Making a Case for Proactive Flow Control in Optical Circuit-Switched Networks

M. Kumar[1], V. Chaube[1], P. Balaji[2], W. Feng[1], and H.-W. Jin[3]

[1] Dept. of Computer Science
Virginia Tech
{mithil,vineetac,feng}@cs.vt.edu

[2] Mathematics and Computer Science Division
Argonne National Laboratory
balaji@mcs.anl.gov

[3] Dept. of Computer Sc. and Engg.
Konkuk University
jinh@konkuk.ac.kr

Abstract. Optical circuit-switched networks such as National LambdaRail (NLR) offer dedicated bandwidth to support large-scale bulk data transfer. Though a dedicated circuit-switched network eliminates congestion from the network itself, it effectively "pushes" the congestion to the end hosts, resulting in lower-than-expected throughput. Previous approaches either use an ad-hoc proactive approach that does not generalize well or a sluggish reactive approach where the sending rate is only adapted based on synchronous feedback from the receiver.

We address the shortcomings of such approaches using a two-step process. First, we improve the adaptivity of the reactive approach by proposing an *asynchronous*, fine-grained, rate-based approach. While this approach enhances performance, its limitation is that it is still reactive. Consequently, we then analyze the predictive patterns of load on the receiver and provide strong evidence that a proactive approach is not only possible, but also necessary, to achieve the best performance in dynamically varying end-host conditions.

Key words: rate-based protocol, circuit switched, optical networks, LambdaGrid

1 Introduction

Rapid advances in optical networking are producing increases in bandwidth that have outpaced the geometric increase in semiconductor chip capacity, as predicted by the Moore's law. This means that the burden is now on the end points of communication networks to process the high bandwidth data being delivered by the network. This trend is more prominent in circuit-switched optical networks, like those found in LambdaGrids [12, 4].

A LambdaGrid is a distributed grid of resources that consists of dedicated high-bandwidth optical networks, computing clusters, and data repositories. Such a distributed supercomputer will enable scientists and engineers to analyze, correlate, and visualize extremely large and remote datasets on-demand and in real time. The dedicated high-bandwidth optical networks found in LambdaGrids translate into no internal network congestion and result in pushing congestion to the end hosts. Thus, the efficiency of a network protocol at an end host is greatly influenced by its ability to adapt its transmission rate to dynamic network and endpoint conditions, thereby minimizing packet loss and maximizing network throughput.

Currently, the computational power of an end host is not sufficient to handle the high network bandwidth that is available in a dedicated circuit-switched network. This puts a cap on the maximum throughput that can be achieved. Additionally, the computational power per process is effectively reduced if there are several processes running on the end host that are competing for CPU resources. Hence, we need to understand end-host contention and come up with effective rate-adaptation techniques, which are critical when networks are fast enough to push congestion to the end hosts.

In this paper, we address the shortcomings of existing approaches in two steps. First, we present an asynchronous, fine-grained, rate-control approach that solves some performance issues but is still *reactive* in nature. Next, we present our observations from a series of experiments and analyze the predictive patterns of load on the receiver node. Based on our analysis, we provide evidence that a proactive approach is possible and required in such environments to achieve the best performance in dynamically varying end-host conditions.

2 Background

A rate-based approach is one where a constant sending rate is negotiated between a sender and receiver. Rate-based protocols perform well for high bandwidth-delay product networks. Reliable Blast UDP (RBUDP) [7] is an example of such a protocol in which the sender transmits data using UDP packets at a rate specified by the user. At the end of data transmission, the receiver sends an acknowledgment via a bitmap of missing packets. The sender then re-transmits the missing packets. The process continues until all packets are received. The mechanism is aggressive and provides reliability, but it is not adaptive to packet loss.

RAPID [1] is an end-host aware, rate-adaptive protocol, where rate adaptation is based on proactive feedback from the receiver. The receiver is monitored by a soft real-time process that attempts to guess the time and duration when the receive process is context-switched and replaced by another process. It then informs the sender, which suspends transmission to avoid packet loss when the receiver process is context-switched and suspended. Although being a proactive approach, a major drawback of this protocol is that it is difficult to predict the exact time when the receive process will be suspended. It is even more difficult to match the sender's transmission suspension with the receive process' suspend interval, especially because the two nodes are physically separated by a reasonable amount of network delay [2]. Moreover, stopping data transmission completely is a drastic step to take when we aim at keeping the network utilization high.

Another RBUDP variant, RBUDP+ [3], uses the same scheme but a different estimate of the time and duration for which the receive process has been rescheduled. Like RAPID, the prediction of time and duration is almost impossible.

An enhancement of RAPID and RBUDP+ is RAPID+ [2], which focuses on dynamically monitoring the packet loss at the receiving end host, so that it can be used to adapt the sending rate. Accurate prediction of when packet loss would occur increases performance and circuit utilization. However, in this case, rate adaptation is initiated only after most of the losses have already occurred.

The Simple Available Bandwidth Utilization Library (SABUL) [6] is a rate-based as well as window-based protocol that has been designed for data-intensive applications over a shared network. Its delay and window-based congestion control makes it TCP-friendly but brings along the same characteristics of TCP that make it inefficient when congestion has been pushed to the endpoints.

The Group Transport Protocol (GTP) [15] is a receiver-driven protocol that performs well in multipoint-to-point and multipoint-to-multipoint environments and ensures fairness between different connections on the same end host. Like other rate-based protocols, SABUL and GTP wait for packet loss to occur before providing feedback to the sender to transmit with revised sending rates.

TCP does not perform well for large bandwidth-delay product networks, as found in LambdaGrids, because of the overhead involved in congestion and flow control. In order to improve TCP performance, significant research has been done to improve TCP congestion control [16, 13, 10, 9]. Other complementary research has been done to improve flow control [11, 14, 5]. However, none of these improvements or variants of TCP were designed for networks with nearly zero congestion. In this paper, we use UDP because it is lighter and faster and can be enhanced to provide reliability without worrying about network congestion.

3 Asynchronous Fine-Grained Rate Control

In this section, we introduce our fine-grained, rate-based control protocol called ASYNCH and compare its performance with existing rate-based protocols.

3.1 Basic Idea

Many existing rate-based protocols such as RAPID+, SABUL and GTP use *reactive* rate control to avoid end-host congestion, where the sending rate is varied *after* an event, such as a packet drop, occurs. Further, these approaches use a coarse-grained feedback mechanism. Specifically, they utilize multi-round communication; in the first round, the sender attempts to send data at the maximum rate. If there are dropped packets in the second round, these dropped packets are sent at a slower pace. These rounds continue till all the data has been successfully communicated.

While such an approach is simple, it has several performance implications. First, a coarse-grained feedback approach, such as that described above, has a high overhead of inaccuracy. For example, if a receiver is only capable of receiving data at 5 Gbps, a sender transmitting a 10-GB file at 10 Gbps would end up dropping half the packets (5 GB). The sender, however, would receive feedback about its high sending rate only at the end of the first communication round, i.e., after the entire file has been sent out. In the second round, the remaining 5 GB has to be retransmitted. Thus, the inaccurate sending rate (in this case, 10 Gbps) can result in a major loss of performance, which is expected to further worsen as the amount of data being communicated increases.

Second, a reactive approach is fundamentally limited in its ability to handle dynamic environments where the receiving end-host is executing other processes. By the time the sender receives feedback for rate adaptation, the receiver's capability to receive data might have already changed.

In order to analyze the behavior of the receiver end-host and its dynamics, we implemented an asynchronous, fine-grained, reactive rate-control protocol named ASYNCH. This protocol, while still reactive, addresses the issue of coarse-grained feedback; it asynchronously sends feedback to the sender at regular intervals of time, instead of waiting for the entire file to be transferred before sending the feedback. This mechanism allows feedback to be independent of the file size. On the other hand in synchronous feedback, such as in RAPID+, the receiver sends feedback only at the end of each communication round.

In ASYNCH, upon receiving feedback, the sender calculates the current loss rate. If an increase in loss rate (compared to the last calculated value) is detected, the sending rate is decreased. The sender considers at least two feedback messages from the receiver for deciding what kind of rate adaptation is required. If a high loss rate has been observed in only one round, it is treated as a temporary loss, and no action is taken. If the receiver does not experience any packet loss for k successive rounds, then the sender will increase its sending rate. This reactive rate adaptation is expected to reduce any further loss at the receiver.

3.2 Performance Evaluation

We evaluate the performance of ASYNCH and RAPID+ for various conditions on the receiver end-host and analyze the results. We first show how throughput degrades when the sending rate increases beyond the receiver's capacity to receive. We then explain the role played by the *round-trip time* (RTT) latency and the receiver's socket buffer size on packet loss rate and network throughput.

Methodology The testbed is a three-node sender-receiver setup with the middle node acting as a wide-area network (WAN) emulator using Netem [8]. A file size of 1GB, RTT of 56ms and MTU of 9000 bytes was used for all experiments, unless otherwise specified. The choice for this setup and configuration is based on our attempt to emulate a real dedicated circuit-switched, long-fat network. In order to obtain consistent results, we bind the receive process to the same core for our experiments. The system configuration is summarized in Table 1.

Table 1. System Configuration

Processor	Dual-core AMD Opteron 2218
Cache Size	1024 KB
RAM	4 GB
Network Adaptor	Myrinet 10Gb
Kernel	2.6.18

High Network Speed One of the primary reasons for packet drops is the discrepancy in the protocol processing requirements for data sending and receiving. Specifically, data transmission with TCP/IP or UDP/IP is typically a lighter weight operation compared to data reception, owing to various optimizations such as integrated checksum and copy and the lack of data demultiplexing requirements that are necessary on the receiver side. In our experiments with a 10 Gbps network, the sender, for example, is able to transmit packets at 7 Gbps. However, the receiver is only able to receive data at 5.5 Gbps.

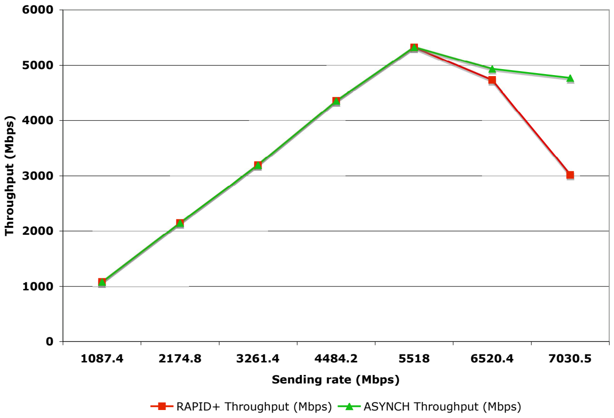

Fig. 1. Network throughput degrades after the sending rate reaches an optimal point

Figure 1 demonstrates this behavior. For sending rates less than 5.5 Gbps, the achieved throughput is about the same as the sending rate, and there is no packet loss. Beyond 5.5 Gbps, however, there is a sharp rise is the packet loss rate, resulting in a decline in throughput. Unlike RAPID+, ASYNCH utilizes a fine-grained feedback mechanism to adapt its rate quickly resulting in up to

58% better throughput as compared to RAPID+. For a sending rate of 7 Gbps, the loss rate for ASYNCH is only 9.5% compared to the 21.49% for RAPID+.

For the rest of the experiments, we are interested in studying the capabilities of the receiver end-host; therefore we use a peak sending rate of 5.5 Gbps.

Socket Buffer Size of the Receiver UDP/IP communication takes place through socket buffers. Data received is stored in the socket buffer until the application reads it. Thus, while a large socket buffer can provide more tolerance to a receiver's slow receiving capability, it can result in memory wastage. Accordingly, an optimal buffer size needs to be chosen to balance memory wastage and packet loss.

Figure 2 shows the effect of the socket buffer size on loss rate. For very small buffer sizes (10-100 KB), substantial packet loss occurs, resulting in poor throughput. For buffer sizes larger than 100 KB, ASYNCH's loss rate drops to about 5-7% or less. While the loss rate for RAPID+ decreases with the socket buffer size as well, we notice that ASYNCH's loss rate is much smaller than that of RAPID+ for all buffer sizes.

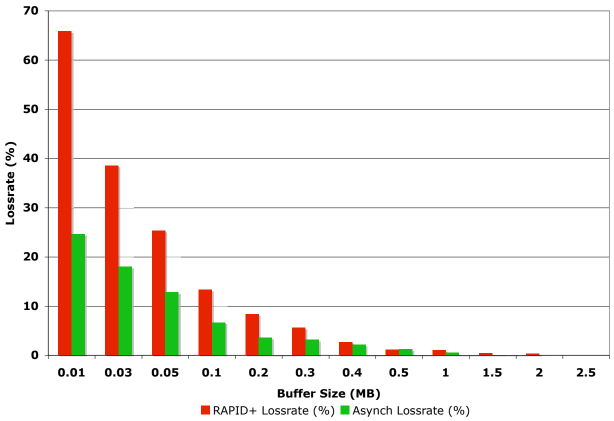

Fig. 2. Effect of socket buffer size on loss rate

Round-Trip Time Unlike TCP, in the case of UDP data transfers, RTT does not play a significant role in UDP throughput due to the lack of congestion- and flow- control mechanisms in UDP. However, to achieve reliability, both RAPID+ and ASYNCH use a TCP control channel; the sender waits for packet-loss feedback from the receiver. This feedback mechanism, however, directly depends on the RTT and causes the throughput to drop with increasing RTT. Figure 3 illustrates this relationship. That is, for both RAPID+ and ASYNCH, the throughput drops by about 65 Mbps for every 20-ms increase in RTT.

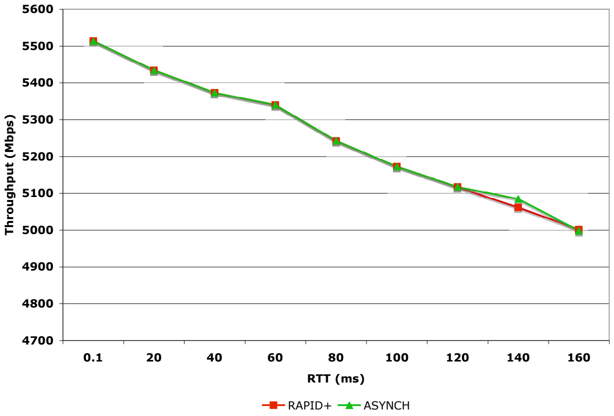

Fig. 3. Effect of RTT on throughput

4 A Case for Proactive Rate Control

Here we analyze the receiver end-host in environments where a number of competing processes are scheduled on the receiver node.

4.1 Effect of Load on the Receiver End-Host

In environments where a number of competing processes are scheduled, the receiver's network process must compete with the other processes on the end host for CPU resources. The competing processes may be of various types and can be I/O-bound or CPU-bound.

We use four types of loads for our experiments. The first is a purely computational workload that runs entirely in user space. The second is a system-call load that reads and writes data to a flat file. The third is a memory-intensive load that reads data from a 1GB buffer. Finally, we have a network load that acts as a forwarder for packets received by the receiver process. We use a four-node setup; two of the nodes perform the actual communication, one node acts as a delay node emulating a high-latency network, and the fourth node acts as a gateway, where the receive and forward processes are running. The gateway node performs some computation on every received packet and then forwards it to the final recipient of the data.

With the exception of network load, all loads are running on the same processor core as the receive process. For network load, we run the receiver on P1C2 (Processor 1, Core 2) and all forwarder threads on P1C1 since cores on the same processor share cache and memory. This helps obtain maximum end-to-end throughput.

Figures 4 and 5 compare the loss rate and throughput of RAPID+ and ASYNCH with increasing load, respectively. For the purely computational load, ASYNCH quickly adapts its sending rate, thus avoiding any severe packet loss.

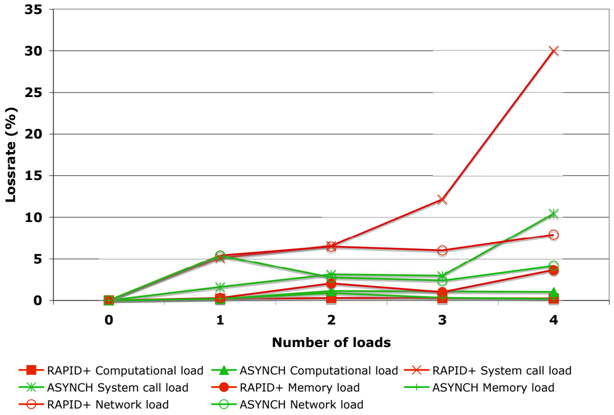

Fig. 4. Effect of various loads on loss rate

However, neither the loss rate nor throughput has any fixed pattern with increasing load.

For the system-call load, the average loss rate is much higher for both ASYNCH and RAPID+. This is attributed to the higher priority assigned by the operating system to system-call workloads. Consequently, fewer CPU resources are allotted to the receiver process, resulting in an increased loss rate. On the other hand, the memory-intensive load demonstrates behavior that is better than the system-call workload, but worse than the compute workload.

For the network load, the throughput, as seen by receiver process, is not affected significantly by the increase in number of forwarder threads, mainly because the forwarder is running on a separate core. However, the end-to-end throughput (not shown in figure) reduces, since the forwarder threads, all running on the same core, compete amongst themselves for CPU resources.

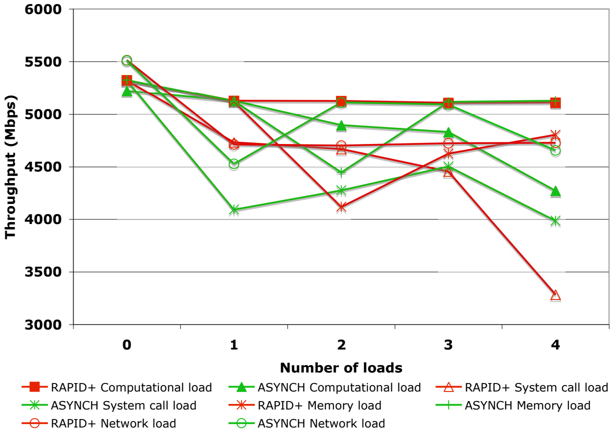

Fig. 5. Effect of various loads on throughput

4.2 Loss Patterns

Figure 6 shows the pattern of packet loss when data is transmitted with rate adaptation disabled. We disable rate adaptation in this experiment in order to observe the loss patterns that would help us understand if the current reactive approaches are able to adapt precisely during intervals of packet losses.

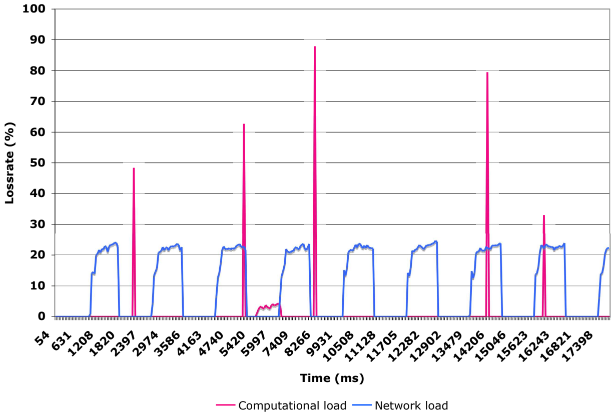

Fig. 6. Loss patterns for computation load and network load

The sharp spikes for pure computational load reveal that all packets in a small interval of time are lost. Similar spikes were observed for system-call load and memory-intensive loads, although not shown in the figure. When rate adaptation is enabled, ASYNCH and RAPID+ interpret the spike as a heavy loss rate and wrongly adapt the sending rate. In general, the presence of spikes in any loss pattern is likely to convey a wrong signal to the reactive rate-adaptation algorithm. The loss pattern for a network load does not have any spikes and flattens on the top. Thus, the reactive rate adaptation is expected to work for this case as the future loss pattern is likely to remain the same as the current loss pattern.

4.3 Reactive versus Proactive Approach

Based on the loss patterns described above, it is clear that a reactive approach is not suitable to adapt the rate, based on dynamic conditions at the receiver end host. A reactive approach will work if the conditions at the receiver are static, resulting in steady loss patterns, e.g., loss patterns due to another network load on the receiver. On the other hand, a proactive approach can potentially predict the time when a loss event will occur and take necessary action to prevent packet drops. However, designing a proactive approach is a difficult problem that requires understanding of the way an operating system scheduler handles processes.

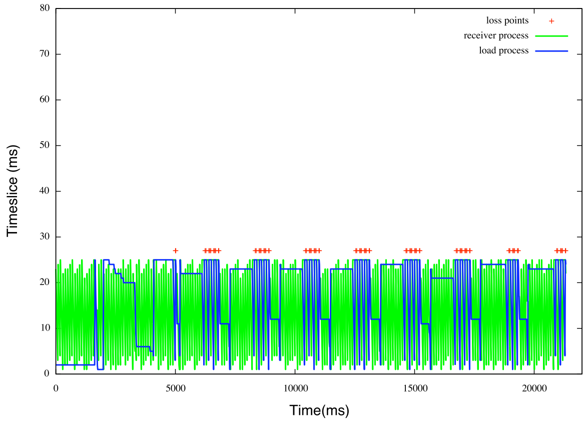

Fig. 7. Timeslice consumption of processes showing intervals when each of them is currently scheduled for execution

Figure 7 shows the timeslices awarded to the receive process and a memory-intensive load process. The exact times when packet loss occurs have been marked in the figure. These points are always located in the same time interval when the load process is running, depriving the network process of CPU resource and thus causing it to drop packets. In other words, packet losses occur exclusively because the receive process gets rescheduled and is replaced by the competing load process.

4.4 A Proactive Approach

In designing a proactive approach, we need to estimate in advance when the receive process will be rescheduled and replaced by another process for execution. We will refer to this time as *context-switch time*, since this rescheduling essentially involves a context-switch. There are two approaches for predicting when a context-switch for the receive process will occur, as discussed below.

Polling Dynamic Priority Polling the dynamic priority of the receive process to estimate the context-switch time has been used by Banerjee et al. [1]. The idea is to note the dynamic priority of the receive process when a loss actually occurred. During polling, if the dynamic priority reaches one less than the previously noted value, the sender is notified to suspend transmission for an amount proportional to the average sleep time of the process. However, as seen in Figure 8 for a memory-intensive load, the dynamic priority of the receive process takes only three values and just prior to getting context switched, there are no changes to the dynamic priority value. This behavior has been observed for other kinds of loads as well and therefore this approach is not reliable.

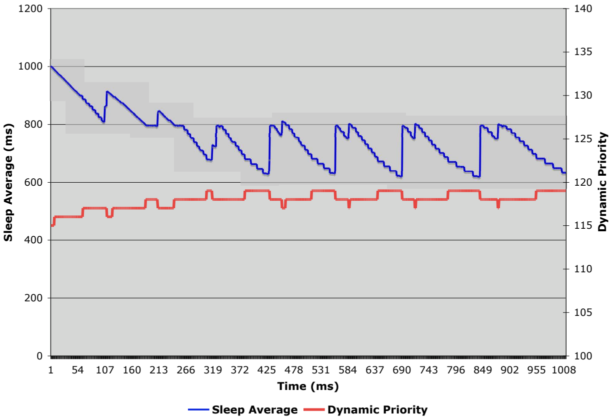

Fig. 8. Sleep average and dynamic priority for a memory access intensive process

Polling Sleep Average Figure 8 shows the average sleep time of the receive process. We see that the average sleep time of a process follows a saw-tooth pattern. A context-switch happens whenever the average sleep time has reached a local minimum. Because of this uniform and periodic pattern, it is possible to estimate the time of an upcoming context-switch.

If at any instance of time, we have the maximum and minimum *average sleep time* (given by MAX_SLEEP_AVG and MIN_SLEEP_AVG), we know that the process started its execution with its *average sleep time* = MAX_SLEEP_AVG and will be rescheduled when its *average sleep time* has reached MIN_SLEEP_AVG. We take an action when the *average sleep time* reaches $(MAX\_SLEEP\_AVG + MIN\_SLEEP\_AVG)/2$. We must constantly update the maximum and minimum values, since they are likely to change.

5 Conclusion and Future Work

In this paper, we presented an asynchronous, feedback-based, reactive, rate-control protocol called ASYNCH that features a fine-grained rate-control mechanism. Our protocol effectively solves some of the problems faced by current rate-based protocols that adapt sending rate, leading to accurate rate adaptation and therefore higher throughput. We also analyzed the end-host behavior in dynamic environments and made a case for a proactive protocol, which is more suitable for handling such environments.

References

1. A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal. RAPID: An End-System Aware Protocol for Intelligent Data Transfer over LambdaGrids. *20th IEEE International Parallel and Distributed Processing Symposium*, pages 10 pp. –, 2006.

2. P. Datta, W. Feng, and S. Sharma. End-System Aware, Rate-Adaptive Protocol for Network Transport in LambdaGrid Environments. *ACM/IEEE Supercomputing 2006*, 2006.
3. P. Datta, S. Sharma, and W. Feng. A Feedback Mechanism for Network Scheduling in LambdaGrids. *IEEE International Symposium on Cluster Computing and the Grid*, 2006.
4. T. DeFanti, C. Laat, J. Mambretti, K. Neggers, and B. Arnaud. TransLight: A Global-Scale LambdaGrid for e-Science. *Communications of the ACM*, pages 34–41, 2003.
5. M. Fisk and W. Feng. Dynamic adjustment of tcp window sizes. *Los Alamos Unclassified Report 00-3321*, 2000.
6. R.L. Grossman, M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang. Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks. *J. Supercomput. (Netherlands)*, 34(3):231 – 242, 2005.
7. E. He, J. Leigh, O. Yu, and T.A. Defanti. Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. *IEEE International Conference on Cluster Computing*, pages 317 – 24, 2002.
8. S. Hemminger. Network Emulation with NetEm. In *Australia's National Linux Conference (LCA '05)*, 2005.
9. U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas Revisited. In *IEEE INFOCOM*, volume 3, pages 1546–1555 vol.3, 26-30 Mar 2000.
10. D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. *Comput. Commun. Rev. (USA)*, 32(4):89 – 102, 2002.
11. J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM '98*, pages 315–323, New York, NY, USA, 1998.
12. N. Taesombut and A.A. Chien. Distributed Virtual Computers (DVC): Simplifying the Development of High Performance Grid Applications. *IEEE International Symposium on Cluster Computing and the Grid, 2004.*, pages 715–722, 19-22 April 2004.
13. K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. *25th IEEE International Conference on Computer Communications.*, 25:1–12, 2006.
14. E. Weigle and W. Feng. Dynamic Right-Sizing: A Simulation Study. *Computer Communications and Networks*, 10:152–158, 2001.
15. R. X. Wu and A. A. Chien. GTP: Group Transport Protocol for LambdaGrids. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 228–238, 2004.
16. L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. volume vol. 4, pages 2514 – 24, Hong Kong, China, 2004.