Using Passive Traces of Application Traffic in a Network Monitoring System

Marcia Zangrilli and Bruce B. Lowekamp[*]
Computer Science Department
College of William and Mary
Williamsburg, VA 23187-8795
{mazang,lowekamp}@cs.wm.edu

Abstract

Adaptive grid applications require up-to-date network resource measurements and predictions to help steer their adaptation to meet performance goals. To this end, we are interested in monitoring the available bandwidth of the underlying networks in the most accurate and least obtrusive way. Bandwidth is either measured by actively injecting data probes into the network or by passively monitoring existing traffic, but there is a definite trade-off between the active approach, which is invasive, and the passive approach, which is rendered ineffective during periods of network idleness. We are developing the Wren bandwidth monitoring tool, which uses packet traces of existing application traffic to measure available bandwidth. In this paper, we demonstrate that the principles supporting active bandwidth tools can be applied to passive traces of the LAN and WAN traffic generated by high-performance grid applications. We use our results to form a preliminary characterization of the application traffic required by available bandwidth techniques to produce effective measurements. Our results indicate that a low overhead, passive monitoring system supplemented with active measurements can be built to obtain a complete picture of the network's performance.

1 Introduction

Adaptive grid applications require timely and accurate information about the available computational and network resources in their environment. To accurately monitor the network resource availability, adaptive applications can rely on techniques that either actively or passively measure available bandwidth. Techniques that actively inject traffic into the network are often very accurate, but may adversely affect the performance of other applications on the network. Although the passive approach avoids the problem of contention by passively observing and collecting network characteristics, it is limited to periods of time when there is traffic on the network between the hosts of interest. We want to develop a new approach to network monitoring that combines elements of passive and active techniques to offer accurate, timely bandwidth measurements while limiting the invasiveness of probes.

The Wren system we are developing will use a hybrid monitoring approach to passively trace existing application traffic and actively inject traffic when necessary to maintain a continuous flow of bandwidth measurements. These measurements can be used by the application to make immediate runtime decisions, and they can also be stored by a central monitoring system for use in the future or by other applications. In order to implement this hybrid system, we need to monitor existing application traffic, calculate bandwidth based on the information collected, and detect when there is insufficient traffic so we can then inject active probes. This paper presents the passive component of this system and evaluates its ability to use packet traces of real application traffic to determine the available bandwidth on a network.

We set two primary requirements for our implementation.

- No modification to the application code or the network infrastructure may be required. Code modification is one of the largest hurdles to grid application development. Our implementation should be as transparent to the user as possible.

- Application performance must not suffer. The monitoring software should not compete with the application for network or CPU resources.

Based on these requirements, we have developed a kernel-level packet trace facility, as an extension to the

[*]This research was supported in part by the National Science Foundation under award ACI-0203974. This work was also performed in part using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

Web100 kernel [11], to provide passive monitoring services. There are several benefits of implementing our facility at the kernel-level:

- Traffic can be captured without modifying the application.
- The amount of additional overhead can be minimized.
- Packets can be accurately timestamped to nanosecond precision.
- Kernel-level protocol information, such as TCP congestion window size and sequence numbers, can be recorded.

To calculate available bandwidth, we must apply techniques designed for use with active probes to our passive traces. Most active bandwidth techniques rely on adjustments to the amount and/or rate of data being sent across the network. Our primary challenge with using passive packet traces is that we have *no control over the traffic pattern*.

This paper provides an overview of the Wren packet trace facility and describes how to apply techniques commonly used to actively measure bandwidth to passive traces of application traffic, a task complicated because we have no control over the application traffic pattern. We have evaluated our approach on LAN- and WAN-based distributed applications and present the results of applying our analysis to traffic captured from these applications. The principle contributions of this paper are

- a system that captures packet traces unobtrusively,
- new algorithms for applying the principles of active probing techniques to passive traces,
- characterization of the network traffic an application must produce for our techniques to provide valid available bandwidth measurements, and
- demonstration that both bulk transfer and bursty BSP-style communication produce packet traces that can be used to measure available bandwidth.

The remainder of this paper will describe the status of the Wren packet trace facility. We will first review related work on monitoring systems. Section 3 will discuss the range of traffic patterns generated by grid applications. In Section 4, we describe our packet trace implementation and the bandwidth techniques implemented in the user-level. In Section 5, we present the results of using Wren to monitor available bandwidth.

2 Monitoring Systems

Packet trace tools, like tcpdump, monitor network traffic to verify the operation of network protocols and to characterize application traffic patterns. Tcpdump uses the libpcap packet capture interface to initiate tracing and process the packet trace into a standard representation. In Linux systems, libpcap interacts with the Linux Socket Filter (LSF), a variation of BSD Packet Filter (BPF) [14], which is composed of a network tap that sits at the link device layer and collects packets specified by user-defined filter rules. The use of the LSF filtering mechanism improves performance because unwanted packets are filtered in the kernel instead of being copied into the user-level for processing by libpcap. A drawback of using LSF to trace packets is the need for applications to be reading the socket to collect the packets as they arrive. In contrast, an application that uses the Wren system can read data from a kernel buffer at any point after the trace is completed. More importantly, it may be difficult for system that uses LSF to coordinate traces of the same range of packets on two machines. We have designed Wren with a triggering mechanism that specifies the same range of packets will be monitored on both machines.

Shared Passive Network Performance Discovery (SPAND) [20] uses information passively collected from several hosts on a network to measure network conditions. Performance data is collected from client applications and packet capture hosts. The performance reports sent by the client applications are based on application-level observations and may lack the detail to provide accurate estimates of available bandwidth. Packet capture hosts, which use BPF to observe all traffic to and from a group of hosts, are the primary means of collecting information. These hosts are not at the end-points of the path, and therefore, must use heuristics to infer end-to-end properties such as achievable bandwidth. The packet capture host is responsible for processing the data before sending the performance report to the server. In our Wren system, the host collecting the kernel-level packet trace can send the data to another machine where all the processing will occur. More importantly, Wren uses a two-sided approach to monitoring traffic at the both end hosts of path, allowing for more accurate measurements of end-to-end properties.

SPAND maintains a central repository of measurements at a performance server, which can be queried by any application in the system. SPAND measurements are shared and passive, features that we have incorporated into the design of Wren. In SPAND, there is an emphasis on the performance the application can obtain rather than the total availability of network resources. However, SPAND does provide an achievable bandwidth metric, which is similar to the bulk transfer capacity metric [12] discussed in Section 4.2.1. Our Wren implementation has the added fea-

Table 1. Information collected during Wren packet trace of TCP traffic.

Incoming packets				Outgoing packets			
timestamp	seq number	ack number	TCP cwnd	timestamp	seq number	ack number	data size

ture of being able to apply several available bandwidth techniques to the same packet trace.

Web100 [11] is designed to monitor, diagnose, and tune TCP connections. Web100 comprises a kernel-level component, which is responsible for exposing the characteristics of the TCP connection, and a user-level component, which retrieves and graphically displays the connection information. The Web100 tool instruments the network stack of the 2.4 series of Linux kernels to capture an instantaneous view of the TCP connection internals and exports that information to the user-level through an interface in the /proc file system. The Web100 tool has an autotuning functionality and also provides a mechanism for hand-tuning kernel variables. The appeal of the Web100 tool is the ability to track the current state of variables in TCP connections and to tune buffers accordingly in real-time at the user-level.

We are developing the Wren bandwidth monitoring tool as an extension to the Web100 kernel so that a single kernel can provide the variety of network services required for high-performance networking. We chose to implement the kernel-level portion of the Wren bandwidth monitoring tool as an additional feature to Web100 because monitoring available bandwidth and buffer tuning are both used in the same situations to improve application performance.

3 Traffic Patterns

Grid applications can generate traffic patterns composed of many long transfers, many short transfers, or some combination of both. At one end of the spectrum are bulk data transfer applications, which move large amounts of data between two points and tax the communication stack by continuously sending data. Many grid applications have communication phases that behave similarly to bulk data transfers, such as the initial distribution of work among processors or the migration of large work units during load balancing.

At the other end of the spectrum of grid applications are bulk synchronous parallel (BSP) applications with compute-communicate phases and sporadic traffic patterns. Common to these applications is the need to periodically synchronize or communicate information with other processors. For example, in a mesh generation application [2], processors may need to be notified about inserted points or other refinements made on the boundaries shared by the processors.

In our evaluation of Wren, we use a bulk data transfer and a distributed eigensolver, so we cover a representative spectrum of the traffic produced by high-performance grid applications.

4 Wren Bandwidth Monitoring Tool

The Wren bandwidth monitoring tool is split into a kernel-level packet trace facility and a user-level trace analyzer. The kernel-level packet trace facility is responsible for gathering information associated with incoming and outgoing packets. In the user-level, once the traces are acquired any or all available bandwidth techniques can be applied. The ability to apply several bandwidth techniques to the same packet trace allows us to determine the relationship and effectiveness of the techniques. The available bandwidth measurements can be given directly to the application that generated the traffic or stored by a monitoring system for use in the future or by other applications.

In the remainder of this section, we review the kernel-level packet trace facility and discuss the user-level trace analyzer, specifying how we passively implemented four bandwidth techniques. We finish this section with a consideration of the security of the system.

4.1 Wren Packet Trace Facility

Wren extends the functionality of the Web100 kernel to incorporate kernel-level packet traces. Because Web100 is a secure, well-accepted system, our Wren packet trace facility should be acceptable to system administrators. In our implementation, we added a buffer to the Web100 kernel to collect characteristics from incoming and outgoing packets. Table 1 shows the information collected during a trace of TCP traffic. In the UDP and TCP code we timestamp the packets using the CPU clock cycle counter and record the timestamps and TCP congestion window size in our buffer. Access to this buffer is through two system calls: one starts the tracing and specifies the connection to trace and the other retrieves the trace from the kernel. The Wren architecture is designed so that user-level components do not need to be reading from the packet trace facility all of the time. Wren can capture full traces of small bursts of traffic or periodically capture smaller portions of continuous traffic and use those to measure available bandwidth. We have previously analyzed the efficiency of our design [23] and found that Wren adds little overhead to the kernel.

The precise timestamp and cwnd values are only possible through kernel-level instrumentation. Another key design feature of the Wren packet trace facility is the ability to coordinate measurements between machines. In our implementation, one machine triggers the other machine by setting a flag in the headers of outgoing packets to start tracing the same range of packets. The other machine traces all packets with the header flag set, which prevents lost packets from adversely affecting the coordinated tracing. This packet coordination ensures that the buffers at each end of the connection store information about the same packets regardless of the bandwidth-delay product of the particular connection.

4.2 Wren Trace Analyzer

The user-level component of the Wren bandwidth monitoring tool is responsible for initiating the tracing, collecting the trace from the kernel, and processing the data. We apply the bandwidth techniques at the application level to avoid slowing the performance of the operating system. The overhead imposed by the user-level code is minimal, and the data can be transferred to another machine for processing, if necessary.

4.2.1 BTC

The Bulk Transfer Capacity (BTC) [12] metric specifies the data rate that a single congestion-aware transport connection can obtain. Tools, such as SPAND [20] and NWS [22], use the BTC metric to measure achievable bandwidth. In Wren, we observe the total number of bytes sent the packet trace and divide the total number of bytes sent by the time elapsed to produce a measure of the actual throughput of the application, which is the same as the BTC when the application is sending at full blast.

4.2.2 TCP window

The TCP window technique uses the TCP protocol's congestion window size variable to measure bandwidth. TCP tracks the congestion in the network by maintaining a congestion window size variable (cwnd), which specifies how much data can be sent per round trip time. The TCP protocol slowly probes for congestion of the network by increasing the congestion window (and thus the sending rate) until loss is detected and then the congestion window size is reduced. Because cwnd and MSS determine the rate at which TCP sends data, dividing the product of the cwnd and MSS by the round trip time (RTT) yields a measure of achievable throughput [13]. The Wren packet trace facility records the cwnd variable when packets arrive in the kernel so that the TCP window technique can be applied.

4.2.3 Packet Pair

Packet dispersion techniques can be used to measure the capacity [1, 3, 9] or the available bandwidth [1, 5, 8, 16, 21] of a network path. The premise of packet dispersion techniques is that if a train of packets is sent at a rate faster than the bottleneck bandwidth, the train will leave the bottleneck link at a rate equal to the bottleneck bandwidth. The rate the packets are traveling is inversely related to the spacing or dispersion between the packets.

Tools such as cprobe [1] use packet dispersion techniques to calculate bandwidth by dividing the size of the packets by the dispersion of the train as measured on the receiving host. However, this approach measures a metric called the asymptotic dispersion rate [3, 9]. More recently, spruce, IGI, and Delphi [18] have used dispersion of packet pairs/train to determine the utilization of the bottleneck link and subtract the utilization from the capacity of the bottleneck link to calculate the available bandwidth.

The success of a packet dispersion technique is dependent on the initial rate of the packets being greater than the bottleneck bandwidth. In TCP bulk data transfer traffic traces, we have observed that packets are often sent out from the OS as a stream of tightly spaced pairs. In our implementation, we analyze the dispersion of these tightly spaced pairs as they arrive at the receiving host. We group packet pairs into trains, average the dispersion of the pairs, and calculate the available bandwidth. We use this approach because:

- The inter-packet spacing is inversely proportional to the sending rate, so the smaller the spacing the faster the rate. This is important for a technique that relies on the sending rate being larger than the bottleneck bandwidth.
- Averaging dispersions of a train helps minimize affects of artifacts in measurements.

We use filtering to eliminate packet pairs that violate assumptions of the packet dispersion technique. Specifically, we remove pairs where the initial packet spacing is larger than the final dispersion because the shrinking of space between packets indicates the packet rate increased.

We recognize that there is still uncertainty in the community about the reliability of measuring available bandwidth using packet dispersion techniques [7, 17]. This is especially true in high bandwidth-delay environments where interrupt coalescing (IC) can cause timestamps to be increased from when packets actually arrive at the NIC. Recently, techniques for detecting IC and removing erroneous measurements have been described [17]. We hope Wren can address these issues by detecting IC or by incorporating NIC timestamping into the packet trace facility and by implementing emerging techniques that rely on other packet spacing, such as exponential, to use on our traces of application-generated trains [4, 19].

4.2.4 SLoPS

Self-Loading Periodic Streams (SLoPS), which is used in pathload [6], is based on the principle that if the rate of a periodic packet stream is larger than the available bandwidth, the one-way delays of this stream will have an increasing trend. If the one-way delays of the stream do not have an increasing trend, the rate of the stream is less than the available bandwidth.

In pathload, SLoPS is an iterative algorithm that requires cooperation of the sender and the receiver hosts. For one iteration, pathload sends out several UDP packets at a fixed transmission rate, determines if the transmission rate is larger or smaller than the available bandwidth, and then adjusts the transmission rate for the next iteration. After several iterations, Pathload's algorithm converges to an available bandwidth range.

In our implementation of SLoPS, we have no control over the initial sending rate of the streams and therefore cannot guarantee that the traffic we are tracing will be sufficient to allow the algorithm to converge to a bandwidth range. However, we hypothesize that TCP traffic can be used to find a maximum limit on the range of available bandwidth because the TCP protocol varies the sending rates of packets according to the congestion window size. In essence, the TCP protocol is doing the same thing that pathload is trying to accomplish: sending a stream of packets that will induce congestion and identifying the rate at which congestion occurs.

In our implementation of SLoPS, we use the timestamps of packets to calculate the one-way delays and the initial sending rate of the stream of packets. We group 10-50 packets that had the same congestion window size when sent into a stream and identify the trend in one-way delays of that stream. Consistent with the pathload implementation, we group several streams together into a one fleet and for that fleet try to identify the maximum range value for the available bandwidth. To test our implementation, we monitored the UDP traffic generated by the pathload tool. In congested and uncongested environments, Wren produced the same measurements as pathload, thus validating the correctness of our implementation. Our SLoPS implementation performs the same convergence tests as pathload and reports results only when TCP naturally produces a sequence of fleets that determines the current available bandwidth.

4.3 Security

We recognize that the interface to packet traces that the Wren bandwidth monitoring tool provides may be considered a security issue. In the current implementation of our Wren bandwidth monitoring tool, there is no restriction on which users can capture traces. The danger here is that any user has access to and can trace any other user's application traffic. However, the amount of information the user can obtain is limited to sequence and acknowledgment numbers and does not include the data segment of the packets. This is not much more information than a user could obtain from the netstat program. Were we to restrict access, deploying a grid-wide monitoring system would require either root access or restrict the monitoring to a single user's applications. But in a production release, we could add the ability to check permissions for access.

5 Results

We have tested the Wren bandwidth monitoring tool by running experiments on LANs and WANs using 100Mb and Gb interfaces. Our results demonstrate that the Wren bandwidth monitoring tool can measure bandwidth using passive traces of application traffic collected on congested and uncongested paths. We use Wren to investigate how our passive bandwidth techniques relate to one another and evaluate each technique in an effort to understand what traffic conditions are necessary for each technique to produce useful measurements.

5.1 Uncongested LAN

We use the Wren bandwidth monitoring tool to collect and analyze traces of iperf traffic between two 2.8 GHz Pentium 4s using 100 Mb and 1 Gb Ethernet interfaces. In this experiment the MTU is set at 1500. In Figures 1 and 2 the packet pair and BTC techniques obtain the same consistent measurement of the bandwidth while the measurements produced by the TCP window technique occasionally dip, reflecting the behavior of the congestion window size being reduced. However, the TCP window technique measurements are quantitatively similar to the BTC and packet dispersion values at the other times. The values obtained by applying the SLoPS technique are in some cases larger than the measurements of the other techniques. In these cases, SLoPS is more likely measuring available bandwidth while the other techniques are measuring achievable bandwidth [10]. In bulk data transfer results shown in Figures 1 and 2, we see that all techniques produce similar bandwidth measurements that are consistent with what we might expect on uncongested 1 Gb and 100Mb LANs.

5.2 WAN

In our WAN experiment, iperf was used to generate traffic from a Pentium 4 machine at the College of William and Mary across the Network Virginia and Abilene networks to

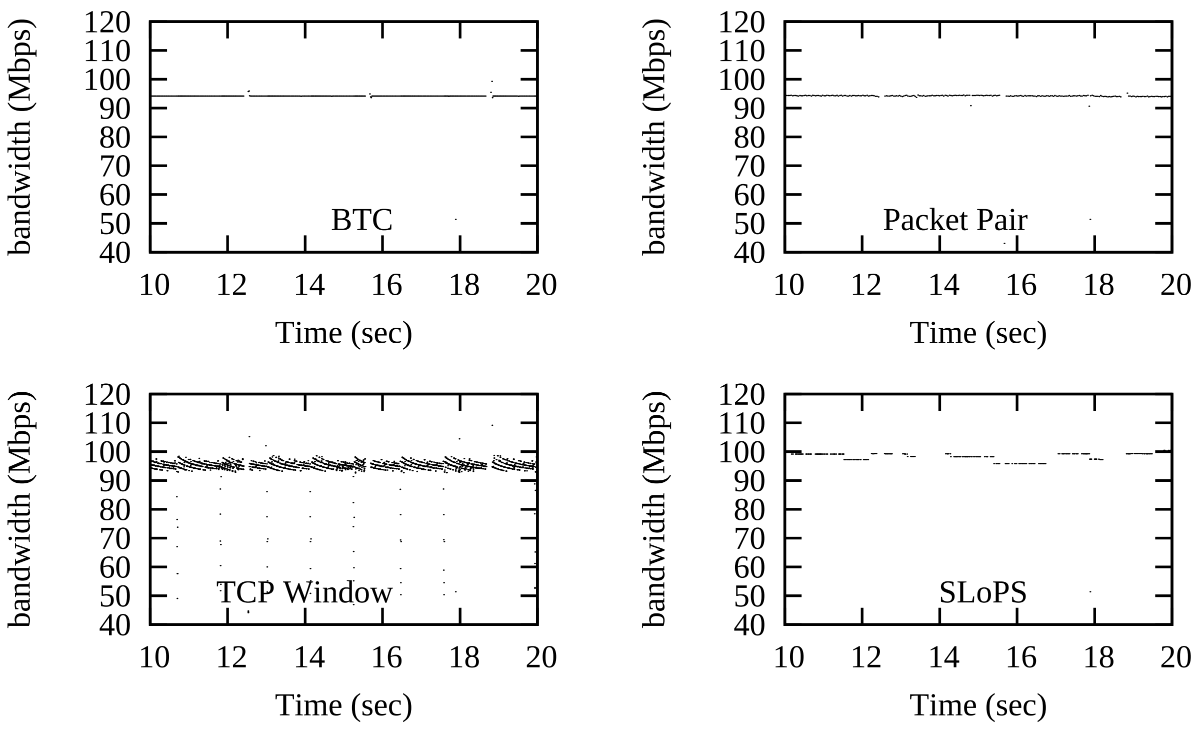

Figure 1. Techniques applied to iperf traffic monitored on uncongested 1OO Mb LAN.

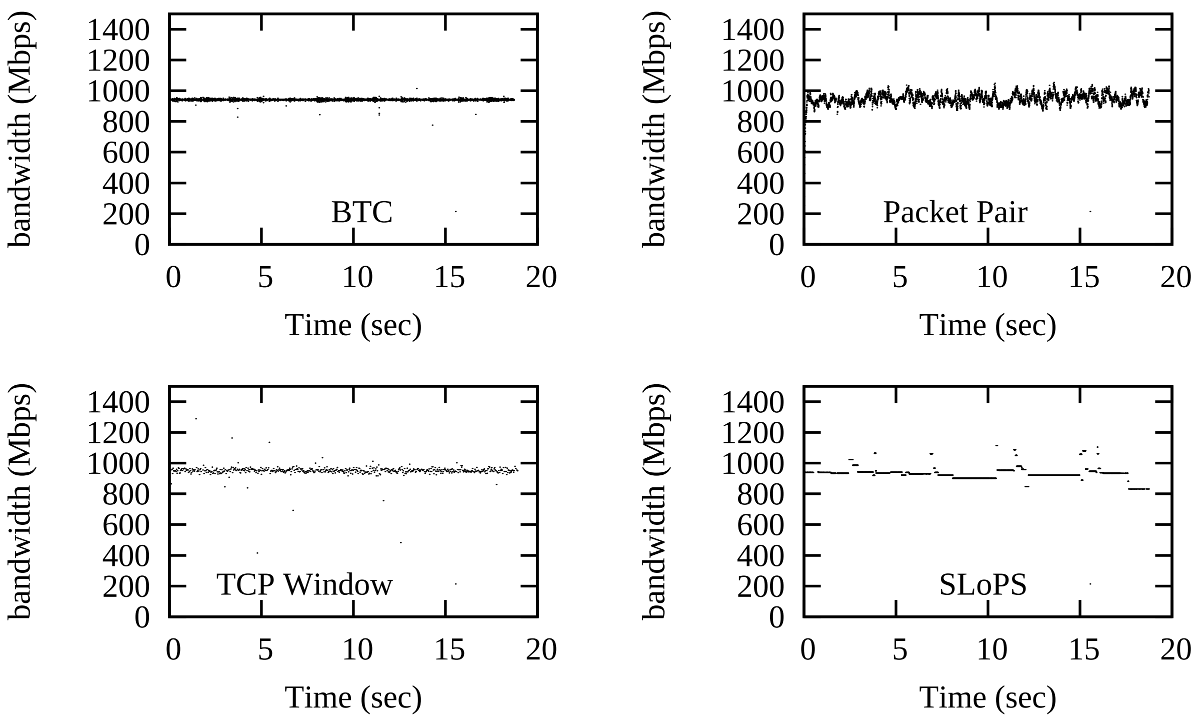

Figure 2. Techniques applied to iperf traffic monitored on uncongested 1 Gb LAN.

a Digital Unix 4.0 machine at Carnegie Mellon University. Typical of modern networks, the capacity bottlenecks of this path are at the endpoints, not the WAN. The bottleneck capacity of this path is 100 Mb, but the obtainable throughput on this path typically maxes out around 70 Mb due to the presence of other traffic on the path.

In Figure 3, the BTC and TCP window techniques illustrate the slow ramp-up and sawtooth patterns associated with TCP's AIMD algorithm. This graph indicates that the BTC and TCP window techniques are measuring throughput instead of available bandwidth until just before a packet loss.

Figure 3 also shows that the packet pair technique produces measurements as soon as back-to-back packets are issued and that both the packet dispersion technique and the SLoPS technique produce values that are quantitatively similar to the other techniques at the height of the ramp-up period. We expect this behavior from SLoPS because the technique is configured to report the maximum value of the available bandwidth range that is computed. The SLoPS and packet pair techniques provide consistent estimates of the available bandwidth, while the other techniques measure the throughput the application is obtaining. Figure 3 shows that the throughput an application achieves is equal the available bandwidth when the TCP window is fully opened.

5.3 WAN Gb

We generated iperf traffic between a dual Pentium III, 800 MHz machine at Vrije Universiteit and an AMD Athlon 2700 MHz machine at Lawrence Berkeley Laboratory, both equipped with Gb NICs. This Gb WAN path has a high bandwidth-delay product, which means it will take longer for the TCP window to open up and for an application to realize its maximum throughput than it would take on a LAN. In our experiments, it took approximately 5 seconds before the TCP window opened fully, at which point the BTC and TCP window techniques were able to measure available bandwidth. These results verify that our software has no problems tracing packets at Gb speed.

The packet pair and SLoPS techniques are not applied to this trace because the packets arriving at the receiver had a constant spacing, which indicates interrupt coalescing is occurring. The inability to effectively apply these techniques is not surprising because of previous results that indicate NIC-level timestamping or IC detection is necessary for high bandwidth connections [7, 17].

5.4 Traffic Requirements on LAN

In this experiment, we monitor TCP traffic produced by iperf running between two Pentium 4s on a 100 Mb LAN with 0, 20 and 40 Mb of TCP cross traffic on the link. The iperf TCP traffic and the TCP cross traffic both traverse the same link, but different machines generate the traffic.

Figure 4 presents the results of applying the BTC, TCP window, SLoPS, and the packet pair techniques to the iperf traces. In each graph, we see distinct bands reflecting the change in the amount of cross traffic demonstrating that all of the techniques can detect the changes in available bandwidth. This graph also shows that the SLoPS algorithm may not always converge, but when it does converge the measurements are correct. All of the techniques produce bandwidth measurements that reflect the availability of the bandwidth on the path when reduced by 20 and 40 Mb.

In Figure 4, the BTC and TCP window measurements have spikes when a congestion event occurs until TCP enters AIMD again, but the packet pair technique is not affected by the congestion event. The important observations are that the packet pair technique measurements are not affected by the congestion window size, but the other techniques require TCP to be in AIMD.

Our next experiment focused on determining the amount of smallest amount of data that must be sent for the packet pair technique to be applied. We are interested in quantifying how many back-to-back packets are required so that the packet pair technique can measure available bandwidth. We refer to a group of back-to-back packets as a burst and the burst size as the total number of bytes in a burst.

For this experiment, we wrote an application, which ran across a 100 Mb LAN between two Pentium 4s and sent 96 bursts of various sizes at a frequency of .2 seconds. For the various burst sizes, we look at the number of measurements produced by applying the packet pair algorithm to traces of the 96 bursts. The number of measurements obtained will depend on how many packets are filtered out to ensure that no assumptions of the packet pair technique were violated.

Table 2. Number of measurements obtained by applying packet pair technique to 96 bursts of traffic sent every .2 seconds.

burst size	number of measurements
70 KB	0
75 KB	75
80 KB	84
90 KB	105
100 KB	119
150 KB	173

Table 2 shows the number of bandwidth measurements we obtained from various burst sizes. While some valid measurements are obtained using bursts of 75 and 80 KB, the fact that there are fewer bandwidth measurements than

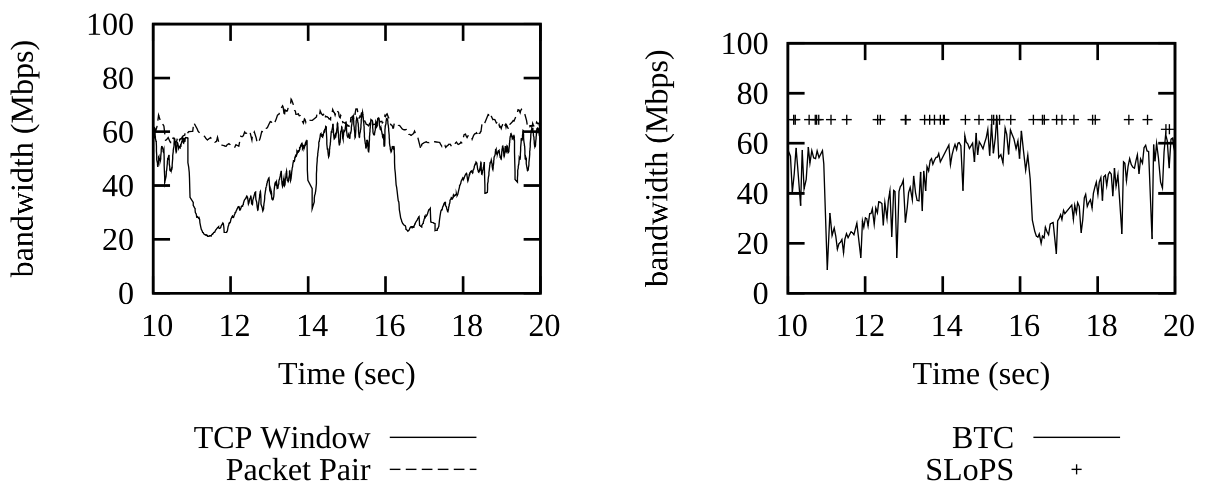

Figure 3. Techniques applied to trace of iperf traffic on a 100Mb WAN between W&M and CMU.

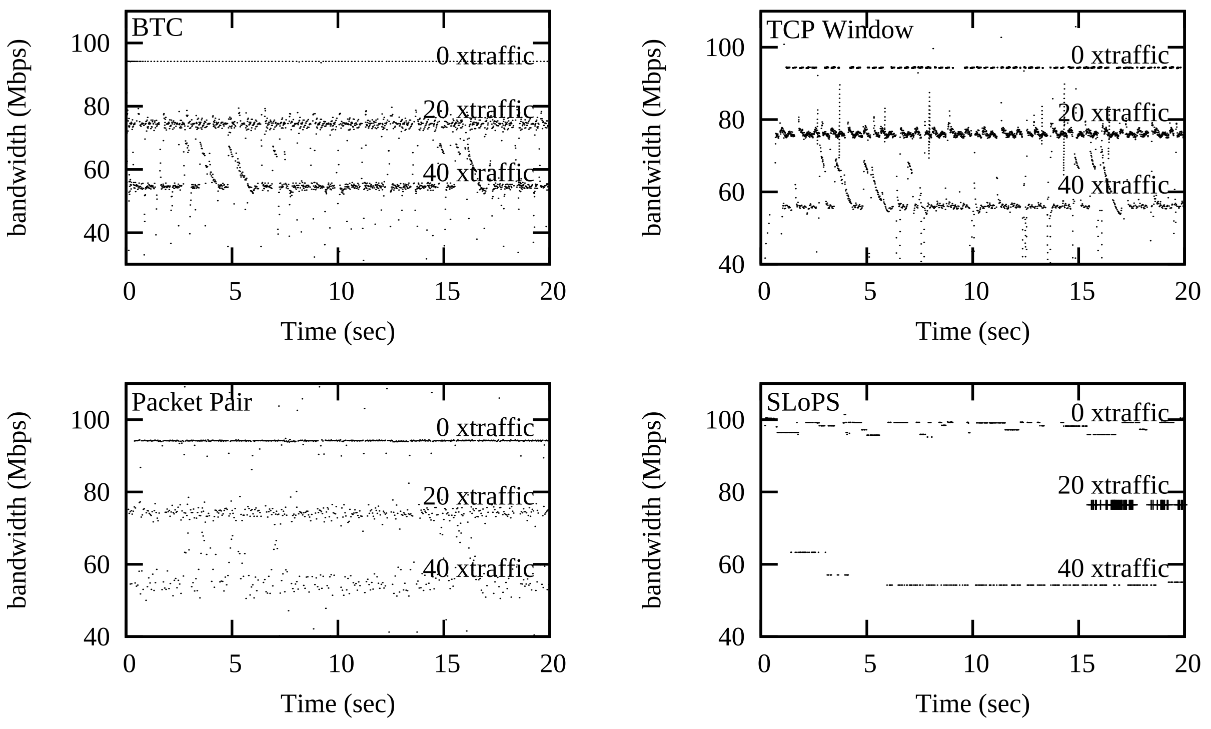

Figure 4. Techniques applied to iperf traces on a 100Mb LAN with 0,20,40 Mb of cross traffic.

number of bursts indicates that the packet pair technique cannot always be applied to bursts this small. The packet pair technique is more reliable using bursts larger than 90 KB because these burst sizes ensure that there is is a one-to-one ratio in terms of burst sent to measurements obtained. Our results for this connection indicate that the packet pair technique can only be reliably applied to bursts of application traffic that are at least 90 KB.

5.5 Bursty Application Traffic

We performed experiments with an adaptive multigrain eigenvalue application, which is designed to find the extreme eigenvalues of large, sparse matrices [15]. This solver is comprised of a projection phase with local computation and a correction phase with communication within clusters. The communication is bursty, with several short messages preceding larger messages. We believe this algorithm has characteristics similar to a large number of latency-tolerant high-performance computing applications that may be run on clusters and grids, and therefore, is an ideal choice to evaluate the performance of our bandwidth measurement techniques when running significant, non-trivial applications with interesting communication patterns.

We ran the eigensolver on a cluster of four Pentium III Linux machines linked together by 100Mb connections and monitored traffic between two of the nodes running the eigensolver. Figure 5 shows the bursty traffic pattern generated by the eigensolver application. Notice that the application only sends at full rate between 6-9 seconds, but the packet pair technique is able to measure bandwidth effectively throughout the duration of the packet trace.

Some of the bursts sent out during the communication phase were more than 90 KB, the burst size qualification for application traffic to be used by the packet pair technique. In Figure 5, the packet pair bandwidth measurements demonstrate that our burst size assessment is valid.

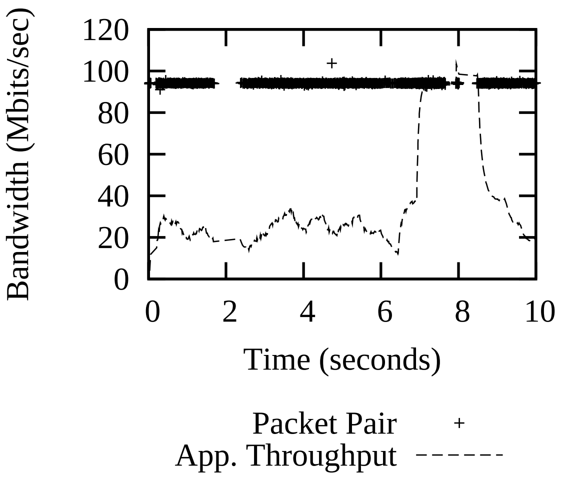

Figure 5. Throughput of an adaptive eigensolver measured on a 100 Mb LAN. This application has a bursty traffic pattern with a maximum rate obtained only between 6-9 seconds, but the packet pair technique is able to provide effective measurements throughout the duration of the trace.

6 Conclusion

This paper describes and evaluates the passive component of the Wren bandwidth monitoring tool. We have implemented the BTC, packet pair, TCP window, and SLoPS algorithms to calculate bandwidth and have analyzed the effectiveness of these techniques on bulk data transfer traffic and bursty application traffic on 100 Mb LANs, 1 Gb LANs, and WANs. Our results show that all of the techniques can identify the presence of competing traffic on a network path and these techniques often produce quantitatively similar measurements.

While our tool does not fail to trace application traffic, the bandwidth techniques can fail to calculate the available bandwidth of the path. We presented a preliminary analysis of the application traffic characteristics needed to apply the bandwidth techniques. We found that the SLoPS, BTC, and TCP window techniques are only able to effectively measure available bandwidth when the TCP window opens up in the AIMD phase. While the packet pair technique could still be applied applied to more bursty traffic that never opened the TCP window, this technique did require a minimum amount of data to be sent in a burst to provide bandwidth measurements. This analysis is important because it specifies the minimum amount of traffic an application must generate to make passive measurements. In the future, we will use this information to determine when active measurements should be started so that the bandwidth information remains up to date.

We are continuing to investigate how to detect and obtain valid results when interrupt coalescing occurs. We also plan to continue with a more rigorous analysis of the limitations of the bandwidth techniques and how various patterns in data streams affect the measurement accuracy.

7 Acknowledgments

We thank those who allowed us to use their hosts as platforms for our experiments: Nancy Miller (CMU), Brian Tierney (LBL), Kees Verstoep, Thilo Kielman and Andrew Tanenbaum (Vrije Universiteit). Thanks also to Claire O'Shea for software development.

References

[1] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, October 1996.

[2] N. Chrisochoides, A. Fedorov, B. Lowekamp, M. Zangrilli, and C. Lee. A Case Study of Computing on the Grid: Parallel Mesh Generation. In *Next Generation Systems Program Workshop, IPDPS*, 2003.

[3] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *INFOCOM01*, 2001.

[4] G. He and J. C. Hou. On Exploiting Long Range Dependence of Network Traffi c in Measuring Cross Traffi c on an End-to-end Basis. In *IEEE Infocom*, 2003.

[5] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 2003.

[6] M. Jain and C. Dovrolis. Pathload: a measurement tool for end-to-end available bandwidth. In *Proceedings of the 3rd Passive and Active Measurements Workshop*, March 2002.

[7] G. Jin and B. Tierney. System capability effects on algorithms for network bandwidth measurement. In *IMC*, 2003.

[8] G. Kin, G. Yang, B. R. Crowley, and D. A. Agarwal. Network characterization server (NCS). In *HPDC11*. IEEE, August 2001.

[9] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *In Proceedings of USENIX Symposium on Internet Technologies and Systems*, 2001.

[10] B. B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. Enabling network measurement portability through a hierarchy of characteristics. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.

[11] M. Mathis. ”Web100” http://www.web100.org, 2000.

[12] M. Mathis and M. Allman. A framework for defi ning empirical bulk transfer capacity metrics. RFC 3148, 2001.

[13] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 1997.

[14] S. McCanne and V. Jacobson. The BSD packet fi lter: A new architecture for user-level packet capture. In *USENIX Winter*, 1993.

[15] J. R. McCombs and A. Stathopoulos. Multigrain parallelism for eigenvalue computations on networks of c lusters. In *Proceedings of the Eleventh International Symposium On High Performance Distributed Computing (HPDC)*, July 2002.

[16] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Internet Symposium*, 2000.

[17] R. Prasad, M. Jain, and C. Dovrolis. Effects of interrupt coalescence on network measurements. In *Passive and Active Measurements Workshop (PAM)*, 2004.

[18] V. Ribeiro, M. Coates, R. H. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk. Multifractal cross-traffi c estimation, 2000.

[19] V. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathChirp:Effi cient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement Workshop (PAM)*, 2003.

[20] M. Stemm, R. Katz, and S. Seshan. A Network Measurement Architecture for Adaptive Applications. In *INFOCOMM*, 2000.

[21] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Internet Measurement Conference (IMC)*, 2003.

[22] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High Performance Distributed Computing Conference (HPDC)*, 1997.

[23] M. Zangrilli and B. B. Lowekamp. Comparing passive network monitoring of grid application traffi c with active probes. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.