

# Comparing Passive Network Monitoring of Grid Application Traffic with Active Probes

Marcia Zangrilli and Bruce B. Lowekamp \*  
College of William and Mary  
{mazang, lowekamp}@cs.wm.edu

## Abstract

*Distributed applications require timely network measurements so that they can adapt to changing network conditions and make efficient use of grid resources. One of the key issues in obtaining network measurements is the intrusiveness of the measurements themselves—how much network performance is “wasted” taking the measurements? Our goal is to combine active and passive monitoring techniques to reduce the need for intrusive measurements without sacrificing the accuracy of the measurements. We are developing a bandwidth monitoring tool as part of the Wren network measurement system that will reduce the burden on the network by passively obtaining measurements from existing application traffic whenever possible, instead of actively probing the network. By using passive measurements when an application is running and active measurements when none are running, we can offer accurate, timely available bandwidth measurements while limiting the invasiveness of active probes. We have completed a prototype of the Wren bandwidth monitoring tool and present our preliminary analysis of its performance in this paper. We provide results from passive implementations of several available bandwidth techniques and demonstrate the close quantitative relationship between the results of both active and passive techniques. We have tested our implementation in a cluster, across a campus, and across the Internet using bulk data transfers as well as an adaptive eigenvalue application. Our results with this diverse set of environments and traffic types show promise toward implementing these techniques as measurement services in production environments.*

---

\*This work was 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. This research was also supported in part by the National Science Foundation under award ACI-0203974, an REU supplement, the ARPA and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-1-0287, and the RI grant number EIA-9972853.

## 1. Introduction

Monitoring network characteristics is critical to the performance of grid applications because grid environments are composed of various resources interconnected by LAN and WAN networks. Distributed applications rely on timely and accurate information about the available computational and network resources to make informed decisions about where to send data for computation and how to adapt to changing conditions. Application middleware can use available bandwidth measurements to choose the best source from which to copy data or to help guide grid scheduling decisions with the overall goal of making efficient use of all the grid resources.

Available bandwidth is measured either by actively injecting data probes into the network or passively monitoring existing traffic. The active approach may cause competition between application traffic and the measurement traffic, reducing the performance of useful applications. Although the passive approach avoids the problem of contention by passively observing and collecting network characteristics, the passive approach is limited to periods of time when there is traffic on the network between the hosts of interest. If we can obtain the same information from both approaches, we can transparently switch between active and passive measurements to avoid using stale data and to reduce the contention between application and measurement traffic. Numerous systems have been developed to coordinate probes and to store and disseminate data so that applications can adapt to available resources [11, 14, 20] and several groups have developed tools that can passively monitor application traffic [1, 8, 9, 18]. Our goal is to extend these projects to combine active and passive measurements into a single tool and thereby maximize the measurement accuracy while minimizing the intrusiveness of the measurements.

Our proposed solution is an architecture where application-generated traffic is captured and used for performance prediction when possible; when no existing traffic is available, active probes are used to provide a constant series of measurements [10]. This architecture allows an

application to:

- modify its adaptation strategy at runtime based on knowledge of additional available bandwidth
- report its measurements to a monitoring system, such as NWS, for use by other or future applications.

Our Wren bandwidth monitoring tool is designed to incorporate the key portions of packet tracing into the kernel, with the analysis and management of the traces handled at user-level. The design goal is easy deployment and integration in a secure, production system so that an individual user can monitor only packets associated with their application. The system should add no overhead to the system when not collecting packets and minimal overhead when collecting a packet trace. We believe that a system meeting these design goals would be as acceptable to system administrators as Web100 [12], a system commonly used to monitor and tune TCP connections.

The prototype implementation of the Wren bandwidth monitoring tool collects data through kernel-level packet traces and can provide information when running on one or both hosts involved in communication. We envision using our packet trace facility on grid computing resources, such as dedicated clusters. In environments where kernel-level packet tracing is infeasible, some of the information we use can be extracted directly from the network through packet capture and analysis hardware that is currently available.

We evaluate the effectiveness of the Wren bandwidth monitoring tool and the impact of its overhead by measuring available bandwidth for a TCP bulk data transfer application. Because TCP bulk data transfers tax the communication stack by continuously sending data, we can discover how several available bandwidth techniques relate to one another and learn what limitations, if any, exist for each technique in various environments. Additionally, many grid applications have communication phases that behave similar to bulk data transfers, such as the initial distribution of submatrices between processors or the migration of large work units during load balancing.

We also use the Wren bandwidth monitoring tool to measure bandwidth for an adaptive multigrain eigenvalue application. Unlike a bulk data transfer application, the eigenvalue application exhibits a sporadic traffic pattern typical of many parallel applications with compute-communicate phases. The eigenvalue application is interesting because it is representative of other grid applications, which allows us to better test the ability of the Wren bandwidth monitoring tool to passively monitor real application traffic.

The results of using the prototype of our Wren bandwidth monitoring tool to measure bandwidth for both the TCP bulk data transfer and adaptive multigrain eigenvalue application indicate the following:

- The Wren packet tracing facilities extend the functionality of the Web100 kernel and add little overhead to the kernel.
- The different techniques produce quantitatively similar results, indicating that we should be able to interchange the measurement techniques according to the information we have available, while keeping the particular technique used for the measurements transparent to the user.
- The Wren bandwidth monitoring tool is able to use brief periods of full network use captured by the kernel-level packet traces to calculate available bandwidth measurements even if an application does not generate continuous traffic.
- The Wren bandwidth monitoring tool is able to use short TCP probes, similar to tools like NWS, to measure available bandwidth, indicating that it is possible to inject such probes when there is insufficient application traffic to passively monitor.

The initial results obtained from our prototype of the Wren bandwidth monitoring tool illustrate the feasibility of combining passive measurements with active measurements in a single monitoring system. Our next step will be deploying these techniques on machines being used for a variety of grid applications so that we can apply our techniques in a production environment.

The remainder of this paper will describe the status of the Wren bandwidth monitoring tool. We will first review related work on collecting packet traces and probing for available bandwidth. Section 3 will describe our implementation and its integration into the Web100 kernel in more detail. In Section 4, we evaluate the overhead imposed by our kernel-level packet trace facility and present the results of using the Wren bandwidth monitoring tool to measure available bandwidth.

## 2. Related Work

In this section, we discuss application monitoring tools, bandwidth probing techniques, and how available bandwidth techniques can be applied to data captured by performance monitoring tools.

### 2.1. Application Monitoring Tools

Web100 is designed to monitor, diagnose, and tune TCP connections [12]. 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` filesystem. It 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. Web100's primary focus is on tuning TCP buffers, but it does collect TCP variables that can be used to calculate available bandwidth.

Other kernel-level tracing tools are MAGNeT [5] and the Linux Trace Toolkit (LTT) [21]. MAGNeT monitors traffic as it passes through each layer in the network protocol stack to help characterize application traffic patterns. LTT is often used to help debug synchronization problems in applications and to measure aspects of overall system performance because it collects information on general kernel events instead of detailed information about internal TCP connection variables and packet headers. These tools make use of shared memory, rather than the `/proc` filesystem, to pass logged events to the user-level.

Paradyn [16] is designed to monitor the performance of large scale parallel programs by dynamically instrumenting applications and automating the search process for bottlenecks. Throughout the duration of the application, Paradyn determines when instrumentation is needed and where the instrumentation should be located. This concept of triggering when and where monitoring occurs reduces the overhead of the monitoring system and could be applied to collecting kernel-level traces.

## 2.2. Bandwidth Probing Techniques

A simple approach to computing available bandwidth is to use the benchmark technique. This technique is similar to active probing because it calculates an application's overall throughput, or available bandwidth, by sending out a block of data from one end host to another, recording how long it takes to transfer the data, and then dividing the size of the data by the time it took to receive the data.

Another approach is the TCP window technique, which measures the available bandwidth of an application that is using the TCP protocol to send data. TCP tracks the congestion in the network by maintaining a congestion window size variable (`cwnd`), which specifies how much data of size maximum segment size (`MSS`) can be sent. 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 available bandwidth [13]. This technique is limited by how fast the congestion window opens and can only be applied to data collected on a host that is sending data.

Packet dispersion techniques can be used to measure the capacity [2–4, 9] or the available bandwidth [2, 7, 8] of a network path. These techniques send back-to-back packets

at a rate higher than the path can sustain and infer the capacity or available bandwidth from the spacing between the packets as they arrive at the receiving host. The difference between measuring capacity and available bandwidth is that when capacity is measured, filtering techniques are used to discard measurements in which traffic was queued between the packets at the bottleneck link. The basic premise to computing capacity and available bandwidth is the same: divide the size of the packets by the difference between the separation of the packets when they were sent and when they were received.

We recognize that there is still uncertainty in the community about the reliability of measuring available bandwidth using packet dispersion techniques. This is especially true in high bandwidth-delay environments where interrupt coalescing can cause timestamps to be increased by some amount up to the coalescing time from when packets actually arrive at the NIC and may also cause packets to arrive in bulk. We do not attempt to address those issues here, and we observe that because the passive packet trace approach relies on an application's own pairs, instead of generating them in back-to-back pairs or trains, producing available bandwidth measurements using packet traces will require more filtering. However, we are investigating methods that rely on other packet spacing, such as exponential, to our traces of application-generated trains [6, 17]. As packet dispersion techniques for measuring available bandwidth are perfected, and their limitations understood, the same results will be applicable to the passive packet traces.

## 3. Wren Implementation

As part of the Wren network measurement system, we are creating a bandwidth monitoring tool that extends the functionality of the Web100 kernel to incorporate kernel-level packet traces that collect information pertinent to monitoring available bandwidth. The Wren bandwidth monitoring tool requires instrumentation at only one end of the path and provides the flexibility of using existing TCP traffic or actively generating TCP traffic. 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.

In our implementation, we added two buffers to the Web100 kernel to collect information, such as sequence numbers and packet timestamps, from incoming and outgoing packets. In our tool, the user-level uses two new system calls to initiate the collection of packets, specify the connection to be monitored, pass a user-level memory reference location to the kernel, and stop the collection of packets. Once the packet trace is initiated, both hosts collect packets based on sequence and acknowledgment numbers. We use

this approach instead of tracing for a fixed period of time because long round trip times between hosts at each end of the connection could prevent the traces from gathering information about the same packets. Our implementation allows us to trigger the measurements exactly when we want them to occur and 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.

In the future, we will investigate whether we can determine when coalescing occurs and compensate by adjusting the timestamps or whether we will have to further modify the kernel to use the NIC clock to time-stamp packets.

### 3.1. User-level Implementation

We apply the available 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 could be transferred to another machine for processing, if necessary. If our user-level code is not incorporated into the application, a separate user-level process is run to collect the data from the kernel component of our system and apply the available bandwidth techniques. In the remainder of this section, we explain how our user-level code applies the benchmark, TCP window, and packet dispersion available bandwidth techniques.

As the TCP flow is generated, we record the elapsed time and the number of bytes sent and use these values to compute the benchmark technique's available bandwidth. The Wren bandwidth monitoring tool can use information collected in either the user-level or the kernel-level to compute available bandwidth. In either case, our user-level code computes the available bandwidth for the benchmark technique by dividing the number of bytes sent or received by the elapsed time.

To apply the TCP window technique, our tool uses the congestion window size and time-stamp values, recorded in the kernel of the sending host. The round trip time is calculated in the user-level code by taking the difference between the time-stamp of a data packet and the time-stamp of the corresponding ACK packet. Alternatively, TCP's own round trip time variable can be used. The available bandwidth is then computed by dividing the product of the congestion window size and maximum segment size by the round trip time.

When the Wren bandwidth monitoring tool is deployed on both end hosts, it implements the packet dispersion technique using packet time-stamps collected on both the sending and receiving host. The available bandwidth is calculated by dividing the size of the packets by the dispersion of the packet train, which is computed by taking the difference between time-stamps of when the packets are sent and are received. We filter the measurements to ensure

that queuing occurred somewhere along the path and that the rate of the packet train did not speed up along the path.

## 4. Results

We have tested the Wren bandwidth monitoring tool by running experiments on clusters and WANs using 100Mb and Gb interfaces. To characterize our tool with different workloads, we measured available bandwidth by passively collecting data from a bulk data transfer and an adaptive multigrain eigenvalue application. We use the bulk data transfer application to show the interoperability and limitations of the three available bandwidth techniques, while our eigensolver experiment demonstrates our tool's ability to use bursty application traffic to passively calculate available bandwidth. We also tested the Wren bandwidth monitoring tool using short TCP probes for comparison with other tools that implement active bandwidth measurements.

### 4.1. Overhead of Wren packet tracing

To get a sense of how much overhead our kernel modifications have imposed, we compared the overhead of our extensions to the Web100 2.4.20 kernel with an uninstrumented 2.4.20 kernel and a Web100 instrumentation of the 2.4.20 kernel when using both 100Mb and Gb Ethernet. We measured the overhead by running iperf for a 90 second interval between two 930 MHz Pentium III Linux machines with a Gb card installed in a 32bit/33MHz PCI slot. In Table 1, we present the bandwidths measured during the 90 second interval using the 100Mb and Gb interfaces. In this table, wren refers to our modified kernel with no packet tracing, wren (send-capt) is our modified kernel with tracing on the iperf server side, and wren (recv-capt) is our kernel with tracing on the iperf client side. Our results demonstrate that our modifications add less than 1% overhead to the Web100 kernel when no tracing occurs, less than 2.6% when traces are collected every 5 seconds on the server side, and less than 2.2% when traces are collected every 5 seconds on the client side. These results show that our tool adds minimal overhead to the system even when we are collecting packets.

### 4.2. Comparing Available Bandwidth Techniques

Figure 1 is an example of measuring bandwidth by applying benchmark, TCP window, and packet dispersion techniques to the first 64K of data sent in a bulk data transfer. This figure exhibits the classic behavior of each prediction technique. The TCP Window algorithm follows TCP's slow start and initial overshoot of the appropriate window size. The sender and receiver benchmark algorithms both

**Table 1: Overhead incurred while running iperf for a 90 second interval between two 930 MHz Pentium III machines.**

kernel	100Mb Ethernet bandwidth	Gb Ethernet bandwidth
stock 2.4.20	94.1 Mb	850 Mb
web100 2.4.20	94.1 Mb	833 Mb
wren 2.4.20	94.0 Mb	829 Mb
wren (serv-capt)	92.2 Mb	812 Mb
wren (recv-capt)	93.3 Mb	815 Mb

follow the amount of data sent according to the TCP Window. The packet dispersion technique provides its first result as soon as back-to-back packets are sent, and those results are immediately at the capacity of the uncongested Ethernet LAN.

Figure 2 shows the bandwidths calculated by collecting data from a bulk data transfer sent from an Athlon Linux machine at the College of William and Mary to a Pentium III Linux machine at Carnegie Mellon University. In this figure, we see that all of the techniques produce quantitatively similar bandwidth measurements during periods when the TCP window is fully opened. This similarity indicates that a production implementation could select the cheapest technique at runtime to produce measurements for a monitoring system.

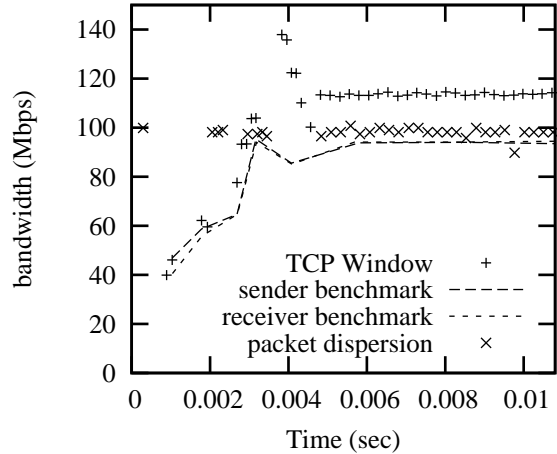
The bandwidths calculated by collecting data from a bulk data transfer sent from an Athlon Linux machine at the College of William and Mary to a Pentium III Linux machine at Lawrence Berkeley National Laboratory are presented in Figure 3. The first observation is that the benchmark and TCP window technique measurements illustrate the slow ramp-up and sawtooth patterns associated with TCP’s AIMD algorithm. Figure 3 also indicates that the benchmark and TCP window techniques are not measuring available bandwidth until just before a packet loss.

Figures 2 and 3 show that the packet dispersion technique is able to provide useful information about network performance as soon as packets are issued back-to-back. In both graphs, the packet dispersion technique measurements provide an upper bound for the available bandwidth.

Our results indicate that the benchmark and TCP window techniques only measure available bandwidth when the TCP window is fully open, but that the packet dispersion technique, with proper filtering, can provide an upper bound for the available bandwidth regardless of the window.

### 4.3. Monitoring Bursty Application Traffic

In addition to testing the Wren bandwidth monitoring tool using a bulk data transfer application, we also per-

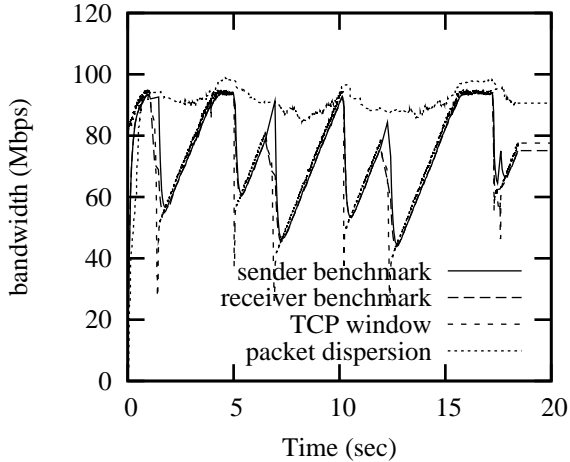


**Figure 1: Using information collected from the Wren packet trace facility, we apply the TCP Window, sender-receiver packet dispersion, and benchmark techniques to measure bandwidth on a 100Mb Ethernet LAN.**

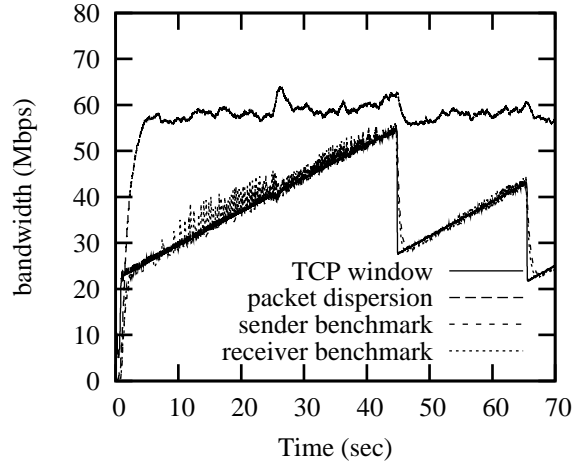
formed experiments with an adaptive multigrain eigenvalue application, which is designed to find the extreme eigenvalues of large, sparse matrices [15]. This application implements a latency-tolerant algorithm that is significantly more efficient when run on clusters than traditional tightly-coupled algorithms. This solver is comprised of a projection phase with local computation and a correction phase with communication within clusters. In addition, communication occurs between clusters before and after the correction phase. Within each phase, 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, therefore, an ideal choice to evaluate the performance of our bandwidth measurement techniques when running significant, non-trivial applications with interesting communication patterns.

Figures 4 and 5 show the results of running the eigensolver on a cluster of four Pentium III Linux machines linked together by 100Mb connections. In Figure 6, we present bandwidth calculations made from observing the eigensolver run on an Athlon Linux machine at the College of William and Mary and a Pentium II Linux machine at LBL. For our experiments, we configured the system so that the matrix is equally partitioned between the processors.

The first observation is that the traffic exhibits the compute-communicate phases typical of many parallel applications. Furthermore, the benchmark results in Figure 5 demonstrate that even during the communication phase, the



**Figure 2: Available bandwidth measurements obtained from applying the benchmark, TCP window, and packet dispersion techniques to data collected from a WAN bulk data transfer between William and Mary and CMU show that all techniques produce similar bandwidth estimates when the congestion window is fully open.**



**Figure 3: The packet dispersion technique is able to measure available bandwidth on this high bandwidth-delay product link between William and Mary and LBL, while the benchmark and TCP Window techniques succumb to the ramp-up and sawtooth behavior associated with TCP’s AIMD algorithm.**

application is not sending a continuous stream of data, but is bursty between each pair of machines. (In this experiment, each node is communicating with three others, and these results are for the communication between a single pair.)

Figure 6 shows that the TCP window technique is unable to measure available bandwidth because the amount of data being sent during the projection phase is not enough to fully open the TCP window. However, the packet dispersion technique, as seen in Figures 4 and 6, can be applied to the eigenvalue traffic to produce useful measurements.

Our results prove that the Wren bandwidth monitoring tool is able to capture and use brief periods of full network communication to accurately measure available bandwidth. As long as these bursts occur as often as measurements are desired, the passive technique can provide the necessary information to calculate available bandwidth.

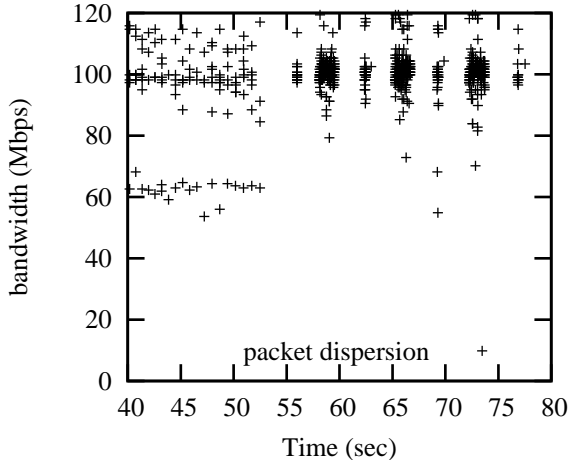
#### 4.4. Actively probing for bandwidth

When there is insufficient application traffic present to monitor available bandwidth at a desired frequency, we need to actively inject and monitor our own traffic into the network. Because we want to be able to alternate between active and passive measurements, the Wren bandwidth monitoring tool must also be capable of obtaining accurate available bandwidth measurements from short probes. To demonstrate our tool’s ability to actively probe the network when an application is not generating sufficient

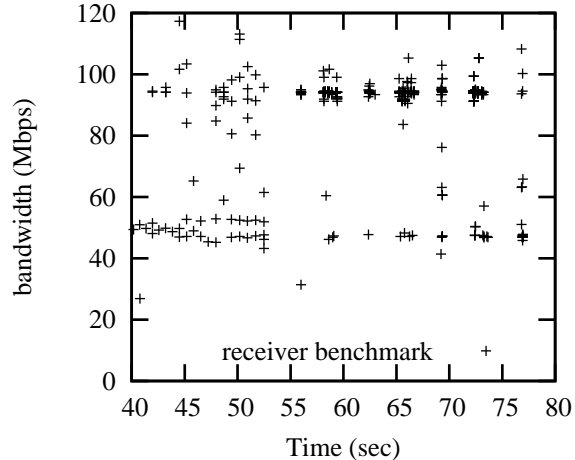
traffic, we applied the available bandwidth techniques to information collected from short, 64K, TCP probes. Swamy and Wolski have also studied using short TCP probes to predict the performance of longer TCP probes in NWS [19] by correlating short and long TCP probes to build a mapping function that can be used in the future. They observe that even when the short TCP probes never fully open their congestion window, their performance is still capable of providing accurate predictions of long-term TCP data transfers using their correlation function. These results are encouraging because they indicate that we can use short, 64K, TCP probes instead of longer, more intrusive probes to actively measure available bandwidth when the passive approach is not viable.

Figures 1 and 7 show the measurements made by applying all techniques to short probes sent on a 100Mb cluster and across a 100Mb WAN, respectively. In these graphs, we see that the TCP window technique is unable to produce valid measurements from the short TCP probes because the ramp-up phase is just beginning and the TCP window never fully opens. However, the packet dispersion technique is able to accurately produce an upper bound for the available bandwidth because the probes send out back-to-back packets immediately.

Our results in Figures 1 and 7 also show that we can use active probe traffic to monitor available bandwidth measurements at short intervals, which indicates that we should be able to transparently switch to active, short measurements when we are unable to passively use existing appli-



**Figure 4: The packet dispersion technique is able to measure available bandwidth on a 100Mb cluster using bursty traffic of an adaptive eigensolver.**



**Figure 5: Receiver benchmark measurements show the bursty communication pattern of the eigenvalue application between two nodes on a 100Mb cluster.**

cation traffic.

## 5. Conclusion

In this paper, we demonstrate that the Wren bandwidth monitoring tool can passively measure available bandwidth from both continuous bulk data transfer traffic and from irregular, bursty grid application traffic. We have evaluated the overhead of our solution and found it to have a negligible impact on application performance while collecting traces. The integration with the Web100 software provides a convenient interface to our tracing facility through the existing Web100 interface.

We have used our kernel extensions to implement the benchmark, TCP window, and packet dispersion available bandwidth calculations. We have evaluated the effectiveness of these measurements on both bulk data transfer and a distributed computing application with bursty communication. Our results show that all of the techniques are useful during bulk data transfer—an important observation because it allows us to choose the cheapest technique to gather bandwidth observations when an application is performing bulk transfer. Capturing precise measurements of available bandwidth for the bursty application is more challenging; however, we find the packet dispersion results encouraging because they indicate useful measurements can be made after only a few back-to-back packets have been sent. Sophisticated filters will be needed to make effective use of these measurements, but our initial observations are promising.

We have shown that we can supplement measurements obtained by passively monitoring existing application traffic with measurements obtained by sending and monitoring

short TCP probes. The next step is to determine when an application is not generating sufficient traffic and inject probes as needed to keep available bandwidth measurements fresh. In this way, sufficient traffic can be quantified on a per-application basis, matching the tolerance level that an application has for stale measurements.

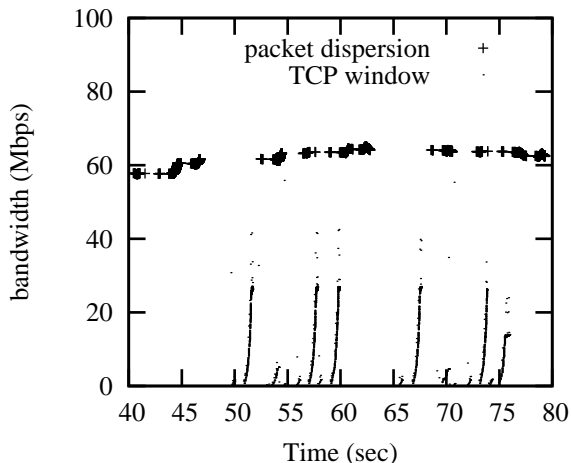
Our evaluation environments have covered a 100Mb cluster and WANs, as well as early tests on a Gb cluster. The results are promising in each of these environments, and we are in the process of adding Gb WAN experiments and evaluating our techniques against a wider range of congestion levels to more fully validate our approach.

## 6. Acknowledgments

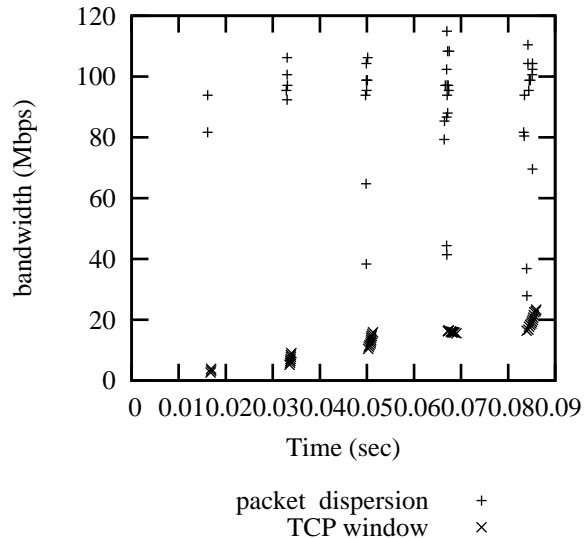
We thank those who allowed us to use their hosts as platforms for our experiments: Nikos Chrisochoides’ PES cluster (College of William and Mary), Nancy Miller (CMU), and Brian Tierney (LBL). Also thanks to Andriy Kot, Jim McCombs and Claire O’Shea.

## References

- [1] J. Bolliger, T. Gross, and U. Hengartner. Bandwidth modelling for network-aware applications. In *Proceedings of Infocomm’99*, 1999.
- [2] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27 and 28, October 1996.
- [3] C. Dovrolis, P. Ramanathan, and D. Moore. Packet dispersion techniques and capacity estimation. *IEEE/ACM Transactions in Networking*. submitted.



**Figure 6: The TCP window technique is not able to measure available bandwidth because the congestion window is not fully open, but the packet dispersion technique can be applied to the multi-grain eigensolver traffic on WAN to monitor available bandwidth.**



**Figure 7: Packet dispersion and TCP window bandwidth measurements of a short TCP probe sent on a WAN. The packet dispersion technique is able to measure available bandwidth, while the TCP window technique shows ramp up behavior.**

- [4] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure? In *INFOCOM01*, 2001.
- [5] M. K. Gardner, W. chun Feng, and J. R. Hay. Monitoring Protocol Traffic with a MAGNeT. In *Passive and Active Measurement Workshop*, 2002.
- [6] G. He and J. C. Hou. On Exploiting Long Range Dependence of Network Traffic in Measuring Cross Traffic on an End-to-end Basis. In *IEEE Infocom*, 2003.
- [7] 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.
- [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. Combining active and passive network measurements to build scalable monitoring systems on the grid. *Performance Evaluation Review*, 30(4):19–26, March 2003.
- [11] B. B. Lowekamp, N. Miller, R. Karrer, T. Gross, and P. Steenkiste. Design, implementation, and evaluation of the remos network monitoring system. *Journal of Grid Computing*, 1(1), 2003.
- [12] M. Mathis. "Web100" <http://www.web100.org>, 2000.
- [13] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.
- [14] W. Matthews and L. Cottrell. The pinger project: Active internet performance monitoring. *IEEE Communications Magazine*, pages 130–137, May 2000.
- [15] J. R. McCombs and A. Stathopoulos. Multigrain parallelism for eigenvalue computations on networks of clusters. In *Proceedings of the Eleventh IEEE International Symposium On High Performance Distributed Computing*, pages 143–149, Los Alamitos, California, July 2002. IEEE.
- [16] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithadadam, and T. Newhall. The paradyn parallel performance measurement tools. In *IEEE Special issue on performance evaluation tools for parallel and distributed computer systems.*, 1995.
- [17] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, and R. Baraniuk. Multifractal cross-traffic estimation, 2000.
- [18] M. Stemm, S. Seshan, and R. H. Katz. A Network Measurement Architecture for Adaptive Applications. In *Proceedings of INFOCOM 2000*, pages 285–294, March 2000.
- [19] M. Swamy and R. Wolski. Multivariate resource performance forecasting in the network weather service. In *IEEE/ACM SC2002 Conference*, November 2002.
- [20] 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)*, pages 316–25, August 1997.
- [21] K. Yaghmour and M. R. Dagenais. Measuring and Characterizing System Behavior Using Kernel-Level Event Logging. In *2000 USENIX Annual Technical Conference*, 2000.