Empirical Characterization of Uncongested Optical Lambda Networks and 10GbE Commodity Endpoints

Tudor Marian, Daniel A. Freedman, Ken Birman, Hakim Weatherspoon
Computer Science Department, Cornell University, Ithaca, NY 14850
{tudorm,dfreedman,ken,hweather}@cs.cornell.edu

Abstract

High-bandwidth, semi-private optical lambda networks carry growing volumes of data on behalf of large data centers, both in cloud computing environments and for scientific, financial, defense, and other enterprises. This paper undertakes a careful examination of the end-to-end characteristics of an uncongested lambda network running at high speeds over long distances, identifying scenarios associated with loss, latency variations, and degraded throughput at attached end-hosts. We use identical fast commodity source and destination platforms, hence expect the destination to receive more or less what we send. We observe otherwise: degraded performance is common and easily provoked. In particular, the receiver loses packets even when the sender employs relatively low data rates. Data rates of future optical network components are projected to outpace clock speeds of commodity end-host processors, hence more and more end-to-end applications will confront the same issue we encounter. Our work thus poses a new challenge for those hoping to achieve dependable performance in higher-end networked settings.

1 Introduction

Optical lambda networks play an increasingly central role in the infrastructure supporting globally distributed, high-performance systems and applications. Scientific, financial, defense, and other enterprise communities are deploying lambda networks for high-bandwidth, semi-private data transport over dedicated fiber optic spans between geographically dispersed data centers. Astrophysicists at Cornell University in New York receive high-volume data streams from the Arecibo Observatory in Puerto Rico or the Large Hadron Collider in Switzerland, process the data at the San Diego Supercomputer Center in California, and retrieve the results for future reference and storage at Cornell. Enterprise technology firms, such as Google and Microsoft, have begun to build proprietary networks to interconnect their data centers; this architecture balances the economics of consolidation against the benefits of end-user proximity, while increasing fault-tolerance through redundancy.

This trend will only accelerate. We are seeing a new wave of ambitious commercial networking initiatives. For example, Google recently announced a fiber-to-the-home test network [7] in the United States to deliver bidirectional bandwidth of 1 Gigabit per second (Gbps), while major Internet providers such as Verizon and Time Warner are projecting significant future improvements in consumer bandwidth. In contrast, as illustrated in Figure 1, the sending and receiving end-hosts themselves are approaching a (single-core) performance barrier. Thus, the future may bring high-speed networks connected to commodity machines powered by an ensemble of slow cores.

One consequence is that, while lambda networks typically have greater bandwidth than required, dedicate their transport for specific use, and operate with virtually no congestion [5] (in fact, the networks are routinely idle), *end-hosts* and applications increasingly find it hard to derive the full performance they might expect [24, 8]. This can be especially frustrating because, unlike the public Internet, traffic across these semi-private lambda networks encounters seemingly ideal conditions; for example, since they operate far from the congestion threshold and employ high quality optical fiber, lambda networks should not drop any packets at all, and one might reasonably believe that, if traffic is sent at some regular rate well below the actual capacity of the lambda network, it will arrive intact and more or less at the same rate. In particular, if end-host Network Interface Controllers (NICs) can reliably communicate at their maximum data rates in the lab, they should similarly do so over an uncongested and lossless lambda network.

In this paper, we show that loss occurs in precisely such situations. Our study reveals that, in most cases, the problem is not due to loss *within* the optical network span itself but instead arises from the interaction of lower-speed commodity end-hosts with such a high-bandwidth optical network: a kind of *impedance mismatch*. This mismatch is further aggravated in situations where the bottlenecks prove

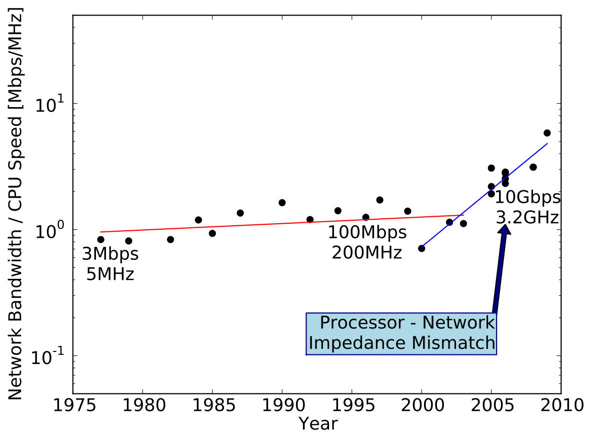


Figure 1. Network to processor speed ratio.

to be end-host memory buses, which are generally even slower than processors. And the situation may soon worsen: end-host performance increase is expected to be achieved mostly through multicore parallelism, yet it can be a real challenge to share a network interface among multiple processor cores. One issue is contention [17], and a second is that the performance-enhancing features of modern multi-queue NICs (like Receive Side Scaling) work best only for a large number of distinct, lower bandwidth, flows.

Our goal here is not to solve this problem, but rather to shed more light on it, with the hope of informing future systems architecture research. Accordingly, we have designed a careful empirical measurement of the end-to-end behavior of a state-of-the-art high-speed optical lambda network interconnecting commodity 10 Gigabit Ethernet (10GbE) end-host servers. Our community has a long history of performing systematic measurements on many prominent networks as they have emerged, including ARPANET, its successor NSFNET [16, 20], and subsequently the early Internet [31]. However, few studies have looked at semi-dedicated lambda networks, and none consider the interactions between the high-bandwidth optical core [26] of a lambda network and 10GbE commodity end-hosts [32].

This study uses a new experimental networking infrastructure testbed—the *Cornell National LambdaRail (NLR) Rings*—consisting of a set of four all-optical end-to-end 10GbE paths, of different lengths (up to 15 000 km) and number of routing elements (up to 13), with ingress and egress points at Cornell University. On this testbed, the core of the network is indeed uncongested, and loss is very rare; accounting for all loss associated with sending over 20 billion packets during a 48-hour period, we observed only one brief instance of loss in the network core, in contrast to significant packet loss observed on the end-hosts themselves.

Our key findings pertain to the relation between end-to-end behavior and fine-grained configuration of the end-host:

- The size of the socket buffer and of the Direct Memory Access (DMA) ring determines the loss rate experienced by the end-host. Similarly, the interrupt affinity policy of the network adapter, that maps interrupts to individual processor cores upon receipt of network traffic, also affects the end-host loss distribution.

- The throughput of the ubiquitous Transmission Control Protocol (TCP) decreases as packet loss increases, and this phenomenon grows in severity as a function of both the path length and the window size. The congestion control algorithm turns out to have only a marginal role in determining the achievable throughput.

- Batching of packets, through both kernel and NIC techniques, increases overall throughput, at the cost of disturbing any latency-sensitive measurements, such as packet inter-arrival times.

This paper first introduces two examples of uncongested lambda networks—the TeraGrid [6] and our own Cornell NLR Rings testbed. In Section 3, we present and discuss our experimental results. Section 4 places this study within the context of past work, and Section 5 concludes.

2 Uncongested Lambda Networks

Lambda networking, as defined by the telecommunications industry, is the technology and set of services directly surrounding the use of multiple optical wavelengths to provide independent communication channels along a strand of fiber optic cable [34]. In this section, we present two examples of lambda networks, namely TeraGrid [6] and the Cornell NLR Rings testbed. Both networks consist of semi-private, uncongested 10Gbps optical Dense Wavelength Division Multiplexing (DWDM) or OC-192 Synchronous Optical Networking (SONET) links.

2.1 TeraGrid

TeraGrid [6] is an optical network interconnecting ten major supercomputing sites throughout the United States. The backbone provides 30Gbps or 40Gbps aggregated throughput over 10GbE and SONET OC-192 links [26]. End-hosts, however, connect to the backbone via 1Gbps links, hence the link capacity between each pair of end-host sites is 1Gbps.

Of particular interest is the TeraGrid monitoring framework [8]; each of the ten sites reports measurements of throughput and loss rates of User Datagram Protocol (UDP) packets performed with Iperf [33]. Every site issues a 60-second probe to every other site once an hour, resulting in a total of 90 overall measurements collected every hour. Figure 2 shows a histogram of percentage packet loss (on a

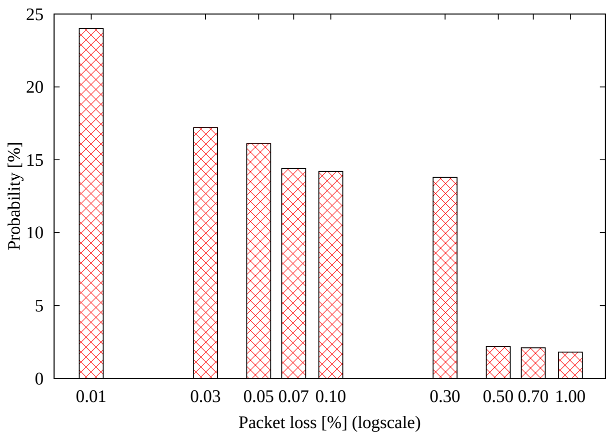


Figure 2. Observed loss on TeraGrid.

logscale) between November 1st, 2007, and January 25th, 2008, where 24% of the measured loss rates had 0.01% loss and a surprising 14% of them had 0.10% loss. Though not shown in the Figure, after eliminating a single TeraGrid site (Indiana University) that dropped incoming packets at a steady 0.44% rate, 14% of the remainder of the measurements showed 0.01% loss, while 3% showed 0.10% loss. Dialogue with TeraGrid operators revealed that the steady loss rate experienced by the Indiana University site was due to a faulty commodity network card at the end-host.

Although small, such numbers are sufficient to severely reduce the throughput of TCP on these high-latency, high-bandwidth paths [10, 27]. Conventional wisdom suggests that optical links do not drop packets. Indeed, carrier-grade optical equipment is often configured to shut down beyond bit error rates of 10^{-12} (one out of a trillion bits). However, the reliability of the lambda network is far less than the sum of its optical parts—in fact, it can be less reliable by orders of magnitude. Consequently, applications depending on protocols like TCP, which require high reliability from high-speed networks, may be subject to unexpectedly high loss rates, and hence low throughput.

Figure 2 shows the loss rate experienced during UDP traffic on end-to-end paths (care should be taken in generalizing these rates to TCP). Furthermore, it is unclear if packets were dropped along the optical path, at intermediate devices (e.g. optical or electrical switches), or at the end-hosts. Finally, loss occurred on paths where levels of optical link utilization (determined by 20-second moving averages) were consistently lower than 20%, making congestion highly unlikely, a conclusion supported by the network administrators [36].

Lacking more detailed information about the specific events that trigger loss in TeraGrid, we can only speculate about the sources of the high observed loss rates. Several hypotheses suggest themselves:

Device clutter: The critical communication path between any two end-hosts consists of many electronic devices, each of which represents a potential point of failure.

End-host loss: Conventional wisdom maintains that the majority of packets are dropped when incoming traffic overruns the receiving end-host. With the NewAPI (NAPI) [3] enabled, the Linux kernel software network stack may drop packets in either of two places: when there is insufficient capacity on the receive (rx) DMA ring, and when enqueueing packets for socket delivery would breach the socket buffer limit. In both, the receiver is overwhelmed and loss is observed, but they differ in the precise conditions that induce loss.

Cost-benefit of service: It may be the case that loss rates are typical of any large-scale networks, where the cost of immediately detecting and fixing failures is prohibitively high. For example, the measurements performed with the faulty network card at Indiana University persisted over at least a three month period.

2.2 Cornell NLR Rings

Clearly, greater control is necessary to better determine the trigger mechanisms of loss in such uncongested lambda networks. Rather than probing further into the characteristics of the TeraGrid network, we chose instead to create our own network measurement testbed centered at Cornell University and extending across the United States; this is the Cornell National LambdaRail (NLR) Rings testbed. In order to understand the properties of the Cornell NLR Rings, we first provide a fairly detailed description of our measurement infrastructure in this section.

Depicted in Figure 3, our Cornell NLR Rings testbed takes advantage of the existing National LambdaRail [4] backbone infrastructure. Two commodity servers are connected to the backbone router in Ithaca, New York, and function as *Ingress* and *Egress* end-hosts; these are four-way 2.4 GHz Xeon E7330 quad-core Dell PowerEdge R900 servers with 32GB RAM, each equipped with an Intel 10GbE LR PCIe x8 adapters (EXPX9501AFXLR). They run a preemptive 64-bit Linux 2.6.24 kernel, with the Intel `ixgbe` driver version 1.3.47. The generic segmentation offload (GSO) was disabled since it is incompatible with the Linux kernel packet forwarding subsystem.

Through a combination of IEEE 802.1Q virtual Local Area Network (VLAN) tagging and source-, policy-, and destination-based routing, we have established four static 10GbE full duplex routes that begin and end at Cornell, but transit various physical lengths: a *tiny* ring to New York City and back, a *small* ring via Chicago, Atlanta, Washington D.C., and New York City, a *medium* ring via Chicago, Denver, Houston, Atlanta, Washington D.C., and

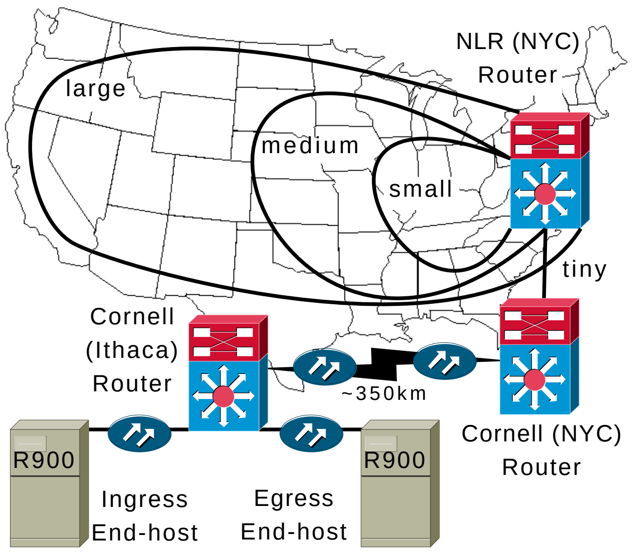


Figure 3. Topology of Cornell NLR Rings.

New York City, and a *large* ring across Chicago, Denver, Seattle, Los Angeles, Houston, Atlanta, Washington D.C., and New York City (Figure 3). The one-way latency (one trip around the ring) as reported by the `ping` utility is 8.0 ms for the tiny path, 37.3 ms for the small path, 68.9 ms for the medium path, and 97.5 ms for the large path. All optical point-to-point backbone links use 10GbE with Dense Wavelength Division Multiplexing (DWDM), except for a single OC-192 Synchronous Optical Networking (SONET) link between Chicago and Atlanta.

The NLR routers are CRS-1 devices, while the Cornell (Ithaca and NYC) backbone routers are Catalyst 6500 series hardware. These routers all have sufficient backplane capacity to operate at their full rate of 10Gbps irrespective of the traffic pattern; the loads generated in our experiments thus far have provided no evidence to the contrary. The Quality of Service (QoS) feature on these routers was disabled, hence in the event of an over-run, all traffic is equally likely to be discarded. In particular, packets are served in the order in which they are received. If the buffer is full, all subsequent packets are dropped, a discipline sometimes referred to as first-in-first-out (FIFO) queueing with drop-tail [15]. Enabling QoS requires wholesale reconfiguration of the production NLR network by NLR engineers, and is not currently feasible.

Figure 4 shows the topology of the large path and highlights the layer-3 load on the entire NLR backbone while we performed controlled 2Gbps UDP traffic experiments over this path. Importantly, the Figure legend also demonstrates that the backbone (and our path) is uncongested. While our tests were performed, the large path, exclusive of the rest of the backbone, showed a level of link utilization of roughly 20%, corresponding directly to our test traffic.

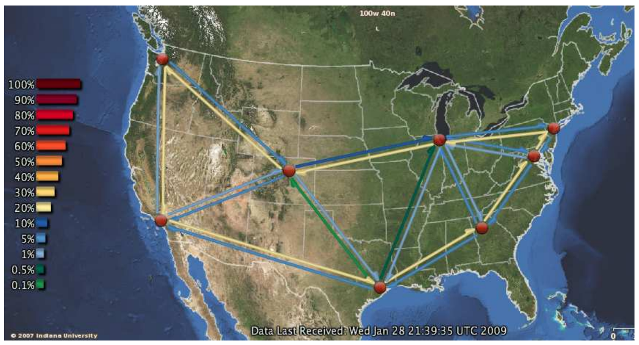

100w 40n
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
5%
1%
0.5%
0.1%
Data Last Received: Wed Jan 28 21:39:35 UTC 2009
© 2007 Indiana University

Figure 4. Test traffic on large NLR Ring, as observed by NLR Realtime Atlas monitor [5].

3 Experimental Measurements

In this section, we use the Cornell NLR Rings testbed to answer the following questions with respect to the traffic characteristics over uncongested lambda networks:

- Under what conditions does packet loss occur, where does packet loss take place, and how is packet loss affected by NIC interrupt affinity? (Section 3.2)
- What is the impact of packet loss, path length, window size, and congestion control variant on TCP throughput? (Section 3.3)
- How does packet batching affect overall throughput and latency measurements? (Section 3.4)

3.1 Experimental Setup

Our experiments generate UDP and TCP Iperf [33] traffic between the two commodity end-hosts over all paths, i.e. between the *Ingress* and *Egress* end-hosts depicted in Figure 3. We modified Iperf to report (for UDP traffic) precisely which packets were lost and which were received out of order. Before and after every experimental run, we read kernel counters on both sender and receiver that account for packets being dropped at the end-host in the DMA ring, socket buffer, or TCP window. The default size of each receive (rx) and transmit (tx) DMA ring is 1024 slots, while the MTU (Maximum Transfer Unit) is set to the default 1500 bytes (we did not use jumbo frames). Unless specified otherwise (Section 3.4), both NAPI and interrupt coalescence packet batching techniques are enabled.

Throughout our experiments, all NLR network segments were uncongested—as a matter of fact, the background traffic over each link never exceeded 5% utilization (computed by the monitoring system [5] every 1-5 seconds). All values are averaged over multiple independent runs, and the error bars denote standard error—they are always present, most of the time sufficiently small to be invisible.

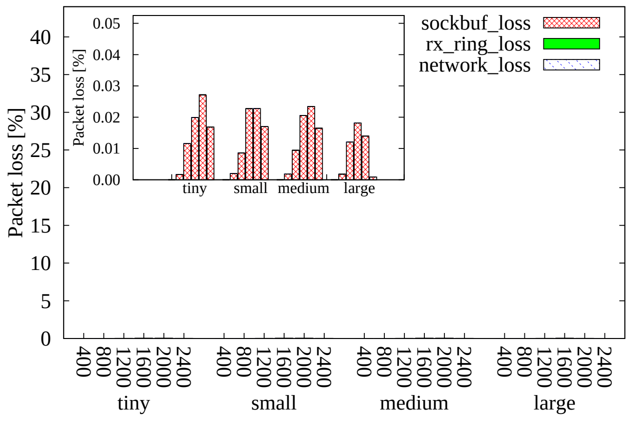
3.2 Packet Loss

To measure packet loss over the Cornell NLR Rings testbed, we performed many sequences of 60-second UDP Iperf runs over a period of 48 hours. We consecutively explored all paths (tiny, short, medium, and large) for data rates between 400Mbps to 2400Mbps, with 400Mbps intervals. We examined the following six different configurations of sender and receiver end-hosts (both identical in all cases): socket buffers sized at 1, 2, or 4MB; and use of either the `irqbalance` [1] daemon or static assignment of interrupts issued by the NICs to specific CPUs. The `irqbalance` daemon uses the kernel CPU affinity interface (through `/proc/irq/IRQ#/smp_affinity`) and periodically re-assigns hardware interrupts across processors in order to increase performance.

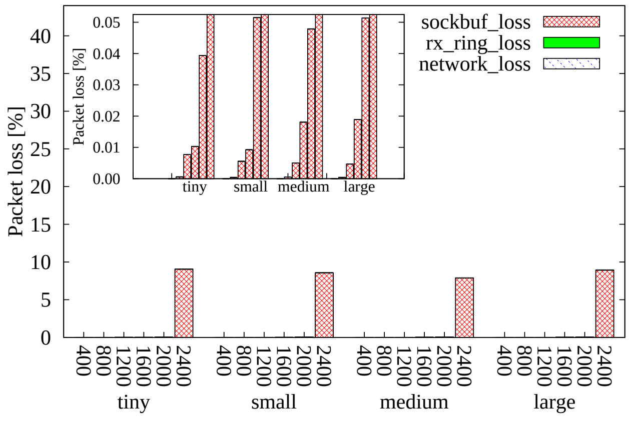
Figure 5 shows our measurements of UDP packet loss, with subfigures corresponding to different combinations of socket buffer size and bound versus balanced interrupts. Each subfigure plots packet loss observed by Iperf on the receiver end-host, as a percentage of transmitted packets, for various sender data rates across each of the Cornell NLR Ring; insets provide rescaled y-axes to better view trends. Packet loss is subdivided into three components denoting the precise location where loss can occur. In particular, loss may be a consequence of over-running the socket buffer (`sockbuf_loss`), over-running the receive (rx) DMA ring (`rx_ring_loss`), or numerous factors within the network core (`network_loss`). Since NAPI is enabled, there is no backlog queue (to over-run) between the DMA ring and the socket buffer. Moreover, we dismiss the remaining possibilities for end-host loss for the following reasons: **i)** the sender socket buffer is never over-run during the entire 48-hour duration of the experiment—in accordance with the blocking nature of the socket API; **ii)** the sender transmit (tx) DMA ring is never over-run during the entire experiment; **iii)** neither the sender nor receiver NIC report any errors (e.g. corruption) or internal (on board) buffer over-runs throughout the experiment; **iv)** the receiver does not transmit any packets (since we used Iperf with UDP traffic).

Interrupts via Irqbalance Figure 5(a) considers the scenario with the `irqbalance` daemon running and the socket buffer size set to 1MB. We observe *zero* loss in the network core; all loss occurs within the receiver's socket buffer. At rates beyond 2000 Mbps, irqbalance spreads the interrupts to many CPUs and the loss decreases. (Of note, omitted for space constraints, irqbalance with 2 and 4MB buffers result in zero loss for all tested data rates.)

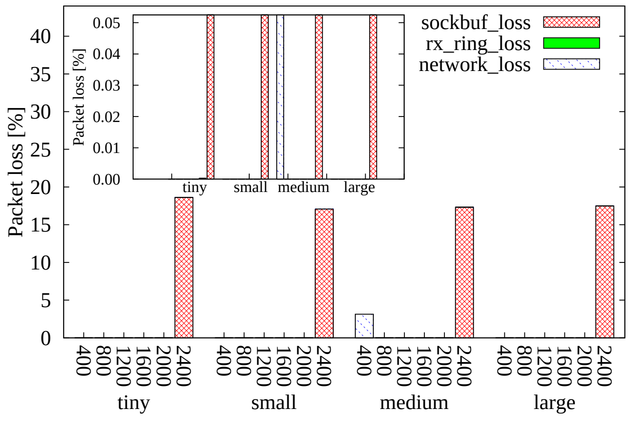
Interrupts Bound to a Single CPU Figures 5(b) and 5(c) consider the more interesting scenario when we assign all interrupts from the NIC to a single core, with 1 and 4MB socket buffers, respectively. (The results for 2MB buffer, not shown, are identical to those of 4MB, but with



(a) 1MB buffers, balanced interrupts



(b) 1MB buffers, bound interrupts



(c) 4MB buffers, bound interrupts

Figure 5. UDP loss as a function of data rate across Cornell NLR Rings: subfigures show various socket buffer sizes and interrupt options for balancing across or binding to cores; insets rescale y-axis, with x-axis unchanged, to emphasize fine features of loss.

$\sim 12\%$ packet loss for a sender data rate of 2400Mbps.)

There are three key points of interest. First, at 2400Mbps there is an abrupt increase in observed loss. Taking a closer look, we noticed that the receiver was experiencing *receive livelock* [25]. On a Linux 2.6 kernel, receive livelock can easily be observed as the network *bottom half* cannot finish in a timely manner, and it is forced to start the the corresponding `ksoftirqd/CPU#` kernel thread. The thread runs exclusively on the same CPU, and picks up the remaining work the softirq did not finish, acting as a rate limiter. As a result, the receive livelock occurs given that all interrupts (rx, tx, rxnobuff, etc.) were serviced by a single overwhelmed CPU—the same CPU that runs the corresponding `ksoftirqd/CPU#` and the user-mode Iperf task. The Iperf task is placed on the same CPU since the scheduler's default behavior is to minimize cache thrashing. Consequently, there is not enough CPU time remaining to consume the packets pending in the socket buffer in a timely fashion. Hence, the bigger the socket buffer, the more significant the loss, precisely as Figures 5(b) and (c) show.

Second, end-host packet loss increases with sender data rate, as visible in the Figure insets. Figure 5(b) corresponds to a relatively small buffer, 1MB, so the effect is clear. Figure 5(c) corresponds to a larger buffer (4MB) for which, with the exception of data rates of 2400Mbps, there is a single negligible packet loss event along the tiny path at a data rate of 2000Mbps (almost unobservable on scale of Figure). Similarly, this trend is evident in Figure 5(a) (irqbalance on); however, at higher data rates, irqbalance spreads the interrupts to many different CPUs and the loss decreases.

Third, Figure 5(c) shows a particular event—the only loss in the core network we experienced during the entire 48-hour period, occurring on the medium path (one way latency is 68.9 ms) for a sender data rate of 400Mbps. During the course of the experiments, this was a single outlier that occurred during a single 60-second run. We believe it could have been caused by events such as NLR maintenance—we have experienced path blackouts due to various path segments being serviced, replaced, or upgraded.

To summarize, the experiments show virtually no loss in the network core. Instead, loss occurs at the end-hosts, notably at the receiver. End-host loss is typically the result of a buffer over-run in the socket, backlog queue, or DMA ring. Unless the receiver is overloaded, a sufficiently large socket buffer prevents loss. NIC interrupt affinity to CPUs affects loss, and is pivotal in determining the end-host's ability to handle load graciously. Our experiments show that, at higher data rates, irqbalance works well (it decreases loss), whereas, at lower data rates, binding NIC interrupts to the same CPU reduces loss more than irqbalance. One benefit of binding all NIC interrupts to the same CPU stems from the fact that the driver (code and data), the kernel network stack, and the user-mode application incur less CPU cache pollution overhead.

3.3 Throughput

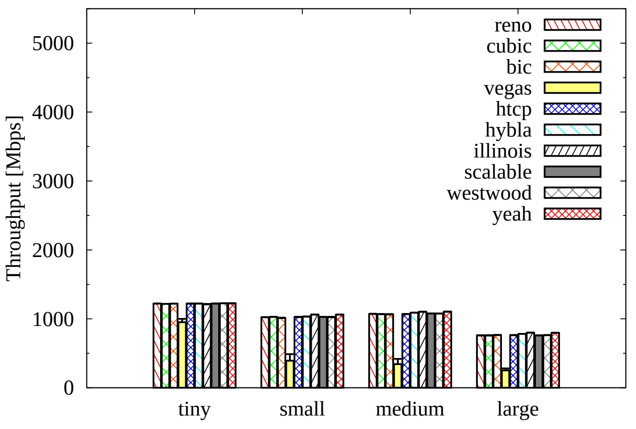
Although UDP is well suited for measuring packet loss rates and indicating where loss occurs, TCP [21] is the de-facto reliable communication protocol; it is embedded in virtually every operating system's network stack. Many TCP congestion control algorithms have been proposed—Fast TCP, High Speed TCP, H-TCP, BIC, CUBIC, Hybla, TCP-Illinois, Westwood, Compound TCP, Scalable TCP, YeAH-TCP—and almost all have features intended to improve performance over high-bandwidth, high-latency links. The existence of so many variants indicate there is as yet no clearly superior algorithm.

To measure the achievable throughput, we used 60-second Iperf bursts to conduct a set of 24-hour bulk TCP transfer tests over all the Cornell NLR Rings; we examined all TCP variants available in the Linux kernel (except for TCP-LP and TCP Veno).

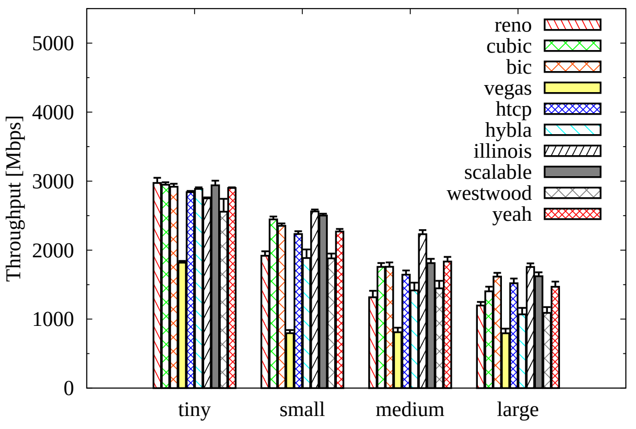
Figure 6(a) shows TCP throughput results for a single flow with window sizes configured, with respect to each path round-trip time (RTT), to allow for a 1Gbps data rate. A higher window translates into larger amount of in-flight (not yet acknowledged) data, which is necessary but not sufficient to yield high throughput on such high-latency, high-bandwidth links. In particular, a single TCP flow of 1Gbps requires a window of at least 2MB on the tiny path, 9.4MB on the short, 17.3MB on the medium, and 24.4MB on the large. Almost all TCP variants yield roughly the same throughput, with the exception of TCP Vegas that underperforms. No packet loss occurs for any of the single-flow TCP variants, yet throughput decreases for longer paths, even though the end-hosts have sufficient window size.

Since TCP window size is a kernel configuration parameter that requires superuser privileges for adjustment, typical user-mode applications like GridFTP [9] strive to maximize throughput by issuing multiple TCP flows in parallel to fetch / send data. To experiment with multiple flows, we issued four TCP Iperf flows in parallel in order to saturate each end-host's capacity and yield maximum throughput. Figure 6(b) depicts the throughput results. Although the window sizes should be sufficient, the overall throughput decreases as the path length increases. Importantly, loss at the end-host does occur for multiple TCP flows. Moreover, some TCP variants yield marginally better aggregate throughput than others when competing with flows of the same type. Note that the TCP throughput over the *tiny* path is identical to the maximum throughput achieved during control experiments (performed by directly linking end-hosts with an optical patch cable).

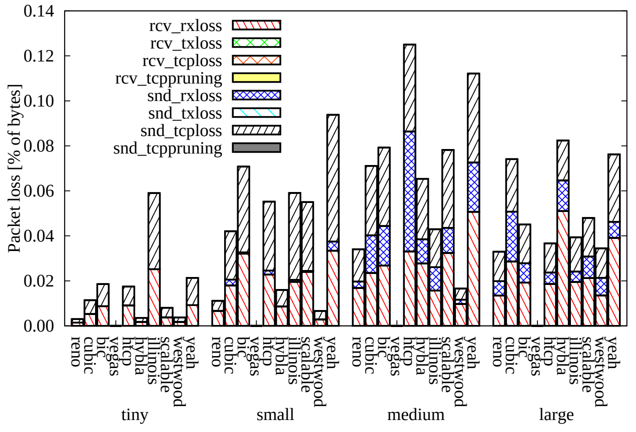
Even though TCP is a reliable transport protocol, packet loss, albeit at the end-host, does affect performance [27].



(a) TCP throughput for single flow



(b) TCP throughput for four concurrent flows



(c) Loss associated with four concurrent flows

Figure 6. TCP throughput and loss across Cornell NLR Rings: (a) throughput for single flow, (b) throughput for four concurrent flows, (c) loss associated with those four concurrent flows; TCP congestion control windows configured for each path round-trip time to allow 1Gbps of data rate per flow.

Figure 6(c) shows the percentage of packet loss corresponding to the TCP throughput in Figure 6(b). Unlike UDP loss, any analysis of TCP loss must account for retransmissions, selective and cumulative acknowledgments, different size of acknowledgments, and timeouts. Figure 6(c) shows percentage of loss *in bytes*, unlike UDP, for which packet count suffices since all UDP packets have identical size. Loss is reported both at the sender (denoted by `snd_`) and receiver (`rcv_`), within the DMA rings (`_txloss` and `_rxloss`), inferred by TCP itself (with `_tcploss`), and due to the inability of the user-mode process owning the socket to read the data in a timely fashion (`_tcppruning`).

Loss occurs solely in one of the following locations: the receiver's receive (rx) DMA ring (`rcv_rxloss`), loss that is then largely inferred by the sender's TCP stack (`snd_tcploss`), and finally, within the sender's receive (rx) DMA ring (`snd_rxloss`). The sender sends MTU-size (1500-byte) TCP data packets and receives TCP empty (52-byte) payload acknowledgments (ACKs), as 20-byte IP header + 20-byte TCP header + 12-byte TCP options.

There are two key observations. First, loss occurs at the end-host in the rx DMA rings—the receiver will drop inbound payload packets, while the sender will drop inbound ACK packets. Recall that the NIC is configured to a default value of 1024 slots per DMA ring. The socket buffer is essentially the TCP window; hence, it is adjusted to a large value in this experiment. Second, there are far more ACK packets (`snd_rxloss`) being lost than payload packets (`rcv_rxloss`). However, since ACKs are cumulative, TCP can *afford* to lose a significant portion of a window worth of ACKs on the rx DMA ring, provided that a single ACK with an appropriate (subsequent) sequence number is delivered to the TCP stack. Note that there is no loss observed by TCP Vegas since its low throughput is insufficient to induce end-host loss, a scenario identical to the one already described in Figure 6(a).

Our experiments show that as path length increases, more data and, importantly, more ACKs are lost since the TCP windows are enlarged to match the bandwidth delay product of the longer paths. This affects performance, and throughput decreases as the path length increases.

3.4 Packet Batching

In this section, we look closely at the impact of packet batching techniques on the measurements reported above.

A CPU is notified of the arrival and departure of packets at a NIC by interrupts. The typical interrupt-driven commodity kernel, however, can find itself driven into a state in which the CPU expends all available cycles processing interrupts, instead of consuming received data. If the interrupt processing overhead is larger than the rate at which packets arrive, *receive livelock* [25] will occur (the interrupt over-

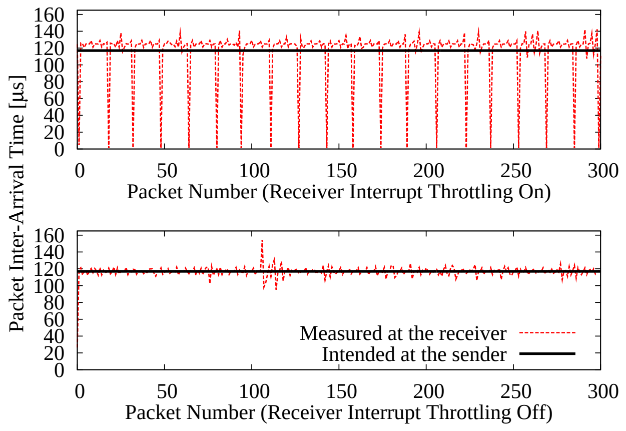


Figure 7. Packet inter-arrival time as a function of packet number; NAPI disabled.

head consists of two context switches plus executing the top half of the interrupt service routine). The typical solution is to batch packets by parameterizing the NIC to generate a single interrupt for a group of packets that arrive during some specified time interval. For example, Intel NICs offer an Interrupt Throttling configuration parameter that limits the maximum number of interrupts issued per second. If the device supports it, the kernel can take it one step further by functioning in NAPI [3] mode. Instead of being exclusively interrupt-driven, a NAPI kernel is interrupt-driven at low data rates, but switches to polling at high data rates.

Packet batching techniques provide the benefit of an increase in maximum achievable bandwidth for a particular commodity architecture. For example, with NAPI and Interrupt Throttling disabled, the maximum achievable TCP throughput on our setup is approximately 1.9Gbps, in control experiments with end-hosts directly connected to each other. With NAPI enabled and Interrupt Throttling set to default parameter values, we achieved around 3Gbps throughput, as shown in Figure 6(b). By default, the NICs in our experiments implement Interrupt Throttling by limiting interrupts to a rate of 8000 per second.

However, this does not mean that packet batching is ideal in all scenarios, even though vanilla kernels and drivers enable batching by default. To illustrate this, consider a standard metric provided by high-end Ethernet test products [2]—the packet inter-arrival time, also known as packet dispersion. To perform this type of measurements, we patched the Linux kernel to time-stamp every received packet as early as possible (in the driver's interrupt service routine) with the CPU time stamp counter (TSC) that counts clock cycles instead of the monotonic wall-clock, thereby achieving cycle (i.e. nanosecond) resolution. Our implementation overwrites the `SO_TIMESTAMPNS` socket option to return the 64-bit value of the TSC register. For the TSC time-stamp values to be monotonic (a validity requirement), they must originate from the same CPU. This means that all NIC interrupts notifying a packet arrival must be handled by the same CPU, since received packets are time-stamped in the interrupt service routine.

Figure 7 shows the packet inter-arrival time for a UDP Iperf experiment consisting of a sequence of 300 packets at a data rate of 100Mbps (about one packet every 120 μs) with and without Interrupt Throttling enabled and NAPI disabled. We see that the interrupt batching superimposes an artificial structure on top of the inter-arrival times, thereby yielding spurious measurement results. This phenomenon may have significant consequences. For example, tools that rely on accurate packet inter-arrival measurements to estimate capacity or available bandwidth yield meaningless results when employed in conjunction with packet batching.

3.5 Summary of Results

Our experiments answer two general questions with respect to uncongested lambda network traffic. First, we show that loss occurs almost exclusively at the end-hosts as opposed to within the network core, typically a result of the receiver being over-run. Second, we show that measurements are extremely sensitive to the configuration of the commodity end-hosts. In particular, we show that:

- UDP loss is dependent upon both the size of socket buffers and DMA rings as well as the specifics of interrupt affinity in the end-host network adapters.
- TCP throughput decreases with an increase in packet (data and acknowledgment) loss, with an increase in path length, and an increase in window size. The congestion control algorithm is only marginally important in determining the achievable throughput, as most TCP variants are similar.
- Built-in kernel NAPI and NIC Interrupt Throttling improve throughput, although they are detrimental for latency sensitive measurements. This reinforces the conventional wisdom that there is no "one-size-fits-all" set of parameters, and careful parameter selection is necessary for the task at hand.

Although this paper limits itself to measurement, we should note that in a previous paper [10], we proposed a practical way to overcome poor end-to-end performance. In that work, we showed that using a perimeter middlebox (or a performance enhancement proxy) can significantly improve end-to-end throughput in the face of packet loss. We achieved this through a combination of Forward Error Correction (FEC) at line speed and TCP segment caching which transparently stores and re-transmits dropped TCP segments without requiring a sender retransmission to travel across the entire network to reach the destination. In fact, we greatly increased both the performance and reliability of wide-area storage using such a technique [35].

4 Related Work

There has been a tremendous amount of work aimed at characterizing the Internet at large by analytical modeling, simulation, and empirical measurements. Measurements, in particular, have covered a broad range of metrics, from end-to-end packet delay and packet loss behavior [11, 14], to packet dispersion (spacing) experienced by back-to-back packets [13], packet inter-arrival time [20], per-hop and end-to-end capacity, end-to-end available bandwidth, bulk transfer capacity, achievable TCP throughput, and other general traffic characteristics [16]. However, there has been little work aimed at characterizing uncongested semi-private or dedicated networks [32], like modern optical lambda networks.

The need for instruments with which to perform such measurements has led to the development of a myriad of tools [13, 23, 18, 19, 22, 30]. These tools are typically deployed in an end-to-end fashion for convenience and often embody a tradeoff between intrusiveness and accuracy [29]. For example, some tools rely on self-induced congestion, while others rely on relatively small probes consisting of packet pairs or packet trains. Tools like these have become essential and provide a solid foundation for measurements; for example, we have saved significant time by working with (and extending) the existing Iperf [33].

Internet measurements provide a snapshot of the characteristics of the network at the time the measurements are performed. For example, in its early days, the Internet was prone to erratic packet loss, duplication, reordering, and the round-trip time delays were observed to vary over a wide range of values [31]. Today, none of these issues remain, although other challenges have emerged.

Historically, networks have been characterized as they became available—ARPANET, its successor, NSFNET [16, 20], and the early Internet [31] have all been the focus of systematic measurements. Murray et al. [26] compared end-to-end bandwidth measurement tools on the 10GbE TeraGrid backbone, while Bullot et al. [12] evaluated the throughput of various TCP variants by means of the standard Iperf, over high-speed, long-distance production networks of the time (from Stanford to Caltech, to University of Florida, and to University of Manchester over OC-12 links of maximum throughput of 622Mbps)—similar to the experiments in Section 3.3.

However, unlike our experiments, Bullot et al. [12] focused on throughput and related metrics, like the stability (in terms of throughput oscillations), and TCP behavior while competing against a sinusoidal UDP stream. Although disregarding loss patterns and end-host behavior, the authors did provide insight into how the `txqueuelen` parameter (i.e. the length of the backlog queue between the IP layer and the DMA rx ring—made obsolete by NAPI) affects throughput stability. In particular, larger values of the `txqueuelen` are correlated with more instability. An equally interesting observation was that reverse-cross-traffic affects some TCP variants more than others, since they alter ACK delivery patterns (e.g. ACK compression due to queueing or loss). It is also worth noting that the authors performed a set of tentative TCP performance measurements on 10Gbps links, using jumbo (9000-byte) frames.

By contrast, relatively few works have investigated the effect of traffic patterns on end-hosts and the their ability to handle such traffic, especially when connected to uncongested lambda networks. Mogul et al. [25] investigated the effect of high data rate traffic on the end-host, noting that a machine would live-lock and spend all available cycles while handling the interrupt service routine as a result of packets being received, only to drop these packets at the subsequent layer, and hence fail to make forward progress. Consequently, NAPI [3] and on-board NIC Interrupt Throttling have been widely adopted, to the point where they are enabled by default in vanilla kernels. On the other hand, an interesting study looked at how “interrupt coalescence” (produced by NAPI possibly in conjunction with Interrupt Throttling) hinders active measurement tools that rely on accurately estimating packet dispersion to measure capacity and available bandwidth [28]. Since the packets were time-stamped in user-space, context switches at the receiver cause similar behavior as packet batching.

5 Conclusion

Optical lambda networks provide high-bandwidth, semi-private transit interconnecting data centers throughout the world and transporting massive quantities of data. They serve critical roles in the infrastructure both of cloud-computing architectures and of traditional scientific, financial, defense, and other enterprise users. In this work, we use our Cornell National LambdaRail Ring testbed to methodically probe the end-to-end behavior of such 10GbE networks, connected to powerful commodity end-hosts to send and receive traffic.

Surprisingly, we observed significant penalties in end-host performance and end-to-end dependability in this scenario, consistently measuring packet loss at the receiving end-host even when traffic was sent at relatively low data rates. Moreover, such effects were readily instigated by subtle (and often default) configuration issues of these end-hosts—socket buffer size, TCP window size, NIC interrupt affinity, and status of various packet batching techniques, with no single configuration alleviating observed problems for all scenarios.

As optical networking data rates continue to outpace clock speeds of commodity end-hosts, more end-to-end applications will invariably face similar issues. This empirical study confronts the difficulty of reliably and consistently maximizing the performance of such networks.

Acknowledgments

We would like to recognize the anonymous reviewers of this manuscript as well as the engineering staff that helped establish and monitor the Cornell NLR Rings testbed: Eric Cronise, Dan Eckstrom, and Ed Kiefer (Cornell Information Technologies); Greg Boles, Brent Sweeny, and Joe Lappa (National LambdaRail); Scott Yoest and Larry Parmelee (Cornell Computer Science Technical Staff); and staff of Cornell and National LambdaRail Network Operations Centers. We also acknowledge Intel and Cisco equipment support and NSF TRUST and AFRL funding.

References

[1] Irqbalance. `http://www.irqbalance.org/`.

[2] Ixia. `http://www.ixiacom.com/`.

[3] NAPI. `http://www.linuxfoundation.org/`.

[4] National LambdaRail. `http://www.nlr.net/`.

[5] NLR PacketNet Atlas. `http://atlas.grnoc.iu.edu/atlas.cgi?map_name=NLR%20Layer3`.

[6] Teragrid. `http://teragrid.org/`.

[7] Think big with a gig: Our experimental fiber network. `http://googleblog.blogspot.com/2010/02/think-big-with-gig-our-experimental.html`.

[8] TeraGrid Performance Monitoring. `https://network.teragrid.org/tgperf/`, 2005.

[9] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *GGF Document Series GFD*, 20, 2003.

[10] M. Balakrishnan, T. Marian, K. Birman, H. Weatherspoon, and E. Vollset. Maelstrom: Transparent Error Correction for Lambda Networks. In *Proceedings of NSDI*, 2008.

[11] J.-C. Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of SIGCOMM '93*.

[12] H. Bullot, R. L. Cottrell, and R. Hughes-Jones. Evaluation of advanced TCP stacks on fast long-distance production networks. In *Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[13] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Perform. Eval.*, 27-28:297–318, 1996.

[14] I. Cidon, A. Khamisy, and M. Sidi. Analysis of Packet Loss Processes in High-Speed Networks. *IEEE Transactions on Information Theory*, 39:98–108, 1991.

[15] Cisco Systems. Buffers, Queues, and Thresholds on the Catalyst 6500 Ethernet Modules, 2007.

[16] K. Claffy, G. C. Polyzos, and H. Braun. Traffic Characteristics of the T1 NSFNET Backbone. In *INFOCOM '93*.

[17] M. Dobrescu, N. Egi, K. Argyraki, B.-g. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of SOSP*, 2009.

[18] C. Dovrolis, P. Ramanathan, and D. Moore. What Do Packet Dispersion Techniques Measure? In *INFOCOM '01*.

[19] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[20] S. A. Heimlich. Traffic characterization of the NSFNET national backbone. *SIGMETRICS Perform. Eval. Rev.*, 18(1):257–258, 1990.

[21] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 25(1):157–187, 1995.

[22] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Tr. Net.*, 11(4):537–549, 2003.

[23] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. CapProbe: a simple and accurate capacity estimation technique. *SIGCOMM Comp. Comm. Rev.*, 34(4):67–78, 2004.

[24] D. A. Lifka. Director, Center for Advanced Computing, Cornell University. *Private Communication*, 2008.

[25] J. C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Syst.*, 15(3):217–252, 1997.

[26] M. Murray, S. Smallen, O. Khalili, and M. Swany. Comparison of End-to-End Bandwidth Measurement Tools on the 10GigE TeraGrid Backbone. In *Proceedings of GRID '05*.

[27] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. *SIGCOMM Comp. Comm. Rev.*, 28(4):303–314, 1998.

[28] R. Prasad, M. Jain, and C. Dovrolis. Effects of Interrupt Coalescence on Network Measurements. In *PAM'04*.

[29] R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. *IEEE Network*, 17:27–35, 2003.

[30] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM'03 Workshop*.

[31] D. Sanghi, A. K. Agrawala, O. Gudmundsson, and B. N. Jain. Experimental Assessment of End-to-end Behavior on Internet. In *Proc. IEEE INFOCOM '93*.

[32] S. C. Simms, G. G. Pike, and D. Balog. Wide Area Filesystem Performance using Lustre on the TeraGrid. In *Teragrid Conference*, 2007.

[33] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf – The TCP/UDP bandwidth measurement tool. 2004.

[34] S. Wallace. Lambda Networking. Advanced Network Management Lab, Indiana University.

[35] H. Weatherspoon, L. Ganesh, T. Marian, M. Balakrishnan, and K. Birman. Smoke and mirrors: Shadowing files at a geographically remote location without loss of performance. In *Proceedings of FAST*, Feb. 2009.

[36] P. Wefel. Network Engineer, National Center For Supercomputing Applications (NCSA), University of Illinois. *Private Communication*, 2007.