Presented at Passive and Active Monitoring Workshop (PAM 2003),
4/6/2003-4/8/2003, San Diego, CA, USA

SLAC-PUB-9733
April 2003

Measuring end-to-end bandwidth with Iperf using Web100*

Ajay Tirumala
CS Dept.
University of Illinois
Urbana, IL 61801

tirumala@uiuc.edu

Les Cottrell
SLAC
Stanford University
Menlo Park, CA 94025

cottrell@slac.stanford.edu

Tom Dunigan
CSMD
ORNL
Oak Ridge, TN 37831

thd@ornl.gov

Abstract

End-to-end bandwidth estimation tools like Iperf though fairly accurate are intrusive. In this paper, we describe how with an instrumented TCP stack (Web100), we can estimate the end-to-end bandwidth accurately, while consuming significantly less network bandwidth and time. We modified Iperf to use Web100 to detect the end of slow-start and estimate the end-to-end bandwidth by measuring the amount of data sent for a short period (1 second) after the slow-start, when the TCP throughput is relatively stable. We obtained bandwidth estimates differing by less than 10% when compared to running Iperf for 20 seconds, and savings in bandwidth estimation time of up to 94% and savings in network traffic of up to 92%.

1 Introduction

Iperf [16] is a bandwidth measurement tool which is used to measure the end-to-end achievable bandwidth, using TCP streams, allowing variations in parameters like TCP window size and number of parallel streams. End-to-end achievable bandwidth is the bandwidth at which an application in one end-host can send data to an application in the other end-host. Iperf approximates the cumulative bandwidth (the total data transferred between the end-hosts over the total transfer period) to the end-to-end achievable bandwidth. We need to run Iperf for fairly long periods of time to counter the effects of TCP slow-start. For example, while running Iperf from SLAC to Rice University using a single TCP stream, with a TCP window size of 1 MB set at both ends, only 48.1 Mbps is achieved during slow-start (slow-start duration was about 0.9 seconds, the Round Trip Time (RTT) for this path was about 45ms), whereas the actual bandwidth achievable is about 200 Mbps. For the cumulative bandwidth to get up to 90% of the end-to-end achievable bandwidth, we need to run Iperf for about 7 seconds.

The other end-to-end bandwidth metric is the bottleneck bandwidth which is the ideal bandwidth of the lowest bandwidth link on the route between the two end-hosts [10] [14]. Generally, packet-pair algorithms are used to measure the bottleneck bandwidth. Packet-pair algorithms generate negligible network traffic. Bottleneck bandwidth does not vary rapidly in the timescales over which people make Iperf measurements, unless there are route changes and/or link capacity changes in the intermediate links of the route. In this paper, we are interested only in the achievable end-to-end TCP bandwidth. Achievable end-to-end TCP bandwidth (bandwidth from hereon) depends not only on the network, but also on the TCP/IP stack, processing power, NIC speed, the number of parallel streams used and the buffer sizes on the end-hosts. We assume sufficient processing power, NIC speed and large enough buffer sizes at the end-hosts to be available and shall not discuss this further.

Tools like Iperf[16] and TTCP[12] measure bandwidth by measuring the amount of data sent for a fixed period of time. They use TCP streams and can make use of parallel TCP connections[1]. Bulk data tools like Iperf work fairly well and are widely used [7] [8] [2]. Our endeavor is to reduce the measurement time and network traffic generated by these tools while retaining or improving the accuracy of measurement.

Web100 [19], which is currently available for Linux kernels, exposes the kernel variables for a particular TCP connection. We can use these variables in Iperf to estimate the end of slow-start. We can then measure the amount of data transferred for a short period

*This work was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information, and Computational Sciences Division under the U.S. Department of Energy. The SLAC work is under Contract No. DE-AC03-76SF00515.

[1] A detailed analysis of end-to-end performance effects of parallel TCP streams can be found in [6].

of time after slow-start, and often dramatically reduce the total measurement time (and network traffic) to make a bandwidth estimate.

The rest of the paper is organized as follows. Section 2 mentions the motivation for attempting this work. Section 3 explains the algorithm in detail. Section 4 has the implementation details, testing environment and the results obtained. We finally conclude in Section 5 with our overall observations and conclusions and a note on future work.

2 Motivation

The popularity and widespread use of Iperf can also be partially attributed to its ease of installation and absence of kernel and/or device driver modifications. The Web100 application library provides functions to query and set values for TCP variables for a particular session. These functions allow an application to tune and monitor its TCP connections which was previously possible only for kernel and/or device drivers.

Importantly, Web100 exposes the current congestion window, maximum, minimum and smoothed RTT, receiver advertised window, maximum segment size and data bytes out (and many other parameters) through the life of a particular TCP connection. In the next section, we show how we can determine the sampling rate and track these variables to determine quickly when the connection is out of slow-start.

As mentioned in the Web100 study undertaken at ORNL [3], the duration of slow-start is roughly $\lceil log_2(ideal\_window\_size\_in\_MSS) \rceil$ * RTT. In practice, it is a little less than twice this value because of delayed ACKs. For example, the bandwidth-delay produce for 1 Gbps network with an RTT of 200 ms is about 16667 1500-byte segments. The slow-start duration, when a single TCP stream is used, will be approximately $\lceil log_2(16667) \rceil * 2 * 0.2$, which is 5.6 seconds. Assuming a stable congestion window after slow-start, the time for the cumulative bandwidth to reach 90% of the achievable bandwidth will be about $10 * slow\_start\_duration - 10 * RTT$, which can be approximated to $10 * slow\_start\_duration$ for high bandwidth networks [3]. This means for the 1 Gbps-200ms network path, it will take over 50 seconds for the cumulative bandwidth to reach 90% of the achievable bandwidth. Using Iperf in quick mode can cut this measurement to under 7 seconds while still retaining the accuracy of measurement. Using parallel TCP streams can reduce the slow-start duration to a certain extent, since the bandwidth is shared between the streams[2].

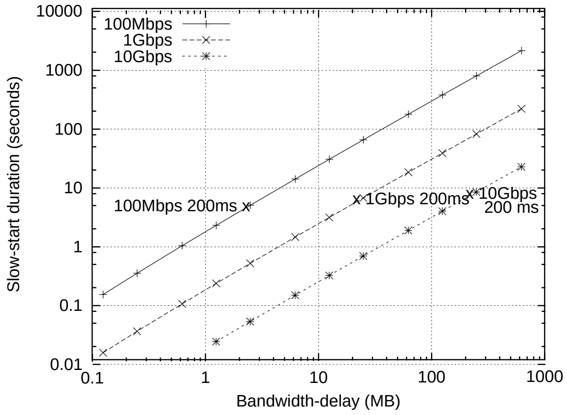

Figure 1: Slow-start times for various bandwidth-delay products (in theory)

Figure 1 gives the slow-start times (in theory) for various bandwidth-delay products. This also illustrates the importance of detecting the end of slow-start before starting the bandwidth measurement. For example, a 10 second measurement (default Iperf measurement time) is not sufficient for a 10 Gbps-200 ms bandwidth-delay network path, since the slow-start duration is itself over 5 seconds. But, a 10 second measurement is more than sufficient for a 1 Mbps-20 ms bandwidth-delay network path.

Varying network conditions like RTT and instantaneous bandwidth will alter the slow-start duration (see [7] for variations in RTT and bandwidth from SLAC to various nodes across the world with time). If we can accurately and quickly determine the end of slow-start for a connection, then by measuring bandwidth for a very short duration after the end of slow-start, we can significantly reduce the amount of network traffic generated and still retain or improve the accuracy.

3 Iperf QUICK mode

Our algorithm works for TCP Reno, which is the default implementation in most operating systems. We have our analysis for TCP Reno. Section 3.1 briefly mentions the behavior of congestion windows in TCP stacks and section 3.2 gives the details of the algorithm to detect the end of slow-start in Iperf Quick mode.

[2] Using parallel TCP connections has other advantages like dropping of a random packet affects only the particular stream which experienced a loss [13]. For example, if only one stream is used and there is a packet loss, the congestion window and thus the instantaneous bandwidth reduces to half the previous value. But, say 8 parallel streams are used and there is a random packet loss, only the stream which experienced a loss will reduce the congestion window to half its previous value and the instantaneous bandwidth will be about 94% of its previous value.

3.1 TCP Congestion Windows

TCP Reno doubles the congestion window every RTT until it reaches the threshold congestion window size or it experiences a retransmission timeout. After the slow-start period, Reno sets the threshold window to half the congestion window where (if) it experienced a loss. It increases its congestion window at a rate of one segment per RTT.

Also, the proposal of limited slow-start by Floyd [5] should result in a congestion-window value nearer to the ideal congestion window at the end of slow-start than in TCP Reno. This is due to the slower acceleration of the congestion window in the slow-start phase than in TCP Reno. Recently, Floyd (HighSpeed TCP) [4] and the Net100 community (TCP tuning daemon)[13] have suggested changes to TCP to improve high performance transfers, wherein the congestion window will increase by more than one MSS every RTT during the congestion avoidance period. With minor modifications, the algorithm mentioned in section 3.2 should be able to work for these variants of TCP. We expect our algorithm to perform better with these newer stacks, since the ideal congestion window is reached faster (in the congestion avoidance phase) than in Reno[3].

3.2 Algorithm to detect the end of slow-start

A knowledge of the behavior of TCP congestion windows in different flavors of TCP in section 3.1 is helpful in determining the sampling rate for kernel variables (explained below), to determine the end of slow-start as soon as possible. Our aim is to quickly and accurately determine the end of slow-start and to minimize the amount of network traffic generated. We cannot use the Web100 slow-start indicator, which will say whether a connection is in or out of slow-start since this will not work when the congestion window is restricted by the receiver window size.

Iperf initially gets the value of the smoothed-RTT(RTT) for the connection and also the MSS. We poll the value of RTT every 20 ms until we get a valid value (value > 0 ms reported by Web100) for the RTT and also note the value of the congestion window at that instant. Once we get the value of the RTT, we poll the value of the congestion window at an interval of $2 * RTT$. We store the values of old and new congestion window sizes. If the connection is still in slow-start during the period, TCP Reno would have doubled its congestion window size twice most likely (but at least once). If the connection is out of slow-start, the increase in congestion window would have been at the most $2 * MSS$. So, it is safe to say that if the old value of congestion window was at least $4 * MSS$ and the difference between the congestion window sizes is less than $3 * MSS$ for a time interval of $2 * RTT$, the connection is out of slow-start. For very slow connections, (or connections with extremely short RTT - like transfer to localhost), the congestion window size never reaches $4 * MSS$. For these kind of connections, an additional check to see if the congestion window size does not change when it is less than $4 * RTT$ will ensure that the connection is out of slow-start[4].

The pseudo-code for the above algorithm is given in the Figure 2.

```
set slow_start_in_progress <- true
set mss <- web100_get_mss()
set new_cwnd <- 0

while (slow_start_in_progress)
begin
  set rtt <- web100_get_smoothed_rtt()
  if (rtt is valid)
    sleep for twice the rtt
  else
    sleep for 20 ms
  set old_cwnd <- new_cwnd
  set new_cwnd <- web100_get_cwnd()
  set diff_cwnd <- abs(new_cwnd - old_cwnd);

  if (rtt is valid)
    if (((old_cwnd >= 4*mss)
          and (diff_cwnd < 3*mss))
        or ((old_cwnd < 4*mss)
          and (diff_cwnd = 0))
       slow_start_in_progress <- false;
end
```

Figure 2: Pseudo-code to determine end of slow-start

[3] The algorithm also works for TCP Vegas [1] without any modifications. Though TCP Vegas is fairer than TCP Reno, TCP Vegas clients may not receive a fair share of bandwidth while competing with TCP Reno clients [11] and hence Vegas clients are not implemented in many practical systems. TCP Vegas doubles the congestion window only every other RTT so that a valid comparison of the actual and expected rates can be made [1]. It calculates the actual sending rate (ASR) and expected sending rate (ESR). It tries to keep $Diff = ASR - ESR$ between α and β. It increases the congestion window linearly if $Diff < \alpha$ and decreases it linearly if $Diff > \beta$. It tries to keep $Diff$ in the range $\alpha < Diff < \beta$. The most common values of α and β are 1 and 3 MSS respectively.

[4] We have also ensured that the RTT does not change drastically between polling times to make our comparisons invalid.

3.3 Analysis of the Quick mode Iperf for TCP Reno

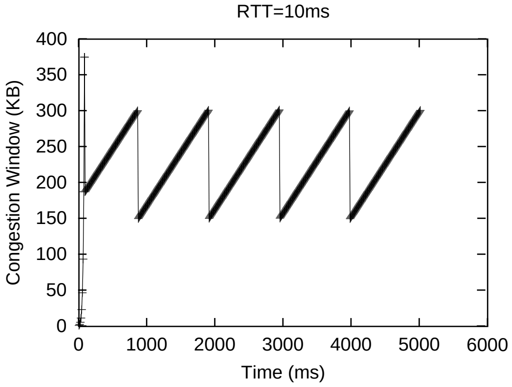

Figure 3: CW for link with $BW * Delay \approx 300KB$ for RTT=10 ms

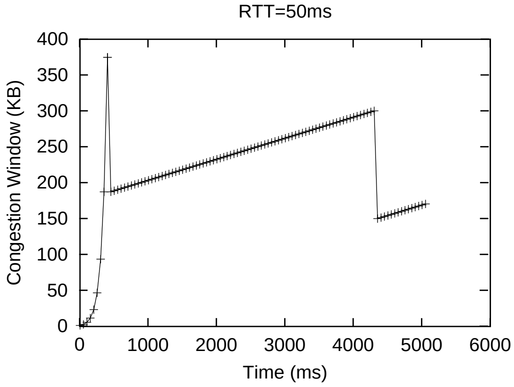

Figure 4: CW for link with $BW * Delay \approx 300KB$ for RTT=50 ms

Assuming constant available bandwidth and a fairly steady RTT for the period of measurement, we measure the bandwidth for a second after the end of slow-start. If the links are lossy and the TCP congestion window is never stable, Iperf does not enter the quick mode, but only reports the bandwidth after 10 seconds or a user specified time[5].

Figures 3 and 4 show the traces of congestion window when RTT is 10 ms and 50 ms respectively where the bandwidth-delay product is about $300KB$. These figures can be used to relate to the terms (for TCP Reno) defined below. Let us denote the congestion window reached during slow-start by ss_cwnd, the value it drops to immediately after slow start by cur_cwnd and the maximum value the congestion window reaches during the congestion avoidance period by m_cwnd. If a large enough receive window is set in the sender side, then the sender will get out of slow-start due to a congestion signal or a slow receiver throttling the sender by advertising a smaller window size. In cases where the receiver throttles the sender by advertising a smaller window size than what the sender assumes to be the ideal congestion window, the congestion window is constant during the congestion avoidance period (does not increase one RTT every MSS). In cases where the congestion window is very stable in the congestion avoidance period, Iperf Quick mode will report near perfect bandwidth values.

If the sender gets out of slow-start due to a congestion signal, then

$$cur\_cwnd = \frac{ss\_cwnd}{2} \tag{1}$$

The congestion window will rise from cur_cwnd to m_cwnd by one MSS every RTT. This means that during the first RTT after slow-start, cur_cwnd amount of data is sent. During the next RTT, $cur\_cwnd + MSS$ amount of data is sent and so on. This will continue until it reaches m_cwnd after which there will be a congestion signal and the congestion window will again drop to $m\_cwnd/2$. The average congestion window during this period will be

$$avg\_cwnd = \frac{m\_cwnd - cur\_cwnd}{2} + cur\_cwnd \tag{2}$$

The time taken to reach the m_cwnd is

$$id\_time\_after\_ss = \frac{m\_cwnd - cur\_cwnd}{MSS} * RTT \tag{3}$$

Thus the amount of data sent during this period is

$$data\_sent = \frac{avg\_cwnd * id\_time\_after\_ss}{RTT} \tag{4}$$

The bandwidth achieved during this period is

$$avg\_bw = \frac{data\_sent}{id\_time\_after\_ss} \tag{5}$$

Even if $id\_time\_after\_ss$ is much larger than 1 second, we will be measuring bandwidth which will be at least 67% of the bandwidth achievable had we allowed the test to run for $id\_time\_after\_ss$. Using limited slow-start [5] will definitely alleviate this problem to a large extent.

To see that measuring bandwidth for a second after the end of slow-start is sufficient in most cases, we

[5] This never occurred while measuring bandwidth to other high performance sites, but occurred once while measuring bandwidth across a dial-up link

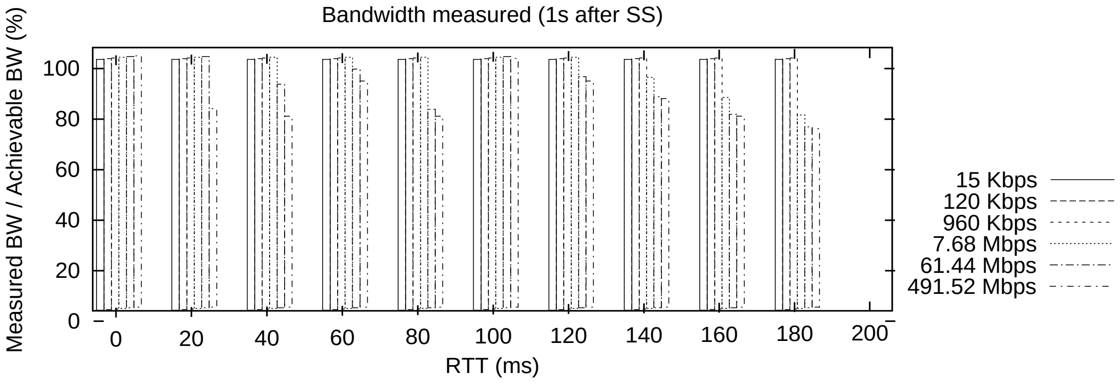

Figure 5: % bandwidth measured 1 second after slow-start (in theory)

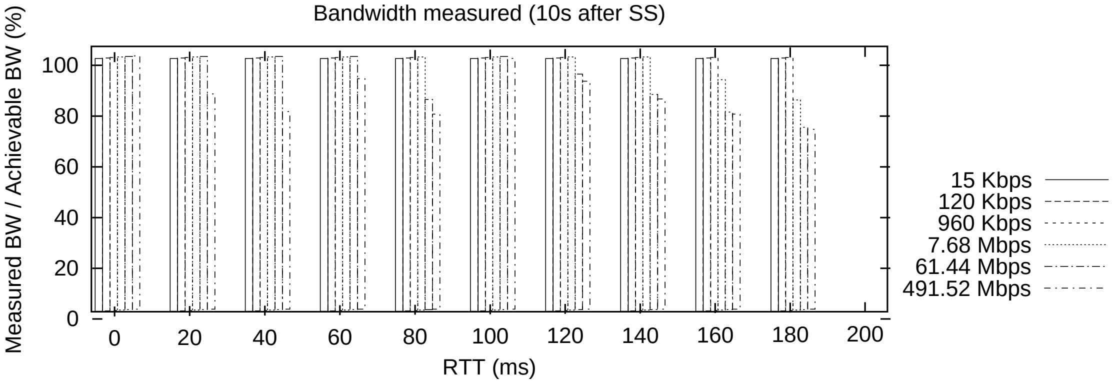

Figure 6: % bandwidth measured 10 seconds after slow-start (in theory)

plot the values of the bandwidth measurable as a percentage of the bandwidth achievable for various RTTs and link bandwidths. Figure 5 shows the bandwidths measurable by running the tests for 1 second after the end of slow-start and Figure 6 shows the bandwidths measurable by running the tests for 10 seconds after the end of slow-start. We assume that we can measure the bandwidth very accurately if the time of measurement after slow-start is greater than or equal to $id\_time\_after\_ss$. We see that the difference between the 1 second and 10 second measurements is not very high. For a bandwidth of about 1 Mbps it makes no difference at all. Even at a bandwidth of 491 Mbps, it makes a difference of less than 10%.

4 Implementation and Results

4.1 Implementation

Iperf runs in client-server mode. All the code changes in Iperf are confined to the client and only the host running the client needs a Web100-enabled kernel. The remote hosts, running Iperf servers, can run the standard Iperf available for various operating systems. Using the Web100 user-library (userland), we can read the variables out of the Linux proc file system. The total changes in code in Iperf is less than 150 lines[6]. We maintain compatibility with all the modes in which Iperf previously ran. The Web100 polling was done in a separate thread since Web100 calls were blocking.

4.2 Testing environment

For the testing environment, Iperf servers were present as a part of the IEPM bandwidth measurement framework [7] to various nodes across the world. Iperf runs in client-server mode and a version of Iperf which runs in a secure mode is available as a part of Nettest[15]. We measured bandwidth from SLAC to twenty high performance sites across the world. These sites included various nodes spread across US, Asia (Japan) and Europe (UK, Switzerland, Italy, France). The operating systems in the remote hosts were ei-

[6] The 150 lines only includes Iperf modifications. Scripts written for data collection are not included here

ther Solaris or Linux. The bandwidths to these nodes varied from 1 Mbps to greater than 400 Mbps. A local host at SLAC which ran Linux 2.4.16 (Web100-enabled kernel) with a dual 1130 MHz Intel Pentium 3 processor was used for the client.

4.3 Results

We measured bandwidth by running Iperf for 20 seconds, since running for 10 seconds (default) would not be adequate for a few long bandwidth-delay networks from SLAC (for example from SLAC to Japan). We have compared the 20 second measurements with the bandwidths obtained by running Iperf in Quick mode (1 second after slow-start). We obtained bandwidth estimates in quick mode differing by less than 10% when compared to the 20 second Iperf test for most cases as shown in Figure 7 [7].

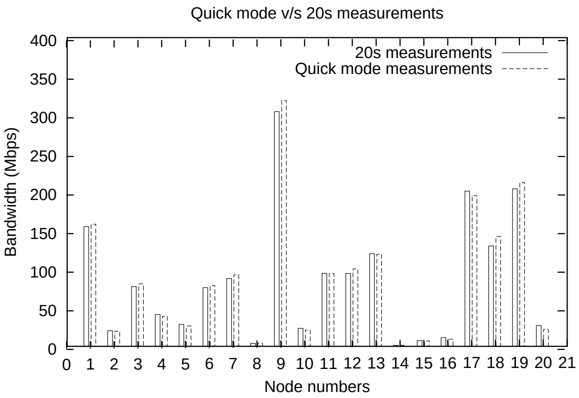

Figure 7: Quick mode measurements v/s 20 second measurements

We plotted the instantaneous bandwidth ($InstBW$ in Figures 8,9 and 10), every 20ms averaged over the last 500 ms, and found that it increases and decreases with the congestion window as expected. We also plotted the cumulative bandwidth, which will be the value reported by standard Iperf at that point of time ($CumuBW$ in Figures 8,9 and 10). The transfer from SLAC to Caltech shown in Figure 8 is the behavior which is expected in most transfers. The maximum TCP window size at both ends was 16MB which was greater than the bandwidth-delay product of about 800KB. The RTT is about 24ms (but Linux TCP stack rounds off the RTTs in 10s of milli-seconds). We have also plotted the transfer across a high latency network, from SLAC to Japan, where the RTT was 140 ms, the bandwidth-delay product was about 6MB and the maximum TCP window size was 16MB. In this case, the quick mode measurement gives us almost the exact value of the bandwidth. Also, we have plotted the transfer from SLAC to Rice University where the constraining factor is the receive window size (256KB). The bandwidth-delay product was 1.6 MB. These graphs are available for all the nodes being monitored at [18].

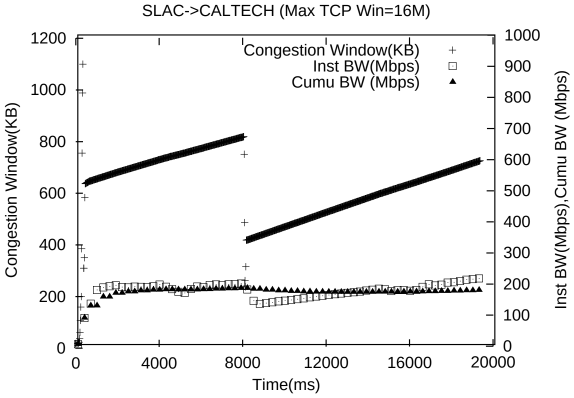

Figure 8: Throughput and cwnd (SLAC->Caltech)

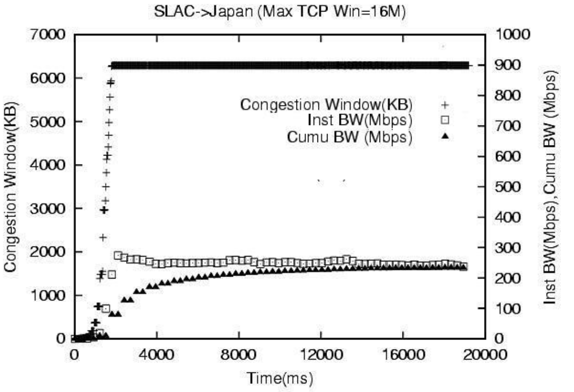

Figure 9: Throughput and cwnd (SLAC->Japan)

5 Conclusions and future work

The results obtained by running Iperf in quick mode for high bandwidth networks are encouraging. Iperf-generated network traffic has been reduced by up to 92% and the measurement time has been reduced by up to 94%. Iperf quick mode was deployed to measure bandwidth from SLAC to various high performace sites across the world, and the difference with

[7] These tests are performed on a regular basis and results are updated in [17]. We initially obtained a mean difference of 19%, and a standard deviation of 20%. This was later found to be a window sizing problem on Solaris where if the requested window size is not available, Solaris sets it to the default size which is generally 8 or 16KB. Linux is better in the sense that it defaults to the maximum allowable window size if the requested size cannot be satisfied.

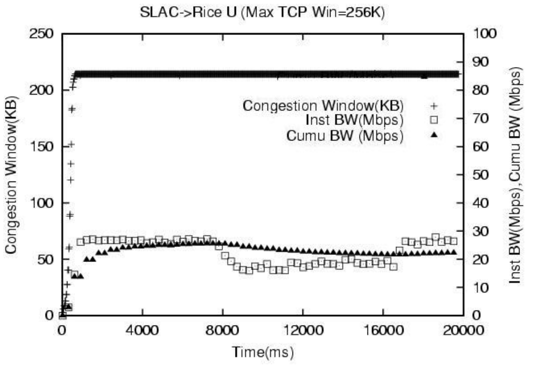

Figure 10: Throughput and cwnd (SLAC->Rice)

standard 20 second Iperf measurements was less than 10% in almost all the cases. These results also demonstrate that the Web100 TCP instrumentation is very useful.

We plan to modify the tool for TCP stacks suggested by the Net100 community and Floyd and analyze the same. We expect better results than in TCP Reno since the congestion window increases to the ideal value much faster in these newer stacks. Also, we plan to do some experiments and performance analysis with gigabit networks with latency in the order of hundreds of milli-seconds. We expect significant savings in measurement time in these cases. We also plan to correlate the congestion signals, loss and retransmission information for an Iperf session using Web100 with the bandwidth achieved.

Acknowledgments

The authors would like to thank Brian Tierney, from LBNL, for his helpful comments.

References

[1] L.S.Brakmo and L.L.Peterson . "TCP Vegas: end to end congestion avoidance on a global internet. *IEEE Journal in Selected Areas in Communication*, 13(8):1465-80, October 1995

[2] "Cooperative Association for Internet Data Analysis". [Online]. Available: http://www.caida.org/index.html

[3] T.Dunigan and F.Fowler. "Web100 at ORNL". [Online]. Available: http://www.csm.ornl.gov/ dunigan/netperf/web100.html

[4] S. Floyd. "HighSpeed TCP for Large Congestion Windows". *Internet Draft.* [Online]. Available: http://www.ietf.org/internet-drafts/draft-floyd-tcp-highspeed-00.txt

[5] S. Floyd "Limited Slow-Start for TCP with Large Congestion Windows". *Internet Draft.* [Online]. Available: http://www.ietf.org/internet-drafts/draft-floyd-tcp-slowstart-01.txt

[6] T. Hacker, B. Athey, and B. Noble. "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network". *In Proceedings of 16th IEEE-CS/ACM International Parallel and Distributed Processing Symposium (IPDPS), 2002*

[7] "Internet End-to-end Performance Monitoring - Bandwidth to the World (IEPM-BW) project". [Online]. Available: http://www-iepm.slac.stanford.edu/bw/index.html

[8] "Internet2 End to End Performance Initiative". [Online]. Available: http://e2epi.internet2.edu/index.shtml

[9] V. Jacobson. Congestion avoidance and control. *Computer Communication Review, 18(4):314-29*, August 1988

[10] K. Lai and M. Baker. "Measuring Bandwidth", *In Proceedings of INFOCOMM '99*, pp 235-245, March 1999

[11] J. Mo, R.J. La, V. Anantharam and J.Walrand. "Analysis and comparison of TCP Reno and Vegas", *Proceedings of IEEE INFOCOMM'99*, pp 1556-63, March 1999

[12] M. Muuss, "The TTCP Program". [Online]. Available: http://ftp.arl.mil/ftp/pub/ttcp

[13] "The Net100 Project". [Online]. Available: http://www.net100.org

[14] V. Paxson. "Measurement and Analysis of End-to-End Internet Dynamics", *Ph.D thesis*, Computer Science Division, University of California - Berkeley, April 1997

[15] "Secure Network Testing and Monitoring". [Online]. Available: http://www-itg.lbl.gov/nettest/

[16] A. Tirumala, M. Gates, F. Qin, J. Dugan and J. Ferguson. "Iperf - The TCP/UDP bandwidth measurement tool". [Online]. Available: http://dast.nlanr.net/Projects/Iperf

[17] A. Tirumala and L. Cottrell. "Iperf Quick Mode". [Online]. Available: http://www-iepm.slac.stanford.edu/bw/iperf_res.html

[18] A. Tirumala and L. Cottrell. “Plots of Congestion Window and Throughput from SLAC to different nodes”. [Online]. Available: http://www-iepm.slac.stanford.edu/bw/cwnd/cwnd.html

[19] “The Web100 project”: [Online]. Available: http://www.web100.org