High Performance TCP in ANSNET

Curtis Villamizar <curtis@ans.net>
Advanced Network & Services, Inc.

Cheng Song <csong@vnet.ibm.com>
Advantis

September 12, 1994

Abstract

This report concentrates on specific requirements and goals of the research networks supported by ANSNET, but applies to any TCP dominated high speed WAN and in particular those striving to support high speed end–to–end flows. Measurements have been made under conditions intended to better understand performance barriers imposed by queueing capacities and queue drop strategies.

The IBM RS/6000 based routers currently supporting ANSNET performed very well in these tests. Measurements have been made with the current software and performance enhanced software. Single TCP flows are able to achieve 40 Mb/s and competing multiple TCP flows achieve over 41 Mb/s link utilization on 44.7 Mb/s DS3 links with delays comparable to US cross continent ANSNET delays. Congestion collapse is demonstrated with intentionally reduced queueing capacity and using window sizes much larger than optimal.

A variation of Floyd and Jacobson's Random Early Detection (RED) algorithm [1] is tested. Performance improved with the use of RED for tests involving multiple flows. With RED and queueing capacity at or above the delay bandwidth product, congestion collapse is avoided, allowing the maximum window size to safely be set arbitrarily high.

Queueing capacity greater than or equal to the delay bandwidth product and RED are recommended. RED provides performance improvement in all but the single flow case, but cannot substitute for adequate queueing capacity, particularly if high speed flows are to be supported.

Contents

1 Introduction

TCP performance over WANs may be limited by numerous factors including both host (end–system) and router (packet switch) limitations.

The NSFNET has been improving upon the performance bounds in wide area IP networking since its inception. Performance capabilities have now reached the point where the current 44.7 Mb/s[1] can be almost fully utilized by either single high speed flows, aggregations of TCP flows, or arbitrary combinations of high speed flows and lower speed flows.

The Maui High Performance Computing Center (MHPCC) has been established in Hawaii to serve the computational needs of the Air Force Maui Optical Station (AMOS). MHPCC also aims to establish itself as a national supercomputing resource. The challenges posed by MHPCC due to it's high propagation delays to the mainland US and high bandwidth needs are described in [2].

Both the NSFNET service and the MHPCC service are provided by ANSNET. Two current requirements of NSFNET and MHPCC are not as easily met as one might expect and are the primary focus of this work. The NSFNET service must accommodate further increases in network load in an IP network dominated by TCP traffic without experiencing congestive collapse (low utilization in the face of high load). High bandwidth applications must be accommodated in support of the NSFNET sponsored supercomputing centers and in support of MHPCC needs.

Statistics gathered for NSF [3] show a doubling of traffic approximately every year. DS3 Link utilizations have exceeded 45% of link capacity over 15 minute intervals and daily peaks in the low 40% range are now common [4]. Congestion loss measured over 15 minute intervals had until recently been well under 10^{-4} (one lost packet in 10^4) for ANSNET DS3 and FDDI links serving the NSFNET [5]. Loss of 10^{-3} for 15 minute intervals and at time close to 10^{-2} has occurred primarily in the NY and Washington areas, indicating that congestion is now beginning to occur. Traffic growth statistics indicate that, though the problem is still acute, congestion is likely to become more common.

The requirement to support large aggregations of traffic is of interest for any large deployment of IP, including existing or planned commercial IP service. The requirement to support very high speed flows may be unique to the scientific fields, but as can be seen later, meeting this more difficult requirement better assures meeting the former.

2 TCP Protocol Dynamics

The Transmission Control Protocol (TCP) is described in RFC–793. TCP is a connection oriented transport protocol capable of providing reliable end–to–end service over a best–effort connectionless protocol such as IP. The foundations of the algorithms most commonly in use prior to the late 1980s are described in RFC–813, RFC–879, RFC–889, RFC–896 and Nagle [6]. The TCP slow start and congestion avoidance algorithms were introduced in 1988 by Van Jacobson [7]. These algorithm affect timing rather than the underlying protocol, and remain interoperable with earlier implementations. Although earlier implementations would continue to function, there has been a concerted effort to eliminate implementations that predate [7] due to their tendency toward congestion collapse under high bottleneck utilizations.

Interoperability with earlier implementations of TCP has remained a priority (despite arguments to the contrary as in [13]). Improvements to the algorithms and extensions to TCP have been discussed in numerous RFCs (RFC–1072, RFC–1106, RFC–1110, RFC–1185, RFC–1323) and in the open literature (too numerous to cite).

Compatibility with earlier TCP implementations can be maintained by using the TCP options defined in RFC–793 for extensibility. The latest commercial TCP implementations support the algorithm improvements and extensions described in RFC–1323 and preserve compatibility with earlier implementations.

Of particular interest to this work is TCP's congestion avoidance, fast retransmit, and fast recovery algorithms and how queueing capacity and queue drop strategy impacts the effectiveness of these algorithms. A large factor in TCP's widespread use is its ability to fully (or nearly fully) utilize a network bottleneck, adapting quickly to changes in offered load or available bandwidth.

TCP is able to quickly detect a dropped segment using the fast retransmit algorithm and is able to avoid waiting a full RTT for acknowledged window pointer to be advanced as a result of the retransmission. These algorithms were originally introduced by Van Jacobson in electronic mail to the end2end-interest list, a working group studying TCP. These algorithms are also described in [19].

A summary of these algorithms and their some of their characteristics is presented here. For a more thorough treatment of the subject see [19] or the cited papers.

2.1 TCP Segment Size

TCP is byte oriented, but prefers to send data in fixed sized packets whose payload plus TCP header overhead is referred to as the TCP segment size. At high speeds a large segment size is generally required.

A large segment size will reduce the per packet overhead at the hosts [20, 21, 22]. For most routers, a larger segment size (as large as can be supported by the path MTU) will more efficiently use available queueing ca-

[1] Clear channel DS3 clocks at 44.7 Mb/s and provides 44.21 Mb/s after DS3 framing.

pacity. Inefficiencies in many routers are due to smaller packets often occupying the same buffer storage as large packets.

Setting the segment size too high will cause packet fragmentation, which is much worse. Packet fragmentation puts a load on the router that has to do the fragmentation. More important, at high speeds fragment reassembly on the receiving host can become the performance limiting factor.

Hosts typically use a segment size which provides a payload of a multiple of 512 or 1024 bytes. For transfers on the same subnet, the MTU of the local interface dictates the segment size. Hosts typically use a TCP payload of 512 bytes when the transfer is not to a directly attached subnet. The small segment size insures that fragmentation will not occur, but can reduce performance relative to an optimal segment size.

For high speed transfers, MTU discovery as defined in RFC–1191 is often used to determine the true MTU of the path and set segment size accordingly. MTU discovery will not overestimate MTU and cause packet fragmentation unless all ICMP generation is disabled in routers on the path (as RFC–1435 warns).

2.2 TCP Maximum Window Size

TCP is an adaptive windowed protocol. TCP will send up to one current window size of data without requiring acknowledgments (ACKs). The current window size and the end–to–end round trip time (RTT) of a TCP flow place a ceiling on the average offered load. Any MAN or WAN experiences transmission delays due to circuit mileage. For example, regardless of link bandwidth, it typically takes a signal 35 msec to cross the continental US, yielding a 70 msec RTT.

In practice TCP will not exceed a maximum window size determined by the amount of queueing space that the sender and receiver are willing to commit to. This practice is not dictated by the TCP protocol. In BSD (and therefore most Unix systems) the default maximum window size is the value of the `tcp_sendspace` and `tcp_recvspace` kernel variables. These can be modified by the `SO_SNDBUF` and `SO_RCVBUF` setsockopt options.

In older BSD and Unix TCP implementations, the default values of the `tcp_sendspace` and `tcp_recvspace` kernel variables were typically 4 kB. These values were increased to 8 kB in more recent implementations. These values yield performance limitations of 0.45 and 0.9 Mb/s respectively, on US cross continent (70 msec RTT) transfers. Shorter delays allow higher throughput to be achieved. For example, a 20 msec RTT would allow 1.6 and 3.2 Mb/sec, respectively.

Implementations that do not support RFC–1323 extensions place an upper limit of 64 kB on maximum window size. If the window size is 64 kB and bandwidth is infinite, the highest throughput a TCP flow could achieve is accomplished by sending 64 kB every 70 msec. This yields a 7.3 Mb/sec limit over a 70 msec path. RFC–1323 eliminates the maximum window size limitation and therefore the bandwidth limitation. For high speed applications, it is therefore critical that RFC–1323 capable TCP implementations be used and that window size limits are increased.

2.3 TCP Congestion Avoidance

Any point in a network where offered load can exceed available bandwidth, even temporarily, forms a bottleneck. With well designed equipment a bottleneck will be a link, but it could also to internal to packet routing or switching equipment.

If a bottleneck forms, some or all TCP flows traversing that bottleneck must reduce their offered load. The TCP congestion avoidance algorithm uses the packet drops that occur as a result of queue overflow as feedback to reduce the sending rate. Such a drop will generally occur if a sustained ingress flow exceeds egress capacity. The reduced sending rate is accomplished by cutting the current window size in half. The window size is then gradually increased in an attempt to find the optimal operating point.

If the queue is large enough, transmission will resume after a short pause and the bottleneck will remain at or very near full utilization. During the pause in transmission, the link is fed by the backlog of data in the queue, allowing the queue to empty somewhat but keeping the link utilization from dropping. Note that if queueing capacity is sufficiently inadequate, a drop can also occur simply due to burstiness and occur well below full average bottleneck utilization.

TCP's slow start and congestion avoidance involves an initial exponential growth in window size starting with a one segment (packet) window. With exponential growth in the window size, a packet drop will soon occur. The packet drop forces a reduction in window size and then gradual increase in window size. If one packet is lost within a window, the window size is halved and and then gradually increased. If a timeout occurs, TCP repeats the slow start agorithm, restarting with a window size of one. Exponential growth in the window size is allowed until half the previous window size is reached and then growth slows from that point on.

2.4 Fast Retransmit and Recovery

When a loss occurs, senders will cut their windows in half. The TCP fast retransmit algorithm detects a loss very quickly by counting three consecutive acknowledgments for the same segment (duplicate ACKs). When the fast retransmit algorithm is triggerred, one full segment just beyond the acknowledgments window is retransmit. The fast recovery algorithm reduces the congestion window by half in response to triggering fast retransmit and keeps the amount of unacknowledged data fixed at that level until the acknowledged window pointer is advanced. Fast recovery retransmits one packet as allowed by the window size for each remaining duplicate ACK, until the acknowledged window pointer is advanced, indicating that the retransmit and retransmit acknowledgment has completed its round trip.

As a result of fast retransmit and fast recovery, when an isolated packet drop occurs, after three duplicate ACKs one packet is retransmitted. The sender then becomes idle until acknowledgments for one half of the previous window arrive in the form of duplicate ACKs. If the queue is not quite large enough, the queue will empty before transmission resumes. After one RTT, the acknowledged window pointer is advanced by a large increment, and linear growth of the congestion window can resume.

If single packet drops are the norm, the TCP window size will oscillate about the optimal size, using close to full bottleneck bandwidth and only occasionally retransmitting. If demand changes at the bottleneck as the result of new flows becoming active, or flows becoming inactive or closing, by oscillating about an optimal window size, TCP adapts quickly.

2.5 Performance Pitfalls

If a drop occurs and is not detected by fast retransmit, a TCP retransmit timeout will occur. The retransmit timer is set to an estimate of RTT times four times the estimated mean deviation of RTT, if an accurate estimate of RTT can be made. If not, the retransmit timer defaults to two seconds. Retransmission timeouts can drastically reduce TCP's performance. TCP will remain idle for the timeout duration, and then slow start.

Fast retransmit and fast recovery improve performance over reliance on TCP timeouts. However these improvements can be lost if the window size is very small or if multiple packet drops occur within one RTT.

If TCP is operating with a very small window (less than 4 segments) and loses a packet, fast retransmit cannot be triggerred. If the window is large enough but a second packet is lost within the same window (a loss prior to the retransmit occurring), acknowlegements following the retransmit will advance the acknowledged window pointer, but only to the second drop. If enough packets have been sent after the second loss and have not been lost, fast retransmit will trigger again. If not a timeout will occur.

With a very large window, if congestion causes a number of packets to be dropped, but always at intervals of four or more packets, fast retransmit and fast recovery can be triggerred a number of times in one RTT. This can cause the window size to be reduced in half a number of times and then grow slowly. This can also reduce utilization by causing TCP to underutilize the bottleneck for a number of RTTs while the window grows.

3 Queue Size Requirements

A parameter critical to TCP performance is the delay bandwidth product (D*BW) of TCP flows. This is the total round trip time (RTT) times the bandwidth available at the bottleneck. If there is a single TCP flow (or more likely a single high speed TCP flow dominating the offered load) at a bottleneck, the window size must be able to reach at least the delay bandwidth product in order to make effective use of the available bandwidth. If the maximum window size is set beyond this limit, the current window size will tend to exceed the D*BW value, resulting in congestion. This will force queueing drops, and cause the window size will oscillate about an optimal value.

For most networks it is impossible for a given user to predict the bandwidth requirements of all other potential users at any given time. Therefore when setting the window size for a high speed application, it is wise (unless queueing capacity is inadequate) to set the window size at or slightly below the D*BW product that would be available if the slowest physical link were otherwise idle and let TCP determine the optimal window size if there is other traffic competing for the bottleneck.

3.1 Multiple TCP Flows

If there are many TCP flows sharing a link, the delay bandwidth product of each flow is that flow's RTT times its share of the link bandwidth. The sum of these D*BW products is proportional to the required queueing capacity. In practice, TCP has been shown to quite equitably share bandwidth among flows (though not perfectly [25]) and achieve fairly high utilization if conditions are right[2].

The queueing requirements of the router or switch at the congestion point is fairly independent of the number of TCP flows. Where there are multiple flows, each flow will use a portion of the bottleneck bandwidth. The sum of these throughputs will be approximately equal to

[2]Random packet loss must be low. Sudden increases in delay of more than the previous average delay should not occur. Delays should not exceed 2 seconds. As we examine later, an important condition is adequate queueing capacity at the congestion point. These are among the most important factors needed to achieve high bottleneck utilization over long delay paths.

the bottleneck bandwidth, if all is working well. Even if bandwidth is unevenly distributed among flows due to bottlenecks elsewhere, if the RTTs are similar the sum of the delay bandwidth products will generally be very close to the average delay times the total bandwidth available at the bottleneck. Except for small numbers of flows, such as 1 or 2, it makes little difference how many simultaneous flows are active. For queueing capacity requirements, the worst case data burst remains approximately equal to the D*BW product using the average delay and the link bandwidth.

If multiple TCP flows are active at a bottleneck, but remain unsynchronized, queueing requirements will be reduced with the number of flows. Unfortunately, TCP flows do tend to become increasingly bursty and can become well synchronized. Synchronization (clustering) is described in [26].

An important factor in bidirectional TCP traffic is "ACK compression", first described in [27]. In ACK compression a series of acknowledgments in one direction are queued behind a burst and become compressed in time. This results in a much larger burst of data in the other direction after the compressed stream of ACKs arrives.

Synchronization can begin when queue overflow occurs and results in loss of packets from more than one flow. If the delays of some of the flows are approximately equal (as is the case for all US East coast to West Coast flows) the TCP flows will become fairly synchronized and result in even larger bursts. Later queue overflows may bring more flows into synchronization with the group.

3.2 Effects of Queueing Capacity

A poorly engineered network bottleneck can become bistable. A period of fairly high (or even full) utilization may be followed by a stable period of loss and substantial bottleneck underutilization. This second condition is a form of congestion collapse. Congestion collapse due to poor TCP congestion control in early TCP host implementations is described in [7]. A similar condition can occur if queueing capacity at the bottleneck (the router) is inadequate, though the performance degredation will not be as severe as the collapse described in [7].

Problems associated with inadequate queueing will not be observed in testing routers or switches unless there are sufficient delays introduced in the test environment. Therefore inadequate queueing capacity can easily be overlooked. When such equipment is later deployed in a WAN the delay is unavoidable and the problems are then observed.

If queueing capacity is grossly inadequate, as link utilization grows, premature loss may occur long before full bottleneck utilization is achieved due to the bursty nature of IP traffic. If queueing capacity is adequate or nearly adequate, loss will tend not to occur until a period of somewhat sustained full bottleneck utilization. This period can be a few hundred milliseconds or longer.

The result of a nearly sufficient queueing capacity can be somewhat less than full average bottleneck utilization, though collapse is avoided. If queueing capacity is truly adequate, full utilization can be sustained indefinitely until offered load simply goes away. If queueing capacity is highly inadequate, some degree of synchronization will occur and alternate periods of very low and very high utilization can occur, resulting in very low average utilization.

Recent work by Floyd and Jacobson [1] aims among other things to reduce the synchronization of TCP flows through actions take by the router or switch. The algorithms are referred to as Random Early Detection (RED). RED limits queue utilization by carefully introducing controlled pseudo-random feedback (in this case packet drops).

4 Performance Testing

Performance testing was conducted to demonstrate effects of queueing capacity and better quantify queueing capacity requirements. Among the more immediate goals are determining whether improvement is needed to the IBM RS/6000 based routers used in ANSNET and what forms of further improvement can be made.

4.1 Test Network Conditions

Testing was done on the ANS test network. This is an unchannelized DS3 network like the ANSNET network that provides the NSFNET service (see [28] for a description of NSFNET). The test network uses routers and other equipment identical to those used on ANSNET. Like many NSFNET sites, some of the test network host have FDDI LAN attachments with direct DS3 connections to the network core and are capable of providing sustained high load on the WAN.

One of the paths used in testing was a 20 msec, 7 IP hop, fairly direct path between NY and Michigan. This path traverses 4 FDDI rings and 3 DS3 circuits. The delay between the same two hosts can be increased to 68 msec by taking down an interface and forcing an 8 hop path that goes by way of Texas. This longer path traverses 4 FDDI rings and 4 DS3 circuits. The 68 msec test network RTT conveniently matches the 70 msec US continental RTT.

Tests have been performed using TCP maximum window sizes in the range of 8 kB to 512 kB. The 20 and 68 msec paths correspond to D*BW products of about 110 kB and 380 kB. A TCP segment size of 4096 plus

designation	description
60	Effective queue capacity per interface intentionally reduced to 60 2 KB buffers. There is a maximum of 44 buffers allowed per peer card on the transmit side.
r60	Effective queue capacity per interface intentionally reduced to 60 2 KB buffers on the transmit side. This build contains an implementation of RED. Additional buffers on the input side serve primarily to buffer bus transfer.
224	Queueing capacity is increased to 224 2 KB buffers for FDDI and 320 2 KB buffers for T3. There is a maximum of 128 buffers allowed per peer card on the transmit side.
r200	Queueing capacity is 200 2 KB buffers on the transmit side. This build contains an implementation of RED. Additional buffers on the input side serve primarily to buffer bus transfer.

Table 1: RS/960 Code Used in Testing

TCP header and a window of D*BW yields windows of about 28 and 98 packets.

The primary bottleneck in all of these tests is the first DS3 circuit fed by the ingress FDDI ring. For the case of a single bottleneck or highly dominant bottleneck, the location of the bottleneck along the path should have no effect on results. The time required for the effects of feedback (drop) to begin to take effect remains one RTT.

4.2 Router Queueing Capacity

The RS/6000 based routers used within ANSNET use intelligent custom interface adapters known as RS/960 cards. These cards contain full routing tables and forward packets directly over the RS/6000 MCA bus without involving the main processor in forwarding.

Each RS960 card provides up to a maximum of 1 MB of packet queueing capacity. Packet buffers are 2 KB each, allocated for either the receive side queueing or the transmit side queueing.

The receive side queueing provides buffering for packets received from the interface prior to being queued for MCA card to card transfer. The transmit side queueing provides buffering for packets waiting for transmission on a network interface.

The buffer allocation scheme is based on dynamic sharing of common pools of buffers with a fixed amount of reserved buffers guaranteed for each card to card packet transfer. On the transmit side, a number of buffers is preallocated for use by a peer card so that packet DMA transfer can be set up immediately. The transmit side code garantees a fixed number of such allocated buffers (a tunable parameter) regardless of peer traffic intensity. To accommodate the dynamic variation of traffic intensity from different peers, more buffers can be allocated from the transmit side common buffer pool. On the receive side, a large number of buffers are reserved to receive packets from network interfaces with non-reserved buffers gathered in a common pool available for packets to be transferred to peer cards. The size of the two common buffer pools are controlled by two parameters, tuned according to the traffic load and characteristics observed on the ANSNET backbone.

A number of builds of software for the IBM RS/960 FDDI and T3 adapters were used. The RS/960 build designations used in this paper contain the approximate queueing capacity and the letter "r" for RS/960 code containing a RED implementation. The RS/960 build designations `60`, `r60`, `224`, and `r200` are described in Table 1. For the builds that do not implement RED, the queue size designation includes both receive and transmit side buffers.

The `r200` build includes a variation of RED implemented on the transmit side [1, 29]. RED relies on tracking average queue utilization. In our experimental code, we avoid the extra software overhead for the transmit side to determine the number of receive side buffers currently in use for a peer traffic. Therefore, RED implementation ignores the receive side buffering and starts dropping packets when the transmit side queue size approaches full. This reduces the effective queue capacity to the capacity of the transmit side.

4.3 Traffic Sources

Simulation of Internet traffic is difficult problem in itself. It is well accepted that packet arrivals over a LAN or WAN cannot be modeled as Poisson arrivals.

Researchers at USC and LBL propose simulating TCP and modeling session arrivals and durations based on empirical studies of traffic [30, 31]. In [32] traffic is modeled as self-similar stochastic processes. Paxson and Floyd provide a review of much of this work in [33]. Partridge sums up the importance of adequate traffic models in [34], pointing out that the use of simple models, particularly the Poisson model, may lead to underestimates of the amount of buffering required in ATM switches and IP routers.

The pragmatic approach taken here is to concentrate on supporting single high speed TCP flows and small numbers of simultaneous TCP flows under the assumption that these cases are likely to be far more bursty than aggregations of very large numbers of slower TCP flows. The cases being studied also constitute worst case situations which may arise if supercomputer centers are encouraged to make use of high bandwidth applications.

designation	description
no-b	No background traffic was used.
revb	A TCP flow was started in the reverse direction with a window size of 256 KB for the 68 msec path and 64 KB for the 20 msec path.

Table 2: Background Traffic Used in Testing

The entry level for hosts that implement RFC–1323 and can saturate a FDDI ring with TCP data are high end workstations. A pair of SGI Indy/SC running IRIX 5.2 was used. IRIX 5.2 TCP is based on BSD Reno with both fast retransmit and fast recovery and contains RFC1323 extensions based on the University of Illinois research prototype written by Thomas Skibo [35].

The IRIX **allnetsarelocal** kernel variable was set (setting **mtudiscovery** would also have worked). Many kernel size limits had to be increased radically, particularly those related to **mbuf** structures. The number of allowed free mbufs had to be made very large to prevent freeing and reallocating mbufs excessively.

The multiple flow tests provide a very measurable background (the other flows). This provides a means of observing how effectively TCP is able to share bandwidth, with different queueing capacities. The effects of incompressible data (data which does not react to congestion drop) and the effects of small packet traffic cannot be measured using multiple flows as background.

A reverse traffic flow is also used. The reverse traffic flow provides a very bursty flow of small packets (TCP ACKs) and forces ACK compression of the forward data flow. The window size for the reverse flow was chosen to be small enough to provide a high link utilization, but below D*BW, so as not to cause congestion in the reverse direction.

Table 2 summarizes the background cases that were used. Other traffic conditions were tried including up to 2.5 Mb/s of UDP small packet traffic, but none produced very interesting results.

4.4 Summary of Test Conditions

The primary measurement tool was the TTCP program. TTCP is a public domain network measurement tool originally written by the US Army Ballistics Research Lab. The version used was obtained from **ftp.sgi.com**. Tests were performed using 1, 4, and 8 TCP flows. In single TCP flow tests window sizes of up to 512 kB were used. In the multiple flow tests, the total of all of the window sizes was allowed to exceed 512 kB (reaching as high a 2 MB), though in these tests the individual TCP flows were limited to windows of 512 kB or less.

New York to Michigan by way of Texas

68msec RTT Single Flow	60	r60	224	r200
no-b	39.89^{66}	40.02	39.95	40.07
revb	22.97^{54}	33.48	35.29	33.59

68msec RTT Four Flows	60	r60	224	r200
no-b	36.28^{32}_{11}	40.84^{26}_{19}	41.16^{19}_{10}	41.12^{11}_{34}
revb	25.16^{14}_{10}	33.36^{12}_{14}	38.68^{15}_{15}	40.56^{13}_{15}

68msec RTT Eight Flows	60	r60	224	r200
no-b	-	41.04^{17}_{10}	41.20^{12}_{09}	41.20^{08}_{19}
revb	28.32^{05}_{07}	36.00^{06}_{07}	39.68^{08}_{07}	40.88^{08}_{17}

Entries list the highest TCP throughput achieved (mb/s) and the window size limit (KB) used to achieve that result. See text for the meaning of the superscripts and subscripts.

Table 3: Test Conditions for 68msec RTT

New York to Michigan direct

20msec RTT Single Flow	60	r60	224	r200
no-b	-	40.78^{61}	40.74^{56}	40.77
revb	38.13^{70}	40.66^{62}	40.77^{57}	40.77

20msec RTT Four Flows	60	r60	224	r200
no-b	-	41.20^{20}_{12}	41.20^{13}_{16}	41.20^{10}_{23}
revb	37.52^{28}_{10}	41.08^{18}_{11}	40.80^{11}_{11}	41.00^{08}_{17}

20msec RTT Eight Flows	60	r60	224	r200
no-b	-	41.12^{06}_{14}	41.12^{02}_{42}	41.20^{01}_{14}
revb	38.96^{15}_{13}	41.04^{06}_{11}	40.88^{02}_{12}	41.04^{01}_{10}

Entries list the highest TCP throughput achieved (mb/s) and the window size limit (KB) used to achieve that result. See text for the meaning of the superscripts and subscripts.

Table 4: Test Conditions for 20msec RTT

5 Test Results

The highest queueing capacity tested provided under one D*BW for the 68 msec path. The D*BW for this path is approximately 380 kB. The 224 2 kB buffers provide an effective queueing capacity of about 300 kB since the TCP packets are over 4 kB in size and require 3 buffers each. This queueing capacity seemed generally adequate in preventing significant congestion degradation but was not sufficient to allow full bottleneck utilization.

Table 3 and Table 4 list the highest TCP throughput achieved in a representative set of tests. Tests were not performed for all possible combinations of conditions nor are results for all builds reported.

Entries in Table 3 and Table 4 marked with a super-

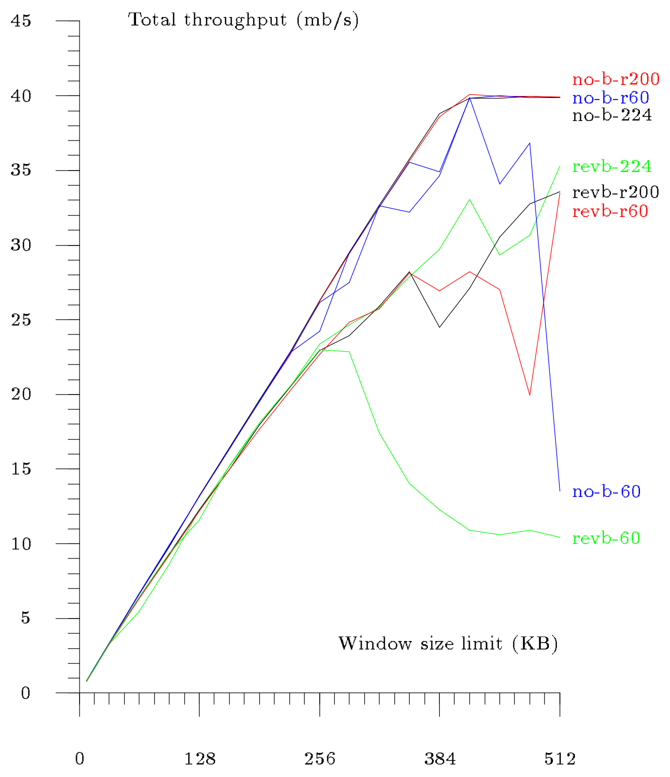

Figure 1: Single TCP Flow, 68 msec RTT

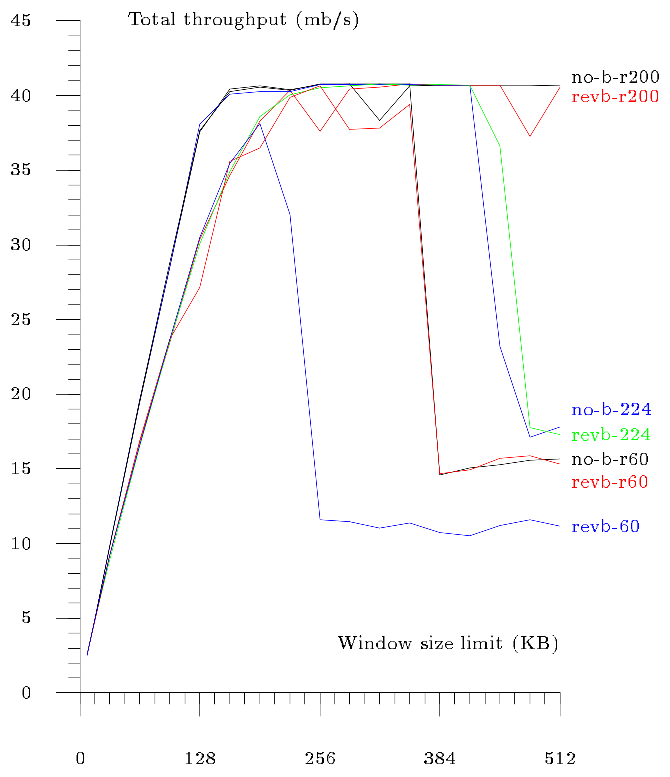

Figure 2: Single TCP Flow, 20 msec RTT

script suffered throughput degradation at higher window size limits. The superscript gives the percent degradation from the peak utilization recorded for the test. Subscripts indicate multiple TCP flow tests in which not all of the flows completed simultaneously. The subscript gives the ratio, expressed as a percent, of the difference between the fastest and slowest flows over the rate of the fastest flow.

Graphs with multiple data sets are labeled with abbreviations to identify the test conditions. The RS/960 build designations found in Table 1 and background condition designations found in Table 2 are used. The abbreviations **1f**, **4f**, and **8f** indicate a single TCP flow, 4 simultaneous TCP flows and 8 simultaneous TCP flows.

5.1 Single High Speed Flows

The single TCP flow tests were run to a window size limit of 512 kB. For the 68 msec path, a window of 512 kB is about 1/3 larger than D*BW. For the 20 msec path, this window size is almost 5 times D*BW.

In tests run with no background, throughput rose linearly with increase in the window size limit up to approximately the delay bandwidth product. If queueing capacity was close to D*BW or exceeded it, the curves would plateau when the TCP window was allowed to exceed D*BW. This can be seen in Figure 1 with the **224** build and the 68 msec path, reaching 38.8 Mb/s at a window size limit of 384 kB and gradually increasing to 40 Mb/s as the window size limit grew to 512 kB.

In Figure 2 the same build is used with the 20 msec path. At 160 kB, 40 Mb/s is exceeded. The plateau remains to 416 kB, just beyond the queueing capacity. At that point, a collapse occurs, despite having nearly 3 time D*BW of queueing, with throughput dropping by about 60%.

When the 68 msec path is used with the **60** build (Figure 1), the linear increase in throughput continues beyond twice the queueing capacity. Above that point, results become somewhat erratic, exceeding 80% utilization before collapsing to just over 25% utilization (a 2/3 degradation from peak performance).

In Figure 1 results are provided with the **224** build with a TCP flow in the reverse direction on the 68 msec path. With reverse traffic only 35 Mb/s is achieved. With reverse traffic and very large windows, performance is somewhat erratic but did not show signs of dropping off.

Figure 1 also shows results for the **60** build with a single flow and reverse traffic. With or without reverse flow severe congestion collapse occurs. With reverse flow only about 23 Mb/s is achieved before collapse begins, compared to 39.9 Mb/s with no background. With no background the collapse is sudden. With reverse flow, the collapse begins earlier and is gradual. Beyond the collapse, with no background performance is reduced to about 16 Mb/s, compared to under 10 Mb/s with reverse flow.

With RED utilization remains high in the single flow tests, but peak utilizations do not improve noticably. In the single flow test with reverse flow, a lower peak of 33 Mb/s is achieved with `r200`, as opposed to 35 Mb/s for the `224` build. Both builds have under D*BW of queueing and in this particular test yielded somewhat erratic results when the window size exceeded queueing capacity. The 2 Mb/s difference may not be significant given the erratic results near the performance peak in this test. In the single flow tests, the most significant difference between `r200` and `224` build performance may be the absence of a collapse in Figure 2 for `r200` where all other builds collapsed to 18 Mb/s or less.

Collapse occurs in a number of the single flow tests. For builds that do not implement RED, there seem to be two conditions under which collapse occurs. Either the the window is much larger than the queue and then exceeds D*BW by a relatively small amount, or the window much larger than D*BW and exceeds the queue size by a relatively small amount. With RED, collapse occurs under only one condition, if the window is much greater than both D*BW and the queue size. This seems to indicate that if RED is implemented and the queue size can be kept greater than D*BW, host window size limits can be set arbitrarily high.

5.2 Multiple Flows

For multiple TCP large window transfers the total utilization of the multiple flows is plotted. The total utilization is approximated by multiplying the number of flows by the lowest throughput achieved by a flow in that test run.

Figure 3 shows the results of running 4 TCP flows using the 68 msec path. With both the `224` and `60` builds, link utilization increases linearly and then begins to drop. With the `224` build 41.2 Mb/s is achieved at a window size limit of 192 kB and with no background traffic, and 38.7 Mb/s is achieved with reverse traffic. As the window size grows to 352 kB, both plots drop back to 33 Mb/s.

Results for 4 TCP flows using the 20 msec path are similar, but with less degredation as the optimal window size is exceeded. Figure 4 shows throughput with `224` quickly rising to 41.2 Mb/s and falling off only slightly starting above 256 kB. With reverse flow, a peak of 40.8 Mb/s is achieved.

When the `60` build is used with 4 TCP flows and the

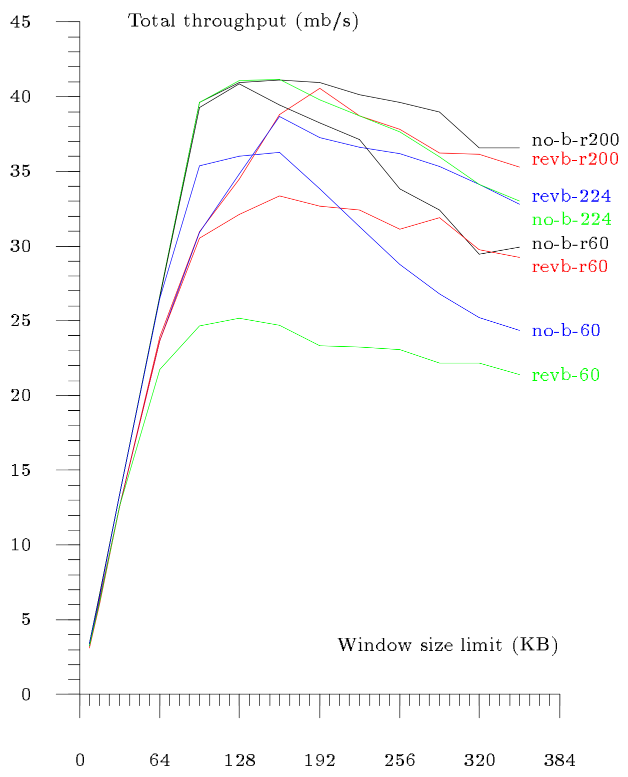

Figure 3: Four TCP Flows, 68 msec RTT

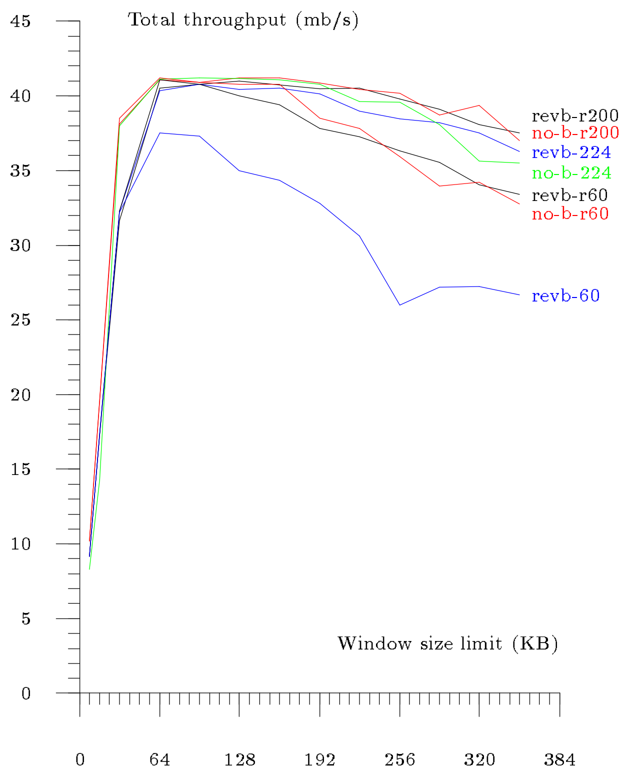

Figure 4: Four TCP Flows, 20 msec RTT

68 msec path (Figure 3), a peak of 36 Mb/s is reached at 128 kB with no background. Beyond that point throughput gradually drops as the window size in further increased, dropping below 25 Mb/s at 352 kB. With a reverse flow, the peak is 25 Mb/s and performance drops slightly to 23 Mb/s beyond the peak.

On the 20 msec path (Figure 4), with 4 flows the **60** build achieves 37.5 Mb/s. On this path, the queueing in the **60** build is much closer to D*BW.

The **r200** build clearly outperforms the other builds in the 4 flow tests. This is most evident on the 68 msec path (Figure 3). The **r200** curves for reverse flow peaks 2 Mb/s higher than the corresponding curve for the **224** build. The **r200** curve remains 2–5 Mb/s avove the **224** curve beyond the peak. On the 20 msec path (Figure 4), both plots peak at 41.2 Mb/s. Beyond the peak, both curves are quite smooth with the curve for **r200** about 1 Mb/s above the **244** curve.

The **r60** build clearly outperforms the **60 build** in the 4 flow tests. With the 68 msec path (Figure 3), the 33.4 Mb/s peak achieved with **r60** is well below the **r200** and **224** results, and performance drops off to under 30 Mb/s. With **r60**, a peak of 41.1 Mb/s is achieved on the 20 msec path (Figure 4), but performance drops off to 34 Mb/s, again well below **r200** and **224** builds.

Both the **r60** and **60** build have slightly inadequate queueing capacity for the 20 msec path and about 1/5 of the queueing capacity needed for the 68 msec path. In the 4 flow tests, **r60** outperformed **60**, by an enormous margin on the 68 msec path. With **r60** performing well under **r200** and **224** it is clear that RED could not make up for inadequate queueing capacity in the 4 flow tests.

Figure 5 provides results for 8 TCP flows on the 68 msec path. Results are similar to the 4 TCP flow results. With 8 flows, the peaks are slightly higher for all of the builds without background or with reverse flow. Performance falls off more slowly beyond the peak for 8 flows than for 4 flows. Again the **r200** build performs best, closely followed by the **224** build. Both **r200** and **224** peak at 41.2 Mb/s. The **r60** build peaks at 36.0 Mb/s and the **60** build peaks at a 28.3 Mb/s. The builds with sufficient queueing again outperform those with insufficient queueing by a significant margin.

Figure 6 shows results for the 20 msec path. With the 20 msec path and 8 flows and no background traffic, there is almost no throughput degredation beyond the peak with the **r200** and **224** builds. The **r60** build performs nearly as well as the **r200** and **224** builds, though a 2 Mb/s difference exists well beyond peak. The **60** build peaks at 39.0 Mb/s, but performance drops of below 35 Mb/s beyond the peak.

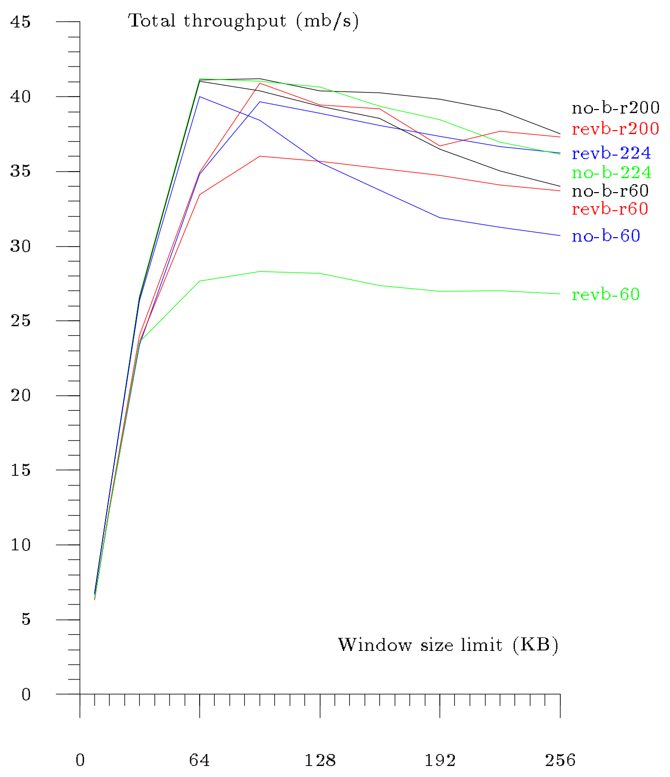

Figure 5: Eight TCP Flows, 68 msec RTT

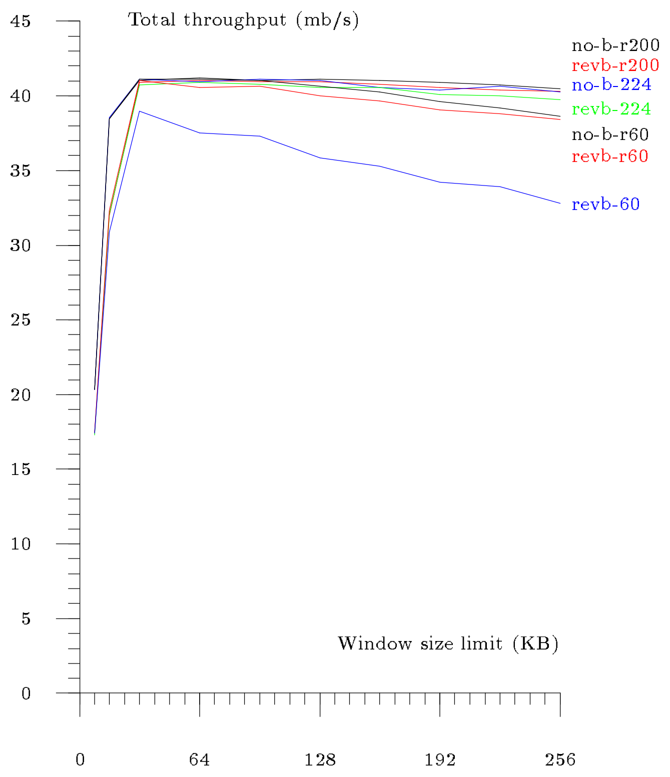

Figure 6: Eight TCP Flows, 20 msec RTT

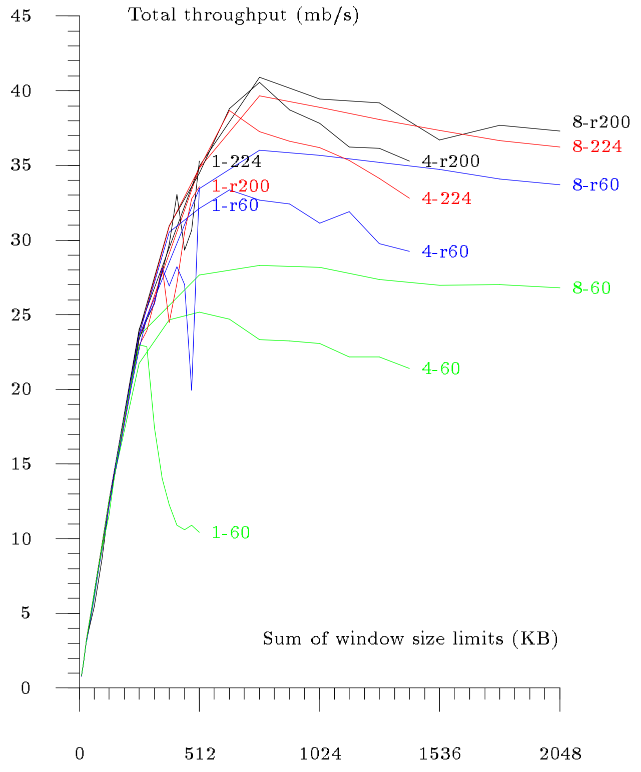

Figure 7: 68 msec RTT, Reverse Flow

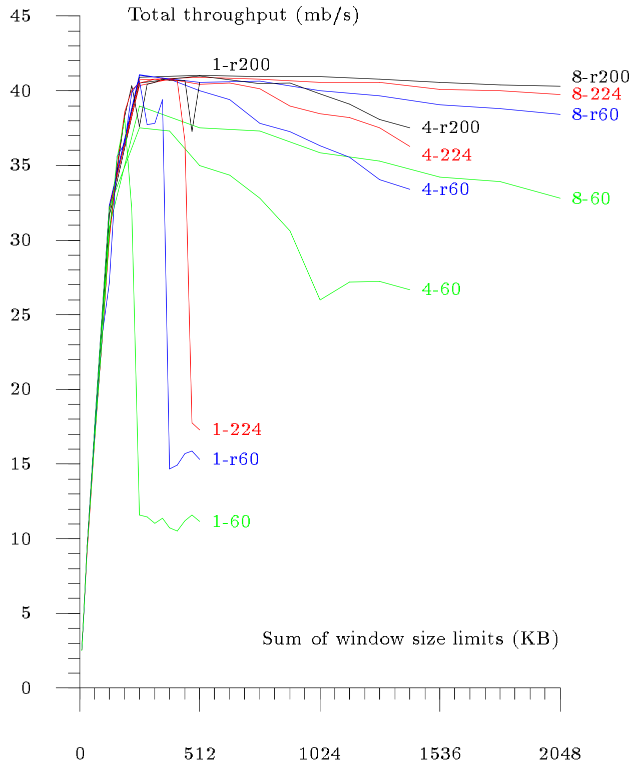

Figure 8: 20 msec RTT, Reverse Flow

5.3 Reverse Flow

The reverse TCP flow had a clear effect. In cases where the maximum window size had exceeded the queueing capacity but was still less than the D*BW product, throughput continued to rise without reverse traffic. This may indicate a spreading of packets throughout the available window, failing to stress the queueing capacity. With the reverse traffic, ACK compression may have caused enough burstiness to stress the queueing capacity when window size limits of individual flows are below D*BW.

Figure 7 shows the link utilization plotted against the sum of the window sizes limits for all tests involving the 68 msec path. For a given build, each set of three plots with 1, 4, and 8 flows all peak within a small area of the graph. For the `60` build, the inflection points are at about 23–28 Mb/s. For the `r60` build, the inflection points are at 36–38 Mb/s. For the `224` build, the inflection points are at about 40–41 Mb/s, very close to full utilization.

For each build, the inflection point occurs at a slightly lower bandwidth for the single flow cases. It also occurs just above D*BW, where in the multiple flow cases, the inflection point moves toward twice D*BW. Beyond the inflection point, there is a fall off in bandwidth. The peak is high for builds with queueing capacity of close to D*BW and highest in tests where queueing capacity is greater than D*BW and with builds that implement RED. The peak are slightly higher and fall off is more gradual as the number of flows increases.

Reverse traffic had a similar effect on tests run on the 20 msec path (Figure 8). For this path, the `r200` and `224` builds have about 3 time D*BW in queueing capacity. The `r60` and `60` builds have just under D*BW in queue capacity. The inflection points for all of the builds are in the range of 38–41 Mb/s.

For tests using the 20 msec path, the major difference between the performance of various builds is in the rate of fall off beyond the inflection point. For single flow tests, all but the `r200` can be considered to have collapsed. In the multiple flow tests, the `60` build shows considerable fall off, to 27 Mb/s in the 4 flow case and 35 Mb/s in the 8 flow case. The `r60` build shows considerably less fall off, to 35 Mb/s in the 4 flow case and to 38 Mb/s in the 8 flow case. The `224` build falls off to 40 Mb/s, and the `r200` build shows no sign of fall of at all.

5.4 Random Early Detection

The effective queueing capacity of the `r200` build implementing RED was slightly smaller than the `224` build.

Figure 1 shows no real difference between the `r200` build and `224` build with no background. With reverse flow, the `224` build is a bit faster. This is likely due to the reduced queueing capacity in the `r200` build and queueing capacity slightly less than D*BW in both builds.

In the 20 msec tests, the small difference in queueing capacity between the `r200` build and the `224` build has no effect since both builds have well over D*BW of queueing capacity. The `r200` generally outperformed the `224` build by a small but noticable margin in the multiple flow tests (see Figure 4 and Figure 6). The most striking difference is the absence of a congestion collapse for very large window sizes in the single flow tests on the 20 msec path that can be seen in Figure 2. Note that with insufficient queueing in the `r60` build, RED is unable to prevent a collapse in Figure 2 as the window size limit is set 3–4 times its optimal value.

In the tests involving multiple flows, the `r200` build performed slightly better than the `224` build both without background traffic and with reverse flow. This can be seen in Figure 3 and Figure 5. In Figure 3, the advantage of RED is most evident as the windows are made too large. Bandwidth falls off beyond the inflextion point less with `r200` than with other builds.

In Figure 7, the `r200` build can be seen to perform best in all of the reverse flow tests on the 68 msec path except the single flow test. In Figure 8 `r200` outperforms `224` in all tests and, unlike `224`, `r200` avoids congestion collapse when window size limits are well beyond optimal. The lower performance in the 68 msec single flow test is due to there being slightly less than D*BW in queueing capacity and RED further reducing queue utilization slightly. When using the 20 msec path, the queueing capacity is larger than D*BW resulting in better performance with RED than without in all tests including the single flow.

5.5 Fairness and Delay

One of the goals of RED is to reduce delay. The variation of RED used here is optimized to improve link utilization, though somewhat reduced delay is a secondary benefit. Delay was not measured. The tradeoff between further reducing delay and impacting high speed flows is discussed further in [29].

The worst fairness experienced in any complete test run (all window sizes for a given set of conditions) is given in Table 3 and Table 4 as the subscripts. While TCP is not perfectly fair in these tests, deviations are typically 10–20%. The only notable exceptions are the 8 flow test with build `224` and no background traffic in which the slowest flow rate was 42% less than the fastest and in the 4 flow test with the `r200` build and no background. A deviation of 10–20% is quite fair. Even

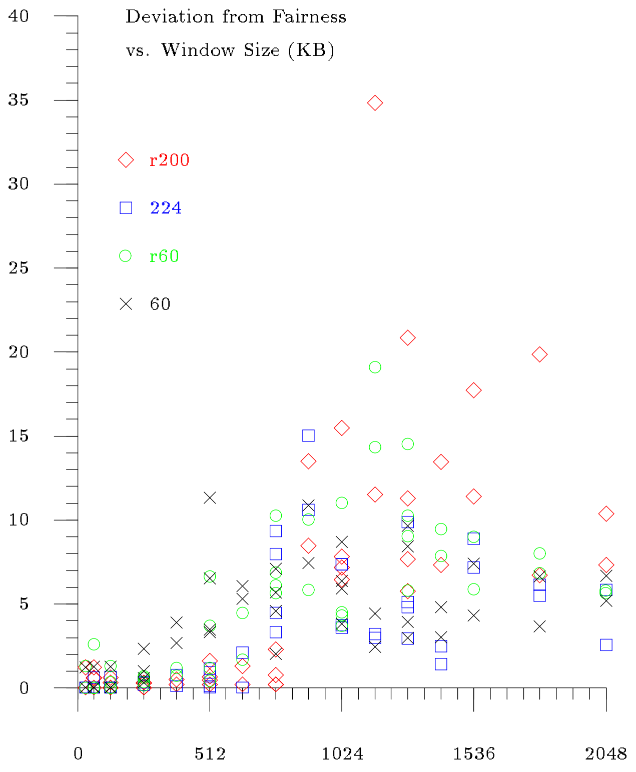

Figure 9: Deviation from Fairness for 68 msec Tests

a 42% difference among 8 flows is not bad.

Figure 9 provides a scatter plot of the deviation from fairness. The ratio of difference in rates over the highest rate achieved expressed as a percent is plotted against the sum of the window size limits.

When the sum of the window sizes are less than D*BW and the queueing capacity, there is no opportunity for inequitable service since there is no need to drop packets. As the window is further increased without RED the queue will overflow. With RED as the average queue size grows, RED will have a greater tendency to perform a packet drop to signal a flow to reduce it's rate. In either case, there is a greater probability of drop for flows contributing more bandwidth, tending to distribute available bandwidth quite fairly.

RED is actually seen to be slightly less fair in these tests. Simulations indicate that for TCP flows with unequal RTTs, RED will remain approximately fair, where the lower RTT would be favored without RED [36]. The simulations indicate that RED will eliminate this tendency to be systematically unfair.

5.6 Link Utilization Estimates

There is some uncertainty associated with the bandwidth figures in the multiple flow cases. Link utilization is estimated by taking the rate of the slowest flow and multiplying by the number of flows. There is good reason not to use an average.

There is a relatively long time in which all of the flows are active. There is then a period of time when a reduced number of flows are active. During the time in which less flows are active, utilization is likely to be different, probably less, than when all flows were active.

If the average rate is used, erroneous results are possible. This can be proven with a simple example. The example is two flows where one dominates completely and the second is totally idle until the first finishes after which the second proceeds at the same rate that the first had. The first flow may proceed at full link rate, in which case the second will will complete at an apparent half that rate. If the average of the two rates times the number of flows were used, the estimate of link utilization would be 1.5 times the link capacity.

The estimate of the average link utilization that is used is the total amount of data transferred divided by the amount of time needed to transfer it. If all of the flows are started simultaneously, this is equal to the transfer rate of the last flow to complete times the number of flows. The utilization is bounded to below link capacity when the sum of the window sizes of the remaining flows is below D*BW. The estimate is therefore a lower bound estimate on the average link utilization during the period when all of the flows were active.

Figure 9 shows a reduced tendency toward fairness with increase in the sum of the window sizes. Since this reduced fairness is likely to also cause a slight underestimate of the link utilization, this is partially responsible for the slight degredation in link utilization seen in Figure 7 and Figure 8. Close examination of the raw data shows that this inaccuracy is small compared to the actual reduction in performance. A more direct measure of link utilization would provide greater accuracy, but does not seem necessary.

6 Recommendations

Whenever high bandwidth is needed, a large TCP segment size should be used. MTU discovery (RFC–1191) if correctly implemented can assure that an optimal segment size is used.

Well connected hosts (those with greater than T1 capacity to much of the Internet) should consider increasing the default `tcp_sendspace` and `tcp_recvspace` kernel variables according to their connectivity. When communicating with hosts behind slower connections, the lower window size limit specified by the other host will be used. When communicating with other well connected hosts, performance can improve dramatically. Care must be taken not to exceed the queueing capacity of routers on the path.

Attempts to set the window size limit either considerably greater than the queueing capacity, or many times greater than the delay bandwidth product can cause congestion collapse at links dominated by a small number of TCP flows. This problem is eliminated if the routers at the congestion point implement RED.

If the window size limit is set correctly, very high performance is possible. If multiple TCP flows attempt very high speed and exceed the capacity of a bottleneck, TCP will share the available bandwidth quite equitably. As flows are added, total utilization will generally rise or remain the same if already at capacity, though slight degradation in utilization may occur as the sum of the window sizes becomes much larger than D*BW. In general, slightly higher link utilizations are achieved using RED. If queueing capacity is greater than D*BW and RED is implemented, little or no degradation in link utilization will occur.

If queueing is inadequate at some bottleneck, utilization will be limited, particularly with small numbers of flows. As load is added, there will be a greater tendency toward degradation or even collapse. Routers should have queueing capacity of more than one D*BW product. Implementing RED is helpful in reducing the queue utilization, particularly for larger numbers of flows, but cannot substitute for adequate queueing capacity.

For ANSNET DS3 connected sites, it should be safe to set the window size limits to about 192 kB to 256 kB provided no routers with reduced queueing capacity are in the path, such as routers on the local campus. This should allow single TCP flows on the order of 25 Mb/s for US cross continental connections, 30 Mb/s to 35 Mb/s for Hawaii to California connections, and single TCP flows of perhaps 15 Mb/s from the US East Coast to Hawaii. Multiple flows, both within the continental US and to Hawaii should be able to drive utilization to 40 Mb/s or more. The risk of congestion collapse is eliminated by RED and seems quite small even without RED.

For sites with high speed connectivity but behind routers with lower queueing capacity, the maximum window size should be set lower. Indications are that with queueing capacity of 80 kB per interface path, appropriate window sizes might be 48 kB to 64 kB at most.

When using a large window size limit to allow full utilization of long paths there is a risk of inducing collapse on shorter paths if the window size is much greater than the delay bandwidth product of the shorter path. There is no practical way to set the window size limit such that it is optimal for a long path but does not pose a risk to a shorter path. This problem is eliminated with RED. Until RED is deployed, DS3 sites must be aware of this.

7 Other Considerations

Our testing was done using large packets, over 4 kB in size. Internet traffic is typically smaller packets, averaging 200–250 B in size. High speed flows (those in which the segment size limit is increased over the default value) using MTU discovery can be expected to be closer to the ethernet or FDDI MTU. Even so, the average packet size will reflect TCP ACKs, and small packets from other services. Most routers reserve a full MTU buffer for each packet regardless of size. For a 200 B average and a 4352 B (FDDI) MTU, this represents a reduction in effective queueing of a factor of about 20. For an ATM MTU of 9180 B, this situation is worsenned.

RED may help address the small packet problem. The Internet small packet traffic is currently a mix of protocols, dominated by an aggregation of TCP flows [37]. RED can help prevent synchronization of the TCP flows and keep queue utilization down should this traffic grow to fill backbone link capacities.

There may be a limit to the number of simultaneous flows that can be supported. The available window capacity can be subdivided among flows until each window has a small integer number of packets. Below a window size of 3 or 4 packets one would expect TCP to perform poorly or erratically. The TCP fast retransmit algorithm would be defeated, causing timeouts. This could produce a ledge in the performance curve. At this point, TCP algorithms are defeated and TCP becomes largely incompressible. The only solutions to this is to decrease the MTU, to increase the number of packets per window, or allow the queue to grow larger, increasing delay and therefore also D*BW.

The problems of multiple bottlenecks is addressed in [36] and other useful results are presented. Without RED TCP favors flows with lower RTTs at a single bottleneck. With RED, TCP is shown to be fair in this situation. Even with RED, TCP flows on long paths with multiple congested gateways may receive an unacceptable low share of bandwidth relative to flows traversing single congestion points on the path. Unpublished work by Floyd, Paxson, and Jacobson is exploring this area further, concentrating on two way traffic. Our current test setup does not provide a simple way to mix flows of different RTTs or provide multiple bottlenecks.

The results reported here included tests involving 2 way traffic, but only congestion in one direction. The behavior under congestion in both directions is a topic for further investigation.

First generation ATM switches gained a reputation for grossly inadequate queueing capacity. This has improved. More recently, ATM cell buffer capacities have increased by more than two orders of magnitude. The effects of cell loss and failure to drop full ATM AAL frames has been shown to be detrimental to TCP performance using simulations [38]. Simulation involving ATM to date have not addressed the use of high speed TCP on long delay paths. Congestion avoidance for ATM UBR service does not exist and is not yet specified for ABR service.

Throughout the evolution of TCP, backward compatibility with earlier host implementations has been of paramount importance. This paper addresses the use of MTU discovery (RFC–1191), high performance extensions to TCP (RFC–1323), and changes to routers to accomplish performance goals for high speed applications and relys only upon changes to routers to achieve better performance using large aggregations of traffic. Changing routers or other equipment in the core of a high speed Internet is generally much easier than changing hosts, since there are simply too many hosts to make host changes a practical alternative.

8 Conclusions

Sufficient queueing capacity is required to support high speed TCP flows or aggregations of TCP flows. Test results demonstrate that nearly full bottleneck utilization is achieved with close to one D*BW product of queueing in paths with delays comparable to US cross continent delays.

Insufficient queueing can result in low bottleneck utilization. Test results demonstrate that insufficient queueing capacity can result in sustained low utilization and that utilizations can drop with increased load. Collapse can occur under certain conditions.

The combination of adequate queueing capacity and RED provide the means to support high bandwidth applications with multiple flows attempting to use full link bandwidth. This combination allows high link utilization and eliminates the possibility of degredation or collapse in link utilization due to excessive window size limits set by end users.

The IBM RS/6000 based routers fared quite well in this testing, including the difficult cases of small numbers of very high speed flows. The newer builds of software further improve performance over the currently deployed build. While queueing capacity is not sufficient to support full bandwidth transfers on the longest ANSNET paths, such as Boston to Hawaii, these improvements will go a long way toward allowing close to full utilization of ANSNET as demand rises.

9 Acknowledgments

The IBM RS/6000 routers used within ANSNET owe their excellent performance to the hard work and dedication of many individuals at Advantis (formerly IBM HPCC). This paper studies only one factor in determining overall performance and documents optimizations done in that area.

This research was sponsored in part by the Phillips Laboratory, Air Force Materiel Command, USAF, under cooperative agreement number F29601-93-2-0001. The U.S. Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Phillips Laboratory or the U.S. government.

The authors are grateful to Sally Floyd (Lawrence Berkeley Laboratory) for numerous discussions and suggestions. We would also like to acknowledge the suggestions and encouragement of Guy Almes (ANS) and Matt Mathis (Pittsburgh Supercomputing Center) and the help of Mike Ida (Maui High Performance Computing Center) and Hans-Werner Braun and Bilal Chinoy (San Diego Supercomputing Center) with earlier testing on ANSNET.

References

[1] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.

[2] G. T. Almes, "Engineering trans-oceanic 44 mb/s tcp/ip networks." presented at the Third Symposium on High Speed Networking for Research in Europe, Brussels.

[3] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "Long-term traffic aspects of the NSFNET," in *Proceedings of the International Networking Conference (INET)*, (San Francisco, California), pp. CBA–1 – CBA–12, Internet Society, Aug. 1993.

[4] ANS Network Operations Center, "Ansnet link utilization daily reports." unpublished reports used for capacity planning.

[5] ANS Network Operations Center, "Ansnet congestion loss daily reports." unpublished reports used for capacity planning.

[6] J. Nagle, "Congestion control in IP/TCP internetworks," *ACM Computer Communication Review*, vol. 14, pp. 11–17, Oct. 1984.

[7] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, pp. 314–329, Aug. 1988. Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[8] J. Postel, "Transmission control protocol," Network Working Group Request for Comments RFC 793, Information Sciences Institute, University of Southern California, Sept. 1981.

[9] D. Clark, "Window and acknowlegement strategy in TCP," Request for Comments RFC 813, Internet Engineering Task Force, July 1982.

[10] J. Nagle, "Congestion control in IP/TCP internetworks," Request for Comments RFC 896, Internet Engineering Task Force, Jan. 1984.

[11] J. Postel, "TCP maximum segment size and related topics," Request for Comments RFC 879, Internet Engineering Task Force, Nov. 1983.

[12] D. Mills, "Internet delay experiments," Request for Comments RFC 889, Internet Engineering Task Force, Dec. 1983.

[13] L. Peterson and S. O'Malley, "TCP extensions considered harmful," Request for Comments (Informational) RFC 1263, Internet Engineering Task Force, Oct. 1991.

[14] R. Braden and V. Jacobson, "TCP extensions for long-delay paths," Request for Comments (Experimental) RFC 1072, Internet Engineering Task Force, Oct. 1988.

[15] R. Fox, "TCP big window and NAK options," Request for Comments RFC 1106, Internet Engineering Task Force, June 1989.

[16] A. McKenzie, "Problem with the TCP big window option," Request for Comments RFC 1110, Internet Engineering Task Force, Aug. 1989.

[17] R. Braden, V. Jacobson, and L. Zhang, "TCP extension for High-Speed paths," Request for Comments (Experimental) RFC 1185, Internet Engineering Task Force, Oct. 1990.

[18] D. Borman, R. Braden, and V. Jacobson, "TCP extensions for high performance," Request for Comments (Proposed Standard) RFC 1323, Internet Engineering Task Force, May 1992. Obsoletes RFC1185.

[19] W. R. Stevens, *TCP/IP illustrated: the protocols*, vol. 1. Reading, Massachusetts: Addison-Wesley, 1994.

[20] D. Clark, V. Jacobson, J. Romkey, and M. Salwen, “An analysis of TCP processing overhead,” *IEEE Communications Magazine*, vol. 27, pp. 23–29, June 1989.

[21] D. A. Borman, “Implementing TCP/IP on a cray computer,” *ACM Computer Communication Review*, vol. 19, pp. 11–15, Apr. 1989.

[22] V. Jacobson, “Some design issues for high-speed networks,” in *Networkshop '93*, (Melbourne, Australia), Nov. 1993.

[23] J. Mogul and S. Deering, “Path MTU discovery,” Request for Comments (Draft Standard) RFC 1191, Internet Engineering Task Force, Nov. 1990.

[24] S. Knowles, “IESG advice from experience with path MTU discovery,” Request for Comments (Informational) RFC 1435, Internet Engineering Task Force, Mar. 1993.

[25] S. Floyd and V. Jacobson, “On traffic phase effects in packet-switched gateways,” *Internetworking: Research and Experience*, vol. 3, pp. 115–156, Sept. 1992.

[26] S. Shenker, L. Zhang, and D. Clark, “Some observations on the dynamics of a congestion control algorithm,” *ACM Computer Communication Review*, vol. 20, pp. 30–39, Oct. 1990.

[27] L. Zhang, S. Shenker, and D. D. Clark, “Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic,” in *Sigcomm '91 Conference: Communications Architectures and Protocols*, (Zürich, Switzerland), pp. 133–147, ACM, Sept. 1991.

[28] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, “Tracking long-term growth of the NSFNET,” *Communications ACM*, vol. 37, pp. 34–45, 1994.

[29] C. Villamizar, “A variation of random early detection congestion avoidance.” work in progress.

[30] R. Cáceres, P. B. Danzig, S. Jamin, and D. J. Mitzel, “Characteristics of wide-area TCP/IP conversations,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Zürich, Switzerland), pp. 101–112, ACM, Sept. 1991. also in *Computer Communication Review* 21 (4), Sep. 1991.

[31] V. Paxson, “Empirically-derived analytic models of wide-area TCP connections,” technical report, Lawrence Berkeley Laboratory and EECS Division, University of California, Berkeley, California, June 1993.

[32] W. E. Leland, M. S. Taqq, W. Willinger, and D. V. Wilson, “On the self-similar nature of Ethernet traffic,” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (San Francisco, California), pp. 183–193, ACM, Sept. 1993. also in *Computer Communication Review* 23 (4), Oct. 1992.

[33] V. Paxson and S. Floyd, “Wide-area traffic: the failure of Poisson modeling,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, United Kingdom), pp. –, ACM, Aug. 1994.

[34] C. Partridge, “Editorial: The end of simple traffic models,” *IEEE Network*, vol. 7, Sept. 1993.

[35] T. Skibo, “Experiences with tcp extensions for high-performance networks.” contained in ftp://vixen.cso.uiuc.edu/pub/tcplw.shar.Z.

[36] S. Floyd, “Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic,” *ACM Computer Communication Review*, vol. 21, pp. 30–47, Oct. 1991.

[37] I. Merit, “Nsfnet statistics repository.” ftp://nis/nsf.net/statistics/nsfnet/.

[38] A. Romanow and S. Floyd, “Dynamics of TCP traffic over ATM networks.” see also 6th IEEE LAN/MAN Workshop, 1993.