edu/stream/.

[14] "TCPDUMP Public Repository," http://www.tcpdump.org.

[9] California Institute of Technology, CERN, Los Alamos National Laboratory, and Stanford Linear Accelerator Center