

# MONITORING AVAILABLE BANDWIDTH OF UNDERLYING GRID NETWORKS TO IMPROVE APPLICATION PERFORMANCE

Marcia Zangrilli

Bruce B. Lowekamp, advisor  
Department of Computer Science  
College of William and Mary

## Abstract

*Grids are becoming increasingly important to research and engineering institutions because they provide a secure way of coupling various geographically distributed resources into a single high-performance environment. Because the performance of distributed systems is intrinsically linked to the performance of the network, applications that have knowledge of the available bandwidth can adapt to changing network conditions and optimize their performance. While several algorithms have been created to actively measure the end-to-end available bandwidth of a network path, they require instrumentation at both ends of the path, and the traffic injected by these algorithms may affect the performance of other applications on the path. As part of the Wren monitoring system, we are developing techniques that use passive traces of existing traffic instead of actively probing the path to measure the available bandwidth. Our technique is designed to use either two-sided or one-sided packet traces, which gives the user flexibility in how our tool is deployed. We have completed a packet trace facility and designed new passive bandwidth algorithms. Our results evaluate the effectiveness of these new algorithms in diverse environments and demonstrate how applications can transparently integrate with the Wren monitoring system to optimize their performance.*

## 1 Introduction

Grids allow users to harness the power of many, often geographically distributed, resources. These grid infrastructures are valuable to chemists, physicists, engineers, and other scientists running large-scale simulations or other high-performance applications. These types of applications are often computationally or memory intensive, which may necessitate that data is distributed between many machines. Computational grids, such as the NASA Information Power Grid<sup>1</sup>, are being developed to provide users with a secure way of sharing data and running intensive simulations. These grids provide more computational power and storage facilities, allowing scientists to tackle larger problems.

While switching to grid computing potentially provides more computational power and storage facilities, the evolution to grid computing introduces new problems for scientific applications because of the heterogeneity and dynamic nature of the grid resources. Grid environments can consist of machines with drastically different architectures connected by an unbalanced communication network. In addition, grids are dynamic in nature; resources can come and go. Network conditions will reflect the amount of other traffic competing for the network, and as a result bandwidth may vary greatly in time. It is this dynamic nature of the grid that makes it challenging for an application to efficiently utilize all resources and maximize its performance.

Grid applications require timely network measurements of the available bandwidth so that they can adapt to changing network conditions and make efficient use of resources. Available bandwidth describes what portion of the path is currently unused by other competing traffic. More precisely, available bandwidth is determined by subtracting the utilization from the capacity of the network path<sup>12:17</sup>. For available bandwidth measurements to be widely used and accepted, one must minimize both the effort needed to acquire available bandwidth measurements and the intrusiveness of the measurement process. The ideal solution will provide available bandwidth information to the application without having to modify the application to make the measurements and without affecting the performance of other applications on the network.

Tools that measure the available bandwidth exist, but they obtain measurements by actively probing the network. While these tools often provide accurate measurements, they still may adversely affect the performance of other applications on the network. Furthermore, these applications require instrumentation on both ends of the path, which may not always be possible.

As part of the Watching Resources from the Edge of the Network (Wren) system, we are developing a monitoring tool that uses passive traces of application traffic instead of actively generating the traffic used to measure available bandwidth. By monitoring the traffic that an application generates or receives, we can calculate the available bandwidth on the path even when the application does not generate enough traffic to saturate the path. Wren uses kernel-level instrumentation to passively trace application traffic and then calculates the available bandwidth in the user-level using information found in those traces. Wren measurements can be used to modify an application's adaptation strategy at runtime and can also be stored by a monitoring system for future use by applications.

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 our initial research are

- a system that captures packet traces unobtrusively,
- new algorithms for applying the principles of active probing techniques to passive traces,
- evaluation of our passive, one-sided available bandwidth algorithm when applied to several types of application traffic on paths with varying amounts of cross traffic, and
- demonstration of how Wren measurements can be used to improve application performance.

The remainder of this paper will describe the status of the Wren bandwidth monitoring tool. We will first review existing available bandwidth algorithms and related work. In Section 4, we describe our packet trace implementation and the new passive, available bandwidth algorithms we developed. In Section 5, we present the results of using Wren to monitor available bandwidth.

## 2 Existing Available Bandwidth Tools

Self-induced congestion (SIC) is a common technique used by active tools to measure available bandwidth. The basic principle of the SIC technique is that if packets are sent at a rate larger than the available bandwidth, the queuing delays will have an increasing trend, and the rate the packets arrive at the receiver will be less than the sending rate. If the one-way delays are not increasing and the rate the packets arrive is the same as the sending rate of the packets, then the available bandwidth is less than or equal to the sending rate. Tools that utilize this concept probe the network path for the largest sending rate that does not result in queuing delays with an increasing trend because this sending rate reflects the available bandwidth of the path.

Tools that use principles similar to self-induced congestion include Netest<sup>11</sup>, Pathchirp<sup>18</sup>, Pathload<sup>8</sup>, PTR<sup>6</sup>, and TOPP<sup>15</sup>.

### 3 Single-sided Bandwidth Measurements

Proposed improvements to the TCP protocol have set a precedent for measuring available bandwidth on a single end host. The emphasis has been on improving the performance of the TCP protocol using information about the path bandwidth to adjust variables in the slow start algorithm to achieve a faster transition into the congestion avoidance phase of TCP.

Hoe<sup>5</sup> and Partridge et al.<sup>16</sup> determine the difference between the separation of a group of packets as they leave the machine and the separation of the corresponding ACKs of those packets when they arrive. This difference is used to infer how much bandwidth is available by dividing the difference by the size of the segments that generated the ACKs. Their formula may actually calculate a metric called the average dispersion rate. Dovrolis et al. have shown that this dispersion rate is a lower bound on capacity and an upper bound of the available bandwidth<sup>2</sup>.

More recently, Paced Start (PaST)<sup>7</sup> proposed incorporating the self-induced congestion principles used by the PTR tool to reduce the amount of time taken before transitioning into the congestion avoidance phase of TCP. PaST groups outgoing packets into trains, using the SIC principle to adjust the gaps between the packets. The returning ACKs are then used to measure the available bandwidth on the path. However, the comparisons made by the PaST approach may falsely indicate that the departure rate is larger than the available bandwidth<sup>9</sup>.

A project that is similar to Wren is Inline measurement TCP (ImTCP). ImTCP integrates an active self-induced congestion algorithm into a TCP stack, waiting until the congestion window has opened large enough to send an appropriate length train and then delaying packet transmissions until enough packets are queued to generate a precisely spaced train<sup>13</sup>. Wren avoids modifying the TCP sending algorithm and, in particular, delaying

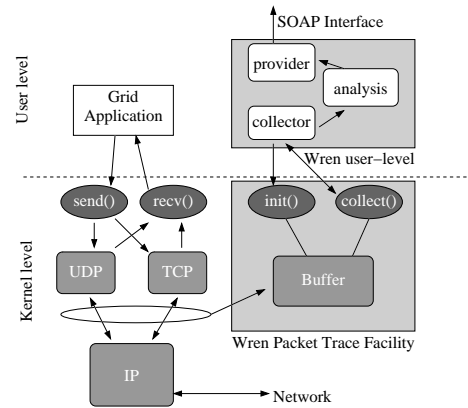


Figure 1: Wren architecture.

packet transmission.

The challenge Wren addresses when compared to ImTCP is that Wren must select the data naturally available in the TCP flow. Although Wren has less control over the trains and selects shorter trains than would deliberately be generated by active probing, over time the burstiness of the TCP process produces many trains at a variety of rates<sup>19;10</sup>, thus allowing bandwidth measurements to be made. There are elements in common with TCP Vegas, Westwood, and FastTCP, but those approaches deliberately increase the congestion window until one-way delay increases, whereas we do not require the congestion window to expand until long-term congestion is observed and can detect congestion using bursts at slower average sending rates.

### 4 Wren Bandwidth Monitoring Tool

The Wren bandwidth monitoring system provides the infrastructure for our passive implementation of available bandwidth techniques. Wren is separated into a kernel-level packet trace component and a user-level analysis component, as shown in Figure 1. The kernel-level packet trace facility is responsible for gathering information associated with incoming and outgoing packets. Once the trace has been collected, it can be analyzed at the user-level or forwarded to a separate machine for analysis.

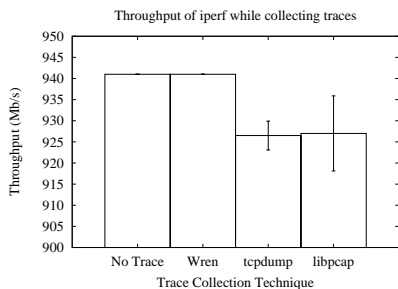


Figure 2: Throughput achieved by iperf with no packet tracing, using Wren, using tcpdump, and using libpcap

## 4.1 Wren Packet Trace Facility

The size of the packets, the time the packets leave the sender, and the time the packets arrive at the receiver are needed to apply the self-induced congestion available bandwidth technique. We use the Wren packet trace facility to collect traces of TCP traffic from applications. Wren timestamps packets in the kernel as they arrive or leave, collects other protocol specific information, and passes the trace to the user-level for analysis. 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. For high bandwidth-delay product networks, the tool can also ensure that it collects data transmission and ACK receipt logs of the same range of sequence numbers.

The key benefit of the Wren packet trace facility is its use of kernel-level timestamps. Wren timestamps outgoing packets in the kernel immediately before they are handed to the layer-2 device driver and similarly timestamps incoming packets as soon as they are passed to the TCP code. This additional accuracy, combined with the efficiency of collecting only the information needed while reading from the buffer at regular intervals, provides the traces required for accurate monitoring without disturbing the running application.

### 4.1.1 Packet Tracing Efficiency

The principle alternative to a system like Wren is using the Berkeley packet filter<sup>14</sup> to capture TCP traffic. The

most straightforward approach is to use *tcpdump* to capture packet header traces and collect the data from the timestamps and packet headers. We have implemented this approach, as well as a direct *libpcap* implementation that extracts only the needed fields for later analysis.

We present the results of experimenting with packet tracing techniques between two 2.8GHz Hyperthreaded P4s running Linux 2.4.29 with 1GB of RAM and an Intel CSA (non-PCI) gigabit Ethernet NIC. The nodes were connected using a Cisco 3750 gigabit switch. Here we analyze only the cost of capturing packets and saving them to the hard drive for offline analysis. Our experience has indicated that performing the actual analysis is relatively inexpensive once the packet trace is in memory, and that cost is constant regardless of the technique with which the data are captured.

At 100Mbps speeds, all three approaches can collect 100% of traffic at full speed with minimal affect on application performance. Therefore, we focus our efficiency analysis on Gbps traffic. Figure 2 presents the throughput achieved by iperf while using each of the three capture techniques, as well as without any packet capture. The Wren approach achieves the same throughput as the untraced connection; however the two packet-filter approaches both affect throughput.

### 4.1.2 Packet tracing accuracy

Beyond efficiency, a packet trace must also be correct. The two aspects of correctness in a packet trace are completeness and accuracy. For completeness, during the throughput experiments both the Wren and libpcap implementations missed approximately 0.3% of the packets, while the tcpdump implementation missed about 1.6%. None of these loss rates are large enough to affect our SIC analysis.

The Wren packet trace facility allows an interesting tradeoff between completeness and ignoring data for improving efficiency. The Wren approach relies on storing data in a circular buffer in the kernel, which is only copied to user-space at the request of an application. This is fundamentally different than the queue approach of

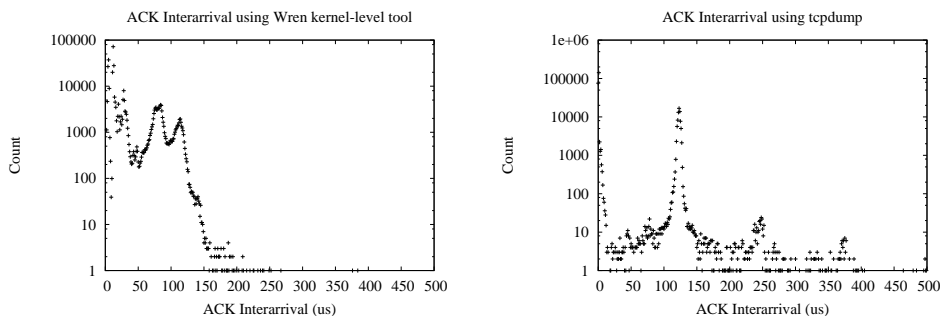


Figure 3: ACK interarrival times for a uncongested gigabit Ethernet path. On the left are the interarrival times recorded by Wren at kernel-level. On the right are the interarrival times recorded by tcpdump at user-level.

the packet-filter. In particular, it allows the collection application to avoid imposing any measurable load except when it runs. Furthermore, on a high-bandwidth path such as a Gbps network, collecting and analyzing all possible packet trains may produce more data than is necessary. In that case, the collecting application may simply opt to collect every few seconds, thus avoiding the overhead of processing data continuously. However, for the efficiency results presented in this paper, Wren has been configured to collect data every 0.5s, which is sufficient to collect all of the packets using our 60,000 packet data buffer.

The most interesting result from comparing the Wren and the packet-filter approaches is the accuracy of the timestamps they produce. Wren uses the CPU’s cycle counter to timestamp the packet immediately before it is queued on the NIC or, for incoming packets, as soon as it is received by the TCP stack. The packet-filter architecture, however, was not designed with precise timestamping in mind. Although the architecture carefully avoids unnecessary copying in the kernel, the packet is queued for the packet-filter without any modifications or annotations. The timestamp is added after the packet is read in by the libpcap code<sup>1</sup> (which we have both used directly and used indirectly through tcpdump’s implementation). The result is that the timestamps more closely reflect the processing and queuing delays of application-level code than the actual packet arrival times. Casual observation

<sup>1</sup>libpcap-0.8.3/pcap-linux.c:626

of the ACK interarrival times indicates that tcpdump typically reports two or three ACKs arriving simultaneously, followed by a large separation before the next group of ACKs. Wren observes the ACKs arriving at much more regular intervals. Figure 3 presents histograms of the interarrival times for the two techniques. There is clearly a significant loss of information by delaying the timestamping of the packets until they reach user-level.

## 4.2 Wren Algorithm Implementation

We apply the self-induced congestion principles to passive traces of application traffic. Instead of using one-way delays of packets, our approach analyzes the trends in the RTTs of packets. One of the primary advantages of our approach is that we only have to have instrumentation on the sender side of the TCP connection. We look for trains that have a non-increasing trend in RTTs because these trains did not incur queuing on the path; the rate at which these trains were sent represents the lower bound of the current available bandwidth.

The first step in our one-sided algorithm is to group packets into trains. We look at the relationship between the inter-departure times of sequential data packets. If inter-departure times of successive pairs are similar, then the packets are departing the machine at approximately the same rate. Let  $\Delta_i$  be the inter-departure time between each successive pair of packets  $i$  and  $i + 1$  in the train. To identify a well-formed train, each  $\Delta_i$  must satisfy the

requirement that  $\min_i(\log(\Delta_i)) > \max_j(\log(\Delta_j)) - \alpha$ , essentially requiring consistent spacing between the packets. The parameter  $\alpha$  is used to tune how consistent the spacing between packets must be for those packets to form a train. For these experiments, we accepted trains where  $\alpha = 1$ . For use with our SIC algorithm we impose a minimum length of 7 packets for valid trains and select maximal length trains subject to the grouping requirement above.

Once the trains are formed, the algorithm enters the processing phase. For each train that is formed, we calculate the the initial sending rate (ISR) by dividing the total number of bits in the train by the difference between the end time and the start time. The start time of the train refers to the time the first data packet in the train departs the machine, and the end time of the train specifies the time that the last data packet in the train departs the machine. The ISR of each train is compared to the ACK return rate. The ACK return rate is calculated by dividing the total number of bits in the train by the difference between the time the first ACK for the train and the last ACK for the train arrive back at the sending machine. Our algorithm only uses packets that have an explicit ACK packet.

Our algorithm compares the ISR of the train with the ACK return rate of the train and checks that they are similar rates; the ACK return rate must be  $\pm 0.5\%$  of the ISR rate. At first glance, this seems counterintuitive, as the goal of SIC is to detect increases in delays. However, this is actually an important step in processing the data. An active SIC implementation carefully probes around the available bandwidth, sending at rates slightly above and below available bandwidth, observing relatively small increases in one-way delay. Our analysis of TCP traffic has indicated that the results of trains with small differences between the ISR and ACK return rate are the most accurate. In particular, because we are ultimately more interested in trains that do not have an increase in ACK return rate, i.e. trains sent below the available bandwidth, discarding the high-error trains that are sent significantly above the available bandwidth is an ideal solution. Fur-

ther experimentation with the best value for this parameter is required.

The next step in processing a train is to use a pairwise comparison test to determine the trend in the RTTs of the packets in that train. If  $\forall i : RTT_i < RTT_{i+1}$  then the train has an increasing trend, indicating its ISR was higher than the available bandwidth. Otherwise, the train trend is labeled as non-increasing. If the train has a non-increasing trend and the ISR is similar in value to the ACK return rate, we know that the train ISR did not cause queuing on the path. Therefore, we report the ISR as a lower-bound available bandwidth measurement.

## 5 Results

We have evaluated our Wren algorithm using both a controlled-load/controlled-latency testbed environment and a real WAN environment. We set up our testbed so that a single link was shared by multiple congestion generators and sinks, as well as the monitored application traffic flow. We used both constant bit rate (CBR) and on/off bursty traffic generators to create congestion in the network.

As validating experimental results on a WAN is difficult due to the inability to confirm the actual available bandwidth on the WAN during the experiment, much of our analysis is based on controlled-load testbed experiments. We simulated a WAN environment on our testbed by using Nistnet to create longer latencies for the congestion and monitored application traffic.

### 5.1 Measuring CBR Congestion

For this experiment, one iperf generated 30 Mbps of uniform (constant bit rate, CBR) cross traffic for the first 20 seconds and the second iperf generated 60 Mbps of cross traffic between 20 seconds and 40 seconds. After 40 seconds, there is no cross traffic on the path. Therefore, we know that there was 70 Mbps available from time 0–20 seconds, 40 Mbps available from time 20–40 seconds, and 100 Mbps available for time 40–45 seconds.

We monitored application traffic that sent 20 200KB

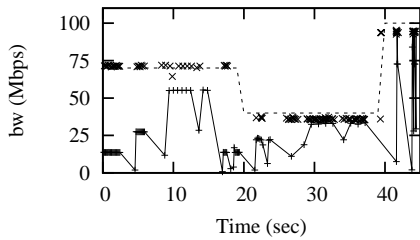


Figure 4: Wren measurements ( $\times$ ) reflect changes in available bandwidth (---) even when the monitored application's throughput (+) does not consume all of the available bandwidth.

messages with .1 second inter-message spacings, paused 2 seconds, 10 500KB messages with .1 second inter-message spacings, paused 2 seconds, and then sent 10 4MB messages with .1 second inter-message spacings. This pattern was repeated twice followed by 500KB messages sent with random inter-message spacings.

In the first 40 seconds of Figure 4, we see that the throughput of the traffic generator varies according to the size of message being sent. The last 5 seconds of this graph show that the throughput of the generator also depends on the inter-message spacings. Figure 4 shows that our algorithm produces accurate available bandwidth measurements even when the throughput of the application we are monitoring is not saturating the available bandwidth. This figure also demonstrates that our algorithm is able to correctly detect changes in available bandwidth.

Even though our TCP application rarely saturates the available bandwidth, the Wren algorithm still correctly identifies the available bandwidth as it changes.

## 5.2 On/Off Bursty Cross Traffic

We created congestion in the network by running on/off bursty UDP traffic generators from multiple machines at one end of the network path. We monitored traffic from an application that sent 500KB messages with .1 second inter-message spacing. We used a modified snmpdelta to obtain the exact available bandwidth of the congested

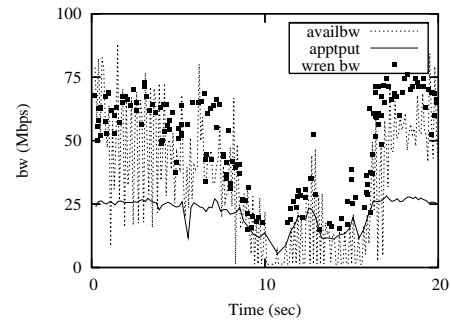


Figure 5: Wren measurements ( $\blacksquare$ ) from monitoring traffic (—) on a path with bursty on/off UDP congestion. SNMP-monitored available bandwidth is shown (---).

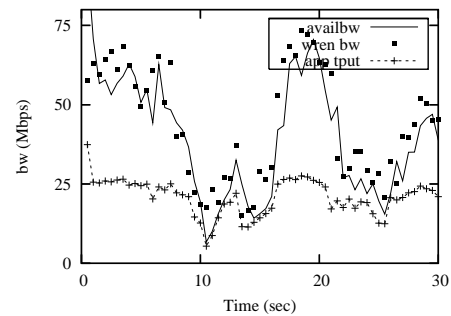


Figure 6: Wren measurements ( $\blacksquare$ ), monitored application throughput (+), and available bandwidth (—) are averaged over .5 second intervals for this trace with on/off bursty UDP congestion.

link on testbed at .1 second intervals.

Figure 5 shows how the Wren measurements ( $\blacksquare$ ) detect changes in the available bandwidth trends of the bursty cross traffic. For clarity, we graphed .5 second averages of the Wren measurements, available bandwidth, and monitored application throughput (+) in Figure 6.

We compared Wren's bandwidth measurements with Pathload, a tool that actively probes the network for available bandwidth. Figure 7 shows that running Pathload produces similar results to using our Wren algorithm on paths with on/off bursty UDP congestion. Pathload reports fewer results because it sends multiple fleets to establish a range for the available bandwidth. A production implementation of Wren might include

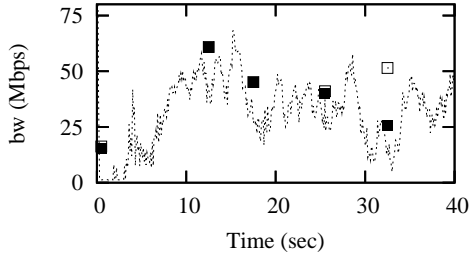


Figure 7: Pathload lower (■) and upper (□) bound available bandwidth measurements of path with on/off bursty UDP congestion.

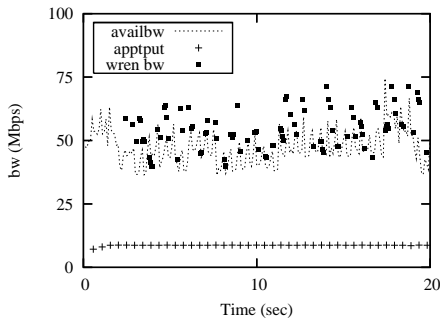


Figure 8: Wren measurements (■) from monitoring application traffic (+) on a simulated WAN. The available bandwidth (- - -) is created by on/off TCP generators.

similar filtering to establish lower and upper bounds, thus also reducing the reported number of observations. These results show that our Wren algorithm can measure available bandwidth as well as active tools by passively monitoring existing application traffic on a single end host. Even though the application TCP stream never utilizes more than 25Mbps and never saturates the available bandwidth except when it drops below that point, the Wren algorithm reports a range of available bandwidths that follows the true available bandwidth as it rapidly changes.

### 5.3 Testbed Simulated WAN Experiment

Because determining the actual available bandwidth on a WAN is difficult, we simulated a WAN environment

using Nistnet to increase the latencies that the cross traffic and monitored application traffic experienced on our testbed. We used on/off TCP traffic generators to create congestion on the path, with Nistnet emulating latencies ranging from 20 to 100ms and bandwidths from 3 to 25Mbps. The application traffic that was monitored sent 700K messages with .1 second inter-message spacing, with Nistnet adding a 50ms RTT to that path. SNMP was used to poll the congested link to measure the actual available bandwidth.

Figure 8 demonstrates how the Wren algorithm can measure the available bandwidth of larger latency paths with variable cross traffic. Due to the higher RTT, our application throughput was lower than in previous graphs, but the algorithm still correctly approximates the available bandwidth.

## 6 Applications Using Wren Measurements

We demonstrate how available bandwidth measurements can be used to improve application performance by integrating our Wren measurements with Virtuoso<sup>20,3</sup>, a virtual machine distributed computing system. In Virtuoso, the virtual machines are interconnected with VNET, a virtual overlay network. The VTTIF (virtual traffic and topology inference framework) component observes every packet sent by a VM and infers from this traffic a global communication topology and traffic load matrix among a collection of VMs. Wren uses the traffic generated by VNET to monitor the underlying network and makes its measurements available to Virtuoso’s adaptation framework.

To validate the combination of Wren monitoring an application using Virtuoso’s VNET, we ran a simple BSP-style communication pattern generator across the Abilene network from W&M to Northwestern University. Figure 9 shows the results of this experiment, with the throughput achieved by the application during its bursty communication phase and Wren’s available bandwidth observations. Although the application never achieved significant levels of throughput, Wren was able to measure the available bandwidth. Validating these re-

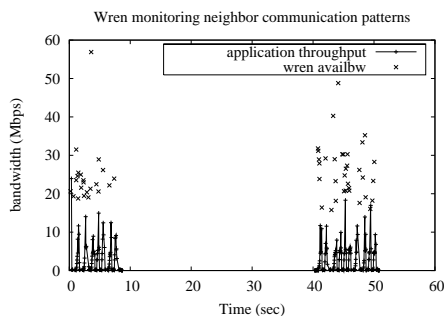


Figure 9: Wren observing a neighbor communication pattern sending 200K messages within Virtuoso’s VNET.

sults across a WAN is difficult, but iperf achieved approximately 24Mbps throughput when run following this experiment, which is in line with our expectations based on Wren’s observations and the large number of connections sharing W&M’s 150Mbps Abilene connection.

The adaptation component of Virtuoso uses both Wren and VTTIF measurements to choose a configuration that maximizes the performance of the application running inside the VMs. Our results have shown that Wren measurements can be used by Virtuoso to help determine the best virtual machine (VM) to VNET mapping and topology for an application. More information about the adaptation strategies used by Virtuoso and the performance benefits gained by incorporating Wren measurements is available<sup>4</sup>.

## 7 Conclusion

We have described how Wren can be used to monitor available bandwidth by passively observing applications’ traffic, even when those applications do not generate enough traffic to saturate the network. We have shown that the Wren packet trace facility is both efficient and accurate enough to monitor traffic on gigabit networks. The Wren analysis algorithm selects well-formed trains from existing application traffic and uses those trains to infer the available bandwidth. We have experimentally evaluated our approach using a variety of traffic types, including fluid and bursty competing traffic on LAN and

WAN networks. Finally, we have demonstrated how Wren measurements can be integrated with Virtuoso to optimize the performance of real applications.

## References

- [1] NASA Information Power Grid (IPG) <http://www.ipg.nasa.gov/>.
- [2] DOVROLIS, C., RAMANATHAN, P., AND MOORE, D. Packet dispersion techniques and a capacity estimation methodology. *IEEE/ACM Transactions in Networking* (2004).
- [3] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003).
- [4] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A., DINDA, P. A., AND LOWEKAMP, B. B. Free Network Measurement for Adaptive Virtualized Distributed Computin. In *IPDPS* (2006).
- [5] HOE, J. C. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *ACM SIGCOMM* (1996), vol. 26, pp. 270–280.
- [6] HU, N., AND STEENKISTE, P. Evaluation and Characterization of Available Bandwidth Techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling* (2003).
- [7] HU, N., AND STEENKISTE, P. Improving TCP Startup Performance using Active Measurements: Algorithm and Evaluation. In *International Conference on Network Protocols (ICNP)* (2003).
- [8] JAIN, M., AND DOVROLIS, C. Pathload: a Measurement Tool for End-to-end Available Bandwidth. In *Proceedings of the 3rd Passive and Active Measurements Workshop* (March 2002).

- [9] JAIN, M., AND DOVROLIS, C. Ten Fallacies and Pitfalls on End-to-End Available Bandwidth Estimation. In *IMC* (2004).
- [10] JIANG, H., AND DOVROLIS, C. Why is the internet traffic bursty in short time scales? In *SIGMETRICS 2005* (August 2005), ACM.
- [11] JIN, G., AND TIERNEY, B. Netest: A Tool to Measure the Maximum Burst Size, Available Bandwidth and achievable Throughput. In *International Conference on Information Technology Research and Education* (2003).
- [12] LOWEKAMP, B. B., TIERNEY, B., COTTRELL, L., HUGHES-JONES, R., KIELMANN, T., AND SWANY, M. Enabling Network Measurement Portability Through a Hierarchy of Characteristics. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)* (2003).
- [13] MAN, C. L. T., HASEGAWA, G., AND MURATA, M. A Merged Inline Measurement Method for Capacity and Available Bandwidth. In *Passive and Active Measurement Workshop* (2005).
- [14] MCCANNE, S., AND JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX Winter* (1993).
- [15] MELANDER, B., BJORKMAN, M., AND GUNNINGBERG, P. A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks. In *Global Internet Symposium* (2000).
- [16] PARTRIDGE, C., ROCKWELL, D., ALLMAN, M., KRISHNAN, R., AND STERBENZ, J. A Swifter Start for TCP. Tech. Rep. BBN-TR-8339, BBN Technologies, March 2002.
- [17] PRASAD, R., MURRAY, M., DOVROLIS, C., AND CLAFFY, K. Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. In *IEEE Network* (June 2003).
- [18] RIBEIRO, V., RIEDI, R. H., BARANIUK, R. G., NAVRATIL, J., AND COTTRELL, L. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement Workshop (PAM)* (2003).
- [19] SHAKKOTAI, S., BROWNLEE, N., AND KC CLAFFY. A study of burstiness in tcp flows. In *Passive and Active Measurement Workshop (PAM2005)* (2005), pp. 13–26.
- [20] SHOYKHET, A., LANGE, J., AND DINDA, P. Virtuoso: A system for virtual machine marketplaces. Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University, July 2004.