TCP Behavior with Many Flows

Robert Morris
Harvard University
rtm@eecs.harvard.edu
Presented at the IEEE International Conference on Network Protocols, October 1997, Atlanta, Georgia.

Abstract

TCP's ability to share a bottleneck fairly and efficiently decreases as the number of competing flows increases. This effect starts to appear when there are more flows than packets in the delay-bandwidth product. In the limit of large numbers of flows, TCP forces a packet loss rate approaching 50%, causing delays that users are likely to notice. TCP's minimum congestion window of one packet is the source of these problems: it causes a few flows to send too fast while the rest wait in re-transmission time-out. The particular packet loss rate is a function of TCP's abrupt transition from exponential backoff to sending with a window of one or more packets, and of the high rate at which TCP increases small congestion windows.

Analysis of packet traces suggests that these aspects of TCP's algorithms contribute substantially to the total loss rate observed on the Internet. One way to work around the problem is to make sure routers have not just one round-trip time of buffering, but buffering proportional to the total number of active flows. A more fundamental cure might make TCP less aggressive and more adaptive when its congestion window is small.

1. Introduction

The main goals of any congestion control algorithm are to maintain high utilization of the bottleneck link, to avoid overloading the bottleneck and thus avoid high queueing delay and packet loss, and to divide bandwidth fairly among competing flows. Congestion control on the Internet is provided by end-to-end mechanisms in TCP [9, 19] in cooperation with queueing strategies in routers [7]. Improvements prompted by a long history of investigation [14, 9, 22, 18, 20] have led to a continuous expansion of the operating conditions under which TCP works well. Most of these investigations consider at most a few dozen competing flows. Some work [6,9,20] has hinted at limits to TCP's ability to handle large numbers of flows, and the idea that window flow control in general has scaling limits is well known [1]. Whether such limits have any practical relevance to the Internet is not well known.

These limits do have a substantial and adverse impact on the Internet, as this paper will argue. The limits start to appear when the number of active flows exceeds the network's delay-bandwidth product as measured in packets. An "active" flow has data to send, and is typically waiting for an acknowledgment or a retransmission time-out. The symptoms are high utilization coupled with high packet loss and high variation in the delay seen by users. High utilization is only good when the packet loss rate due to queue overflow is low. Each lost packet consumes network resources before it is dropped, contributing to lowered efficiency in other parts of the network. Sufficiently high packet loss rates also cause long and unpredictable delays in time-out-based protocols such as TCP.

The main contribution of this work is a simulation study of TCP's behavior when many active flows compete for bandwidth over the same link. The simulations predict unacceptably frequent packet loss and high variation in bandwidth. The second contribution of this work is evidence that such conditions exist on the Internet: some points in the network regularly experience hundreds or thousands of active flows. Finally, analysis of the causes of these problems suggests two solutions less costly than increasing network bandwidth. As a short-term solution, routers could be provisioned with buffer space proportional to the maximum number of active flows; simulations suggest ten times as many buffers as flows. This solution would have to be coupled with per-flow limits on buffer use. We also speculate that when TCP's congestion window is small it should use rate control and a less aggressive window increase policy. These changes would reduce TCP's dependence on exponential retransmission time-out backoff, which we argue makes a poor congestion control mechanism.

2. TCP Algorithms

TCP's algorithms [19] and the behavior they produce are too involved to describe fully here. The following is a summary of the points most relevant to this work.

A TCP sender avoids overloading the network by sending new data only after getting an acknowledgment from the receiver indicating receipt of previous data. The

network's available bandwidth governs the rate at which data arrives at the receiver, and thus the rate at which the receiver sends acknowledgments, and thus the rate at which the sender transmits. Waiting for an acknowledgment after sending each packet would leave the network idle much of the time if the network's available bandwidth were high enough that the sender could transmit a packet in substantially less than the round trip propagation delay. For this reason TCP senders do not wait for acknowledgments until they have sent a window of packets.

The amount of bandwidth a TCP flow uses is roughly equal to the window size divided by the round-trip propagation delay. TCP should use a window size that makes this product equal to its fair share of the network bandwidth. Since TCP does not know its fair share or the propagation delay, it uses an adaptive algorithm to pick a good window size. After every second window of data it sends, TCP increases its window size by one packet. It decreases the window size by 50% each time it detects that the network has dropped a packet; TCP assumes that packet loss is caused by router queue overflow. In favorable circumstances, particularly when its window is larger than 4 packets, TCP can recover from the loss of a packet in one round trip time (RTT). This is called fast retransmit. In general, however, TCP pauses for at least one second before it concludes that a packet was lost and resumes sending. The intended result is that TCP's window size oscillate around a value that gives it a fair share of the network bandwidth.

TCP alone cannot ensure that it competes fairly with other flows: routers must cooperate by dropping packets fairly and thus causing TCP flows to reduce their window sizes. A router with a single FIFO queue that drops packets only when the queue is full turns out to be unfair. The state of the art in fair dropping for FIFO queues is a router packet dropping policy called Random Early Detection (RED) [7]. RED drops each incoming packet with a probability governed by the router's recent average queue length. This ensures that each flow sees the same packet loss rate.

3. Simulation Parameters

Most of the results in this paper are derived from simulations using NS 1.2a2 [11]. In each simulation, *N* TCP senders transmit through a shared bottleneck to an equal number of receivers, as shown in Figure 1. Congestion will occur in router A at the input to link L. Each sender starts at a randomly chosen time in the first 5% of the simulation time. Each sender has an unlimited amount of data to send.

The TCP version involved is Tahoe [19]. Receivers acknowledge every other packet. The routers use RED

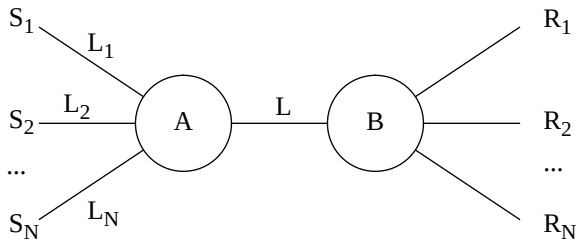

Figure 1: Simulation topology

and a FIFO queue. The routers have buffering equal to the average end-to-end delay-bandwidth product, as recommended by [20]. Five low-bandwidth telnet sessions compete with the main flows to help avoid deterministic behavior. Other simulation parameters are summarized in Figure 2.

Packet size	576 bytes
Maximum window	64 kilobytes
TCP time-out granularity	0.5 seconds
Propagation delay of L	25 ms
Bandwidth of L	10 megabits/second
Propagation delay of L_i	random, 0 to 50 ms
Bandwidth of L_i	10*(10/*N*) mbits/second
Router buffers	217 packets
RED min_thresh	217/4 = 54 packets
RED max_thresh	217/2 = 108 packets
RED max drop rate	0.2%
Simulation length	200 seconds

Figure 2: Simulation Parameters

The intent is that the bottleneck represent a customer access link or a heavily loaded backbone link. The sender links L_i have capacity proportional to each sender's fair share of the bottleneck on the theory that users are unlikely to pay for much more access capacity than they will be able to use. This means that the ratio of input capacity to output capacity in router A is fixed at 10 to 1 regardless of *N*, so any effects caused by varying *N* are due only to the number of flows. The packet size and propagation delay are chosen to approximate those typical on the Internet.

The numbers in Figure 2 imply that the average

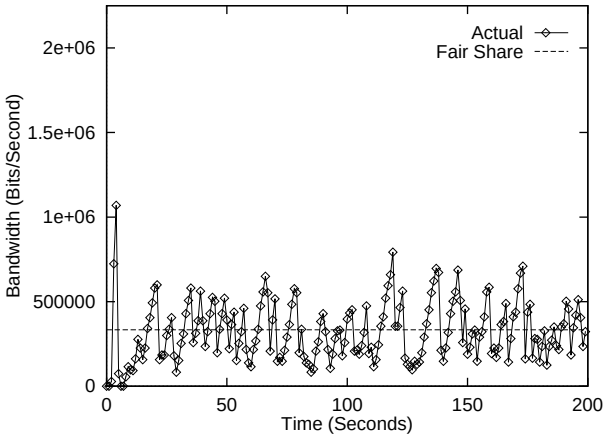

Figure 3: Bandwidth of One of the 30 Flows

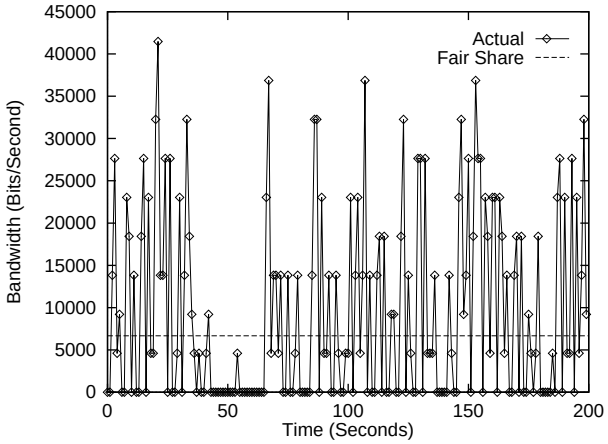

Figure 5: Bandwidth of One of the 1500 Flows

connection's one-way propagation time is 0.05 seconds. A 10 megabit link can forward 2170 576-byte packets per second, so a sender on an otherwise idle network can transmit 108.5 packets before the receiver sees the first packet. It takes another 0.05 seconds (or 108.5 packet times) for the sender to get the receiver's first ACK. A sender must keep a window of 217 packets in flight if it wishes to keep the network busy; if there are multiple senders, their windows must total at least 217. The network links effectively store these 217 packets.

4. Behavior with Few Flows

As a baseline for comparison with larger numbers of flows, we describe simulation results for just 30 competing TCPs. The bottleneck link efficiency, or goodput, is 97%; that is, the bottleneck link spends 97% of its time sending useful data and 3% of its time either idle or sending copies of packets already in some receiver's possession. Router A drops 0.7% of all packets.

This low loss rate is good. That it is also approximately correct is easy to prove; the key to the analysis is the frequency at which the TCPs' windows oscillate. RED ensures that Router A's queue length averages somewhere between the two RED thresholds, or roughly (108+54)/2 = 81 packets. The network also stores 217 packets in flight on the links, for a total of 298 packets. Each TCP's congestion window averages one 30th of 298, or about 10 packets, by oscillating between 7 and 13 packets. It takes TCP 12 round trip times to grow its window from 7 to 13 packets, after which (on average) the router will drop one of its packets. A round trip time is the propagation delay of 0.1 seconds plus a queuing delay averaging about 81 packet times, or 0.037 seconds. Thus each TCP can expect a drop every 12*0.137 or 1.7 seconds, and the total drop rate will be 30 packets per 1.7 seconds. The link capacity is 2170 packets per second, so 0.8% of all packets should be dropped. This is close to the simulation results.

TCP is also fair in this configuration. Figure 3 shows the bandwidth of one of the flows averaged over one second intervals. The horizontal line is one flow's fair

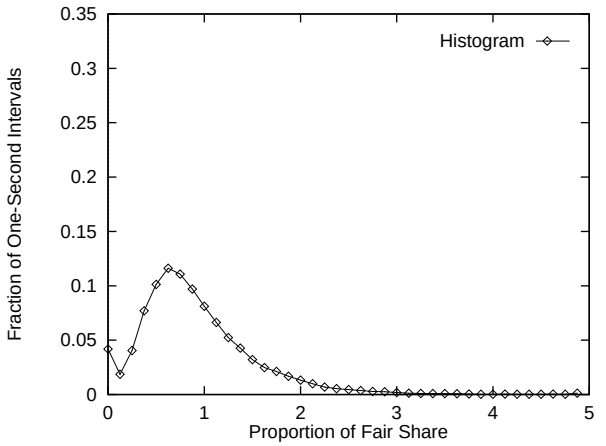

Figure 4: Fairness Histogram for 30 Flows

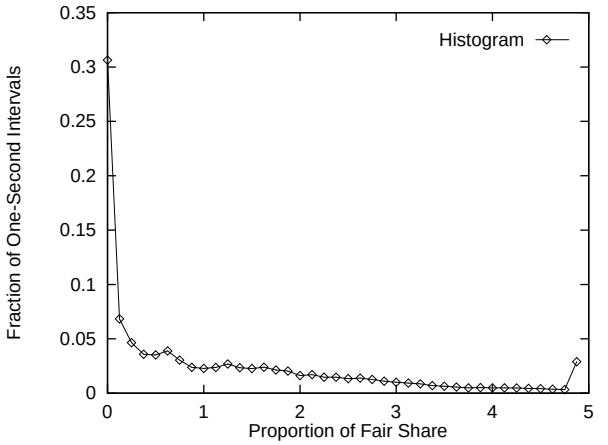

Figure 6: Fairness Histogram for 1500 Flows

share. Except during start-up, the flow never sustains a rate much higher or lower than its fair share, and never has to time out. The former means that TCP chooses reasonable window sizes; the latter indicates that TCP's fast retransmit mechanism is working well.

The bandwidth distribution shown in Figure 4 demonstrates that the 30 TCPs get fair shares even over relatively short periods. The x-axis units are fair shares of the link; an x value of 1 represents one 30th of 10 megabits/second. The y-axis is the probability of a flow in the simulation seeing a particular bandwidth for one second. Ideally all the flows would see exactly their fair share in every one-second interval, and Figure 4 would have a sharp peak at an x value of 1. One second is a guess at the smallest interval over which users might notice degraded performance. The actual simulation results in Figure 4, though not ideal, are good: a user would only encounter a whole second of zero bandwidth and high delay about 5% of the time.

5. Behavior with Many Flows

A simulation with 1500 flows also gets a total goodput of over 97%. In this case, however, the packet loss rate is 17%, high enough to cause TCP retransmission time-outs and wasted bandwidth along the paths to the bottleneck. The configuration also exhibits high variation in the bandwidth achieved by each flow, causing it to be unfair over intervals many seconds long. Figure 5 shows the bandwidth of one of the 1500 flows, averaged over one second intervals. Comparison with Figure 3 reveals much larger variation around the fair share: the flow often sends much too fast, and is often idle due to time-outs. Note that the y axes of these two graphs have the same scale relative to the fair share.

The bandwidth distribution in Figure 6 quantifies this high variation. Compared with Figure 4, Figure 6 shows that users will encounter substantially more intervals in which they get substantially more or less than their fair share. Figure 6 also predicts that users will encounter delays of one second or more with a probability of just over 30%.

The proximate causes of this variation are the high packet loss rate and consequent retransmission time-outs. Figure 7 shows the relationship between number of flows and packet loss rate for the simulated topology considered in this paper. These loss rates are enough to cause high time-out delays, and the steady upward trend is not a desirable feature of a congestion control system. Any evidence that real networks operate in this range would be a strong argument for understanding and fixing whatever causes this behavior.

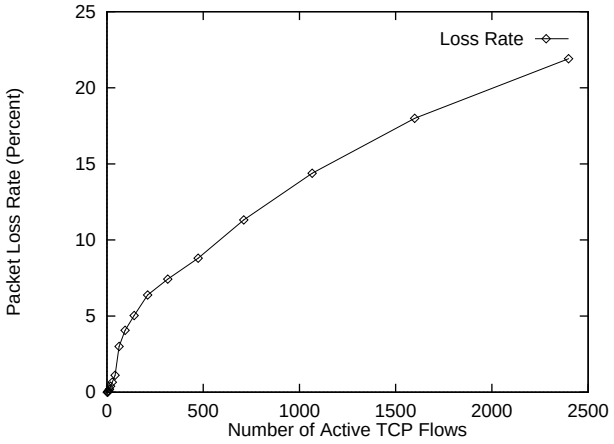

Figure 7: Number of Flows vs. Loss Rate

6. Internet Flow Counts

Analysis of three Internet packet traces suggests that some Internet links support large numbers of active flows. Each distinct combination of source and destination IP addresses and ports counts as a flow. The first trace [12], made available by Digital Equipment Corporation, was collected at 14:00 PST on a day in March 1995 on a 10 megabit Ethernet in Palo Alto that carried most of DEC's traffic to the Internet. Packets from 200 to 300 distinct TCP flows are visible in this trace in each second. The second trace [13] was collected at 14:40 EST on March 13 1997 on a 10 megabit Ethernet connecting Harvard's main campus to the Internet. This trace has packets from 300 to 400 distinct TCP flows in each second. The third trace [4] was collected on an FDDI ring at the FIX-West interchange point in September 1996. It has packets from over 2000 distinct TCP flows in each second. Pointers to these traces and the software used to analyze them may be found at http://www.eecs.harvard.edu/~rtm/traces.html.

A count of distinct flows in each second approximates the number of simultaneous active flows. Counts covering less than one second would miss TCPs in retransmission time-out. Counts over longer intervals would tend to treat short flows that do not overlap in time as simultaneous, though this effect will be present for any interval longer than a few round trip times. These counts treat the two directions of a bidirectional flow separately, and do not ignore acknowledgment packets. This seems appropriate since all the media involved were half-duplex, and since acknowledgments consume packet buffers and bandwidth. It might not be correct when considering full duplex transmission systems with separate buffer pools for each direction of each router port. The flow counts for the two Ethernets are comparable to other published numbers [5].

Assuming the traced networks are bottlenecks, these counts and Figure 7 predict that the router feeding the DEC network imposes a 6% packet loss rate and that feeding the Harvard network 8%. These are just the losses attributable to the interaction of TCP with router queuing policies; the simulations include no misbehaving or non-flow-controlled senders. These predicted loss figures are similar to those from empirical Internet studies [3, 8, 16, 21]. The FIX-West data could not be interpreted due to a lack of information about the FIX-West exchange topology.

Since there is evidence that Figure 7 is relevant, it is worth pursuing an explanation and a more general formulation of the problem.

7. Discussion

The general shapes of Figures 5 and 6 are an inevitable side-effect of window flow control with limited buffer space [1]. The storage in the simulated network totals 434 packets: 217 on the links and 217 in router A's buffers. Even if all flows used the minimum possible window size of one packet, only 434 of them could coexist. Another way to see this is that a window of one packet means that TCP must send a packet every round trip time. A round trip time is 0.2 seconds in this simulation when the maximum queuing time is included, so a TCP must send at least a packet of 576 bytes every 0.2 seconds, or 23,040 bits per second. This is one 434th of the bottleneck capacity. These considerations require that TCPs alternate between sending at more than their fair share and timing out. TCP has no mechanism that allows it to send slower than a few packets per round trip time, but faster than one packet per time-out. The result is the behavior in Figure 5.

Figure 7, however, is not inevitable. Sufficiently conservative algorithms might allow the loss rate to be constant as the load increases. TCP uses time-outs of at least one second with exponential backoff on repeated loss. TCP causes a high loss rate despite aggressive backoff because it jumps too abruptly from sending one packet per time-out to sending one or more packets per round trip time.

7.1. Proof of High Loss with Large *N*

We can gain some insight into the impact of TCP's time-out algorithms on packet loss by proving that the loss rate will approach 50% with sufficiently many flows. The proof depends only on the structure of TCP's backoff mechanism.

Let p be the steady-state loss rate with N flows. Let the state s_i be the set of TCPs in a 2^i second time-out, and n_i be the size of s_i. Note that a TCP moves between states in only two ways. A TCP in state s_i re-transmits a packet 2^i seconds after entering s_i. If the receiver acknowledges the packet, the TCP leaves s_i and resumes sending with a window. Otherwise the TCP moves to state s_{i+1}.

Lemma 1: n_0 is bounded by some constant independent of N. Each of the n_0 TCPs will retransmit a packet every second. For sufficiently large values of n_0, each retransmission must cause a loss, since the network has fixed capacity of less than n_0 packets per second. For any $p < 1$, there is some value of n_0 which will cause a loss rate of p. If n_0 grows with N, p must eventually exceed 0.5, in which case the theorem to be proved is true.

Lemma 2: $n_i = 2\,pn_{i-1}$. Each TCP in s_{i-1} retransmits after 2^{i-1} seconds. pn_{i-1} of these TCPs will suffer another drop and move to s_i. Thus TCPs move into s_i at a rate of $pn_{i-1}/2^{i-1}$ per second. Each will stay in s_i 2^i seconds. By Little's Theorem and the assumption that the system is in steady state, there must be $2pn_{i-1}$ TCPs in s_i.

Lemma 3: $n_i = (2\,p)^i n_0$ by recursive expansion of Lemma 2.

Lemma 4: The total number of TCPs in time-out is

$$\sum_i n_i \;=\; n_0 \sum_i (2p)^i$$

The maximum number of TCPs actively transmitting is a constant independent of N, as described in Section 7. Thus the number of TCPs in time-out must grow with N. The sum in Lemma 4 converges to a constant independent of N if $p < 0.5$. Thus the loss rate p must approach or exceed 50% as N grows large.

This proof lends credibility to Figure 7. It also suggests that TCP's behavior with large numbers of flows depends largely on its backoff algorithm. The high loss rate could be addressed in at least three ways. The backoff constants could be increased, thus decreasing the value of p needed to make the sum in Lemma 4 diverge. This doesn't seem wise: a few losses could drive a TCP into long backoff. We speculate that the backoff algorithm could use a linear decrease in inter-packet time after a success, rather than jumping directly from a rate of one packet per 2^i seconds to one packet per round trip time. Best would be to avoid reliance on retransmit time-out for congestion control. This could be done by making TCP's congestion window work well for rates less than one packet per round trip time.

7.2. N^2 Loss and Window Policy

Figure 8 is a detail from Figure 7. With fewer than 60 flows the loss rate seems to increase more than linearly with N. This is no illusion, as can be shown with

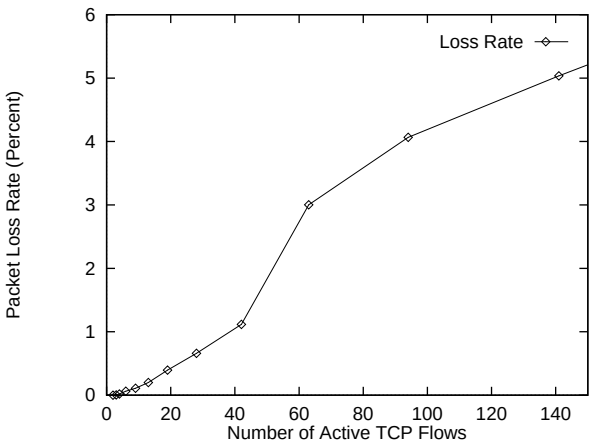

Figure 8: Detail of Flows vs. Loss Rate

an argument adapted from [9]. For values of N much less than the total buffering divided by four, fast retransmit allows N TCPs to share the bottleneck without time-outs. Each TCP increases its window by half a packet each round trip time, so the sum of the windows increases by $N/2$ each RTT. Since the number of router buffers is fixed, the frequency of loss episodes is proportional to N. Each loss episode involves the loss of one packet from each flow. Thus for modest numbers of flows the loss rate is proportional to N^2.

TCP's window increase policy is linear in the sense that it increases the congestion window by a fixed amount per round trip time. However, with very small windows the increase as a fraction of window size is large. In order to keep losses low, TCP should increase its window no more than linearly in the amount of data sent. For instance, when the window is small, TCP could increase the window at a rate that will double it every 100 packets.

The regrettable features of Figure 7, high loss rate with large N and loss proportional to N^2 with small N, arise from the interaction of limited network buffering with specific TCP mechanisms: window-based congestion control, exponential backoff, and the window increase policy.

8. Provisioning Buffer Space

The most obvious way to support large numbers of users is to provide more storage in the network with higher bandwidth or more buffering. For our purposes they are equivalent; we consider increasing buffering here because it is more practical.

Even with small numbers of flows, bottleneck routers should have at least one delay-bandwidth product of buffering [20]. In this case, the TCPs sharing the bottleneck will have windows summing to two delay-bandwidth products just before each congestion episode. After the TCPs cut their windows in half, they will sum to one delay-bandwidth product, just enough to allow full utilization of the bottleneck link. This is clear from

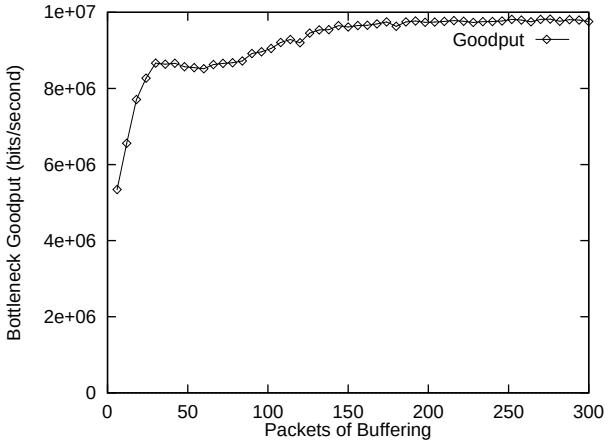

Figure 9: Buffers vs. Goodput for 30 TCPs

Figure 9, which shows the average bottleneck utilization as a function of the amount of buffer space. This graph was derived from simulations of 30 TCPs in the configuration described in Section 3. Recall that the delay-bandwidth product is 217 packets.

With large numbers of users, total goodput is good even with small numbers of buffers: there are always more than enough TCPs sending packets. The buffer space does affect the drop rate and the fairness. Figure 10

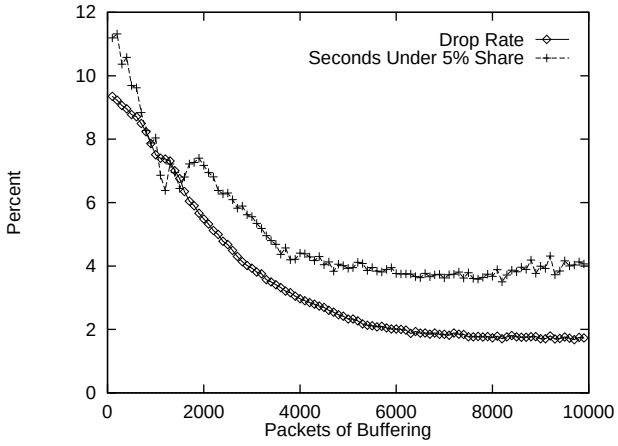

Figure 10: Buffers vs. Dropping and Fairness, 500 TCPs

shows the results of simulations with 500 TCPs and varying amounts of router buffer space. The line marked Drop Rate is the percent of packets dropped by the bottleneck router. The line marked Seconds Under 5% Share indicates the percentage of seconds over all the TCPs in which a TCP experienced less than 5% of its fair share. This amounts to the probability of a TCP being in a time-out. Both measures improve significantly with

increased buffer space until there are about 10 buffers per TCP. The decrease in drop rate with increase in sustainable window size is well known [9]. That fairness should improve as drop rate decreases seems natural, since drops can cause time-outs. The most curious feature of the fairness graph is the irregularity between 1500 and 2000 packets. At this point TCP's fast retransmit mechanism allows most of the flows to recover from losses without time-outs; fast retransmit requires a window of at least four packets.

A drawback of building routers with very large buffers is that excessive queues may build up even with few flows. Solutions to this problem are known [15, 2, 10]; one approach is for routers to impose a higher loss rate on flows with many packets queued than on those with few queued.

9. Conclusion and Future Work

Some of the known theoretical limits of window flow control and of TCP's algorithms may be actual problems on the Internet, as demonstrated by observation of Internet traffic and simulation. Predicted loss rates and unfairness suggest that these limits may cause a substantial loss in Internet performance. The easiest solution to these problems may be radically increased router buffer space coupled with limits on per-flow queue length.

More analysis of Internet traffic is needed to count simultaneous flows and confirm their relationship to loss rate.

There is room for more work on flow control mechanisms that maintain a low loss rate regardless of load. TCP would adapt better to large numbers of flows if it used rate control when the window size is small [6]. Its window increase algorithm could be modified to maintain a low loss rate at small window sizes. Finally, TCP could use the same exponential/linear approach for retransmit backoff as for window size, or it could merge the two mechanisms.

10. References

[1] BERTSEKAS, D. and GALLAGER, R., *Data Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[2] BRAKMO, L., O'MALLEY, S., AND PETERSON, L., TCP Vegas: New Techniques for Congestion Detection and Avoidance, *Proceedings of the SIGCOMM 1994 Conference*.

[3] BOLOT, J-C., End-to-end Packet Delay and Loss Behavior in the Internet, *Proceedings of the SIGCOMM 1993 Conference*.

[4] claffy, k., FIX-West Network Traces, ftp://ftp.nlanr.net/Traces/FR+/960926, National Laboratory for Applied Network Research, 1996.

[5] CLAFFY, K., BRAUN, H-W., and POLYZOS, G., A Parameterizable Methodology for Internet Traffic Flow Profiling, *IEEE Journal on Selected Areas in Communications*, March 1995.

[6] ELDRIDGE, C., Rate Controls in Standard Transport Layer Protocols, *ACM Computer Communication Review*, v. 22 n. 3, July 1992.

[7] FLOYD, S., AND JACOBSON, V., Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, August 1993.

[8] HANDLEY, M., *An Examination of MBone Performance*, University of Southern California Information Sciences Institute Research Report ISI/RR-97-450.

[9] JACOBSON, V., Congestion Avoidance and Control. *Proceedings of the SIGCOMM 1988 Conference.*

[10] LIN, D., AND MORRIS, R., Dynamics of Random Early Detection, *Proceedings of the SIGCOMM 1997 Conference*.

[11] MCCANNE, S., and FLOYD, S. NS (Network Simulator), 1995, http://www-nrg.ee.lbl.gov/ns/.

[12] MOGUL, J. DEC-PKT-4, http://town.hall.org/Archives/pub/ITA/html/contrib/DEC-PKT.html, Digital Equipment Corporation, March 1995.

[13] MORRIS, R., and WANG, S., Harvard Network Traces, http://www.eecs.harvard.edu/net-traces/, Harvard University, March 1997.

[14] NAGLE, J., Congestion Control in TCP/IP Internetworks, *ACM Computer Communication Review*, October 1984.

[15] NAGLE, J., On Packet Switches with Infinite Storage, *IEEE Transactions on Communication*, Vol. 35, pp 435-438, 1987.

[16] PAXSON, V., End-to-End Internet Packet Dynamics, *Proceedings of the SIGCOMM 1997 Conference*.

[17] POSTEL, J., Transmission Control Protocol, RFC793, 1981.

[18] ROMANOW, A., AND FLOYD, S., The Dynamics of TCP Traffic over ATM Networks, *Proceedings of the SIGCOMM 1994 Conference*.

[19] STEVENS, W. R., *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.

[20] VILLAMIZAR, C., and SONG, C., High Performance TCP in ANSNET. *ACM Computer Communication Review*, v. 24 n. 5, October 1995.

[21] YAJNIK, M., KUROSE, J., AND TOWSLEY, D., Packet Loss Correlation in the MBone Multicast Network, *IEEE Global Internet Conference*, London, Nov 1996.

[22] ZHANG, L., SHENKER, S., AND CLARK, D., Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic, *Proceedings of the SIGCOMM 1991 Conference*.