

# PERFORMANCE TRADE-OFFS OF TCP ADAPTATION METHODS

NAGESWARA S. V. RAO

*Center for Engineering Science Advanced Research  
Computer Science and Mathematics Division, Oak Ridge National Laboratory  
Oak Ridge, Tennessee 37831-6364, USA E-mail: raons@ornl.gov*

WU-CHUN FENG

*Computer and Computational Sciences Los Alamos National Laboratory  
Los Alamos, New Mexico 87545, USA E-mail: feng@lanl.gov*

Parallel streams and the flow-control adaptation have been proposed as methods for overcoming the throughput limitations of existing transport protocols. While these methods have been very effective in simulations and implementations, only a few high-level explanations exist in understanding the actual network conditions that give rise to the performance improvements. By utilizing qualitative models for TCP and a generic simulation scenario, we demonstrate the conditions under which each of the techniques excels. In short, parallel streams perform best when the network suffers from high packet-loss conditions while the adaptation of flow-control windows performs best under low packet-loss rates.

## 1 Introduction

Many high-performance network applications use only a small fraction of the available network bandwidth, particularly over large Bandwidth-Delay product (BDP) network connections. Obtaining a larger fraction of the available bandwidth, and hence closing the *wizard gap*<sup>a</sup>, generally requires network and systems experts to tune end-host software, and potentially, institutional firewalls. To close this gap, network researchers have proposed two generic classes of adaptation techniques for TCP: (a) parallel TCP<sup>7</sup>, Pockets<sup>9</sup>, and multiple paths<sup>7</sup>, and (b) tuning of buffer sizes via dynamic right-sizing<sup>1</sup>, auto-tuning<sup>8</sup>, net100<sup>6</sup>, and ENABLE<sup>10</sup> to optimize a single stream.

Although the above techniques are available as individual modules and have even been integrated into tools such as GridFTP<sup>2</sup>, they can actually degrade performance if inappropriately deployed. For instance, transfers between Stanford Linear Accelerator Center (SLAC) and University of Wisconsin demonstrate that parallel TCP streams offer better throughput than tuned buffers. In stark contrast, transfers between SLAC and Rice University ben-

---

<sup>a</sup>The wizard gap is the difference between the network performance that a network “wizard” can achieve by appropriately tuning buffer sizes and that of an untuned application.

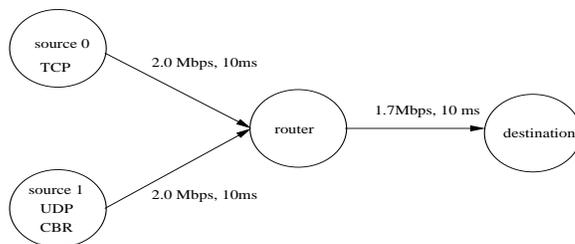


Figure 1. *Simulation set-up.*

enefit most from tuned buffers rather than parallel streams. Finally, transfers between SLAC and Los Alamos National Laboratory (LANL) provide optimal throughput when an appropriate combination of the two is used. Therefore, a qualitative understanding of the network conditions under which each of the techniques delivers the best performance is critical to applications that rely on networks with large BDPs.

In this paper, we will focus on parallel TCP and tuning of buffer sizes via dynamic right-sizing (DRS). Parallel TCP uses parallel connections while DRS automatically tunes the flow-control buffers. By adopting an appropriate model for TCP, we will demonstrate that parallel TCP offers throughput advantages over DRS in the following cases: (1) the slow-start phase of TCP, (2) the fast recovery phase of TCP, and (3) when packet-loss rates are high. On the flip side, a tuned DRS stream achieves better throughput in high-bandwidth, low-loss environments, e.g., Abilene and Internet2. For instance, by setting the flow-control window to a value slightly below the bottleneck bandwidth, a tuned connection provides a smooth flow (i.e., relatively constant bandwidth) as opposed to the sawtooth-like flow typical of TCP. For parallel TCP, the sawtooth profile is particularly pronounced, which in turn, leads to throughput levels well below the bottleneck bandwidths. Finally, we discuss ways to combine the two techniques to reap the benefits of both.

Figure 1 shows the network topology used in ns-2 simulations to perform our experiments. Source nodes 0 and 1 generate TCP and UDP traffic, respectively, that goes to a drop-tail router over 2.0 Mbps links and then to a bottleneck link of 1.7 Mbps before reaching the destination. Source node 1 generates UDP traffic at a constant bit rate (CBR) and serves as the background traffic that the TCP connection must adapt to.

## 2 Macroscopic View of a Single TCP

Two parameters control the transport process in TCP: the flow-control window and congestion-control window, denoted by  $F_S(t)$  and  $C_S(t)$ , respectively, as functions of time  $t$ . Here the suffix  $S$  denotes a single TCP stream. The *throughput* given by the rate of packets transmitted is  $\eta_S(t) = \min\{F_S(t), C_S(t)\}$ . For typical TCP implementations,  $F(t)$  is fixed at the initiation of the connection between source and destination.

Figure 2 shows the congestion window of the TCP connection corresponding to UDP CBR rates 0, 0.5, 1.0, and 1.5 Mbps. In the top-left figure, the flow-control window keeps the throughput below the bottleneck bandwidth of 1.7 Mbps, and hence no losses are encountered, resulting in monotonically increasing congestion window. As the volume of UDP traffic increases, the aggregate UDP and TCP throughput demands oftentimes exceed 1.7 Mbps, resulting in packet losses. In response to each packet loss, TCP reduces its congestion window in half and re-enters its congestion-avoidance mode where the congestion window grows linearly in size.

Figure 3 presents aggregate throughput profiles when the corresponding UDP CBR rates are 0, 0.5, 1.0, and 1.5 Mbps, respectively. In each graph, the upper line represents the aggregate throughput emitted from the source while the lower line captures the goodput at the destination. A source node infers that a packet has been lost in the network via packet loss, timeout, or duplicate acknowledgements.

Let  $T_1, T_2, \dots, T_p$  be the times at which a loss is inferred. Let  $r_S(t)$  denote the *growth rate* of the congestion window at  $t$ , where  $r_S(t)$  is defined to be 0 when the congestion window decreases in size. Let  $T_0$  and  $T_{\max}$  be the start and end times of the transmission from source to destination, respectively.

We first consider the case where the  $F_S(t)$  per round-trip time is larger than the bottleneck bandwidth  $B_r$  between the source and destination. This case corresponds to the top-right, bottom-left and bottom-right plots in Figure 3. Figure 4 shows that TCP dynamics can be visualized over two intervals,  $[0, T_S^{SS}]$  and  $(T_S^{SS}, T_{\max}]$ , corresponding to the *slow-start* and *congestion-control* phases, respectively. We follow the usual convention where  $[t_1, t_2]$  denotes the closed interval that includes the endpoints and  $(t_1, t_2)$  denotes the open interval that excludes the endpoints.

The function  $r_S(t)$  spans across two different sets of values, i.e., the fast growth of slow start and the slow linear growth during congestion avoidance. At high load, the congestion profile dominates because each inferred loss reduces  $C_S(t)$  by a fraction, and multiple losses cause  $C_S(t)$  to stay below  $F_S(t)$ , as shown in the top-right, bottom-left and bottom-right plots of Figure 2.

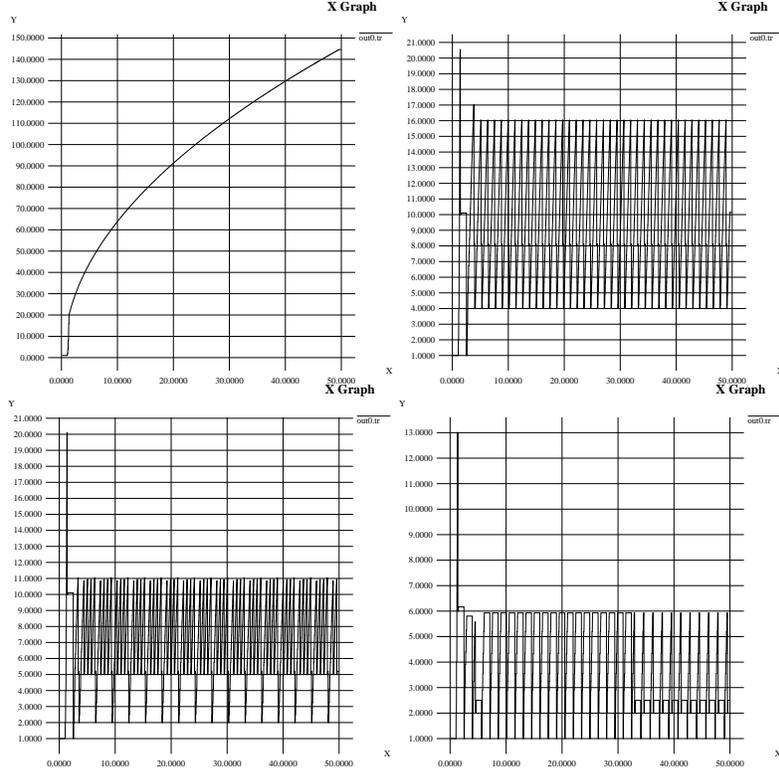


Figure 2. Congestion window profiles for UDP CBR rates of 0, 0.5, 1.0 and 1.5 Mbps, in top-left, top-right, bottom-left and bottom-right plots, respectively. X-axis represents the simulation time  $t$  in seconds and Y-axis represents  $C_S(t)$ .

Thus, for a single TCP, we have  $T_S^{SS} = T_1$ . While the exact values of  $r_S(t)$  vary somewhat within each phase, we approximate the rate in the slow-start and congestion-control phases by  $r_{SS}$  and  $r_{CC}$ , respectively.

Let  $U_{[t_1, t_2]}(t)$ , for  $t_1 \leq t_2$ , denote the function which is 1 if  $t \in [t_1, t_2]$  and 0 elsewhere, then the growth rate of the congestion window for a single TCP is  $r_S(t) = r_{SS}U_{[T_0, T_S^{SS}]}(t) + r_{CC}U_{(T_S^{SS}, T_{\max}]}(t) = r_{SS}U_{[0, T_1]}(t) + r_{CC}U_{(T_1, T_{\max}]}$  when flow windows are sufficiently large such that  $F_S(t) > C_S(t)$  for all  $t \in [T_0, T_{\max}]$ . Note that in this case  $\eta_S(t) = C_S(t)$  for all  $t \in [T_0, T_{\max}]$ .

If flow windows are smaller, then  $F_S(t)$  affects  $\eta_S(t)$  the most such that  $\eta_S(t) = F_S$  for all  $t \geq T_1$ . Figures 5(a) and (b) demonstrate this phenomenon. Suppose that  $F_S(t)$  is much smaller than the bottleneck bandwidth  $B_\tau$ , then throughput is clamped by the  $F_S(t)$ . Hence, TCP does not cause overflow at

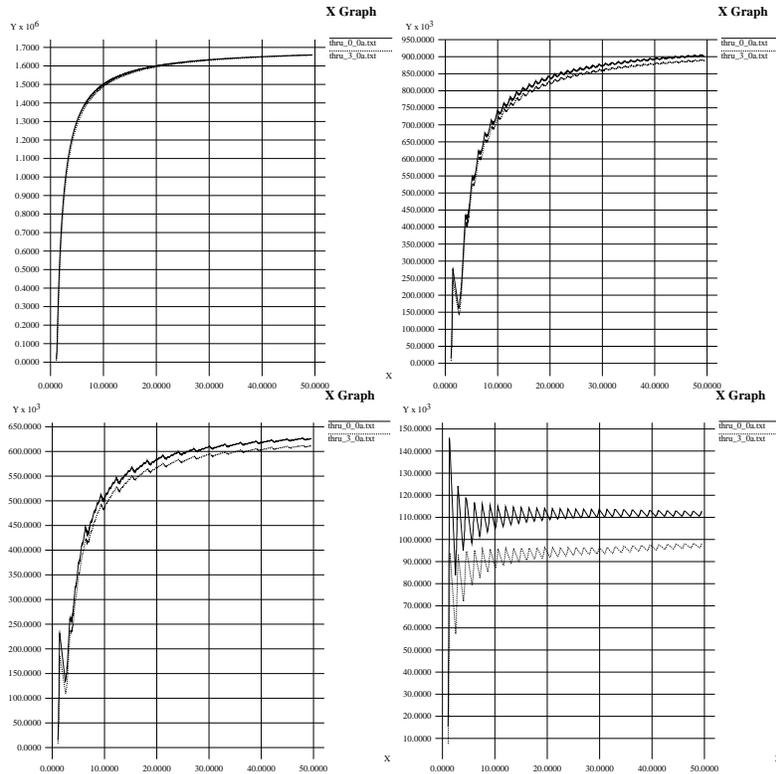


Figure 3. Throughput profiles for UDP CBR rates of 0, 0.5, 1.0, and 1.5 Mbps in top-left, top-right, bottom-left and bottom-right figures, respectively. X-axis represents the simulation time  $t$  in seconds and Y-axis represents the throughput expressed in bits per second. In each figure, top plot corresponds to the throughput of source 0 and bottom plot corresponds to the goodput received at destination.

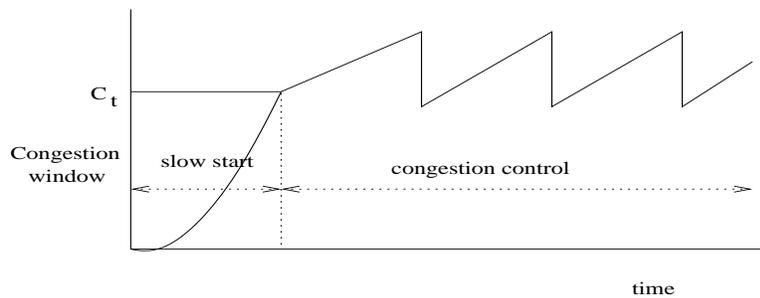


Figure 4. Congestion window profile of TCP under congestion losses.

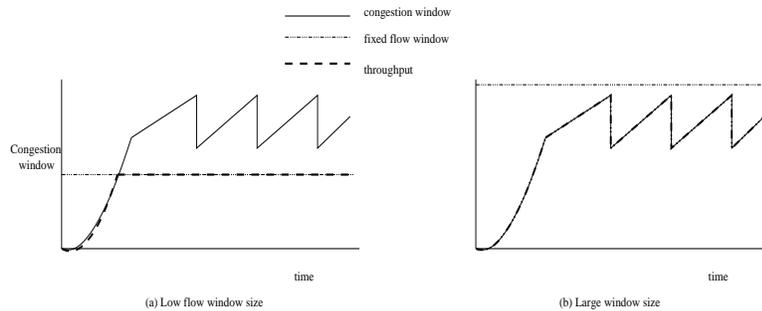


Figure 5. Congestion window profiles of TCP under different loads.

the bottleneck links and does not infer loss. Thus the slow-start phase will be prolonged until the threshold  $C_\tau$  is reached, and then TCP switches to the congestion control mode. But the latter will not have much effect on  $\eta_S(t)$  which would have been already curtailed by  $F_S$ . Let  $T_t$  be the earliest time such that  $C_S(t) \geq F_S$ . We have two cases: (a) if  $T_1 > T_t$ , then  $r_S(t) = r_{SS}$  for  $t \in [T_0, T_t]$ ; and (b) if  $T_1 < T_t$ , then  $r_S(t) = r_{SS}$  for  $t \in [T_0, T_1]$  and  $r_S(t) = r_{CC}$  for  $t \in [T_1, T_t]$ . In summary, we observe two qualitatively different behaviors in a single TCP stream:

(i) If the size of the flow window is small,  $\eta_S(t)$  consists of fast increase followed by a constant flow. This behavior may be caused either by  $F_S$  being set low or by a low-loss rate with high bottleneck bandwidth. We refer to this region as *low- $F_S$* .

(ii) If flow window is large,  $\eta_S(t)$  consists of fast increase followed by a sawtooth profile due to TCP's Additive Increase and Multiplicative Decrease (AIMD) behavior. This behavior is due to high values for  $F_S$ , high-loss rates, or low bottleneck bandwidth. We refer to this region as *high- $F_S$* .

### 3 Single and Parallel TCP

In contrast to the previous section, here we consider  $n$  parallel-TCP streams in place of a single TCP stream. In the simulation, we increased the UDP CBR rate from 0 to 1.75 Mbps and the corresponding throughputs are shown for  $n = 1, 2, 3, 4$  in Figure 6. The throughput of parallel TCP improves with the number of streams as the UDP traffic increases. When the UDP traffic is low, however, a single stream performs better.

When the UDP traffic rate is high, i.e., high- $F_S$  region, parallel TCP

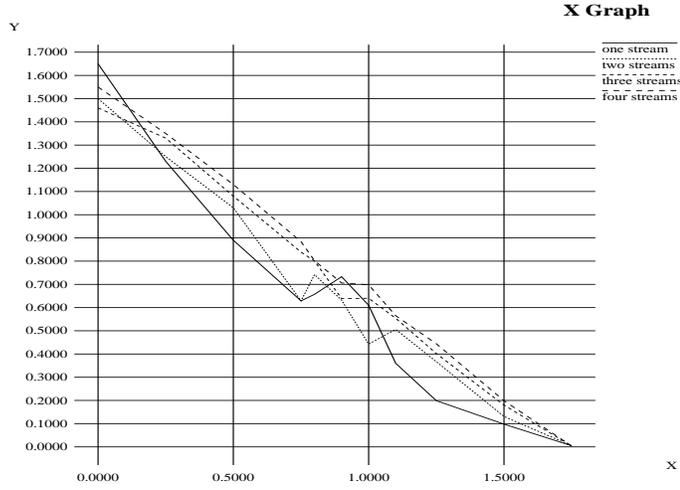


Figure 6. Throughput improvements due to parallel TCP. X-axis represents the UDP CBR traffic in Mbps and Y-axis represents the goodput received at the destination in Mbps.

performs better than a single, tuned TCP stream. Conversely, when the UDP traffic rate is low, i.e., low- $F_S$  region, the tuned TCP stream outperforms the parallel TCP streams.

### 3.1 High $F_S$ Region

Initially, the losses inferred at times  $T_1, T_2, \dots, T_m$  affect many streams. Each stream's first loss causes its TCP to transition from slow start to congestion control. The growth rate for parallel TCP is given by

$$r_P(t) = nr_{SS}U_{[T_0, T_1]} + [(n-1)r_{SS} + r_{CC}]U_{[T_1, T_2]} + \dots + nr_{CC}U_{(T_n, T_{\max})}.$$

The growth rates of a single TCP stream versus parallel TCP streams over different intervals are as follows.

time interval	Single TCP	$n$ Parallel-TCP
$[0, T_1]$	$r_{SS}$	$nr_{SS}$
$[T_1, T_2]$	$r_{CC}$	$(n-1)r_{SS} + r_{CC}$
$[T_2, T_3]$	$r_{CC}$	$(n-2)r_{SS} + 2r_{CC}$
$[T_{n-1}, T_n]$	$r_{CC}$	$r_{SS} + (n-1)r_{CC}$
$[T_n, T_F]$	$r_{CC}$	$nr_{CC}$

In summary, the advantages of parallel TCP over a single (tuned) TCP in the high- $F_S$  region are three-fold:

(a) **Robust and sustained slow start:** It takes at least  $n$  losses for all streams to transition into congestion-control mode. Thus, the duration of slow start in parallel TCP is  $T_P^{SS} = T_n$  in the worst case, which is significantly larger than that of a single TCP given by  $T_S^{SS} = T_1$ .

(b) **Faster slow start:** For the duration of slow start  $[0, T_P^{SS}]$ , the growth rate for  $n$  parallel streams reduces by  $r_{SS} - r_{CC}$  with every inferred loss, as shown in Table 3.1. Compared to the growth rate of  $r_{SS}$  during the slow-start period  $[0, T_1]$  of a single TCP, the growth rate of parallel TCP is  $r_P(t) = nr_{SS}$ .

The average growth rate of parallel TCP is  $r_P(t) \frac{1}{n} \sum_{i=1}^n (n-i)r_{SS} = \frac{n(n-1)}{2}r_{SS}$ .

(c) **Faster recovery:** After all streams enter congestion-control mode,  $n$  streams recover at an aggregated growth rate of  $nr_{CC}$ , offering a factor of  $n$  improvement over a single TCP. Similar results are presented in <sup>3,4</sup>.

### 3.2 Low $F_S$ Region

In this section, we consider the low- $F_S$  region, where the flow windows per round-trip time (RTT) are significantly smaller than the bottleneck bandwidth  $B_\tau$ . As shown in Figure 6, if the same flow-window size is used for all streams, the parallel streams compete with each other and induce losses because of each other. As shown in the leftmost portions of Figure 6, this phenomenon causes the overall throughput for parallel streams to drop below that of a single TCP. The ideal case is to set the  $F_{P_i}(t) = B_\tau/n$  for all streams  $i$ , then the throughputs are given as follows where  $t_{B_\tau}$  is the earliest  $t$  such that  $\eta_S(t) = B_\tau$ :

Time Interval	Single TCP	$n$ Parallel TCP
Growth Rate during $[0, T_{B_\tau}]$	$r_{SS}$	$nr_{SS}$
Throughput during $[T_{B_\tau}, T_{\max}]$	$B_\tau/n$	$B_\tau$

Now, compare the parallel TCP case to that of a single TCP with the flow window (ideally) tuned to be just below  $B_\tau$ . While both single and parallel streams will eventually reach the optimal throughput of  $B_\tau$ , the latter reaches it quicker. However, for optimal throughput in parallel TCP, the sum of the sizes of flow windows must be chosen to be slightly below  $B_\tau$ . If the number of streams and the flow-window sizes are not properly chosen, additional streams can in fact result in throughput reduction. To understand this phenomenon, consider two parallel streams obtained by adding a stream to one with  $F_S = B_\tau - 1$ . The additional stream will result in loss at the bottleneck link, and consequently, at least one of the streams reduces its

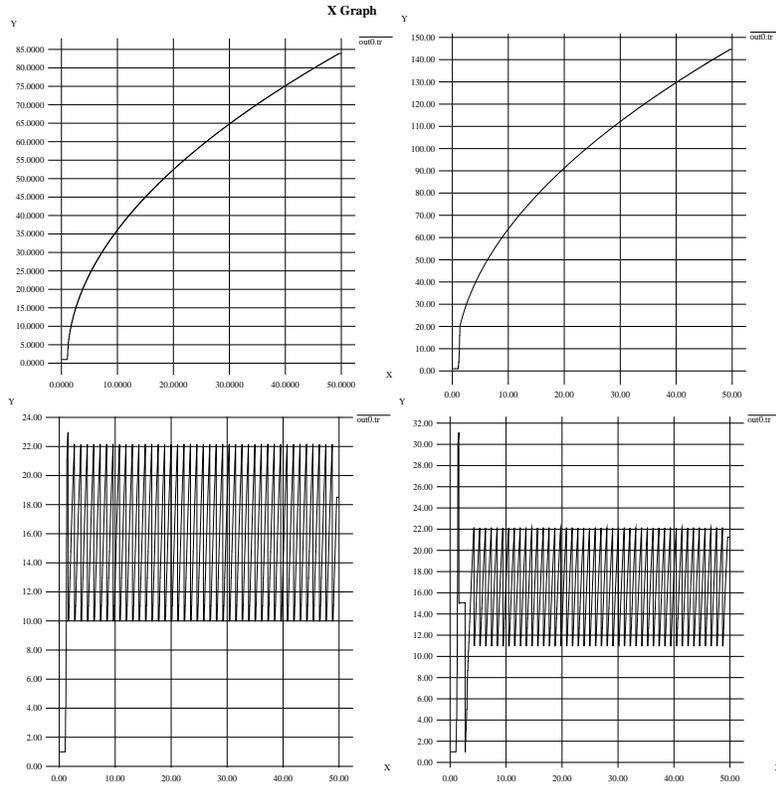


Figure 7. Congestion profiles for flow-window sizes of 5, 20, 21, and 30 in top-left, top-right, bottom-left and bottom-right plots respectively.  $X$ -axis represents the simulation time  $t$  and  $Y$ -axis represents  $C_S(t)$ .

$C_S(t)$  by a multiplicative factor  $a$ . Assume that both streams equally share the losses and that  $a = 1/4$ . Then, a straightforward computation shows that the average throughput of these two TCP streams is approximately  $7/8B_\tau$  whereas that of single TCP is approximately  $B_\tau$ .

#### 4 Dynamic Right-Sizing

As shown in Section 2, a fixed low value for  $F_S$  makes TCP less responsive to the available bandwidth as well as loss conditions, resulting in artificially reduced throughput. By varying  $F_S$  via dynamic right-sizing (DRS)<sup>1</sup>, the effect on throughput could be dramatic, as will be demonstrated here.

Figure 7 shows the congestion-window profiles when the flow-window sizes are  $F_S = 5, 20, 21, 30$  (expressed in packets), and Figure 8 shows the corresponding throughputs. For  $F_S = 20$  the throughput is about 1.65 Mbps and that for  $F_S = 21$  it is below 1.5 Mbps. This is explained by the qualitative difference in the flow rates under the control of flow and congestion windows. As shown in Figure 7, for  $F_S = 20$  the smooth flow is a result of control by the flow window. For  $F_S = 21$ , the flow is controlled by the congestion window which oscillates around the optimal flow window, thereby resulting in a non-smooth flow. In general, this interesting case occurs when  $F_S(t) = F_{S;B_\tau-}$  is just below the bottleneck bandwidth  $B_\tau$  as shown in Figure 5(b). Here  $C_S(t)$  continues to grow since there will be no losses. But  $\eta_S(t) = \eta_{S;B_\tau-}(t)$  grows until it reaches  $F_S$  and stays constant. Increasing  $F_S$  slightly above the bottleneck bandwidth  $B_\tau$  to  $F_{S;B_\tau+}$  will result in losses since  $\eta_S(t) > B_\tau$  for some  $t$ , which consequently reduces  $C_S(t)$ . Then, the throughput will be subsequently reduced below  $B_\tau$  and will be later increased to exceed  $B_\tau$ , and this behavior repeats. Let  $\bar{\eta}_S$  denote the average value of  $\eta_S(t)$  over the period  $[T_0, T_{\max}]$ . Thus we have  $\bar{\eta}_{S;B_\tau-} > \bar{\eta}_{S;B_\tau+}$  although the flow window in the latter case is larger, i.e.  $F_{S;B_\tau-} < F_{S;B_\tau+}$ . For certain TCP implementations the reduction in throughput when  $F_S$  is increased above  $B_\tau$  could be as much as 25%. In general, if  $c$  is the multiplicative factor by which the window size is reduced when a loss occurs, the resulting throughput is approximately  $(1 - c/2)B_\tau$  due to the sawtooth profile of TCP's congestion control. Thus, by ensuring  $F_S(t) = F_{S;B_\tau-}$ , we achieve smoother and higher throughput.

If  $F_S$  is set too low, the available bandwidth is not fully utilized. But if  $F_S$  is too large initially, it leads to unnecessary losses in slow start especially if the latency to the bottleneck link is high. In such a case, it will take some time for the dropped packet to reach the source, and meanwhile TCP sends packets at an increasing rate corresponding to  $r_{CC}$ . DRS<sup>1</sup> achieves a balance by starting  $F_S(t)$  at a low value and increasing it carefully as the packets flow without losses. This method allows  $F_S(t)$  to track  $C_S(t)$ , and generally ensures that flow windows are not the limiting factors to the throughput. Furthermore, in slow-start phase, right-sizing encourages  $C_S(t)$  to grow as fast as possible without causing unnecessary buffer overflows at the bottleneck link. In the case of sustained losses,  $C_S(t)$  dips below  $F_S(t)$  thereby backing off in response to the increased traffic so as not to take unfair share of the bandwidth, and the resultant performance is identical to the usual TCP.

## 5 Right-Sizing Parallel-TCP Streams

We now first characterize the scenarios that are favorable to either parallel TCP or DRS. Under high loads, the performance of DRS is the same as that of a single TCP, and hence parallel TCP in place of a single TCP will provide better performance. Under low loss conditions, parallel TCP using fixed values of  $F_S$  has several disadvantages. If the number of parallel streams is not suitably limited, the throughput will increase much faster than the sustainable rate, resulting in unnecessary losses and subsequent reduction in the throughput. This problem is particularly acute if  $F_S$  is fixed to a large value because the parallel streams increase their throughput at a much faster rate during slow start. Dynamic right-sizing a single stream, on the other hand, prevents the throughput from growing too fast to unsustainable levels. Also, once the parallel TCP streams enter the congestion-control mode, such that their total peak rate is above the available bandwidth, losses will result, which will in turn reduce the throughput rate. However, by right-sizing the flow windows such that the total throughput of parallel TCP is slightly below the available bandwidth, losses can be avoided and the available bandwidth can be entirely utilized. Such right-sizing requires a delicate balance because the available bandwidth could vary rapidly, and hence, it must be dynamically inferred to adjust the flow windows.

If we consider replacing a single TCP stream with  $n$  parallel TCP streams, each of which is dynamically right-sized, then the individual streams are guided by the right-sizing during slow start from growing too fast. But at the same time, in cases of loss, all the virtues of parallel TCP are preserved during slow start. During high loss situations, the performance is identical to that of parallel TCP.

### Acknowledgments

This work was supported by the High-Performance Networking program of U.S. Dept. of Energy, and Laboratory-Directed Research and Development at Oak Ridge National Laboratory and Los Alamos National Laboratory.

### References

1. W. Feng, M. Fisk, M. Gardner, and E. Weigle. Dynamic right-sizing: An automated, lightweight, and scalable technique for enhancing grid performance. In *Lecture Notes of Computer Science*. 2002 (in press).
2. Grid ftp. <http://www.globus.org/gridftp.html>.

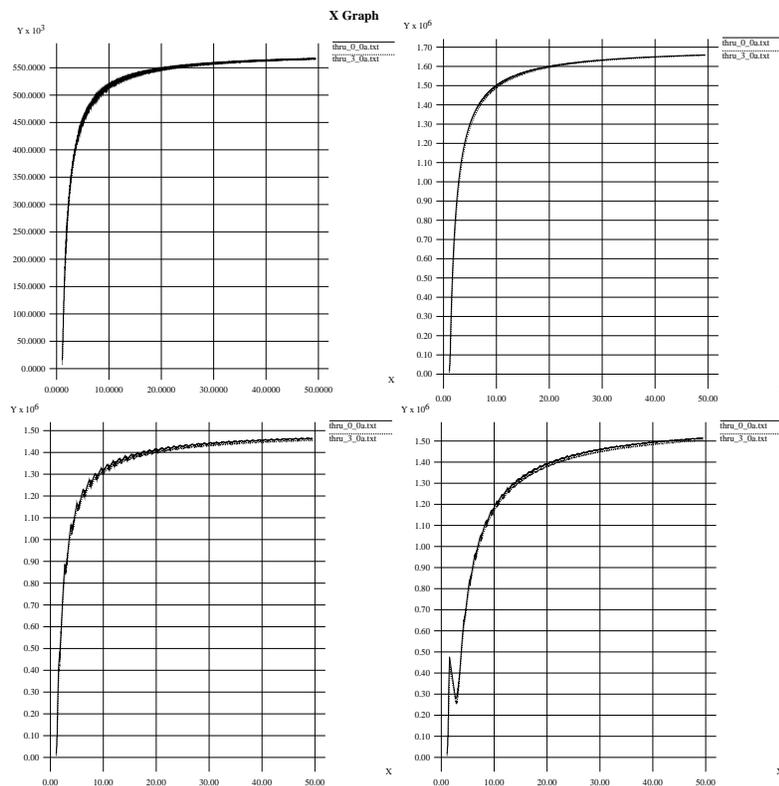


Figure 8. Throughputs for flow-window sizes of 5, 20, 21, and 30 in top-left, top-right, bottom-left and bottom-right plots, respectively. X-axis represents the simulation time  $t$  and Y-axis represents the throughput in bps.

3. T. J. Hacker and B. D. Athey. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Proc. of Int. Parallel and Distributed Processing Symposium*. 2002.
4. F. P. Kelly. Mathematical modelling of the Internet. In *Proc. Int. Congress on Industrial and applied mathematics*. 1999. <http://www.statslab.cam.ac.uk/frank/mmi.html>.
5. W. Matthews and L. Cottrell. The PingER project: Active Internet performance monitoring for the NENP community. *IEEE Communications Magazine*, 130–136 (May 2000).
6. Net100 project. <http://www.net100.org>.
7. N. S. V. Rao. NetLets for end-to-end delay minimization in distributed

- computing over Internet using two-paths. *International Journal of High Performance Computing Applications* **16**, 3 (2002). in press.
8. J. Semke, J. Mdhavi, and M. Mathis. Automatic TCP buffer tuning. In *Proc. of ACM SIGCOMM 1998*. 1998.
  9. H. Sivakumar, S. Bailey, and R. L. Gorssman. Psockets: The case for applicaiton-level network striping for data intensive applications using high speed wide area networks. In *Proc. of SC2000*. 2000.
  10. B. L. Tierney, D. Gunter, J. Lee, and M. Stoufer. Enabling network-aware applications. In *Proc. of HPDC 01*. 2001.