SLAC-PUB-10198

Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters, and Grids: A Case Study [*]

Wu-chun Feng,[†] Justin (Gus) Hurwitz,[†] Harvey Newman,[††] Sylvain Ravot,[††] R. Les Cottrell,[‡‡]
Olivier Martin,[‡] Fabrizio Coccetti,[‡‡] Cheng Jin,[††] Xiaoliang (David) Wei,[††] and Steven Low[††]

[†]Los Alamos National Laboratory (LANL)
Computer & Computational Sciences Division
Research and Development in Advanced Network Technology
Los Alamos, NM 87545
{feng,ghurwitz}@lanl.gov

[††]California Institute of Technology (CalTech)
Pasadena, CA 91125
Charles C. Lauritsen Laboratory of High Energy Physics
{newman,ravot}@caltech.edu
Computer Science & Electrical Engineering Dept.
{chengjin,weixl,slow}@caltech.edu

[‡]European Organization for Nuclear Research (CERN)
Information Technology Division
Geneva, Switzerland
{Olivier.Martin}@cern.ch

[‡‡]Stanford Linear Accelerator Center (SLAC)
SLAC Computing Services
Menlo Park, CA 94025
{cottrell,cfabrizo}@SLAC.stanford.edu

Abstract

This paper presents a case study of the 10-Gigabit Ethernet (10GbE) adapter from Intel®. Specifically, with appropriate optimizations to the configurations of the 10GbE adapter and TCP, we demonstrate that the 10GbE adapter can perform well in local-area, storage-area, system-area, and wide-area networks.

For local-area, storage-area, and system-area networks in support of networks of workstations, network-attached storage, and clusters, respectively, we can achieve over 7-Gb/s end-to-end throughput and 12-μs end-to-end latency between applications running on Linux-based PCs. For the wide-area network in support of grids, we broke the recently-set Internet2 Land Speed Record by 2.5 times by sustaining an end-to-end TCP/IP throughput of 2.38 Gb/s between Sunnyvale, California and Geneva, Switzerland (i.e., 10,037 kilometers) to move over a terabyte of data in less than an hour. Thus, the above results indicate that 10GbE may be a cost-effective solution across a multitude of computing environments.

[*]This work was supported by the US DOE Office of Science through LANL contract W-7405-ENG-36 Caltech contract DE-FG03-92-ER40701, and SLAC contract DE-AC03-76SF00515. Additional support was provided by NSF through grant ANI-0230967, AFOSR through grant F49620-03-1-0119, and ARO through grant DAAD19-02-1-0283.

This paper is also available as the following LANL technical report: LA-UR 03-5728, July 2003.

1. Introduction

Thirty years ago in a May 1973 memo, Robert Metcalfe described the technology that would evolve into today's ubiquitous Ethernet protocol. By 1974, Metcalfe and his colleague, David Boggs, built their first Ethernet; and by 1975, they demonstrated what was at the time a dazzling 2.94 Mb/s of throughput over the 10-Mb/s Ethernet medium. Since that time, Ethernet has proliferated and evolved tremendously and has done so in virtual lockstep with the ubiquitous TCP/IP (Transmission Control Protocol / Internet Protocol) protocol suite which was started at Stanford University in the summer of 1973. Today's Ethernet carries 99.99% of Internet packets and bears little resemblance to the original Ethernet [11]. About the only aspect of the original Ethernet that still remains is its packet format.

So, even though the recently ratified 10-Gigabit Ethernet (10GbE) standard differs from earlier Ethernet standards, mainly with respect to operating only over fiber and only in full-duplex mode, it still remains Ethernet, and more importantly, does not obsolete current investments in network infrastructure. Furthermore, the 10GbE standard ensures interoperability not only with respect to existing Ethernet but also other networking technologies such as SONET (i.e., Ethernet over SONET), thus paving the way for Ethernet's expanded use in metropolitan-area networks (MANs) and wide-area networks (WANs). Finally, while 10GbE is arguably intended to ease migration to higher aggregate performance levels in institutional network-backbone infrastructures, the results in this paper will demonstrate 10GbE's versatility in a myriad of computing environments.

The remainder of the paper is organized as follows: Section 2 briefly describes the architecture of the Intel 10GbE

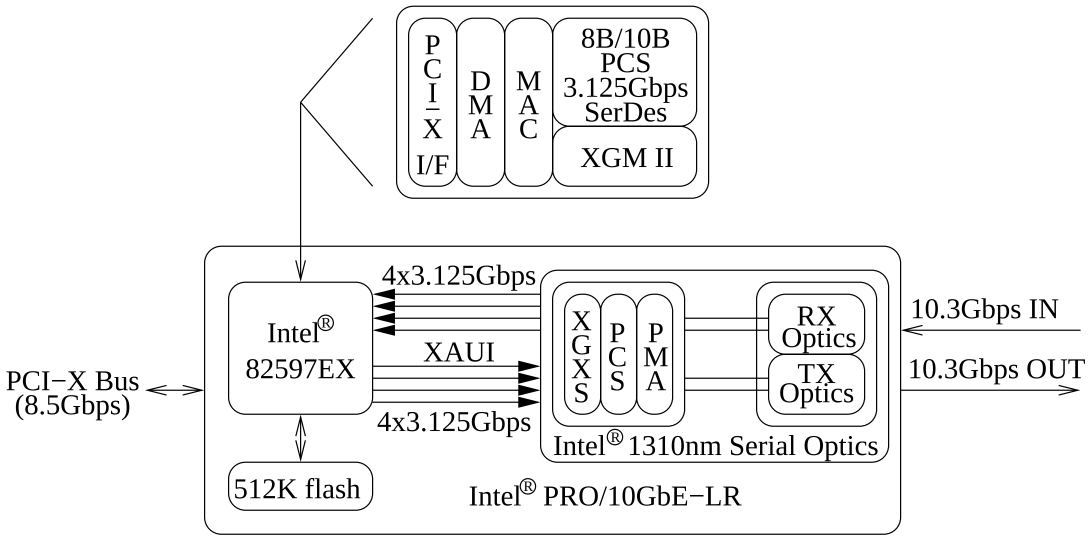


Figure 1. Architecture of the 10GbE Adapter

adapter. Section 3 presents the local-area network (LAN) and system-area network (SAN) testing environments, experiments, and results and analysis, and Section 4 does the same for the wide-area network (WAN). Finally, we summarize and conclude in Section 5.

2. Architecture of a 10GbE Adapter

The recent arrival of the Intel® PRO/10GbE LR™ server adapter paves the way for 10GbE to become an all-encompassing technology from LANs and SANs to MANs and WANs. This first-generation 10GbE adapter consists of three major components: Intel 82597EX™ 10GbE controller, 512-KB of flash memory, and Intel 1310-nm serial optics, as shown in Figure 1.

The 10GbE controller provides an Ethernet interface that delivers high performance by providing direct access to all memory without using mapping registers, minimizing programmed I/O (PIO) read access required to manage the device, minimizing interrupts required to manage the device, and off-loading the host CPU of simple tasks such as TCP checksum calculations. Its implementation is in a single chip and contains both the medium-access control (MAC) and physical (PHY) layer functions, as shown at the top of Figure 1. The PHY layer, to the right of the MAC layer in Figure 1, consists of an 8B/10B physical coding sublayer and a 10-gigabit media independent interface (XGM II). To the left of the MAC layer is a direct-memory access (DMA) engine and the “peripheral component interconnect extended” interface (PCI-X I/F). The former handles the transmit and receive data and descriptor transfers between the host memory and on-chip memory while the latter provides a complete glueless interface to a 33/66-MHz, 32/64-bit PCI bus or a 33/66/100/133-MHz, 32/64-bit PCI-X bus.

As is already common practice with high-performance adapters such as Myricom’s Myrinet [2] and Quadrics’ QsNet [17], the 10GbE adapter frees up host-CPU cycles by performing certain tasks (in silicon) on behalf of the host CPU. In contrast to the Myrinet and QsNet adapters, however, the 10GbE adapter focuses on host off-loading of certain TCP/IP tasks[1] rather than on remote direct-memory access (RDMA) and source routing. As a result, unlike Myrinet and QsNet, the 10GbE adapter provides a general-purpose, TCP/IP-based solution to applications, a solution that does *not* require any modification to application codes to achieve high performance, e.g., as high as 7 Gb/s between end-host applications with an end-to-end latency as low as 12 μs.

As we will see later, achieving higher throughput will require either efficient offloading of network tasks from software to hardware (e.g., IETF’s RDMA-over-IP effort, known as RDDP or remote direct data placement [19]) and/or significantly faster machines with large memory bandwidth. Achieving *substantially* higher throughput, e.g., approaching 10 Gb/s, will *not* be possible until the PCI-X hardware bottleneck in a PC is addressed. Currently, the peak bandwidth of a 133-MHz, 64-bit PCI-X bus in a PC is 8.5 Gb/s (see left-hand side of Figure 1), which is less than half the 20.6-Gb/s bidirectional data rate (see right-hand side of Figure 1) that the Intel 10GbE adapter can support.

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

3. LAN/SAN Tests

In this section, we present our LAN/SAN experimental results and analysis. The results here show that we can achieve over 7 Gb/s of throughput and 12-μs end-to-end latency with TCP/IP.

3.1. Testing Environments

We evaluate the performance of the Intel 10GbE adapter in three different LAN/SAN environments, as shown in Figure 2:

(a) Direct single flow between two computers connected back-to-back via a crossover cable,
(b) Indirect single flow between two computers through a Foundry® FastIron™ 1500 switch,
(c) Multiple flows through the Foundry FastIron 1500 switch,

where the computers that host the 10GbE adapters are either Dell® PowerEdge™ 2650 (PE2650) servers or Dell PowerEdge 4600 (PE4600) servers.

(Recently, we have also conducted additional back-to-back tests on computer systems, provided by Intel, with a slightly faster CPU and front-side bus (FSB). Given that we only had these systems for a few days, we merely use the results from these systems for anecdotal purposes as well as a "sanity-check" on our more exhaustive tests on the Dell PowerEdge servers.[2])

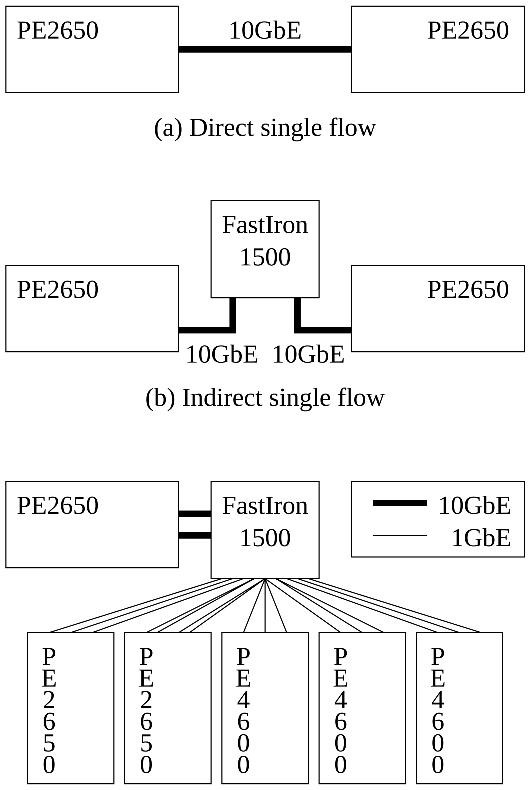
Each PE2650 contains dual 2.2-GHz Intel Xeon™ CPUs running on a 400-MHz front-side bus (FSB), using a ServerWorks® GC-LE chipset with 1 GB of memory and a dedicated 133-MHz PCI-X bus for the 10GbE adapter. Theoretically, this architectural configuration provides 25.6-Gb/s CPU bandwidth, up to 25.6-Gb/s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

(The systems provided by Intel contain dual 2.66-GHz Intel Xeon CPUs running on a 533-MHz FSB, using Intel's E7505 chipset with 2 GB of memory and a dedicated 100-MHz PCI-X bus for the 10GbE adapter. This architecture provides theoretical bandwidths of 34-Gb/s, 25.6-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.

[2]We also have even more promising (but again, preliminary) 10GbE results on a 1.5-GHz Itanium-II system.



(a) Direct single flow

(b) Indirect single flow

(c) Multiple flows through the switch

Figure 2. LAN/SAN Testing Environments

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.

3.2. Experiments

In this paper, our experiments focus on the performance of bulk data transfer. We use two tools to measure network throughput — NTTCP [16] and Iperf [8] — and note that the experimental results from these two tools correspond to another oft-used tool called `netperf` [14].

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 [15] 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 [10].

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` [21] and MAGNET [6]. `tcpdump` is commonly available and used for analyzing protocols at the wire level. MAGNET is a publicly available tool developed in part by the co-authors from Los Alamos National Laboratory. MAGNET allowed us to trace and profile the paths taken by individual packets through the TCP stack with negligible effect on network performance. By observing a random sampling of packets, we were able to quantify how many packets take each possible path, the cost of each path, and the conditions necessary for a packet to take a "faster" path. (We note that this is just one of many possible uses for MAGNET.)

3.3. Experimental Results

This section presents an abridged account of the optimizations that we implemented to achieve greater than 4 Gb/s of throughput for a single TCP/IP flow between a pair of low-end 2.2-GHz Dell PE2650s. For a more in-depth discussion of each optimization step, we refer the reader to [7].

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, as shown in Figure 2(a).

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.

3.3.1 Bandwidth

Before commencing our formal testing, we tune the TCP window sizes by calculating the ideal bandwidth-delay product and setting the TCP window sizes accordingly [22]. Running in a LAN or SAN, we expect this product to be relatively small, even at 10GbE speeds. The initial latency numbers that we observed are 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 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.

Stock TCP

We begin with single-flow experiments across a pair of unoptimized (stock) Dell PE2650s using standard 1500-byte and 9000-byte (jumboframe) maximum transfer units (MTUs). In their stock (i.e., default) 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 3 shows the baseline results. Using a larger MTU size produces 40-60% better throughput than the standard 1500-byte MTU. 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.

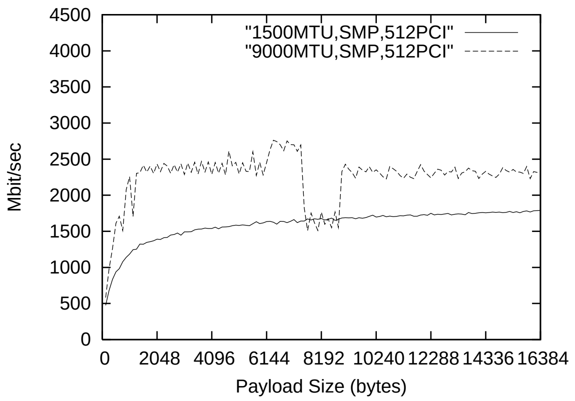


Figure 3. Throughput of Stock TCP: 1500- vs. 9000-byte MTU

Stock TCP + Increased PCI-X Burst Size

Next, we increase the PCI-X burst transfer size (i.e., MMRBC register) from 512 bytes to 4096 bytes. Although this optimization only produces a marginal increase in throughput for 1500-byte MTUs, it dramatically improves performance with 9000-byte MTUs. The peak throughput increases to over 3.6 Gb/s, a throughput increase of 33% over the baseline case, while the average throughput increases to 2.63 Gb/s, an increase of 17%. The CPU load remains relatively unchanged from the baseline numbers reported above.

Stock TCP + Increased PCI-X Burst Size + Uniprocessor

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. Consequently, our next *counterintuitive* optimization is to replace the SMP kernel with a uniprocessor (UP) kernel. This change further improves average throughput for 9000-byte MTUs by approximately 10% to 2.9 Gb/s. For 1500-byte MTUs, the average and maximum throughputs increase by about 25% and 20% to 2.0 Gb/s and 2.15 Gb/s, respectively. In addition, the CPU load was uniformly lower than in the SMP tests.

TCP with Oversized Windows + Increased PCI-X Burst Size + Uniprocessor

Though we calculated that the default window sizes were much larger than the bandwidth-delay product, we improve throughput further by setting the window size to be four times larger than the default setting (and roughly ten times larger than the actual bandwidth-delay product). That is, we set the receive socket buffer to 256 KB in `/proc/sys/net/ipv4/tcp_rmem`. With a 256-KB socket buffer, the peak bandwidth increases to 2.47 Gb/s with 1500-byte MTUs and 3.9 Gb/s with 9000-byte MTUs, as shown in Figure 4. A detailed discussion of this particular optimization will be presented in Section 3.5.1.

Tuning the MTU Size

We achieve even better performance with non-standard MTU sizes. Figure 5 shows that the peak bandwidth achieved is 4.11 Gb/s with an 8160-byte MTU.[3] This result is a direct consequence of Linux's memory-allocation system. Linux allocates memory from pools of "power-of-2" sized blocks, An 8160-byte MTU allows an entire packet, i.e., payload + TCP/IP headers + Ethernet headers, to fit in a single 8192-byte block whereas a 9000-byte MTU requires the kernel to allocate a 16384-byte block, thus wasting roughly 7000 bytes.

The above discussion leads us to our next logical step — using the largest MTU that the Intel 10GbE adapter can support, namely 16000 bytes. With a 16000-byte MTU, the peak throughput achieved is 4.09 Gb/s, virtually identical to the 8160-byte MTU case. However, the average throughput with the larger MTU is clearly much higher, as shown in Figure 5. The surprisingly marginal increase in throughput is due to the sender's congestion window artificially limiting throughput, as discussed in more detail in Section 3.5.1. It is worth noting that larger MTUs, and consequently, larger block sizes are not without consequences. Using larger blocks places far greater stress on the kernel's memory-allocation subsystem because it is generally harder to find the contiguous pages required for the larger blocks.

(Note: As points of reference, Figure 5 also labels the theoretical maximum bandwidths for Gigabit Ethernet, Myrinet [12, 13], and QsNet [17]. A more detailed discussion of these interconnects versus 10-Gigabit Ethernet is presented in Section 3.5.4.)

[3]8160-byte MTUs can be used in conjunction with any hardware that supports 9000-byte MTUs.

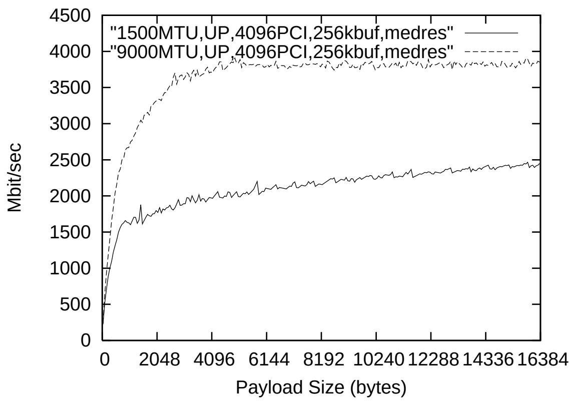


Figure 4. Throughput of TCP with Oversized Windows and Increased PCI-X Burst Size Running on a Uniprocessor Kernel

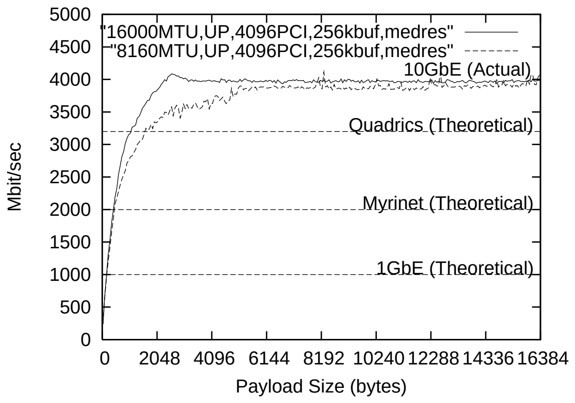


Figure 5. Throughput of Cumulative Optimizations with Non-Standard MTUs

3.3.2 Latency

Although our experiments did not focus on optimizations with respect to latency, we are acutely aware that low latency is critical for scientific applications such as global climate modeling [24]. Therefore, we report our preliminary latency results here, demonstrate how end-to-end latency can be improved, and suggest other avenues for improvement.

Our latency measurements, running NetPipe between a pair of 2.2-GHz Dell PowerEdge 2650s, produce 19-μs end-to-end latency when the machines are connected back-to-back and 25-μs end-to-end latency when going through the Foundry FastIron 1500 switch. As the payload size increases from one byte to 1024 bytes, latencies increase linearly in a stepwise fashion, as shown in Figure 6. Over the entire range of payloads, the end-to-end latency increases a total of 20% such that the back-to-back latency is 23 μs and the end-to-end latency through the switch is 28 μs.

To reduce these latency numbers even further, particularly for latency-sensitive environments, we trivially shave off an additional 5 μs (i.e., down to 14-μs end-to-end latency) by simply turning off a feature called *interrupt coalescing* (Figure 7). In our bandwidth tests and the above latency tests in Figure 6, we had configured the 10GbE adapters with a 5-μs interrupt delay. This delay is the period that the 10GbE card *waits* between receiving a packet and raising an interrupt to signal packet reception. Such a delay allows multiple packet receptions to be coalesced into a single interrupt, thus reducing the CPU load on the host at the expense of latency.

From the perspective of the host system, newer versions of Linux (which we have yet to test) implement a New API (NAPI) for network processing. This NAPI allows for better handling of network adapters (or network interface cards) that support hardware-based interrupt coalescing by coalescing the software processing of packets from the network adapter's ring buffer. The older API queues each received packet separately, regardless of whether multiple packets are received in a single interrupt, resulting in wasted time in an interrupt context to process each individual packet. In the NAPI, the interrupt context only queues the fact that packets are ready to be processed and schedules the packets to be received from the network interface card later, outside the scope of the interrupt context. This approach provides two major benefits: (1) less time spent in an interrupt context and (2) more efficient processing of packets, which ultimately decreases the load that the 10GbE card places on the receiving host. (In systems where the host CPU is a bottleneck, it would also result in higher bandwidth.)

Newer versions of Linux also support TCP Segmentation Offload (TSO, also known as Large Send), another offload feature supported by Intel's 10GbE adapters. TSO allows the transmitting system to use a large (64 KB) "virtual" MTU. The 10GbE card then re-segments the payload into smaller packets for transmission. As the NAPI does for receiving systems, the implementation of TSO should reduce the CPU load on transmitting systems, and in many cases, will increase throughput.

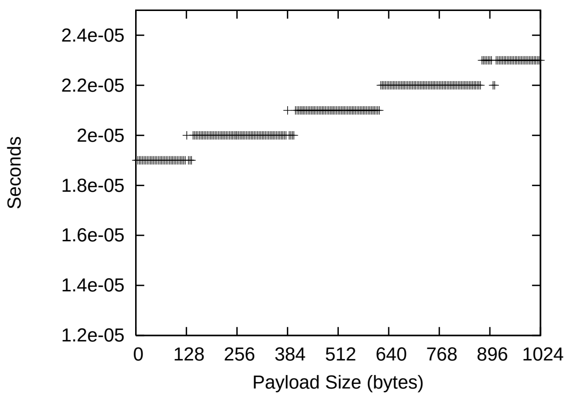


Figure 6. End-to-End Latency in Test Configuration

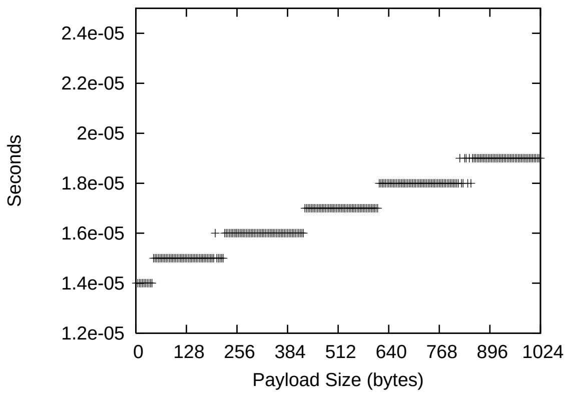


Figure 7. End-to-End Latency without Interrupt Coalescing

3.4. Anecdotal Results

In addition to the more thorough experimental results above, we have preliminary results on machines that we had very limited access to. The first set of these machines, provided by Intel and described briefly in Section 3.1, primarily differ from the Dell PE2650s by having dual 2.66-GHz CPUs and a 533-MHz FSB. The essentially "out-of-box" performance of the 10GbE adapters on these machines was 4.64-Gb/s in a back-to-back configuration. (It is worth noting that this performance required TCP timestamps to be disabled because enabling timestamps reduced throughput by approximately 10%. The reason for this behavior is explained in Section 3.5.2). Latency numbers between these machines appeared to be up to 2 μs less than those seen between the Dell PE2650s, indicating that end-to-end latencies could reach as low as 12 μs.

We also have anecdotal 10GbE results for a 1-GHz quad-processor Itanium-II system. Specifically, multiple 1GbE clients were aggregated through a 10GbE switch into a single 1-GHz quad-processor Itanium-II system with a 10GbE adapter. Performing the same set of optimizations, as described above, on the Itanium-II system produces a unidirectional throughput of 7.2 Gb/s.

3.5. Discussion of LAN/SAN Results

This subsection presents a more detailed analysis of our experiments, proposes ways to improve performance further, and puts our 10GbE results in context with respect to other network interconnects.

3.5.1. LAN/SAN TCP Windows

Back in Section 3.3, the "9000-byte MTU" throughput results in Figure 3 showed a marked dip for payload sizes between 7436 and 8948 bytes (as well as a series of smaller, but higher frequency, dips across all payload sizes). Even optimizing the PCI-X burst transfer size and using a uniprocessor kernel did not eliminate the marked dip; however, oversizing the TCP windows *did* eliminate the marked dip, as shown in Figure 4.

Using `tcpdump` and by monitoring the kernel's internal state variables with MAGNET, we trace the causes of this behavior to inefficient window use by both the sender and receiver. Briefly, the throughput dips are a result of (1) a large Maximum Segment Size (MSS[4]) relative to the ideal window size and (2) Linux's TCP stack keeping both the advertised and congestion windows MSS-aligned.[5]

On the receive side, the actual advertised window is significantly smaller than the expected value of 48 KB, as calculated in Section 3.3. 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.[6] The larger that the MSS is relative to the advertised window, the harder it becomes to increase the advertised window.[7]

On the sender side, performance is similarly limited because the congestion window is kept aligned with the MSS [1]. For instance, with a 19-μs latency, the theoretical ideal window size for 10GbE is about 48 KB. With a 9000-byte MTU (8948-byte MSS with options), this translates to about 5.5 packets per window. Thus, neither the sender nor the receiver can transfer 6 complete packets; both can do at best 5 packets. This immediately attenuates the ideal data rate by nearly 17%.

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

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

[6]The window is aligned by $advertised\_window = (int)(\frac{available\_window}{MSS}) * MSS$. This rounds the window *down* to the nearest increment of MSS bytes.

[7]We note that this is actually an argument for *not* increasing the MTU size.

The effect can be even more severe with smaller bandwidth-delay products, as shown in Figure 8. Such a situation can arise on either the send or receive side. We describe the send-side case as it is easier to understand — the theoretical (or ideal) window size is a hard limit that the sender cannot exceed (e.g., 26 KB in Figure 8). Because the congestion-control window must be MSS-aligned on the sender side, the actual congestion-control window is only 18 KB, or 31% less than the allowable 26-KB window.

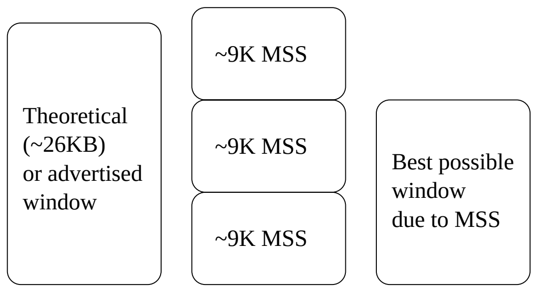

Theoretical (~26KB) or advertised window
~9K MSS
~9K MSS
~9K MSS
Best possible window due to MSS

Figure 8. Ideal vs. MSS-allowed Window

In addition to the above inefficiencies, the *full* socket-buffer memory (as allocated for window use by the kernel) on the receive may *never* be available to be advertised as the window. That is, although the window that the receiver advertises is a function of the available buffer memory, there is no requirement that this available buffer memory be a multiple of the MSS. As a result, the amount of memory that is *not* a multiple of the MSS is simply wasted.

To further complicate matters, the MSS is not necessarily constant between communicating hosts, i.e., the sender's MSS is no necessarily equal to the receiver's MSS. Why is this an issue? Consider a sender MSS of 8960 bytes and a receiver MSS of 8948 bytes and let the receiver have 33,000 bytes of available socket memory.[8] The receiver will then advertise a window of $(int)\frac{33000}{8948} * 8948 = 26,844$ bytes, or 19% less than the available 33,000 bytes. With a sender MSS of 8960 bytes, the maximum size that the congestion window can be, due to the fact that the congestion window must be kept MSS-aligned, is $(int)\frac{26844}{8940} * 8940 = 17,920$ bytes, or 33% less than the receiver's advertised window and nearly 50% smaller than the actual available socket memory of 33,000 bytes. If window scaling is used (as is becoming more prevalent with the proliferation of gigabit-per-second networks in the wide-area network), the situation is exacerbated even further because the accuracy of the window diminishes as the scaling factor in-

[8]We frequently observe such a situation, i.e., the sender using a larger MSS value than the receiver, in our tests. This is apparently a result of how the receiver estimates the sender's MSS and might well be an implementation bug.

creases. Even without scaling, in this example, the sender and receiver are both artificially limiting the bandwidth by a total of nearly 50%.

To overcome the above problems, network researchers routinely (and in many cases, blindly) increase the default socket buffer sizes even further until performance improves. However, this is a poor "band-aid" solution in general. 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 impact performance, as noted in Table 1. Furthermore, the low latencies and large MSS in LAN/SAN environments are contrary to this "conventional wisdom" of setting the window/buffer size so large. In addition, the above solution does not prevent the sender from being artificially limited by the congestion window, as noted earlier. Better solutions might include the following:

- Modifying the SWS avoidance and congestion-window algorithms to allow for fractional MSS increments when the number of segments per window is small.
- Making "better" (or at least, more conservative) adaptive calculations of the MSS on the receive side.
- Allowing the sender to incrementally decrease the MSS if the congestion window is locked in a steady state.

In summary, although larger MTUs tend to improve network throughput and reduce CPU load, they magnify problems that were not as apparent with the standard 1500-byte MTU, particularly in a LAN/SAN environment. Specifically, a large MSS relative to the ideal window size (as in a LAN/SAN) and TCP's "enforcement" of MSS-aligned windows results in lower-than-expected, highly-fluctuating throughput.

(Interestingly, we first ran into these problems 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 [9], this problem will only exacerbate itself further.)

3.5.2. 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 to indicate that the CPU is not a primary bottleneck. This is supported by the fact that disabling TCP timestamps on the PE2650s 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. Furthermore, by moving from 1500- to 9000-byte MTUs, we typically expect a performance increase of 3x-6x if the CPU were the bottleneck. Our results show an increase of only 1.5x-2x.

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.[9]

Multiplexing GbE flows across two 10GbE adapters on independent buses in a single machine 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 and Intel-provided dual 2.66-GHz systems, 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 [10] 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.

The Intel-provided systems, however, further confirm that memory bandwidth is not a likely bottleneck. STREAM results for the PE2650s and Intel-provided systems are within a few percent of each other. However, the Intel-provided sys-

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

tems achieved 4.64 Gb/s with virtually no optimizations while the heavily optimized PE2650s only reached 4.11 Gb/s. This difference in performance, better than 13%, cannot be accounted for by differences in the memory bandwidth alone. A more likely, but related, explanation is the change in front-side bus (FSB) speed. Further investigation is needed, however, before making such a conclusion.

All of the above results are supported by Linux's packet generator. The packet generator bypasses the TCP/IP and UDP/IP stacks entirely. It is a kernel-level loop that transmits pre-formed "dummy" UDP packets directly to the adapter (that is, it is single-copy). We observe a maximum bandwidth of 5.5 Gb/s (8160-byte packets at approximately 88,400 packets/sec) on the PE2650s when using the packet generator. This rate is maintained when additional load is placed on the CPU, indicating that the CPU is not a bottleneck. Because the packet generator is single-copy (as opposed to the IP stack's triple-copy), memory bandwidth is not a limit to its performance. Our observed TCP bandwidth, which is *not* single-copy, is about 75% of the packet generator bandwidth. It is reasonable to expect, however, that the TCP/IP stack would attenuate the packet generator's performance by about 25%. While this does not demonstratively rule out memory bandwidth as a bottleneck, it does indicate the the observed performance is in line with what we should expect were the memory bandwidth not a bottleneck.

Overall, we are left to believe that the host software's ability to move data between every component in the system is likely the bottleneck. Given that the Linux kernel's packet generator reports a maximum total bandwidth of approximately 5.5 Gb/s on the PE2650s, this movement of data 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.

3.5.3. Breaking the Bottlenecks

To improve network performance, contemporary network adapters provide various means for reducing the load on the host operating system and hardware. These methods, oftentimes referred to as "offload" techniques, attempt to move networking tasks from the host processor to the adapter. A few of these techniques have been discussed above, e.g., TSO, checksum offloading, and interrupt coalescing. While each of these techniques do offer measurable performance gains, their main benefit is in decreasing the load on the host CPU rather than substantially improving throughput and end-to-end latency.

However, in all of our experiments, we have seen that CPU load is not the primary bottleneck. Thus, the host system's bandwidth is more likely I/O-limited than CPU-limited. So, the key to improving bandwidth is to either improve the host system's ability to move data, decrease the amount of data that needs to be moved, or decrease the number of times that the data needs to be moved across the memory bus [18, 19].

The two prevailing approaches towards improving TCP/IP performance are TCP offload engines (TOEs) and RDMA over IP [19]. The former will offload the entire TCP stack into the silicon of the network adapter while the latter will leverage the use of a processor on the network adapter to run TCP software and enable direct data placement from the network adapter into application space (RDMA over IP), thus eliminating excessive copying across the memory bus and virtually eliminating processing load from the host CPU.

It is our belief that TOE is *not* the best solution for a general-purpose TCP. Many previous attempts at TOE engines failed because design and implementation errors are virtually impossible to change once they are cast in silicon (unless an FPGA, field-programmable gate array, or other field-upgradable processor is used). In a TOE, the TCP implementation is effectively hidden from the host CPU software, making it difficult to interface with. Hardware design and production costs are significantly larger than with so-called "dumb" adapters. Most importantly, the adapter must still transfer data across the memory and I/O buses, introducing a potential source of data errors, *errors that a TOE has no way to detect or correct*.

Our experience has shown that hardware can have design defects that lead to data errors. In high-load environments, heat, high bit rates, or poor hardware designs often contribute in error rates far higher than predicted by hardware manufacturers [20]. This is especially the case with "bleeding edge" and other high-performance hardware. For instance, in our educated opinion, received TCP data should *not* be checksummed in the adapter; rather they must be computed once the data has reached the system's main memory. Unfortunately, current proposals for TOEs perform checksums in the adapter.

Rather than implement a complete TOE, we would like to see an implementation of a TCP header-parsing engine, e.g., *a la* ST [18]. Briefly, such an engine would use a hash table of established sockets to transfer the payload of incoming packets directly into user memory. The headers would then be passed on to the kernel for normal processing. In the event of out-of-order packets, or other TCP circumstances that cannot be handled on a fast path, the adapter passes the entire packet on to the kernel for traditional processing.

Such an implementation requires a small amount of logic and buffer memory on the adapter itself as well as a simple interface for interacting with the kernel. Furthermore, this approach keeps the TCP logic on the host while allowing the adapter to transfer the payload. It also isolates TCP from hardware-dependent design decisions and allows for an easy upgrade, maintenance, and development path for the TCP.

An additional possibility that we hope to see implemented in the future is the placement of network adapters on the Memory Controller Hub (MCH), typically found on the Northbridge. Intel's Communication Streaming Architecture

(CSA) [5] is such an implementation for Gigabit Ethernet. Placing the adapter on the MCH allows for the bypass of the I/O bus. By eliminating the I/O bus, we eliminate both an important bottleneck and a source of data errors. In addition, the adapter is better enabled to compute the checksum of the payload once placed in system memory. Such a computation would significantly improve the accuracy and reliability of the checksums.

3.5.4. Putting the 10GbE LAN/SAN Numbers in Perspective

We now turn to discuss the actual performance that one can expect out of Gigabit Ethernet, Myrinet, and even QsNet (rather than the theoretical maximums shown in Figure 5) in order to provide a better reference point for the 10GbE results.

Our extensive experience with 1GbE chipsets (e.g., Intel's e1000 line and Broadcom's Tigon3) 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 but with a 9000-byte MTU. [10]

For comparison to Myrinet, we report Myricom's published performance numbers for their adapters [12, 13]. 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 to 7 μs. To use this API, however, may oftentimes require rewriting portions of legacy applications' code.

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 may require 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 [17].

In summary, when comparing TCP/IP performance across all interconnect technologies, our established 10GbE throughput number (4.11 Gb/s) is over 300% better than GbE, over 120% better than Myrinet, and over 80% than QsNet while our established 10GbE latency number (19 μs) is roughly 400% better than GbE and 50% better than Myrinet and QsNet. Our preliminary 10GbE throughput number of 7.2 Gb/s on the Itanium-II systems is nearly 700% better than GbE, nearly 300% better than Myrinet, and over 200% better than 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 — generally better with respect to throughput and slightly worse with respect to latency.

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

4. WAN Tests

In this section, we provide a brief account of our wide-area network tests, and more specifically, our Internet2 Land Speed Record effort back in February 2003.

4.1. Environment

In this section, we present the performance of our 10GbE adapters across a WAN that stretched from Sunnyvale, California to Geneva, Switzerland (i.e., 10,037 kilometers), as shown in Figure 9. The WAN utilized a loaned Level3 OC-192 POS (10 Gb/s) circuit from the Level3 PoP at Sunnyvale to StarLight in Chicago and then traversed the transatlantic LHCnet OC-48 POS (2.5 Gb/s) circuit between Chicago and Geneva.

At Sunnyvale, the OC-192 POS circuit originated at a Cisco GSR 12406 router and was connected to a Juniper T640 (NSF TeraGrid router) at Starlight in Chicago. The TeraGrid router then crossed over to a Cisco 7609 router before heading overseas to a Cisco 7606 router in Geneva. In traversing roughly halfway across the world, our WAN traffic crossed two different autonomous systems: AS75 (TeraGrid) and AS503 (CERN).

At each end point, we had a dual 2.4-GHz Intel Xeon PC with 2 GB of memory and a dedicated 133-MHz PCI-X bus for the 10GbE adapter. Each node ran Linux 2.4.19 with jumbo frames and optimized its buffer size to be approximately the bandwidth-delay product, i.e.,

```
echo ''4096 87380 128388607'' >
  /proc/sys/net/ipv4/tcp_rmem
echo ''4096 65530 128388607'' >
  /proc/sys/net/ipv4/tcp_wmem
echo 128388607 > /proc/sys/net/core/wmem_max
echo 128388607 > /proc/sys/net/core/rmem_max

/sbin/ifconfig eth1 txqueuelen 10000
/sbin/ifconfig eth1 mtu 9000
```

4.2. Experiment & Result

The additive increase, multiplicative decrease (AIMD) algorithm governs TCP's bandwidth through a sender-side state variable called the congestion window. The AIMD algorithm modifies the size of the congestion window according to the network conditions. Without any packet loss, the congestion

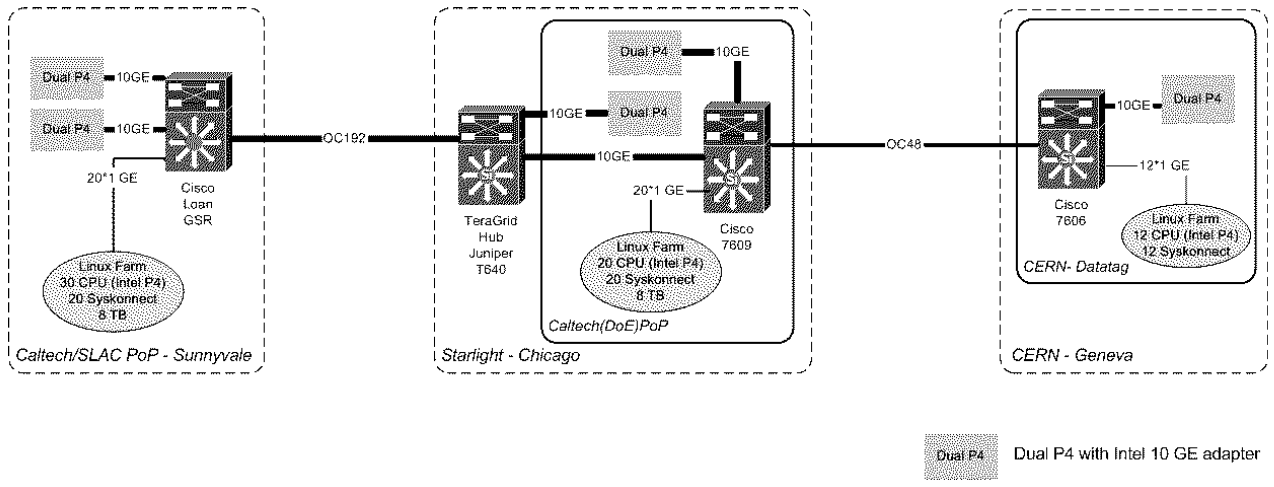


Figure 9. 10-Gigabit Ethernet WAN Environment

window normally opens at a constant rate of one segment per round-trip time; but each time a congestion signal is received (i.e., packet loss), the congestion window is halved.

However, as the bandwidth and latency increase, and hence, increasing the bandwidth-delay product, the effect of a single packet loss is disastrous in these long fat networks with gargantuan bandwidth-delay products. For example, in future scenarios, e.g., 10-Gb/s connection from end-to-end between Geneva and Sunnyvale, Table 1 shows how long it takes to recover from a packet loss and eventually return to the original transmission rate (prior to the packet loss), assuming that the congestion window size is equal to the bandwidth-delay product when the packet is lost.

To avoid this problem, one simply needs to reduce the packet-loss rate. But how? In our environment, packet loss is due exclusively to congestion in the network, i.e., packets are dropped when the number of unacknowledged packets exceeds the available capacity of the network. In order to reduce the packet-loss rate, we must “stop” the increase of the congestion window before it reaches a congested state. Because explicit control of the congestion-control window is not possible, we turn to the flow-control window (TCP buffer sizing) to implicitly cap the congestion-window size to the bandwidth-delay product of the wide-area network so that the network approaches congestion but avoids it altogether.

As a result, using only a single TCP/IP stream between Sunnyvale and Geneva, we achieved an end-to-end throughput of 2.38 Gb/s over a distance of 10,037 kilometers. This translates into moving a terabyte of data in less than one hour.

Why is this result so remarkable? First, it is well-known that TCP end-to-end throughput is inversely proportional to round-trip time; that is, the longer the round-trip time (in this case, 180 ms, or approximately 10,000 times larger than the round-trip time in the LAN/SAN), the lower the throughput. Second, given that the bottleneck bandwidth is the transatlantic LHCnet OC-48 POS at 2.5 Gb/s, achieving 2.38 Gb/s means that the connection operated at roughly 99% payload efficiency. Third, the end-to-end WAN throughput is actually *larger* than what an application user typically sees in a LAN/SAN environment. Fourth, our results smashed both the single- and multi-stream Internet2 Land Speed Records by 2.5 times.

5. 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 over 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 12-μs (best-case) end-to-end latency, which is 1.7 times slower than Myrinet/GM (but over two times faster than Myrinet/IP) and 2.4 times slower than QsNet/Elan3 (but over two times faster than QsNet/IP). These performance differences can be attributed mainly to the host software.

In recent tests on the dual 2.66-GHz CPUs with 533-MHz FSB Intel E7505-based systems running Linux, we have achieved 4.64 Gb/s throughput “out of the box.” 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 im-

Path	Bandwidth Assumption	RTT (ms)	MSS (bytes)	Time to Recover
LAN	10 Gbps	1	1460	428 ms
Geneva - Chicago	10 Gb/s	120	1460	1 hr 42 min
Geneva - Chicago	10 Gb/s	120	8960	17 min
Geneva - Sunnyvale	10 Gb/s	180	1460	3 hr 51 min
Geneva - Sunnyvale	10 Gb/s	180	8960	38 min

Table 1. Time to Recover from a Single Packet Loss

possible 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, like RDMA over IP, implemented over 10GbE would result in throughput approaching 8 Gb/s, end-to-end latencies below 10 μs, and a CPU load approaching zero. 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 in support of networks of workstations, clusters, distributed clusters, and grids, respectively. The unprecedented (commodity) performance offered by the Intel PRO/10GbE server adapter also enabled us 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

First and foremost, we would like to thank the Intel team — Patrick Connor, Caroline Larson, Peter Molnar, and Marc Rillema of the LAN Access Division — for their tremendous support of our research efforts and Eric Weigle for his assistance throughout this project.

With respect to the wide-area network, *none* of the research (e.g., Internet2 Land Speed Record) would have been possible without the generous support and contributions of many people and institutions. From an infrastructure standpoint, we thank Linda Winkler and Tom DeFanti from Argonne National Laboratory and the University of Illinois of Chicago, respectively, for the use of a Juniper T640 TeraGrid router at Starlight in Chicago. Equipment-wise, in addition to Intel, we thank

- Paul Fernes from Level(3) Communications for the OC-192 link from Sunnyvale to Chicago,
- Doug Walsten from Cisco Systems for the 10GbE switch support in Sunnyvale, and
- Peter Kersten and John Szewc from Foundry Networks for providing Los Alamos National Laboratory with a 10GbE switch to run their LAN/SAN tests on.

If any of the above infrastructure pieces would have fallen through, no attempt at the Internet2 Land Speed Record would have ever been made.

Last, but not least, we gratefully acknowledge the invaluable contributions and support of the following colleagues during our record-breaking effort on the Internet2 Land Speed Record: Julian Bunn and Suresh Singh of Caltech; Paolo Moroni and Daniel Davids of CERN; Edoardo Martelli of CERN/DataTAG; Gary Buhrmaster of SLAC; and Eric Weigle and Adam Engelhart of Los Alamos National Laboratory. In addition, we wish to thank the Internet2 consortium (http://www.internet2.edu) for creating a venue for "supernetworking" achievements.

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, June, 1989, pp. 23-29.

[5] "Communication Streaming Architecture: Reducing the PCI Network Bottleneck," Intel Whitepaper 252451-002. Available at: http://www.intel.com/design/network/papers/252451.htm.

[6] 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.

[7] G. Hurwitz, and W. Feng, "Initial End-to-End Performance Evaluation of 10-Gigabit Ethernet," *Proceedings of Hot Interconnects 11 (HotI'03)*, August 2003.

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

[9] M. Mathis, "Pushing Up the Internet MTU," *Presentation to Miami Joint Techs*, Feburary 2003.

[10] J. McCalpin, "STREAM: Sustainable Memory Bandwidth in High-Performance Computers," http://www.cs.virginia.edu/ stream/.

[11] R. Metcalfe and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, No. 5, July 1976.

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

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

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

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

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

[17] 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.

[18] I. Philp and Y.-L. Liong, "The Scheduled Transfer (ST) Protocol," *3rd International Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC'99), Lecture Notes in Computer Science, Vol. 1602*, January 1999.

[19] A. Romanow, and S. Bailey, "An Overview of RDMA over IP," *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003),* Feburary 2003.

[20] J. Stone, and C. Partridge, "When the CRC and TCP Checksum Disagree," *Proceedings of ACM SIGCOMM 2000*, August 2000.

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

[22] B. Tierney, "TCP Tuning Guide for Distributed Application on Wide-Area Networks," USENIX **;login:**, Vol. 26, No. 1, February 2001.

[23] S. Ubik, and P. Chimbal, "Achieving Reliable High Performance in LFNs," *Proceedings of Trans-European-Research and Education Networking Association Networking Conference (TERENA 2003),* May 2003.

[24] W. Washington and C. Parkinson, *An Introduction to Three-Dimensional Climate Modeling*, University Science Books, 1991.