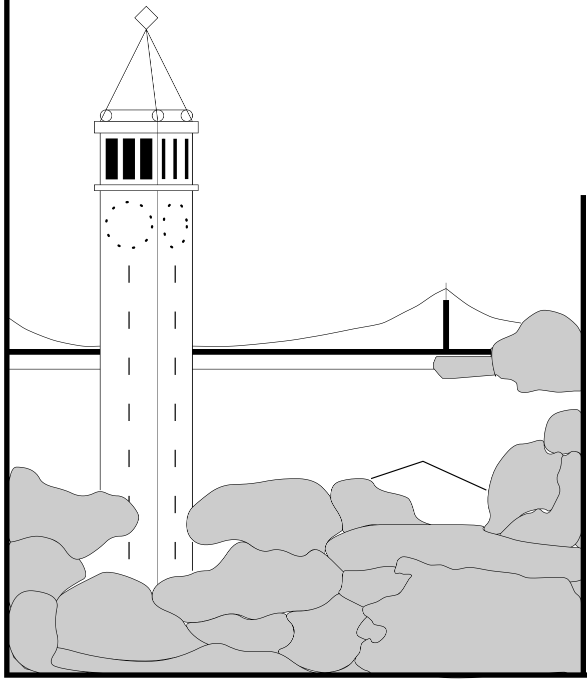
TCP Performance over Satellite Channels

Thomas R. Henderson and Randy H. Katz



Report No. UCB/CSD-99-1083

December 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

TCP Performance over Satellite Channels

Thomas R. Henderson and Randy H. Katz
$\{$*tomh,randy*$\}$*@cs.berkeley.edu*
Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720

Abstract

Technological advances are enabling new satellite communications systems that combine broadband data rates with small terminals. These novel systems are being designed to provide affordable "last-mile" network access to home and small business users worldwide. In particular, two types of advanced broadband satellite systems are under development: high-power satellites deployed at traditional geostationary (GEO) orbits, and large constellations of satellites deployed at much lower (LEO) orbits. In this report, we explore research problems that have arisen from the attempt to use GEO satellites to provide Internet access. In particular, performance of the Internet's Transmission Control Protocol (TCP) is degraded by the high latency and high degree of bandwidth asymmetry present in such systems, and the degradation is compounded by the packet-level congestion found in today's Internet. We first study whether TCP's congestion avoidance algorithm can be adjusted to provide better fairness when satellite connections are forced to share bottleneck links with other (shorter delay) connections. Our data suggests that adjustments of the policy used in that algorithm may yield substantial fairness benefits without compromizing utilization. We next demonstrate how minor variations in TCP implementations can have drastic performance implications when used over satellite links (such as a reduction in file transfer throughput by over half), and from our observations construct a satellite-optimized TCP implementation, using standardized options, that achieves good file transfer performance when congestion is light. We explore the performance of TCP for short data transfers such as Web traffic, and find that two experimental options relating to how TCP starts a connection, when used together, could reduce the user-perceived latency by a factor of two to three. However, because not all of these options are likely to be deployed on a wide scale, and because even the best satellite-optimized TCP implementation is vulnerable to the fairness problems identified above, we explore the performance benefits of splitting a TCP connection at a protocol gateway within the satellite network, and find that such an approach can allow the performance of the satellite connection to approach that of a non-satellite connection. Our results illustrate many of the remaining challenges facing TCP performance over GEO satellite links while also demonstrating techniques that can be used to optimize performance in many satellite networks.

1 Introduction

Several companies (e.g., Lockheed Martin, Hughes, Ka-Star) have announced plans to build large satellite systems to provide commercial broadband data services distinct from narrowband voice services. These systems are expected to offer Internet access to remote locations and to support virtual private networks for widely scattered locations. However, the performance of data communications protocols and applications over such future systems is the subject of heated debate in the research community. Nowhere has this debate raged more than in discussions regarding the transport-level protocol in the Internet TCP/IP protocol suite (namely, the Transmission Control Protocol [39]). Some researchers insist that TCP will work suitably in a satellite environment, while others have suggested satellite-specific protocol options for improved performance, and still others claim that TCP cannot work effectively over satellite channels. There is, however, no disagreement in that the large latencies, bandwidth and path asymmetries, and occasionally high error rates on satellite channels provide TCP with a challenging environment in which to operate.

In this report, we evaluate just how well TCP performs in a satellite environment composed of one or more satellites in geostationary orbit (GEO) or low-altitude earth orbit (LEO), in which the end-to-end connection may traverse a portion of the wired Internet. We first discuss our assumptions concerning future broadband satellite systems that plan to provide direct-to-user Internet access, focusing on characteristics that impact transport layer protocol performance. In Section 3, we describe the various ways in which latency and asymmetry can impair the performance of TCP, and survey the related work that has yielded extensions to standard TCP that alleviate some of these performance problems. Through analysis, simulation, and experiments described in Sections 4-7, we quantify the performance of state-of-the-art TCP in a satellite environment, both for large file transfers and short Web transactions. A key part of our experimental method is the use of traffic models, empirically derived from Internet traffic traces, to provide moderate levels of background traffic. We identify scenarios where TCP can be

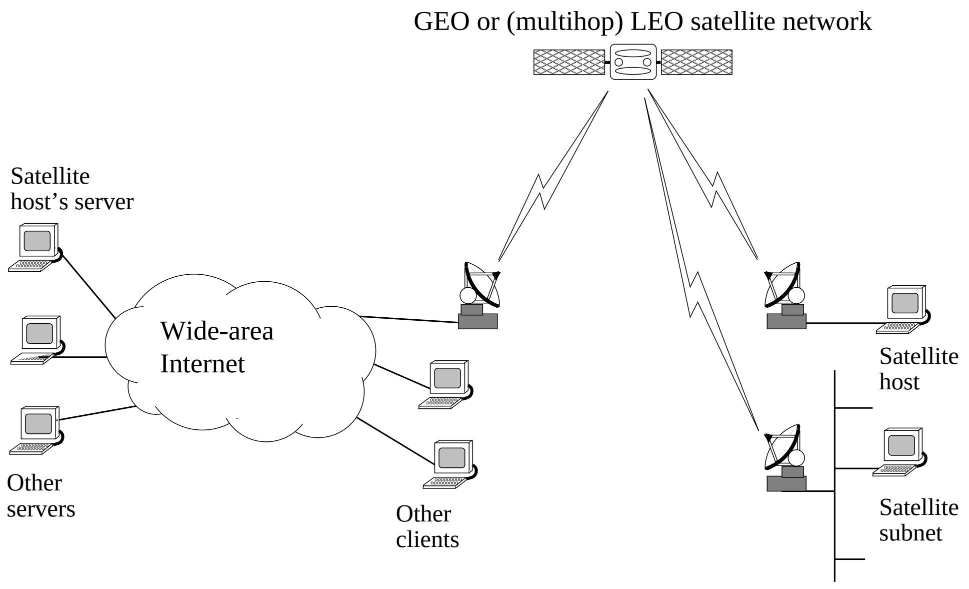


Figure 1: Example of a future satellite network in which a satellite-based host communicates with a server in the Internet

expected to perform reasonably well, and where it can suffer serious performance degradation due to either suboptimal protocol configuration or congestion in the wide-area Internet.

The following are our main contributions:

- The fairness of TCP's congestion avoidance algorithm when multiple connections with different RTTs share a bottleneck path is a longstanding research problem. Using simulation models, we provide evidence that, while TCP fairness problems may not be easily solvable in a manner that can be incrementally deployed by making changes to host implementations, small changes to a satellite connection's congestion avoidance algorithm can substantially improve the fairness of a bottleneck link without compromising link utilization. In particular, in congestion situations we were frequently able to double the throughput of satellite connections by making their congestion avoidance policies slightly more aggressive than normal, but not so aggressive as to unfairly penalize other competing connections.

- Previous studies of TCP performance over satellite channels have focused on the large file transfer performance of a single connection in isolation, often on channels with high bit error rates. Our data indicates that, despite the use of satellite-optimized TCP implementations on relatively error-free channels, the presence of other competing TCP connections in the wide-area Internet can dominate the satellite connection's performance. We also illustrate how subtle implementation details can have a major effect on TCP performance over satellite channels, and describe a standards-compliant TCP implementation that is able to obtain good performance over a satellite channel when there is little congestion along the path.

- We quantify the effects of TCP latency on small data transfers by performing analysis and experiments based on traces of HTTP connections (the Hypertext Transfer Protocol [42], used for Web browsing), and evaluate the relative merits of proposed TCP options that reduce the latency of short HTTP connections. We find that the user-perceived latency of Web transfers can be reduced by a factor of two to three through the combination of a larger TCP initial window and the use of TCP for Transactions.

- Because satellite TCP connections are susceptible to network congestion, and because satellite-friendly TCP options are not likely to be universally adopted in the near term, we explore the potential benefits of splitting TCP connections at a gateway within the satellite subnetwork. We find that, although such gateways may be expensive to implement, particularly when IP security protocols are implemented, the performance problems related to satellite latency and asymmetry can be effectively avoided.

2 Transport Environment of Future Satellite Systems

Our assumptions about future satellite network characteristics are shaped by projections of future commercial systems (e.g., Teledesic [44], Spaceway [13]). These future systems will offer Internet connections at up to broadband (tens of Mb/s) data rates via networks of LEO or GEO satellites (or hybrid constellations). Users may contact other hosts in either the satellite network or the wide-area Internet. In general, we have considered an architecture based on packet switching that is fully compatible with the TCP/IP protocol suite. We also were primarily concerned with architectures that scale to serving many thousands of users (e.g., direct-to-user services rather than carrier trunking). Figure 1 illustrates the general topology, in which users or small networks access the wide-area Internet via the satellite system.

The main characteristics of the end-to-end path that affect transport protocol performance are latency, bandwidth, packet loss due to congestion, and losses due to transmission errors. If part of the path includes a satellite channel, these parameters can vary substantially from those found on wired networks. We make the following assumptions about the performance characteristics of future systems:

- **Latency:** The three main components of latency are propagation delay, transmission delay, and queueing delay. In the broadband satellite case, the dominant portion is expected to be the propagation delay. For connections traversing GEO links, the one-way propagation delay is typically on the order of 270 ms, and may be more depending on the presence of interleavers for forward error correction. Variations in propagation delay for GEO links are usually removed by using Doppler buffers. Therefore, for connections using GEO links, the dominant addition to the end-to-end latency will be roughly 300 ms (one way) of fixed propagation delay. In the LEO case, this can be an order of magnitude less. For example, satellites at an altitude of 1000 km will contribute roughly an additional 20 ms to the one way delay for a single hop; additional satellite hops will add to the latency depending upon how far apart are the satellites. However, the delay will be more variable for LEO connections since, due to the relative motion of the LEO satellites, propagation delays will vary over time, and the connection path may change. Therefore, for LEO-based transport connections, the fixed propagation delay will generally be smaller (such as from 40-400 ms), but there may be substantial delay variation added due to satellite motion or routing changes, and the queueing delays may be more significant [19].

- **Asymmetry:** With respect to transport protocols, a network exhibits asymmetry when the forward throughput achievable depends not only on the link characteristics and traffic levels in the forward path but also on those of the reverse path [5]. Satellite networks can be asymmetric in several ways. Some satellite networks are inherently bandwidth asymmetric, such as those based on a direct broadcast satellite (DBS) downlink and a return via a dial-up modem line. Depending on the routing, this may also be the case in future hybrid GEO/LEO systems; for example, a DBS downlink with a return link via the LEO system causes both bandwidth and latency asymmetry. For purely GEO or LEO systems, bandwidth asymmetries may exist for many users due to economic factors. For example, many proposed systems will offer users with small terminals the capability to download at tens of Mb/s but, due to uplink carrier sizing and the cost of power amplifiers, will not allow uplinks at rates faster than several hundred Kb/s or a few Mb/s unless a larger terminal is purchased.

- **Transmission errors:** Bit error ratios (BER) using legacy equipment and many existing transponders have been poor by data communications standards; as low as 10^{-7} on average and 10^{-4} worst case. This is primarily because such existing systems were optimized for analog voice and video services. New modulation and coding techniques, along with higher powered satellites, should help to make normal bit error rates very low (such as 10^{-10}) for GEO systems. For LEO systems, multipath and shadowing may contribute to a more variable BER, but in general those systems are also expected to be engineered for "fiber-like" quality most of the time.[1]

- **Congestion:** With the use of very high frequency, high bandwidth radio or optical intersatellite communications links, the bottleneck links in the satellite system will likely be the links between the earth and satellites. These links will be fundamentally limited by the uplink/downlink spectrum, so as a result, the internal satellite network should generally be free of heavy congestion. However, the gateways between the satellite subnetwork and the Internet could become congested more easily, particularly if admission controls were loose.

In summary, we assume future satellite networks characterized by low BERs, potentially high degrees of bandwidth and path asymmetry, high propagation delays (especially for GEO based links), and low internal network congestion. These assumptions were used to drive our protocol design and performance analyses described in the rest of the paper.

3 Satellite TCP Fundamentals

This section describes basic TCP operation, identifies protocol options helpful to satellite performance, and discusses some outstanding performance problems. For a more comprehensive overview of TCP, the interested reader is directed to [41].

3.1 Basic TCP Operation

TCP provides a reliable, end-to-end, streaming data service to applications. A transmitting TCP accepts data from an application in arbitrarily-sized chunks and packages it in variable-length segments, each indexed by a sequence number, for transmission in

[1]With advances in error correction, links are more likely to be in one of two states: error free, or completely unavailable.

IP datagrams. The TCP receiver responds to the successful reception of data by returning an acknowledgment to the sender, and by delivering the data to the receiving application; the transmitter can use these acknowledgments to determine if any segments require retransmission. If on the sending side the connection closes normally, the sending application can be almost certain that the peer receiving application successfully received all of the data.

TCP has been heavily used in the Internet for over a decade, and a large part of its success is due to its ability to probe for unused network bandwidth while also backing off its sending rate upon detection of congestion in the network; this mechanism is known as "congestion avoidance" [25]. An additional mechanism known as "slow start" is used upon the start of the connection to more rapidly probe for unused bandwidth. The operation of these mechanisms is described in detail in [41], and is briefly summarized here. TCP maintains a variable known as its *congestion window*, which is initialized to a value of one segment upon connection startup. The window represents the amount of data that may be outstanding at any one time, which effectively determines the TCP sending rate. During slow start, the value of the congestion window doubles every round trip time (RTT), until congestion is experienced (a loss occurs). Upon detection of congestion, the missing segment is retransmitted, the window is halved, and the congestion avoidance phase is entered. During this phase, the congestion window is increased by at most one segment per RTT, and is again halved upon detection of further congestion. Finally, if any retransmissions are lost (which may indicate more serious congestion), the TCP sender is forced to take a timeout, which involves again retransmitting the missing packet, but this time reducing the window to one segment and resuming slow start. For satellite connections, this timeout period and the following slow start result in several seconds during which the throughput is very low.

As originally specified, TCP did not perform well over satellite networks (or high latency networks in general) for a number of reasons related to the protocol syntax and semantics. Over the past decade, a number of TCP extensions have been specified which improve upon the performance of the basic protocol in such environments:

- **Window scale [26]:** TCP's protocol syntax originally only allowed for windows of 64 KB. The window scale option significantly increases the amount of data which can be outstanding on a connection by introducing a scaling factor to be applied to the window field. This is particularly important in the case of satellite links, which require large windows to realize their high data rates.

- **Selective Acknowledgments (SACK) [31]:** Selective acknowledgments allow for multiple losses in a transmission window to be recovered in one RTT, significantly lessening the time to recover when the RTT is large.

- **TCP for Transactions (T/TCP) [6]:** TCP for Transactions, among other refinements, attempts to reduce the connection handshaking latency for most connections, reducing the user-perceived latency from two RTTs to one RTT for small transactions. This reduction can be significant for short transfers over satellite channels.

- **Path MTU discovery [33]:** This option allows the TCP sender to probe the network for the largest allowable Message Transfer Unit (MTU). Using large MTUs is more efficient and helps the congestion window to open faster.

The IETF is in the process of creating an informational standard that identifies which standardized TCP options should be used in future implementations [2]; Partridge and Shepard also discuss some of these transport improvements [37].

In this work, we are interested in quantifying the performance of TCP implementations were they to use these latest standard enhancements. Note that even though some of these options have been specified for over five years, not all implementations use them today. The lack of widespread vendor support for satellite-friendly protocol options has historically been a hindrance to achieving high performance over satellite networks.

3.2 Unresolved Problems

Despite the progress on improving TCP, there remain some vexing attributes of the protocol that impair performance over satellite links. For these problems, there are no standardized solutions, although some are currently under study:

- **Slow start "ramp up":** TCP's slow start mechanism, while opening the congestion window at an exponential rate, may still be too slow for broadband connections traversing long RTT links, resulting in low utilization. This problem is exacerbated when slow start terminates prematurely, forcing TCP into the linear window growth phase of congestion avoidance early in the connection [37]. Researchers are now considering allowing a TCP connection to use an initial congestion window of 4380 bytes (or a maximum of 4 segments) rather than one segment [1]. Transfers for file sizes under roughly 4K bytes (many Web pages are less than this size) would then usually complete in one RTT rather than two or three. In the following, we refer to this policy as "4K slow start" (4KSS). Other researchers have investigated the potential for caching congestion information from a recently used connection in order to start the new connection from a larger initial window size [35],[46].

- **Link asymmetry:** The throughput of TCP over a given forward path is maximized when the reverse path has ample bandwidth and a low loss rate, because TCP relies on a steady stream of acknowledgments (ACKs) to advance its window and clock out new segments in a smooth manner. When the reverse path has limited bandwidth, the TCP acknowledgment stream becomes burstier, as ACKs are clumped together or dropped. This has three effects: i) the sending pattern becomes more bursty, ii) the growth of the congestion window (which advances based on the number of ACKs received) slows, and iii) the "fast retransmit" mechanism that avoids retransmission timeouts becomes less effective. Since TCP acknowledgments are cumulative, researchers have recently studied ways to reduce the amount of ACK traffic over the bottleneck link by "ACK congestion control" and sender algorithms that grow the window based on the amount of data acknowledged and that "pace out" new data transmission by using timers [5]. This has the drawback of requiring transport-layer implementation changes at both ends of the connection. An alternative approach reintroduces the original ACK stream at the other end of the bottleneck link ("ACK filtering and reconstruction"). This does not require changes at the TCP sender, but is more challenging to implement [5]. Finally, if the MTU for the constrained reverse channel is small, the Path MTU discovery mechanism will select the small MTU for the forward path also, reducing performance.

- **Implementation details** In many implementations, applications must explicitly request large sending and receiving buffer sizes to trigger the use of window scaling options. For example, default socket buffer sizes for many TCP implementations are set to 4KB [22]. Unfortunately, this requires users to manually configure applications and TCP implementations to support large buffer sizes; moreover, some applications do not permit such configuration, including common Web servers [22]. Also, because TCP can only negotiate the use of window scaling during connection setup, unless it has cached the value of the RTT to the destination, it cannot invoke window scaling upon finding out that the connection is a long RTT connection. In addition, even if T/TCP is present in an implementation, applications based on the sockets Application Programming Interface (API) often use system calls that prevent the usage of T/TCP. Because the TCP standard is not rigorously defined or followed, different vendor implementations often have different (and buggy) behavior (see, for example, [38]). The subtle performance effects of these variations can significantly manifest themselves over satellite channels.

- **TCP fairness** Perhaps the most challenging problem is that TCP's congestion avoidance algorithm results in drastically unfair bandwidth allocations when multiple connections with different RTTs share a bottleneck link. The bias goes against long RTT connections by a factor of RTT^{α}, where $\alpha < 2$ [27]. This problem has been observed by several researchers (e.g., [27],[17]), but a viable solution has not yet been proposed, short of modifying network routers to isolate and protect competing flows from one another [45]. Furthermore, bandwidth asymmetry exacerbates the fairness problems by shutting out certain connections for long periods [28]. While theoretical results suggest that it may be possible to design a distributed algorithm that simultaneously converges to fair allocations in bandwidth with high utilizations of bottleneck links [32], no such algorithm has been successfully constructed in practice.

3.3 Outline of Experiments

In remainder of this report, we address the following research questions:

1. (Section 4) Satellite TCP connections for which a portion of the connection traverses the wired Internet are subject to severe throughput degradation if the packets flow through a queue that is being congested by connections with a short round-trip time (RTT). Can this bias against long RTT connections be overcome by simple changes to the congestion avoidance algorithm in end hosts?

2. (Section 5) In the current Internet, there exist a wide variety of TCP implementations with various options that interact in different ways. What implementation options contribute to poor performance? What is the best combination of (standard) TCP options and implementation guidelines for use over satellite channels?

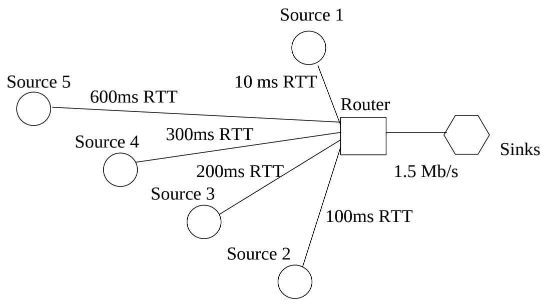
3. (Section 6) How much performance advantage can be gained by "splitting" a TCP connection at a gateway located at the satellite terminal equipment connected to the wired Internet, thereby shielding the satellite subnetwork from the rest of the Internet?

4 TCP Fairness in a Heterogeneous Environment

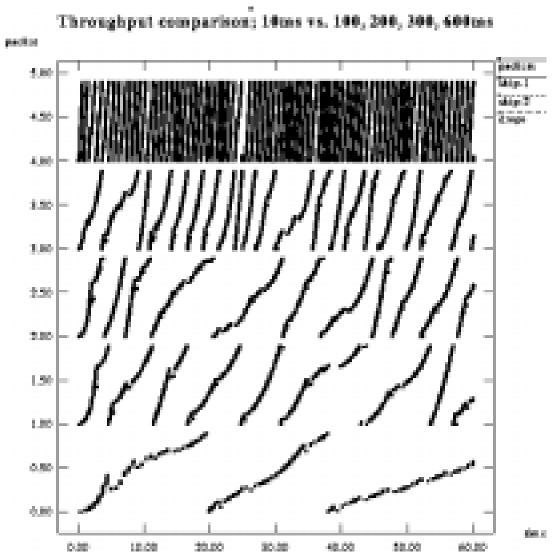
4.1 Introduction

The fairness problem in TCP is rooted in its congestion avoidance mechanism, which we described above in Section 3.1. The congestion avoidance phase is sometimes referred to as "additive increase and multiplicative decrease," because, in the absence of congestion, segments are added to the window over time, while in the presence of congestion, the window value is multiplied by one half.

The "additive increase and multiplicative decrease" algorithm in TCP derives from a similar algorithm in the DECnet protocol [25]. Chiu and Jain showed that this algorithm leads to fair allocations of network bandwidth even though it operates in a distributed manner [10]. However, their analysis presumes that all connections in the network share the same additive increase rate and multiplicative decrease factor. In TCP, the multiplicative decrease factor (1/2) is the same for all connections,



(a) Congested network topology.



(b) TCP sequence number evolution vs. time.

Figure 2: A demonstration of the unfairness of the current TCP congestion avoidance algorithm. The connections, from top to bottom, have RTTs of 10, 100, 200, 300, and 600 ms respectively.

but the policy of an additive increase of one segment per round trip time (RTT) does not provide a uniform increase in the rates of TCP connections with different RTTs[2]. In particular, connections with long RTTs open their window more slowly than those with short RTTs. And if a mixture of such short and long RTT connections share a bottleneck link, severe unfairness is inevitable as the short RTT connections grab the available bandwidth well before the long RTT connections have a chance [21].

Figure 2 illustrates an example of this problem by showing simulation results for a 60 second trace of TCP connections over the illustrated topology. In Figure 2, the evolution of the sequence number is plotted for 5 connections (from top to bottom, with RTTs of 10, 100, 200, 300, and 600 ms) sharing the same bottleneck link. The sequence number in this simulation is on a per segment basis, and the plots wrap after every 90 segments. The long RTT connections do not obtain an allocation close to their fair share of the bottleneck link, and their overall throughput performance suffers drastically.

To combat the bandwidth inequities that result from heterogeneous RTTs, Floyd proposed a modification to TCP's window adjustment algorithm that counteracts the RTT bias. In this section, we elaborate Floyd's "Constant-Rate" algorithm [14] with a thorough investigation of the performance achievable by both universally and selectively (i.e., incrementally) deploying a TCP with a modified window increase policy in the congestion avoidance phase of the connection.

Floyd [14] developed a fairly general characterization of window increase algorithms that facilitates the discussion of fairness. Although TCP maintains its send window in units of bytes, we find it more convenient herein to maintain it in units of segments. A key assumption is that a number of segments approximately equal to the send window size is sent every RTT; this is generally true for long RTT connections. Let c be the increase (in segments) in the size of the send window for a connection in one round trip time RTT. Therefore the window grows at a rate of c/RTT segments per second when additive increase is in effect. In conventional TCP, $c = 1$. Floyd refers to the standard TCP policy as an "increase by 1" policy.

If one were to scale the window growth rate c/RTT by RTT, the effect would be to build the window at a constant rate of c segments per second, independent of the RTT. However, the window growth rate does not equal the growth rate in data transmission. As [15] points out, it results in a "linear-in-RTT" bias in the sending rate. Because each connection can send a full window's worth of segments each RTT, shorter RTT connections achieve greater throughput over a common time interval. To fully remove the bias, we must change the additive increase to $c * RTT$ segments per second; i.e., a factor of RTT^2 faster than the original algorithm. Floyd defines such an increase as a *Constant-Rate (CR) increase policy*, since it can roughly be interpreted as causing the rate of segment transmission to increase at a constant rate.

Figure 3 demonstrates the behavior of two of these policies for two connections with different round trip times. The figure plots the equation

$$P_{c_i,RTT}(t) = \sum_{k=0}^{\lfloor t/RTT \rfloor} (cwnd + kc_i),$$

for the different RTTs and policies c_i. This equation describes the number of segments sent ($P_{c_i,RTT}(t)$) assuming a window of segments is sent each RTT and there are no losses, where c_0 is equal to 1 (the standard policy), and c_1 implements the constant

[2]In the following, we distinguish between the overall congestion avoidance *algorithm* and the *policies* implemented in this algorithm such as the multiplicative decrease factor of 1/2 and the additive increase of one segment per RTT.

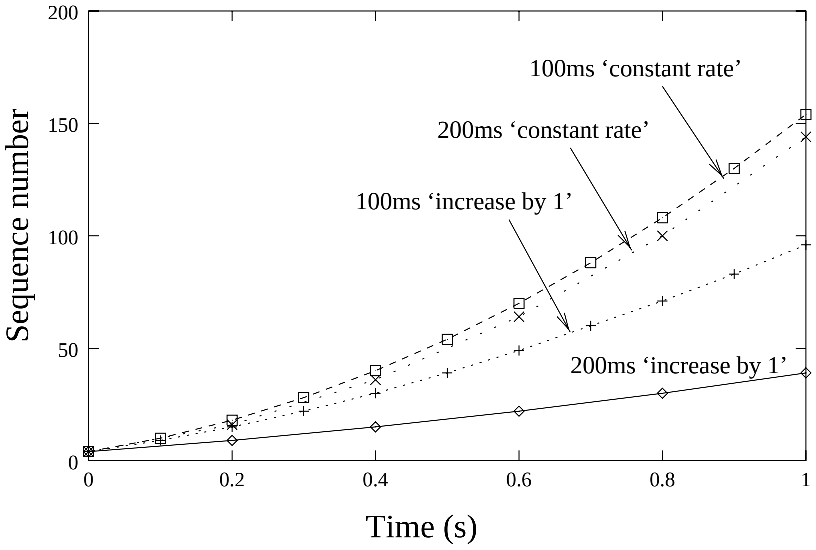


Figure 3: Simulated sequence number evolution for connections with different RTTs under both the constant rate and increase-by-one policies.

rate policy (and hence is a different value for each RTT). While only a rough approximation, the graph confirms the shape of the increase rate curve for each policy.

4.2 Methodology

We used the UCB/LBNL Network Simulator "ns" [3] to evaluate TCP performance. In addition to using the standard ns modules, we ported the HTTP traffic generator module from Bruce Mah's INSANE simulator[4], which gave us the ability to add a mix of realistic background traffic to our simulations.

Our study focused on large file transfer performance. While short HTTP transfers and Telnet connections over long RTT paths are also subject to performance degradation during periods of congestion, this degradation is due more to the fundamental latency of long RTT connections than to problems with congestion avoidance. Additionally, HTTP protocol implementations are migrating towards "persistent-HTTP" and longer duration TCP connections. We also did not assume the implementation of fair scheduling and TCP-friendly buffer management that can isolate flows or classes of flows from one another (e.g., as discussed in [45]), or pricing structures that might give network providers incentives to protect the throughput of paying customers. In short, we assumed an environment similar to the present day Internet, with the addition of Random Early Detection (RED) queues [18], and the latest in standardized TCP improvements (Selective Acknowledgments (SACK) [31] and large window enhancements [26]).

4.2.1 Performance Metrics

A number of metrics for quantifying fairness have been proposed but no single metric has common acceptance [14]. In this paper, we consider a "fair share per link" metric; i.e., if there are n flows through a bottleneck link, each flow has the right to $1/n$th of the capacity of that bottleneck link. Jain's metric of fairness [10] is applicable in this context. For n flows, with flow i receiving a fraction b_i on a given link, the fairness of the allocation is defined as:

$$Fairness \equiv \frac{(\sum_{i=1}^{n} b_i)^2}{n * (\sum_{i=1}^{n} b_i^2)}.$$

This metric ranges continuously in value from $1/n$ to 1, with 1 corresponding to equal allocation for all users. Utilization is another important metric, since high fairness is of little use if the link capacity is grossly underutilized. Utilization is defined herein as the number of original bits (i.e., not counting retransmissions) successfully transferred over a link during some time

[3]http://www-mash.cs.berkeley.edu/ns/
[4]http://http.cs.berkeley.edu/~bmah/Software/HttpModel/

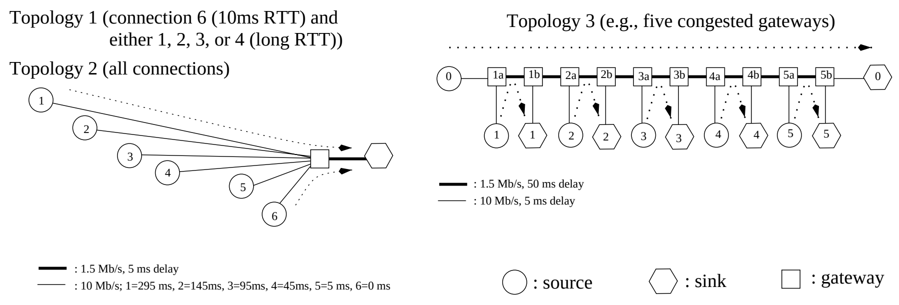

Topology 1 (connection 6 (10ms RTT) and either 1, 2, 3, or 4 (long RTT))
Topology 2 (all connections)
1
2
3
4
5
6
: 1.5 Mb/s, 5 ms delay
: 10 Mb/s; 1=295 ms, 2=145ms, 3=95ms, 4=45ms, 5=5 ms, 6=0 ms
Topology 3 (e.g., five congested gateways)
0
1a
1b
2a
2b
3a
3b
4a
4b
5a
5b
0
1
1
2
2
3
3
4
4
5
5
: 1.5 Mb/s, 50 ms delay
: 10 Mb/s, 5 ms delay
: source
: sink
: gateway

Figure 4: Three simulation topologies.

interval divided by the product of link rate and that time interval; this is often called "goodput."

$$Utilization \equiv \frac{(segments\_acked) * (segment\_size)}{(link\_rate) * time}.$$

4.2.2 Topologies

We explored a number of test configurations that allowed us to isolate selected behavior of both the standard and our proposed window adjustment policies. We selected the topologies illustrated in Figure 4; similar test configurations have previously been used by the research community to study the effects of congestion. The first test configuration (Topology 1) was used to study the effects of two long duration connections, one with a short RTT (10 ms) and one with a long RTT (100-600 ms), sharing a single bottleneck link. The second test configuration (Topology 2) was used to examine the effects of many competing connections over a single bottleneck link. The six connections have RTTs of 10, 20, 100, 200, 300, and 600 ms. The third configuration (Topology 3) was used to examine the effects of long RTT connections that must also traverse a number of network hops populated by short RTT connections. This topology is very similar to the one previously used by Floyd to study the CR algorithm [14]; the number of congested gateways could vary between 1 and 10 (the figure illustrates 5 congested gateways).

4.2.3 Configuration Details

We studied the performance of two different TCP variants: TCP NewReno, and TCP SACK. We note that other researchers have detected problems with using TCP Reno (a version of TCP that does not perform adequately when multiple drops occur in a window of data) in combination with congestion avoidance mechanisms that try to add more than one segment per RTT [8]; therefore, we avoided such implementations. We also examined two different queueing schemes: traditional "first-in, first-out" (FIFO) queueing, and Random Early Detection (RED) with packet discard.[5] In our simulations, data packet sizes were fixed at 1000 bytes, and the bottleneck link speed was 1.5 Mb/s. We examined a range of queue sizes from 4 to 50 packets, but in the data that follows, we concentrate on a RED queue size of 50 packets with a "minimum threshold" of 20 packets and a "maximum threshold" of 40 packets; all other RED parameters were set to the ns defaults. 20 packets in this case is approximately 100 ms at our output line rate.

4.2.4 Data Analysis and Presentation

TCP throughput in an environment containing random traffic can be quite variable, because small changes in initial conditions can cause wide variations in resulting behavior. Therefore, we computed the utilization and fairness of a particular configuration as follows. We first ran enough independent simulations such that the sample standard deviation of each connection's throughput was within 5% of its sample mean (this generally required around fifty runs). We then used these sample means to compute the fairness and utilization of a given topology. In the remainder of Section 4, if the experimental data does not explicitly list error bars, the reader may assume that the sample standard deviation is within 5% of the value listed. In the following subsections, we first provide some benchmark data, followed by an analysis of the Constant-Rate policy, followed by experiments aimed at selectively increasing the aggressiveness of a long-delay TCP connection.

[5]RED queues operate by computing an exponentially weighted moving average of the queue size. When the average queue size is below some minimum threshold, the queue does not drop any packets. When the average queue size is between the minimum and maximum threshold, the queue probabilistically drops incoming packets according to an algorithm described in [18]. When the average queue size exceeds the maximum threshold, the queue drops every incoming packet. The instantaneous queue depth can exceed the maximum threshold if the average queue depth is below the maximum threshold.

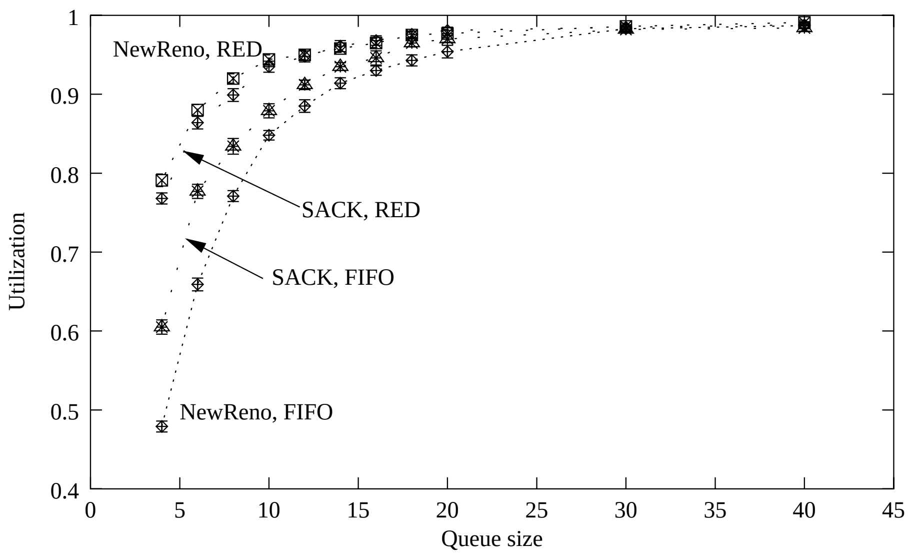


Figure 5: Benchmark performance results using Topology 2: Utilization vs. queue size.

4.3 Benchmark Results

To calibrate our simulation studies, we established a set of benchmark performance results for each of our test topologies. Though used principally to gauge the efficacy of our proposed policies, the benchmark data itself reveals some interesting effects. In this section, we examine benchmark performance data from Topology 2 in Figure 4.

Figures 5 and 6 plot the utilization and fairness of the standard TCP window adjustment policy for Topology 2. The error bars on these figures represent 99% confidence intervals, and are very small. We show results from the combination of two TCP variants, NewReno and SACK, with two queueing disciplines, FIFO and RED. In this representative data set (and in our other benchmark data), the following trends are evident:

- The utilization of the bottleneck link improves with increased queue size, because, under congestion, large queues keep the link busy as they drain out while TCP sources back off their sending window. Additionally, larger queues absorb bursts of packets and prevent coarse timeouts, which cause long idle periods. The capability of absorbing traffic bursts also cause RED queues to outperform FIFO queues.

- Network fairness is poor in almost all cases, except when queue sizes are very large. In general, the two short delay connections obtained roughly 50% of the bandwidth, and the background WWW traffic consumed 20% of the bandwidth. The remaining 30% was split $unequally$ among the 4 long RTT connections, with the longest connection receiving only about 24 to 64 kb/s (2 to 4%) for TCP NewReno with FIFO queueing. While even larger queue sizes may help further, they would also introduce more significant delay variability.

- In general, when queue sizes are reasonably large and when all TCPs use SACK instead of NewReno, the network fairness is marginally better. This is most likely due to SACK's superiority in recovering from multiple drops in a single window. Since multiple congestive losses in a single window are more likely to occur in a connection with a long RTT, the use of SACK helps such connections. However, the use of SACK by all TCP connections does not, by itself, remove the bias against long RTT connections.

In general, RED queues perform much better in terms of utilization and fairness than do FIFO queues. However, we found that the use of RED and SACK alone, without modifications to TCP sending behavior, still leaves much room for improvement in fair bandwidth allocation. RED queues equalize the bandwidth of flows with similar RTTs, but do not do so for flows with heterogeneous RTTs, as pointed out in [18]. In the remainder of this section, we focus on enhancing the performance of TCP implementations that use SACK and networks using RED queues.

4.4 Performance of the Constant-Rate Policy

In this section, we describe the case in which each foreground and background TCP connection uses a Constant-Rate (CR) policy. In TCP implementations, an additive increase to the TCP variable snd_cwnd (send congestion window) of approximately

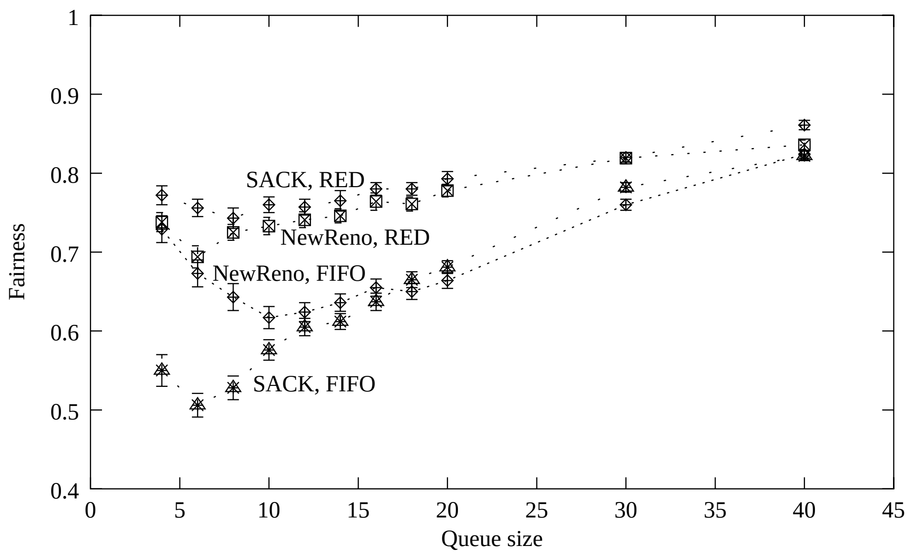


Figure 6: Benchmark performance results using Topology 2: Fairness vs. queue size.

one segment per RTT, assuming an acknowledgment (ACK) is received for each segment, is effected by executing the following pseudocode upon receipt of a new ACK:

```
snd_cwnd = snd_cwnd + 1/snd_cwnd.
```

In this manner, the TCP connection gradually adds to its congestion window at the rate of approximately one segment per RTT; this approach to building the congestion window reduces transmission burstiness [25]. To implement a CR policy, we can modify the window increase algorithm to account for the RTT bias:

```
snd_cwnd = snd_cwnd + (c*rtt*rtt)/snd_cwnd,
```

where c is the constant that controls the rate. This policy causes an additive increase in the throughput rate that is the same for all connections. After initial experiments, we observed that the second term of the above equation could lead to very bursty send patterns, which led to increased losses. For example, if the RTT is large and the value of snd_cwnd is small, each ACK can trigger the transmission of several segments. To avoid this behavior, we bounded the increase per ACK by 1 segment; i.e.:

```
snd_cwnd = snd_cwnd +
min((c*rtt*rtt)/snd_cwnd, 1 segment).
```

With this constraint on the sender's behavior, the TCP connection is never more bursty than a TCP connection in slow start. Another approach, with which we did not experiment, would be to smooth the sending of several segments across a longer time period.

One question previously raised by Floyd is how to pick the proper value for the constant c. One way to think of the value of c for CR connections is how the aggressiveness of the CR connection would compare to that of a standard TCP connection with a certain RTT. For example, if $c = 100$, the value of the RTT that makes the numerator equal to 1 in our pseudocode above is 100 ms. Therefore, an environment in which $c = 100$ would have connections that were about as aggressive as normal TCP connections with 100 ms RTTs. We chose to experiment with a range of values, between $c = 4$ (as aggressive as standard 500 ms connections) and $c = 1600$ (25 ms).

In addition to varying the constant c, we experimented with several other variations in an effort to identify which types of environments were suitable for the CR policy:

- TCP NewReno vs. TCP SACK,
- RED vs. FIFO gateways,
- bottleneck queue lengths (or RED maximum queue thresholds) from 4 to 50 packets, and
- TCP RTT timer granularity of 500 ms (standard in many TCP implementations) vs. 10 ms.

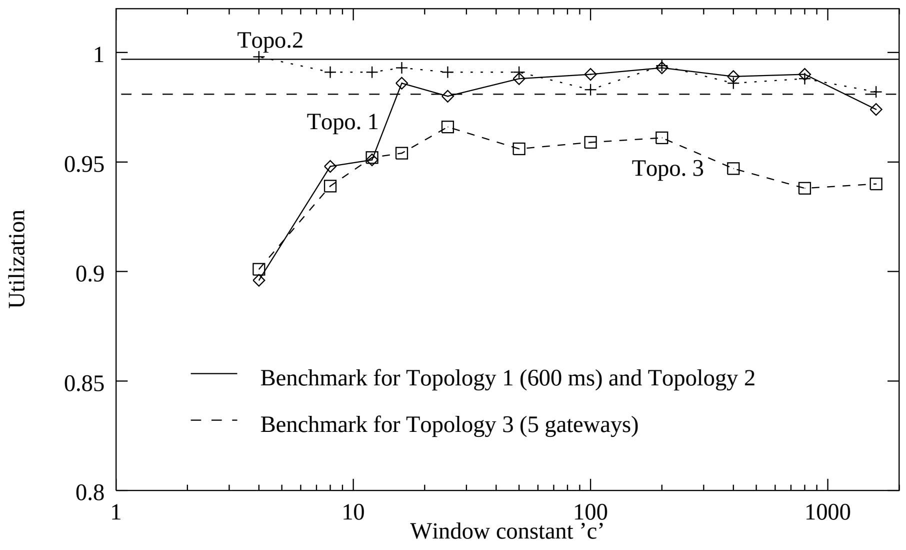


Figure 7: Utilization vs. window constant of TCP SACK with fine-grained RTT estimates over topologies with bottleneck RED queues.

As mentioned above, Topology 3 conforms closely to one with which Floyd experimented [14], and we were able, when using a similar value for the CR constant ($c = 4$), to confirm their results that CR can substantially improve the fairness of connections traversing multiple gateways when all connections use a CR policy. However, we also observed the following general trends in our data:

- Deep RED queues appear to be a prerequisite for good performance of the CR policy. Performance when FIFO queues or short queues were used was very inconsistent, in the sense that there was often no value of c that simultaneously yielded high fairness and high utilization. Moreover, we could not determine strong correlations between the value of c and the fairness and utilization metrics we were using; i.e., the performance was highly sensitive to the particular simulation topology.

- TCP SACK and fine-grained RTT timers were the next most important indicators of good CR performance. The use of SACK helps TCP recover from losses more quickly, which leads to improved and more consistent performance. Also, many existing TCP implementations use a coarse estimate of the RTT, which impairs the ability of our modified congestion avoidance algorithm to determine the true RTT of the connection.

Figures 7 and 8 plot the utilization and fairness performance of TCP SACK over bottleneck RED queues when all connections, including background HTTP traffic, use the same CR policy, constant c, and RTT timer granularity of 10 ms. Topology 1 corresponds to the case in which the long RTT connection has a round trip propagation delay of 600 ms, while Topology 3 in this case corresponds to the topology with 5 congested gateways (the trace is taken from the first congested gateway). For comparison, we also plot as horizontal lines the utilization and fairness achieved when all TCP connections use the standard algorithm (i.e., benchmarks). The data indicates that the fairness can be substantially improved if all connections adopt the CR policy. However, the utilization suffered for small c when there were only two foreground connections in the topology (Topologies 1 and 3). When statistical multiplexing was in full effect (Topology 2), both fairness and utilization were near optimal for small values of c. Additionally, in Topology 3, although the fairness improved substantially, equal allocations were not obtained by the CR policy because the long RTT connection is also traversing multiple congested gateways.

In these experiments, as c became larger, connections became more aggressive, to the point that the bound in our policy of adding 1 segment per ACK was in effect nearly all of the time. Consequently, since the CR policy was no longer being applied, the unfairness reappeared. In general, we observed that the fairness properties were best when the value of c was below 100. However, if too few connections are using the link, such as in Topologies 1 and 3, such a small value of c can lead to lower utilization. Because it is difficult in practice for a given connection to determine the number and type of connections against which it is competing, we conclude the following negative result: a good choice of the constant c cannot be determined with high confidence on an operational basis.

Not only does the CR policy appear difficult to manage in a distributed network, we also found it susceptible to the presence of TCP connections operating under the standard policy. For example, Figure 9 illustrates the fairness performance when a single additional connection using the standard window increase policy was introduced into each of the topologies (and

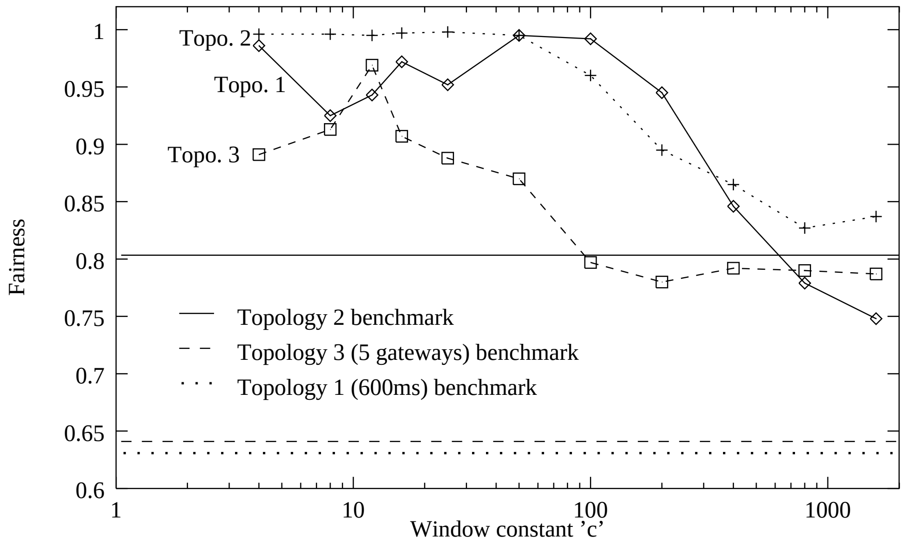


Figure 8: Fairness vs. window constant of TCP SACK with fine-grained RTT estimates over topologies with bottleneck RED queues.

also included in the fairness computation). Although this additional connection slightly improved link utilization, much of the fairness improvement due to CR was lost when this competing connection was introduced, as it not only used a disproportionate share of the bandwidth itself but also acted as a "trailblazer," improving the performance of short RTT connections that were using the CR policy by a disproportionate amount. We also observed similar performance degradation if no extra file transfers were introduced, but instead the HTTP background traffic (20% of the bottleneck link rate, on average) used the standard policy. Similar effects (passive connections competing with more aggressive connections using the standard congestion avoidance algorithm) are also responsible for the poor performance of TCP "Vegas" in a long RTT environment characterized by a mix of heterogeneous TCP implementations [49].

4.5 Selectively Modifying the Additive Increase Policy

We next investigated whether the throughput of an individual long RTT connection could be improved by modifying the additive increase policy of only the long RTT connection. We were interested in two questions:

- Can an individual connection improve its own throughput by becoming more aggressive during additive increase?
- If so, how does the individual connection's more aggressive behavior affect the performance of other (unmodified) connections using the same path?

To study the first question, we experimented with an "increase-by-K" (IBK) policy rather than the standard "increase-by-one," again limited by a maximum increase of one segment per ACK. In other words, we used the following pseudocode in our implementation:

```
snd_cwnd = snd_cwnd +
min((K/snd_cwnd),1 segment).
```

For example, by setting $K = 2$, we built the window by roughly 2 segments per RTT.[6] Again, the increase is bounded by 1 segment per ACK, but this is the general trend.

Figure 10 illustrates fairness results, calculated over the entire simulated network, for the case in which only one connection in the simulated topology used the IBK policy. In particular, we enabled the IBK policy on the longest RTT connections in each of the three topologies, and then repeated the experiment by enabling the IBK policy on only the 300 ms connection in Topology 2. In the graph, the value of $K = 1$ (left-most data points) corresponds to the normal (benchmark) case. We observed that the long RTT connection was able to steadily improve its performance over that of the benchmark case by increasing K across the range of values we considered. This resulted in improved fairness in all topologies for small values of $K > 1$. For larger values, even though the throughput of the long RTT connection continued to improve, fairness actually decreased in some topologies as the more aggressive connection began to take more bandwidth than its fair share.

[6]If delayed acknowledgments are being used, this is akin to correcting the window growth penalty that is due to delayed acknowledgments.

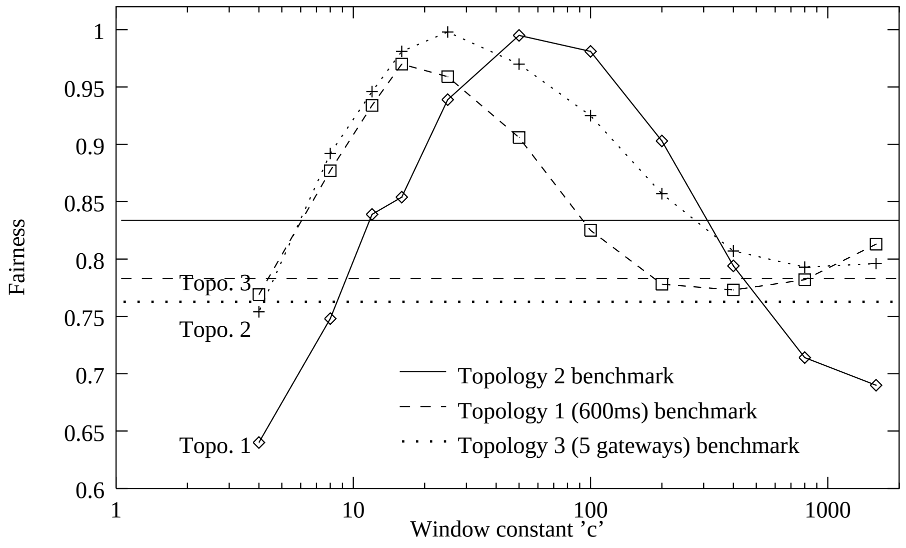


Figure 9: The sensitivity of the proper selection of the constant c when the effects of a standard TCP connection are added to the performance shown in Figure 8.

	benchmark	**300 ms**	**600 ms**
Utilization	0.99	0.99	0.99
Fairness	0.807	0.861	0.873
	Throughput (kb/s)		
600 ms	**68.1 (0.9)**	59.6 (0.6)	**138.7 (2.0)**
300 ms	**116.2 (1.1)**	**240.8 (2.8)**	109.1 (1.1)
200 ms	156.6 (1.7)	139.8 (1.5)	151.6 (1.1)
100 ms	217.1 (2.1)	188.1 (1.9)	207.0 (1.8)

Table 1: Effect of the IBK policy on throughput (Topology 2, K = 4). 99% confidence intervals are shown in parentheses.

In Figure 10 we plotted the performance of TCP SACK over RED queues, and found that there was no limit to the improvement that a more aggressive connection could obtain for itself. We repeated the experiment for TCP NewReno over FIFO queues, and found that connections could increase their own performance, independent of the topology, by using a value of K of up to 4 or so. However, for higher values of K, performance degraded, because the sending behavior became too bursty for the FIFO queues to successfully absorb.

Because the performance improvements result from increasing TCP's aggressiveness, we should be concerned that this can have a negative impact on other peer connections. Remarkably, we found that, in every case we examined, the average fairness index *always* improved, and the average utilization held relatively constant, when the more aggressive connection used a modest value of K (less than 8 or so). This improvement occurred regardless of whether TCP SACK or NewReno was used, or whether FIFO or RED queues were present. In fact, the majority of the redistributed bandwidth came from connections that were already using more than their fair share. The effect on other connections was similar to what they would have experienced had the long RTT connection actually been a connection with a somewhat shorter RTT.

For example, Table 1 provides an example of the magnitude of the performance gains achievable. For the value $K = 4$, we tabulate the utilization, fairness, and throughput of the four longest RTT connections in Topology 2. In the first column are results from when each connection used the standard policy, the second column shows the results from when only the 300 ms connection used an IBK policy, and the third column is from when only the 600 ms connection used an IBK policy. This table illustrates that the more aggressive connections did not seriously harm the throughput of the peer connections. The throughput gains (highlighted in bold font) are substantial; in many cases, throughput increases (and hence reductions in user-perceived latency) exceeded 100% when RED queues were used, and 50% when FIFO queues were traversed.

We next investigated whether the performance gains were sustainable when multiple connections become more aggressive by examining this case with Topology 2. We found that the aggressive connections were able to simultaneously improve their

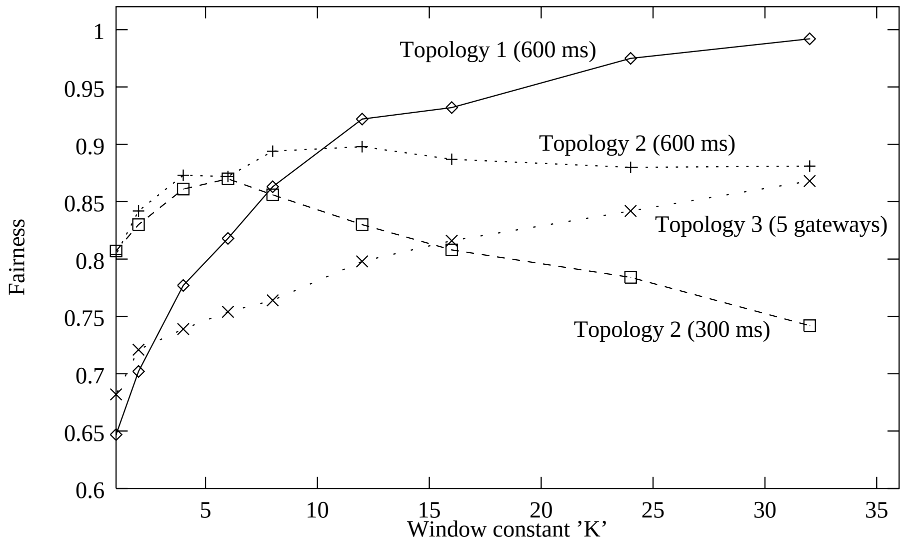


Figure 10: Improvement in fairness vs. window constant due to the IBK policy.

own performance, although their relative performance gains were not what they would have achieved had they been the only aggressive connection. Of course, if every connection adopted an IBK policy, the fairness situation would be back to the standard case, so there must be some RTT threshold beyond which connections can become more aggressive if such a policy is to work in practice. Finally, we experimented with the case in which congestion was induced in both directions of data transfer. This had the effect of disrupting the ACK stream to some extent, but did not significantly affect our main results.

4.6 Implementation Issues

We have already discussed some minor implementation changes to the code segment which builds the congestion window. Throughout the discussion, we implicitly assumed that the TCP connection had an accurate estimate of its RTT. In practice, this is not the case. TCP does maintain a smoothed round trip time ($srtt$), but because of the timer granularity of 500 ms in TCP, this value is not very accurate. It is, however, rather easy to improve the RTT accuracy through use of the TCP timestamps option [26]. A sending TCP implementation can put a more accurate timestamp in the TCP timestamp field, which is merely reflected by the receiver; such a technique is suggested for TCP Vegas[7]. However, it is important not to base the retransmission timer value on this accurate timestamp, because TCP fast retransmit and fast recovery rely on the $srtt$ variable being much larger than the actual RTT. We experimented in ns with running the TCP timer granularity at 1 ms instead of 500 ms, and found that the more accurate $srtt$ value caused coarse timeouts to trigger before fast recovery could be accomplished.

One practical issue is that modifications to the sending algorithm of an implementation have little use if the implementation is a client of a large data transfer rather than the source of the data. However, it is possible to indirectly modify the sender's behavior by actions taken at the receiver. For example, by sending more ACKs back to the sender, the receiver can "speed up" the sender; e.g., if one were to receive a segment of 1000 bytes, sending ACKs 1:250, 250:500, etc. would quadruple the window build rate. We did not experiment with this technique, and note that it is of limited utility when the reverse channel is constrained and cannot handle the additional ACK traffic.

Another alternative to increasing the aggressiveness of a single connection is to run multiple connections in parallel, coordinated by the application or some type of session manager. This technique, sometimes called "striping", has been in use for some time by satellite operators and WWW browser software. One advantage of this approach is that it overcomes small offered windows by the receiver. This technique, however, can impact the network more than our approach since multiple slow starts are launched into the network simultaneously. Recent results in which congestion window state is shared across the multiple connections can potentially combat the problem [5]. In general, the use of striping without the constraints outlined in [5] is not favorably viewed by the research community.

4.7 Summary

In this section, we have presented the results of our investigation of simple changes to TCP's congestion avoidance algorithm in an effort to improve its fairness properties. While we found that the Constant-Rate (CR) policy could improve fairness dramatically, we faced two practical difficulties that would likely prevent universal deployment of this scheme in its current

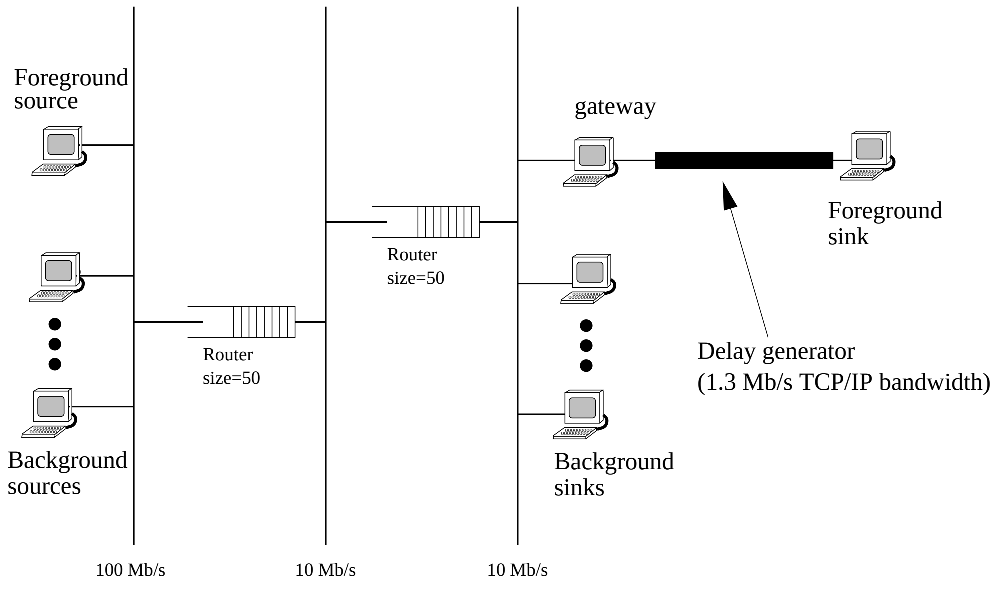


Figure 11: Configuration for network experiments.

form: i) the proper selection of a constant is dependent upon the network topology and the number of peer connections and is therefore difficult to determine in a distributed manner, and ii) the fairness benefits of the CR policy can be confounded by competing connections using standard congestion avoidance, thereby making it disadvantageous to deploy CR in an existing heterogeneous environment. However, when we instead made only certain long RTT connections slightly more aggressive, we were always able to improve network fairness while keeping bottleneck link utilization relatively constant by using an increase-by-K (IBK) policy. Interestingly, the effects on other unmodified connections that were sharing the bottleneck link were similar to what they would have experienced had the modified connection actually been a connection with a shorter RTT.

Our results indicate that it may be beneficial for long RTT connections (running TCP SACK) to become slightly more aggressive during the additive increase phase of congestion avoidance. While our data set is not comprehensive enough to allow us to advocate a particular policy at this time, the IBK policy for small values of K (such as 2 or 4) may significantly improve the throughput while not significantly impacting other flows. Such a policy could be invoked in practice when the TCP implementation detects that the connection has an RTT above a certain threshold (for example, connections traversing a GEO satellite link have a much larger RTT– at least 500 ms– than terrestrial connections). A TCP receiver could even induce the sender into an IBK policy by acknowledging data in smaller chunks. Determining appropriate values for K as a function of RTT, as well as determining the accuracy and resolution required of TCP's RTT estimates, could be the focus of future work.

5 End-to-End TCP Performance over Satellite Links

5.1 Introduction

In this section, we quantify how well current TCP implementations perform in a satellite environment composed of one or more satellites in geostationary orbit (GEO) or low-earth-orbit (LEO), particularly when the satellite channel forms only a part of the end-to-end connection, as depicted in Figure 1. We focused on two types of workload found most commonly in the Internet: large file transfers, and short Web connections.

Our assumptions about future satellite network characteristics are shaped by projections of future commercial systems that will offer Internet connections at up to broadband (tens of Mb/s) data rates via networks of LEO or GEO satellites (or hybrid constellations). Users may contact other hosts in either the satellite network or the wide-area Internet. We discussed some of our assumptions about the transmission and congestion characteristics of the end-to-end path using such satellite systems in Section 2. In short, we assume future satellite networks characterized by low BERs, potentially high degrees of bandwidth and path asymmetry, high propagation delays (especially for GEO based links), and low internal network congestion. These assumptions were used to drive our protocol design and performance analyses described in the rest of this chapter.

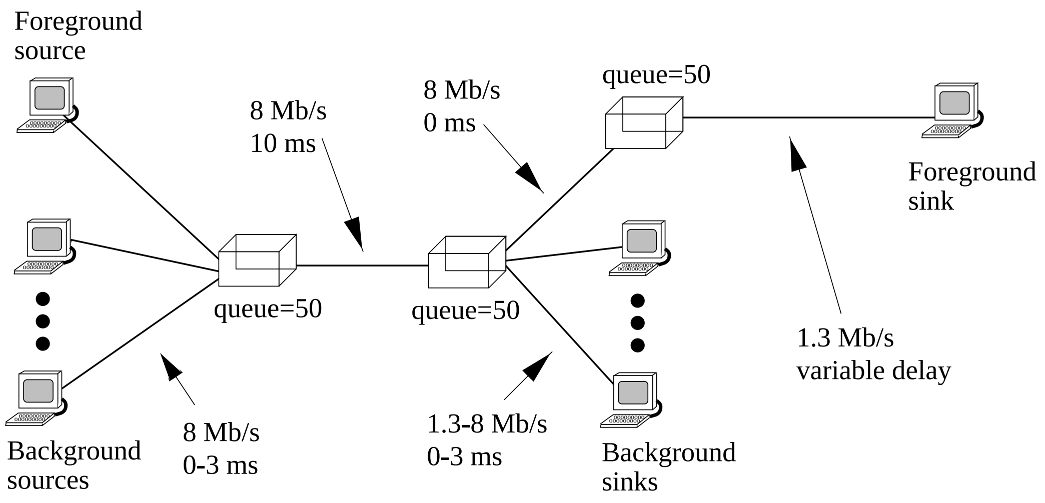


Figure 12: Configuration for simulation experiments.

5.2 Methodology

This experimental methodology discussed in this subsection pertains to the experiments of both Sections 5 and 6.

5.2.1 Experimental Setup

Our experiments were conducted using hosts, running BSD/OS 3.0 UNIX, connected to Ethernets in a local-area subnet at Berkeley. The TCP implementations on these machines are derived from 4.4BSD-Lite (also known as Net/3 [47]), with modifications to support our experiments. We configured the receivers to offer the largest window possible (240 KB) to the senders. For the experiments, traffic sources were connected to a 100 Mb/s Ethernet, and traffic sinks were on a 10 Mb/s Ethernet separated by a 10Mb/s transit Ethernet segment. Figure 11 illustrates the experimental topology. To generate traffic, we used a combination of the `sock` program [41] for bulk file transfers and a HTTP traffic generator for testing of 4KSS and T/TCP. This traffic generator generated small file transfers according to empirical distributions drawn from Bruce Mah's HTTP traces [30]. We implemented STP in the BSD/OS UNIX kernel.

For investigating satellite transport protocol performance, it is usually sufficient to experiment with delay and error simulators rather than with detailed emulators of the transmission channel. To emulate satellite links, we used modified device drivers that delayed sending a packet onto the Ethernet for a deterministic amount of time. These drivers can also constrain the maximum rate at which a host can send data. We modeled GEO satellite links by a constraint of 1.3 Mb/s of TCP/IP bandwidth (i.e., approximately T1 rate at the physical layer), on a 600 ms RTT link. LEO satellites were modeled by a constraint of 1.3 Mb/s with a fixed RTT in the range of 40-400 ms [19]. Our links had no bit errors or variation in propagation delay, which, while not representative of all satellite links, exemplifies the common case.

5.2.2 Simulation Configuration

We used *ns*, described above in Section 4.2, to test simulated topologies that matched our experimental setup. We aligned the TCP modules to match our implementations, and wrote a STP simulation module to closely emulate the implementation used in the experiments. We also used a background HTTP traffic generator, similar to that used in the experiments, to lightly load the network topology and to break up any TCP phase effects [17]. Our simulation topology, which conformed closely to the experimental setup, is shown in Figure 12.

5.3 Performance for Large File Transfers

TCP is the underlying protocol supporting the file transfer protocol (FTP) in the wide-area Internet. In this section, we describe simulations and experiments used for characterizing file transfer performance over satellite links.

To maintain high throughput for large file transfers, the TCP congestion window must be large. This implies that the congestion avoidance and loss recovery mechanisms are very important in determining performance. In this section we examine the performance of four variants of TCP loss recovery and congestion control.

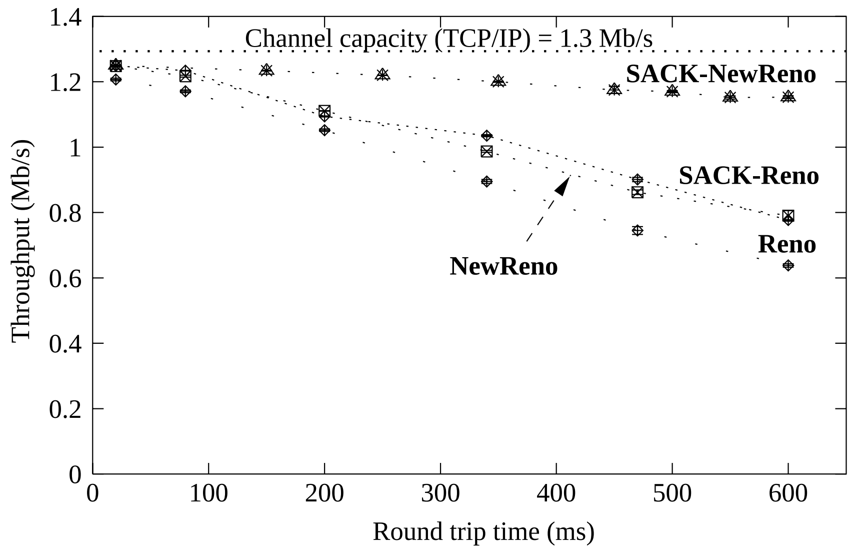


Figure 13: Throughput performance of TCP SACK NewReno, TCP SACK Reno, TCP NewReno, and TCP Reno over an experimental path with a TCP/IP bandwidth of 1.3 Mb/s and no transmission errors. Data points represent the sample means from 20 independent transfers of 10 MB each. In this and subsequent figures representing a large number of experimental results, error bars represent 95% confidence intervals.

- **TCP Reno** The unmodified TCP implementation in our BSD/OS 3.0 operating system is commonly known as TCP Reno. Many modern TCP implementations are largely based on this version of TCP. Of the satellite-friendly TCP extensions described above, BSD/OS 3.0 supports window scale and path MTU discovery.
- **TCP NewReno** TCP "NewReno" is a collection of bug fixes and refinements for how TCP Reno handles the fast recovery phase of congestion avoidance. Our TCP NewReno implementation is similar to TCP Reno except that it avoids false fast retransmissions [24], multiple window reductions in one window of data [11], and constrains the burstiness of the sender upon leaving fast recovery [11]. Specifically, it implements the "Less Careful, Slow-but-Steady" variant of NewReno described in [16].
- **TCP SACK-Reno** Reno congestion avoidance algorithms may be combined with the SACK option for loss recovery to form TCP "SACK-Reno."
- **TCP SACK-NewReno** Likewise, this corresponds to TCP NewReno congestion avoidance with the SACK option for loss recovery.

It is important to emphasize that all of the above implementations would be regarded as conformant to the TCP standards; in practice, many more variants of TCP exist.

For our file transfer experiments, we repeatedly transferred 10 MB files across our testbed while varying the latency of the emulated satellite channel. The file transfers lasted at least 60 seconds, allowing the low throughput of the initial slow start phase to be amortized across the lifetime of the connection. In the simulations, we added a number background HTTP traffic generators to the topology so as to introduce low levels of cross traffic (approximately 10% of the forward throughput of the channel). These traffic generators did not by themselves congest the forward path; the TCP losses were periodically self-induced by the greedy nature of the congestion avoidance mechanism of the persistent file transfers. In the experiments, which were conducted on operational networks during early morning periods of light network activity, the low amounts of live traffic on the networks and the variable processing delays of the hosts sufficed to add variability to the experiments.

5.3.1 Behavior of Several TCP Variants

We plot the results of these experiments in Figure 13. In all of our figures, throughput is defined as "application-level" throughput. For values of RTT less than 100 ms, the performance is relatively high for all four variants. However, for GEO delays (600 ms) and for LEO delays greater than 100 ms, the difference in performance for different TCP implementations is quite evident. By analyzing packet traces in both the simulations and the experiments, we determined that the main distinction between the implementations was in their behavior immediately upon leaving the slow start phase of congestion avoidance. It is

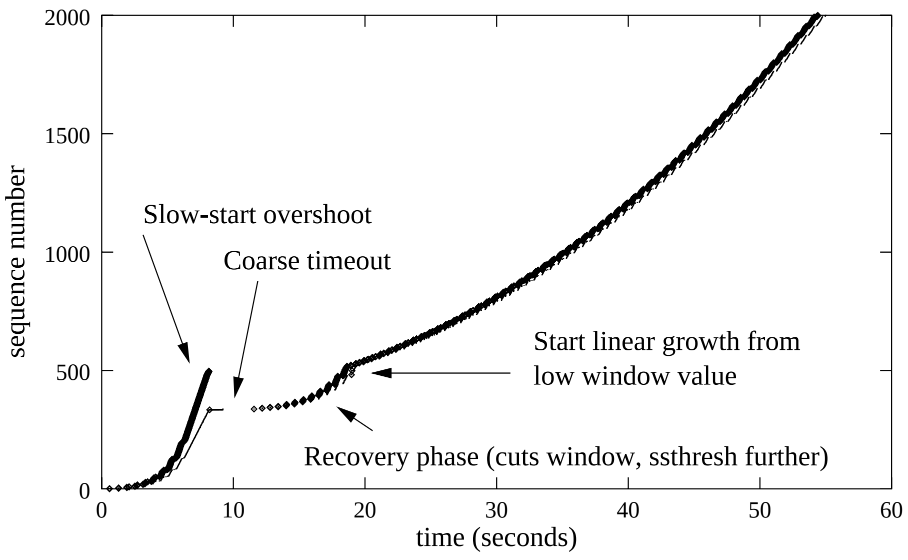


Figure 14: Typical performance, using a standard BSD TCP (Reno) implementation, of a large file transfer over a GEO satellite channel.

critical that TCP transition from slow start to congestion avoidance in a smooth manner, with a congestion window close to the bandwidth-delay product of the path. We found the performance of SACK-NewReno congestion avoidance to be the best; in this case, when a slow start overshoot occurs, the protocol cuts its window in half once and smoothly moves to congestion avoidance after recovering all losses. There is little penalty for using a high-bandwidth, high-latency GEO satellite link in this case. When SACK was used without NewReno enhancements (SACK-Reno), we observed that the slow start termination, which is characterized by several bursts of packet losses, resulted in the implementation cutting its congestion window in half several times, rather than just once. As a result, TCP was forced to rebuild its window linearly from a very low value. The performance of NewReno without SACK was similar but for a different reason. In this case, the slow start overshoot resulted in similar bursty patterns of losses, but since NewReno, unlike SACK, can only recover one loss per RTT, it spent a large portion of time recovering from the slow start losses. Finally, TCP Reno rarely avoided a retransmission timeout and multiple reductions in its window after the first slow start, resulting again in slow window growth.

A closer look at the behavior of these different TCP variants is informative. Figure 14 illustrates a “time-sequence” plot of an individual connection– the initial 60 seconds of a large file transfer using an unmodified BSD/OS UNIX TCP implementation (TCP Reno without SACK) over the topology illustrated in Figure 11. Two plots are overlaid– the evolution of the sender’s sequence number, and the evolution of the acknowledgments received (the trace of the sender’s sequence number generally lies to the left of the acknowledgment trace and is marked by larger points). The connection initially starts in slow-start, and although the connection takes several seconds to make noticeable progress, within the first 10 seconds the connection has already overshot by a wide margin the capacity of a router along the path, resulting in many packet drops (not necessarily contiguous in the sequence space). The implementation performs fast retransmission, but since many packet losses have occurred, the implementation invariably is forced to recover with a coarse timeout because it does not interpret the arrival of a partial acknowledgment (that is, an acknowledgment that does not cover all of the data that was outstanding at the time of the retransmission) as a sign that the next unacknowledged packet is missing. The timeout cuts the window to one segment and the slow start threshold in half; normally this would allow the sender to rapidly ramp up after the timeout to half of its previous window (that caused congestion). However, because the receiver’s buffer has many out-of-order packets (and holes to fill), as the post-timeout TCP sender starts to send more data, it can receive a number of duplicate acknowledgments that push it in and out of fast retransmission again (these are sometimes called “false fast retransmissions” [24]), each time cutting the window and slow start threshold in half. The result is, by the time that the sender has recovered from all of the original losses, it has a very low congestion window and slow start threshold value, and is forced to build its window linearly from a very low value, resulting in poor throughput.

TCP NewReno was devised to correct this oversight in the TCP Reno implementation; it defines a “recovery phase” that ends when all of the packets that were outstanding when the first loss was detected are acknowledged. Figure 15 illustrates the typical performance of this algorithm. No timeouts occur during the recovery and the window is not reduced multiple times for the same burst loss. However, since the recovery takes one round trip time for each gap in the sequence space to be recovered, the result is a TCP connection that takes over half a minute to recover from a single burst of losses. For this reason, as pointed out by Floyd [16], it may be beneficial to prevent this behavior from occurring by forcing TCP to take a timeout if

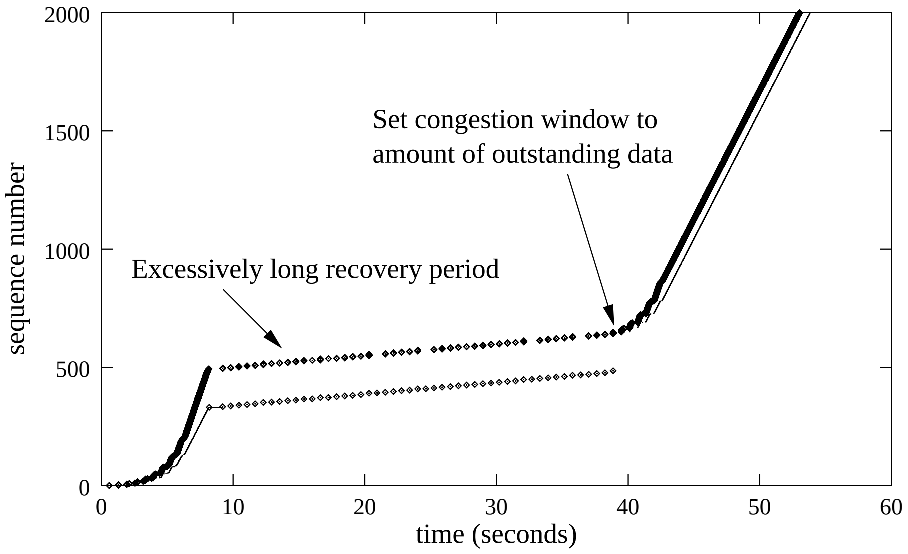


Figure 15: Correct NewReno behavior, using a modified BSD TCP implementation, of a large file transfer over a GEO satellite channel.

it requires too many round trips to recover. Second, in our experiments with this algorithm as specified by [24] and [11], we noticed undesirable behavior that occurred at the end of the recovery phase. This behavior (immediate reentry into a burst loss situation) can be seen in Figure 16, and it is due to a burst of packets that can occur at the end of recovery. This burst occurs if, during window inflation of the recovery phase, the transmission of new segments was constrained by the receiver's offered window (because the "window inflation" step of TCP Reno can result in very large artificial windows being generated). As a result, when the hole is plugged in the reassembly buffer and the TCP sender resets its congestion window upon receipt of the acknowledgment, it is eligible to immediately send many segments. The solution to this problem, as shown in Figure 15, is to constrain the congestion window at the end of recovery to be no larger than the amount of outstanding data at that time (plus one segment, to allow a new transmission). This proposal was first described in [29].

The best behavior is obtained by combining both the SACK and NewReno algorithms, as illustrated by Figure 17. In this case, TCP recovers very rapidly from bursty losses because the extra information present in the SACK option gives the TCP sender a more complete picture of what is missing. Figure 18 illustrates this recovery in more detail; the burst of losses is recovered in less than two seconds (that it requires more than a round trip delay is due to queueing delays that have built up), and a large window is preserved for the subsequent connection to use in linear growth phase.

Finally, we illustrate in Figure 19 the close correspondence between simulation and experimental results for file transfers. In the remainder of this section, we present only our experimental results since our simulation results were generally in close agreement. The TCP implementations of the *ns* simulator are very realistic, to the point that bugs found in common implementations can also be enabled in our simulations.

5.3.2 Effect of a Competing Connection

The above experiments are appropriate to model connections entirely within a satellite subnetwork, but do not accurately portray conditions found when using the satellite network to access sites on the wired Internet, where competition for bandwidth from many different connections (with shorter round trip delays) can lead to network congestion and unfairness in bandwidth allocation. For our next experiments, we added a single, large-window persistent connection from a background source to a background sink in the same direction as the foreground file transfer. In our topology, this caused the first router in the network to occasionally become congested. Note that this background connection does not traverse any portion of our emulated satellite subnet. The results in this case are strikingly different. It only takes one low delay (in this case, 20 ms RTT) connection to drastically reduce the achievable throughput for SACK-NewReno, as shown in Figure 20. This is the TCP fairness problem identified earlier in this chapter. TCP's fairness properties can be the first-order determinant of how well a large-window satellite TCP connection can do in the wide-area Internet. Even though the satellite connection was successful in avoiding timeouts in almost all of the transfers, the window reductions due to recurring fast retransmits substantially reduced the throughput. The throughput is also much more variable under these conditions, as represented by the error bars. The main problem is that the connection with the long RTT is too sluggish to rebuild its window and push data through the congested queue before it takes another loss.

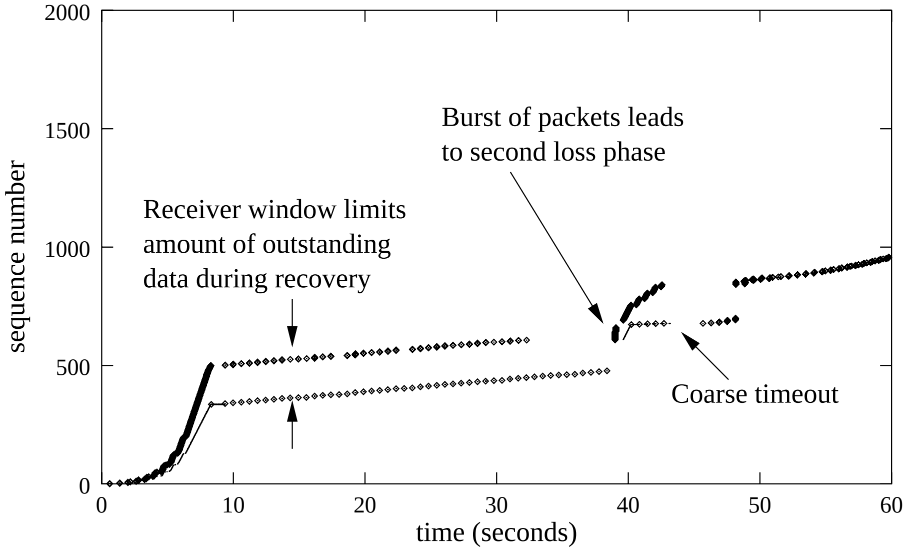


Figure 16: Incorrect NewReno behavior, using a modified BSD TCP implementation, of a large file transfer over a GEO satellite channel. This behavior is due to a burst of packets transmitted at the end of recovery.

In summary, we observed that TCP SACK with NewReno congestion avoidance is able to sustain throughputs at close to the bottleneck link rate even for GEO-like delays. This is because TCP is able to amortize the low throughtput of the initial window build across a longer period of high throughput. However, our data illustrates that the use of SACK alone is not sufficient to enable high performance. Specifically, NewReno helps to avoid coarse timeouts and multiple window reductions, while SACK accelerates the loss recovery phase. Specific details of our SACK-NewReno implementation can be found in Appendix A. Finally, the result we would like to emphasize, which agrees with our analysis in Section 4.1, is that it only takes very moderate levels of congestion in the wide-area Internet to drastically impair the performance of even well-configured TCP connections.

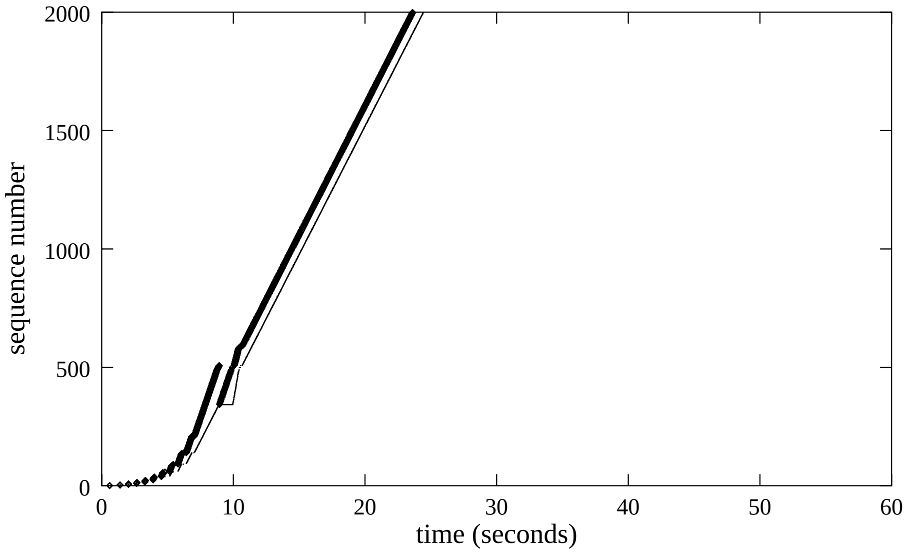


Figure 17: Correct SACK behavior, using a modified BSD TCP implementation (including NewReno loss recovery), of a large file transfer over a GEO satellite channel.

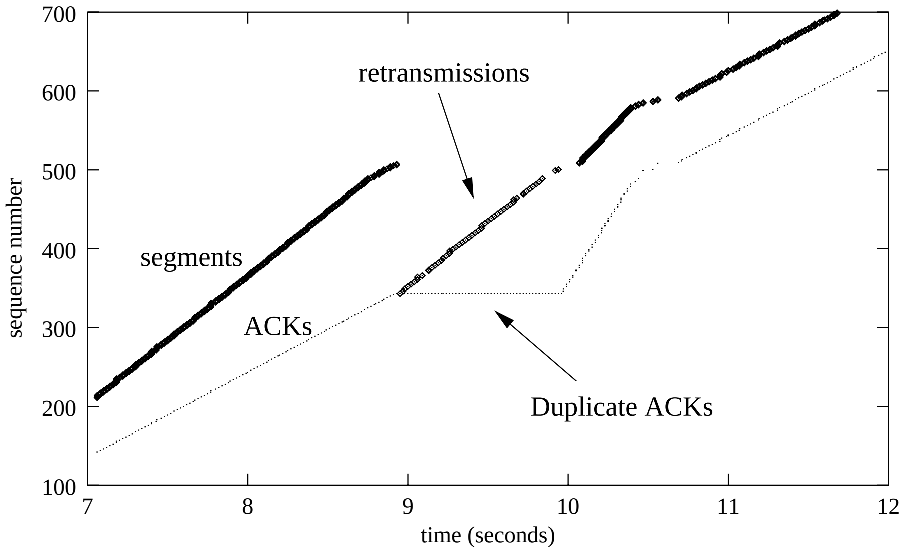


Figure 18: Closeup of rapid SACK recovery of multiple losses.

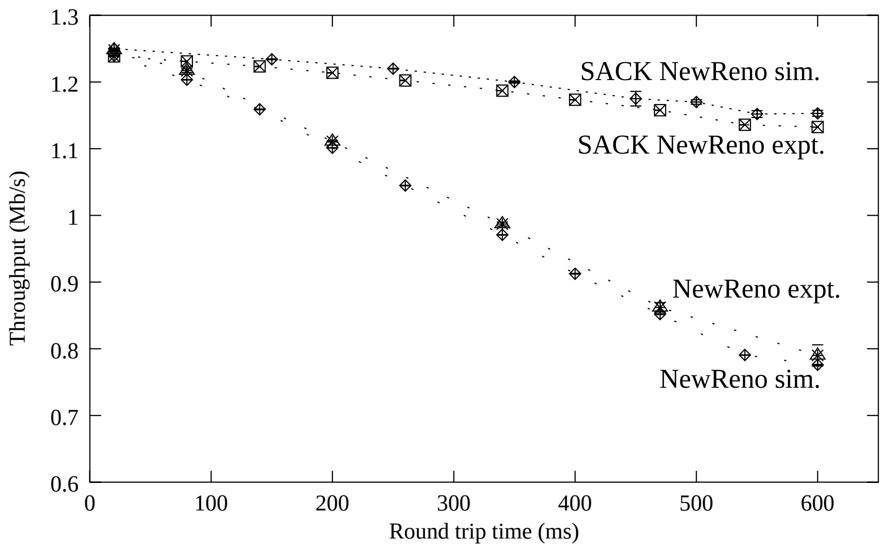


Figure 19: Agreement between simulation and experimental results for TCP SACK and TCP NewReno.

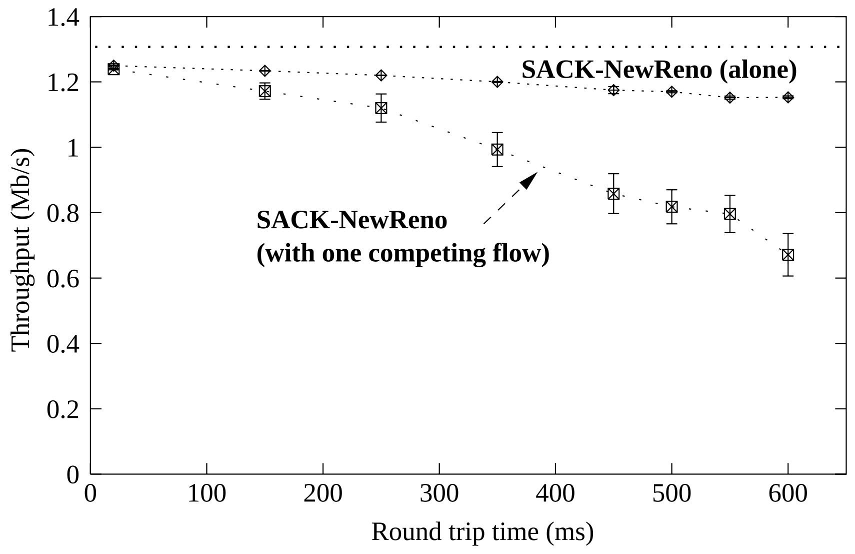


Figure 20: The effect of a single competing short-delay connection on the satellite connection's throughput. The competing connection was a persistent file transfer using TCP SACK NewReno with a nominal 20 ms RTT between a background source and sink in the experimental topology.

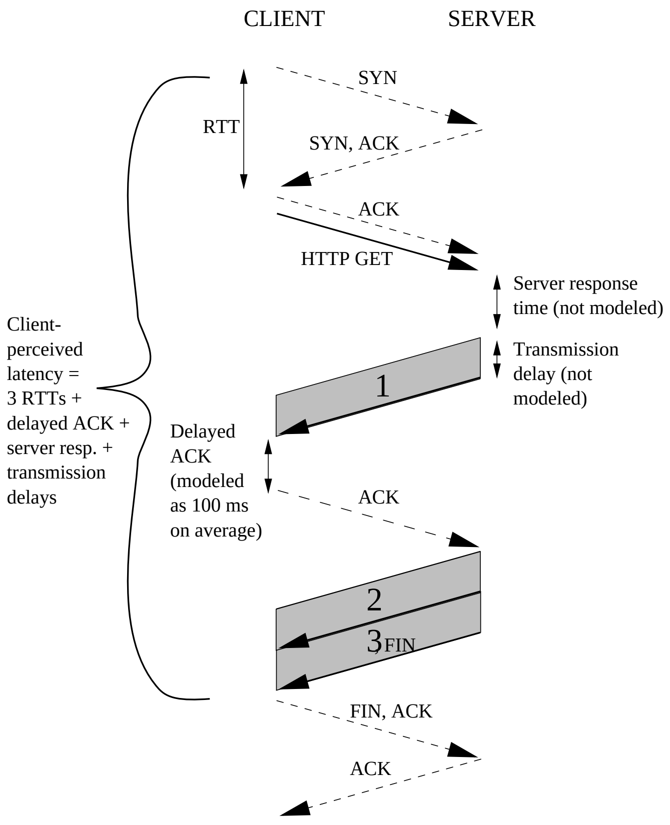

CLIENT
SERVER
SYN
RTT
SYN, ACK
ACK
HTTP GET
Server response time (not modeled)
Transmission delay (not modeled)
1
Client-perceived latency = 3 RTTs + delayed ACK + server resp. + transmission delays
Delayed ACK (modeled as 100 ms on average)
ACK
2
3FIN
FIN, ACK
ACK

Figure 21: TCP latency of a 3 segment server reply using standard TCP.

5.4 Performance for Web Transfers

Besides file transfers, most of the rest of the TCP traffic in the Internet is driven by Web transfers. Such connections are very different from file transfers. Typically, an Web client issues a small request to a server for an HTML (HyperText Markup Language) page. The server sends the initial page to the client on this first connection. Thereafter, the client launches a number of TCP connections to fetch images that fill out the requested page or to obtain different pages. Each item on the page requires a separate connection.[7] Many common Web browsers allow a user to operate multiple (typically, four) TCP connections in parallel to fetch different image objects. Basically, the data transfer model is “client request, server response.”

Using standard TCP, any connection requires a minimum of two RTTs until the client receives the requested data (the first RTT establishes the connection, and the second one is for data transfer). As the RTT increases, the RTT can become the dominant portion of the overall user-perceived latency, particularly since average Web server response times are much smaller than one second [20]. Two mechanisms described in Section 3 attempt to alleviate the latency effects of TCP for short connections. The first, T/TCP, does away with the initial handshake (RTT) of the connection. The second, 4KSS, allows the TCP server to send up to 4380 bytes in the initial burst of data. If the size of the transfer is no more than 4380 bytes, the transfer can complete in one RTT. By using some simple analysis, we can quantify the beneficial effects that these TCP mechanisms have on the user-perceived latency.

Figure 21, adapted from a similar figure in [23], illustrates the latency in a hypothetical three segment reply using standard TCP. We make the following assumptions:

- We do not model server response times or segment transmission times. We assume an environment in which the RTT is the dominant latency in the transfer.[8] Server response times and segment transmission delays are a constant offset to the latencies we calculate; i.e., the same offset must be added no matter what version of TCP we are considering.
- We assume no packet losses and a fixed RTT. Therefore, these latencies are the best case.

[7]We will discuss shortly a modification to this approach, known as Persistent-HTTP (P-HTTP), which reuses the same TCP connection for multiple items.
[8]This is not always true in practice. Even for fast links, server responses can take several seconds, but on average, the server response time is much less than a second [20].

	Geostationary orbit (600 ms RTT)							
	1500 byte segments				**500 byte segments**			
	minimum	median	mean	expt. mean	minimum	median	mean	expt. mean
Standard TCP	1.2	1.9	1.9	2.0 (0.1)	1.2	2.5	2.5	2.6 (0.1)
T/TCP	0.6	1.2	1.2	1.4 (0.1)	0.6	1.8	1.7	2.0 (0.1)
TCP “4KSS”	1.2	1.2	1.4	1.6 (0.1)	1.2	1.8	1.7	1.9 (0.1)
T/TCP “4KSS”	0.6	0.6	0.8	1.0 (0.1)	0.6	1.2	1.1	1.3 (0.1)
	Low-earth orbit (80 ms RTT)							
Standard TCP	0.16	0.34	0.31	0.37 (0.02)	0.16	0.42	0.42	0.55 (0.02)
T/TCP	0.08	0.16	0.17	0.28 (0.02)	0.08	0.24	0.25	0.47 (0.02)
TCP “4KSS”	0.16	0.16	0.18	0.25 (0.01)	0.16	0.24	0.23	0.31 (0.01)
T/TCP “4KSS”	0.08	0.08	0.10	0.16 (0.01)	0.08	0.16	0.15	0.23 (0.01)

Table 2: TCP latency effects on HTTP transfers for GEO and LEO satellite connections. Trace data is taken from [27]. All latencies are in seconds. For the experimental results, 95% confidence intervals are shown in parentheses.

- We do not model some of the bugs that have appeared in early HTTP implementations and that are discussed in [23], under the assumption that they will gradually disappear. For example, one quite prevalent bug allows the connection to start with an initial congestion window of two segments [42].

With these assumptions in mind, consider Figure 21, in which dashed lines denote control packets and solid lines indicate data packets. The first RTT is consumed by a SYN exchange, after which the client issues an HTTP GET request. Upon receiving and responding to this request, the server at this point has a congestion window of one segment. Assuming that the TCP implementation implements delayed acknowledgments (delayed ACKs) of up to 200 ms [41], the client on average will acknowledge this data after 100 ms. Upon receiving the acknowledgment, the congestion window grows to 2, and the server sends the second and third segments, followed by a FIN, which closes its half of the connection. The client must close its own half of the connection, but we do not model this delay since it does not contribute to user-perceived latency. Therefore, the total amount of TCP-related latency is 3 RTTs + 100 ms in this case. Using either T/TCP or 4KSS would reduce the latency to 2 RTTs, and using both mechanisms would reduce it to a single RTT.

We used HTTP traces to compute probability mass functions (pmfs) for the number of bytes transferred per HTTP connection. We then computed the average TCP latency for all of these file sizes, based on a simple analysis of how the congestion window builds over time. Because some transfers were very long, we eliminated those over 100 segments (only 2-4% of the data set, in general). For these cases, it is more realistic to consider them as large file transfers. Our trace data was gathered from two different user populations. The first, collected by Mah in 1995 [30], comes from a well connected Berkeley subnet. The second set, collected by Gribble in 1997 [20], comes from Berkeley residential usage over dial-up modems. By using this trace data with our model, we estimated the minimum, median, and mean latency effects of TCP on user-perceived latency. For GEO networks, we modeled the RTT as a fixed 600 ms, and for LEO networks we assumed a RTT of 80 ms. To verify the analytical results, we also performed measurements using similar pmfs to drive a TCP traffic generator in our experimental topology, and we recorded the latency experienced along with the file size for each file transfer. For the experiments, we did not cull the large transfers from our trace data. The experimental setup captured the effects of not only the propagation delay but also the processing delays in real end systems.

In Table 2, we present the results from an analysis of the data set provided by Mah [30]. The first three columns of data list the minimum, median, and mean TCP transfer times required, according to the analysis of the trace file and assuming a maximum segment size of 1500 bytes. These values were calculated by first determining the TCP related latency for a connection of a given size, and then by weighting these latencies according to the pmfs derived from the trace data. The fourth column lists experimental results corresponding to this data set. These values are the mean (and 95% confidence interval) of 1000 independent transfers, in which the size of the transfer was generated randomly according to the pmfs drawn from the trace data. The last four columns are similar to the first four, except for the use of a maximum segment size of 500 bytes. This data indicates that the use of either T/TCP or TCP with 4KSS improves mean latency by a small amount, but the combination of both options yields an improvement by a factor of two to three. The relative improvement is similar whether GEO or LEO networks are assumed (because the analysis is based on RTT). Because the mean latencies using the assumed LEO network are already rather small, the improvements due to TCP optimizations are less likely to be perceived by users. The data set provided by Gribble [20] contained slightly larger transfers, on average, but the same trends in TCP latency were present.

Finally, the most recent version of the HTTP specification (version 1.1 [12]) recommends that servers and clients adopt the persistent connection and pipelining techniques known as “persistent-HTTP” (P-HTTP) [36]. Rather than using separate TCP connections for each image on a page, P-HTTP allows for a single TCP connection between client and server to be reused for multiple objects. The shift to P-HTTP offers a tradeoff in performance for satellite connections. On the one hand, P-HTTP

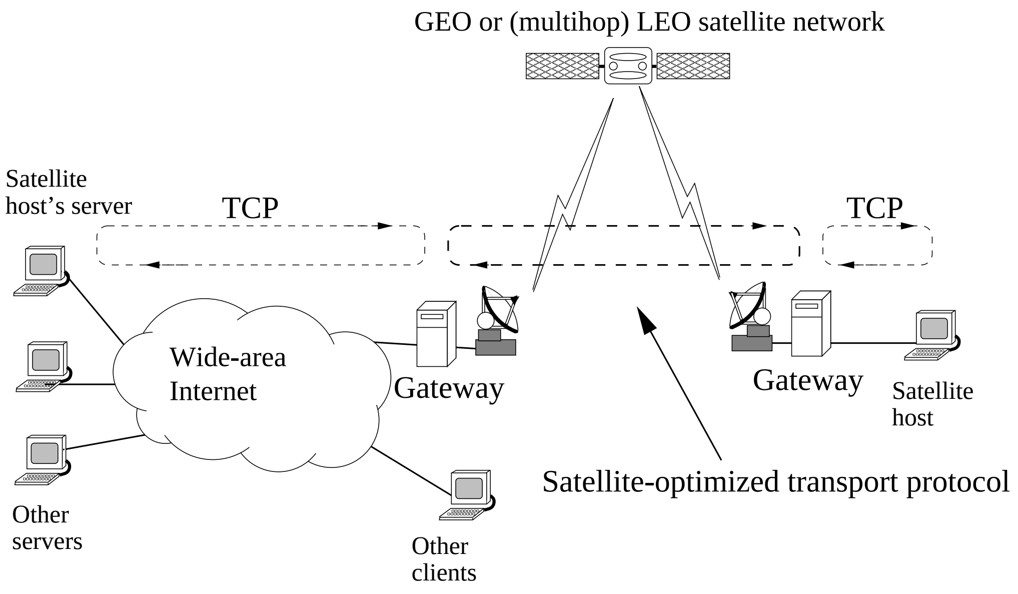


Figure 22: Future satellite networking topology in which a satellite-based host communicates with a server in the Internet through a satellite protocol gateway.

is potentially much more bit-efficient than HTTP with standard TCP, because connections are not set up and torn down as frequently (the connection establishment costs are identical to those of T/TCP [23]). However, in terms of latency, the use of T/TCP and multiple, concurrent connections may yield faster Web page loads under some scenarios. The capability of many Web browsers to support multiple, concurrent connections is an example of a general technique known as "striping," which has been a strategy for transport protocol improvement known to satellite network operators for some time, and which has most recently been studied in the context of FTP [3]. Because TCP and HTTP optimizations such as T/TCP, and TCP with 4KSS do not yield major performance improvements for most users of the Internet [23], it is unclear whether they will see deployment. In fact, Padmanabhan recently studied the potential benefit of not using P-HTTP but instead reverting back to multiple, concurrent TCP connections that share congestion window and other state information [34].

In summary, for connections using GEO satellite links, TCP optimizations such as T/TCP and 4KSS, especially when used together, can yield a reduction of two to three times in in user-perceived latency and can also reduce the bandwidth overhead of HTTP connections. For LEO satellite links, optimizations to reduce the number of unnecessary control packets are desirable, but optimizations to reduce latency will not have as perceptible of an effect for users because propagation delays are smaller. However, since such optimizations benefit only a small user community, it is possible that they will not see widespread deployment.

6 Split TCP Connections

Although TCP can work well over even GEO satellite links under certain conditions, we have illustrated that there are cases for which even the best end-to-end modifications cannot ensure good performance. Furthermore, in an actual network with a heterogeneous user population, users and servers cannot all be expected to be running satellite-optimized versions of TCP. This has led to the practice of "splitting" transport connections. This concept is not new; satellite operators have deployed protocol converters for many years. In this section, we describe how TCP connections may be split at a satellite gateway, identify some drawbacks to split connections, and quantify how much improvement can be obtained.

6.1 Split Connection Approaches

The idea behind split connections is to shield high-latency or lossy network segments from the rest of the network, in a manner transparent to applications. TCP connections may be split in a number of ways. Figure 22 illustrates the most general case, in which a *gateway* is inserted on the link between the satellite terminal equipment and the terrestrial network. On the user side, the gateway may be integrated with the user terminal, or there may be no gateway at all. The goal is for end users to be unaware of the presence of an intermediate agent, other than improved performance. From the perspective of the host in

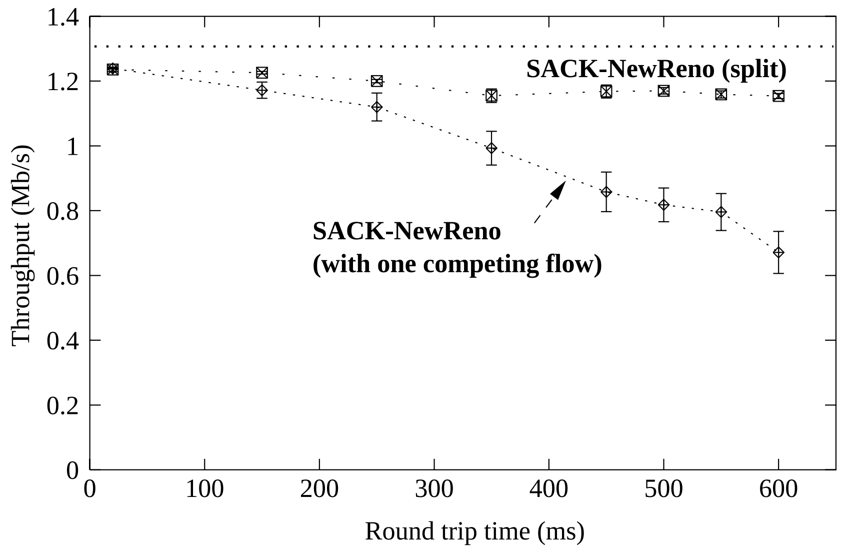


Figure 23: Forward throughput performance of split TCP in the presence of a short-delay competing connection. TCP SACK NewReno with large windows was used on both connection portions.

the wide-area Internet, it is communicating with a well-connected host with a much shorter latency. Over the satellite link, a satellite-optimized transport protocol can be used.

TCP may be split in the following ways:

- **TCP spoofing** In this approach, the gateway on the network side of the connection prematurely acknowledges data destined for the satellite host, to speed up the sender's data transmission [48]. It then suppresses the true acknowledgment stream from the satellite host, and takes responsibility for resending any missing data. As long as the traffic is primarily unidirectional, TCP datagrams are passed through the gateway without alteration. In the reverse direction, the same strategy is followed. No changes are needed at the satellite client.

- **TCP splitting** Instead of spoofing, the connection may be fully split at the gateway on the network side, and a second TCP connection may be used from the satellite gateway to the satellite host. Logically, there is not much difference between this approach and spoofing, except that the gateway may try to run TCP options that are not supported by the terrestrial server. Modern firewall implementations often perform a type of TCP splitting (such as sequence number remapping) for security reasons.

- **Web caching** If satellite-based Web users connect to a Web cache within the satellite network, the cache is effectively splitting any TCP connection for requests that result in a cache miss. Therefore, Web caching not only can reduce the latency for users in fetching data from the Web, it has the side benefit of splitting the transport connection for cache misses.

Furthermore, when the TCP connection is fully split at a gateway or cache, it is possible to use an alternative protocol for the satellite portion of the connection. While this requires the use of a satellite gateway or modified end-system software on the satellite host's side, this approach may provide better performance by improving on TCP's performance in ways not easily achieved by remaining backward compatible with existing implementations. Set-top boxes or other user terminal equipment may provide a natural point for the implementation of protocol conversion (back to TCP, if necessary) on the satellite host's side of the connection.

In all three approaches, the amount of per-connection buffering required at the gateway is roughly 2-3 times the bandwidth-delay product of the satellite link or the Internet path, whichever is smaller. The computing resources required to support a large set of users (approximately 200-500 KB of memory per active connection, plus processing) can form a significant portion of the hardware requirements of a satellite Internet gateway. In addition, although persistent-HTTP connections will reduce the number of connections that need to be set up and torn down, they will also drastically lower the duty cycle of each TCP connection, requiring the gateway to keep resources allocated for idle connections. However, it is important to emphasize that if Web caches or other proxies are already part of the satellite network architecture, there would be no need for extra equipment to support transport-level gateways.

Besides the resource consumption noted above, split connections are not without other hazards. First, from an architectural standpoint, a split TCP connection that is not explicitly associated with a proxy or a cache breaks the end-to-end semantics

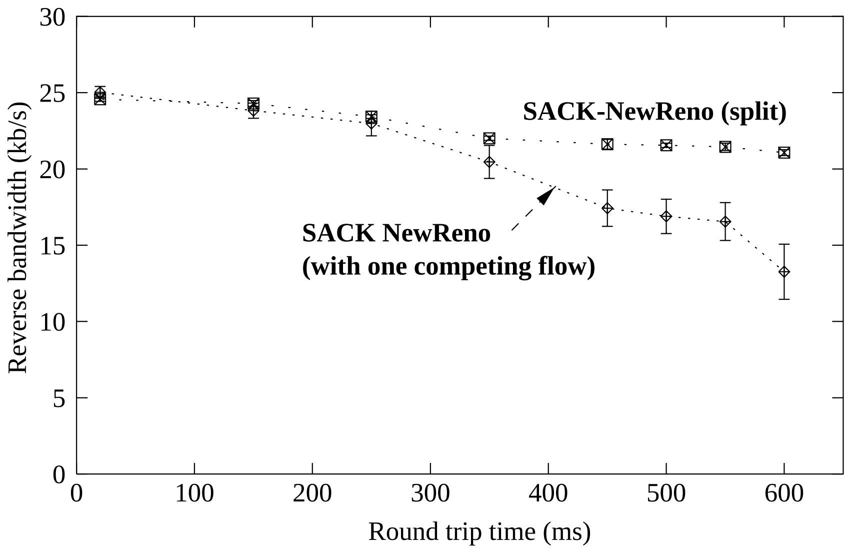


Figure 24: Reverse channel utilization of split TCP in the presence of a short-delay competing connection. TCP SACK NewReno with large windows was used on both connection portions.

of the transport layer. Although approaches for TCP improvement over local area wireless links, such as Berkeley's "snoop" protocol [4] and "mobile TCP" [9], can preserve end-to-end semantics, it is more difficult to do so in the satellite environment because combating the fairness problem and speeding up slow-start both rely on the early acknowledgment of data. However, steps can be taken to ensure that the connection does not close normally unless all data has been received; for example, the gateways can allow the FIN segment of TCP to pass end-to-end. Furthermore, higher layer protocols typically have mechanisms to restart a transport connection if it prematurely fails. Second, gateways introduce a single point of failure within the network, and require all traffic for a given connection to be routed through them (i.e., there can be no alternate packet routing). Third, protocol conversion gateways are ineffective if IP-level encryption and authentication protocols are operating on a link, although they can still function normally if the encryption and authentication is performed at the transport layer. In the case of IP-level security, the transport gateway must be included as part of the "trust infrastructure" to operate. Typically, however, if a satellite network is used to provide "last-mile" access to a large diverse set of users, transport-level security protocols will be used instead of IP-level security; in this case, protocol gateways can operate correctly.

6.2 Split Connection Performance

In Figure 23, we illustrate the performance gains achievable when the TCP connection is split at the gateway between the satellite network and the Internet, under the same conditions as shown in Figure 20 (a competing short delay connection in the Internet). We replotted the relevant data from Figure 20 for comparison. Note that the presence of the gateway allows the split connection to compete for bandwidth in the wide area and obtain its fair share. However, as shown in Figure 24, the reverse channel usage required for this TCP connection is roughly 20 Kb/s. This usage scales linearly with the forward throughput, and for 1000 byte segments, is roughly 2% of the forward throughput achieved. For bandwidth-constrained reverse channels as will be the case in most satellite systems, this sets an upper bound on the forward throughput achievable if TCP relies on a stream of acknowledgments to clock out new data. This suggests that it would be useful either to make modifications to TCP to reduce its reverse channel usage (such as using modifications to handle TCP asymmetry [5]) or to use a protocol over the satellite portion of the connection that uses less bandwidth.

7 Summary

In this report, we have investigated the performance of IP-compatible transport protocols over satellite links from several perspectives. Our main results are as follows:

i) We observed little degradation in TCP performance for connections with RTTs in the range of future LEO systems (40-200 ms), although we did not investigate potential problems due to large RTT variations. However, maintaining good TCP performance over GEO paths (or long LEO paths) is challenging.

ii) While we found that the Constant-Rate (CR) policy could improve TCP fairness dramatically, we faced two practical difficulties that would likely prevent universal deployment of this scheme in its current form. First, the proper selection of a constant is dependent upon the network topology and the number of peer connections and is therefore difficult to determine in a distributed manner. Second, the fairness benefits of the CR policy can be confounded by competing connections using standard congestion avoidance, thereby making it disadvantageous to deploy CR in an existing heterogeneous environment.

iii) When we instead made only certain long RTT connections slightly more aggressive, we were always able to improve network fairness while keeping bottleneck link utilization relatively constant by using an increase-by-K (IBK) policy. Interestingly, the effects on other unmodified connections that were sharing the bottleneck link were similar to what they would have experienced had the modified connection actually been a connection with a shorter RTT.

iv) If the right TCP options are used and congestion is light, TCP can work well for large file transfers even over GEO links. In particular, in our large file transfer experiments with TCP SACK plus NewReno congestion control, average throughput decreased by no more than 10% when the RTT was increased from 20 ms to 600 ms. However, we showed that even low levels of competition from short delay flows (in the form of cross-traffic in the wide-area Internet) significantly degrades the satellite connection's performance.

v) Concerning the latency due to HTTP exchanges, we found that the use of both T/TCP and modified slow start performed much better than either option used separately, and could cut the average TCP-related latency by a factor of two to three for GEO links.

vi) We showed that the performance problems due to mis-configured TCP or network congestion can be alleviated by splitting the TCP connection at a gateway within the satellite subnetwork. Even with congestion in the wide-area Internet, the end-to-end connection is still able to maintain high throughput.

TCP has proven to be a very robust protocol in a variety of network environments. However, this report has illustrated that obtaining good performance using standard end-to-end TCP connections is very challenging in a GEO satellite environment. For file transfers, the best performance results that we obtained were based on splitting the connection at a gateway, where the long round trip delay of the satellite portion of the path can be isolated from the portion of the connection that traverses the Internet. For short transactions, we found that the best performance requires TCP enhancements (T/TCP, 4KSS) that are not implemented in the current Internet– again leading us to consider split connection solutions.

As for future work, our results suggest the following interesting directions:

i) Our results above indicated that it may be worthwhile to selectively increase the aggressiveness of certain TCP connections to improve network fairness. However, our results are preliminary and need to be experimentally validated before deployment can be recommended. For example, a wider range of topologies should be considered, and experiments as well as more simulations are needed to decide on the best policy. The impact of this algorithm on connections used for short data transactions (such as many small Web transfers) should be studied more. Finally, mechanisms for more accurately determining a connection's RTT, as well as policies that might be invoked as a function of the RTT observed, require more study. It should be emphasized that mechanisms for per-flow fair sharing of congested links would obviate the need for improvements to TCP's end-to-end algorithm, so further work on the design and deployment of these mechanisms would also be very useful. Also, Mo has demonstrated the potential existence of a fair, distributed flow control algorithm but has not been able to construct such an algorithm [32]; if such an algorithm were discovered, it may be a substantial improvement over the current one.

ii) We have identified the performance of the slow start and congestion avoidance algorithms, as well as implementation details such as correct sizing of socket buffers and use of the correct TCP options, as the biggest hurdles to overcome to improve TCP performance over satellite links. Further work on automating the correct configuration of TCP, such as described in [40], could help the deployment of more satellite-friendly implementations.

iii) The main obstacle to the deployment of split-connection protocol gateways is their interaction with a security infrastructure. In particular, any IP security protocols that encrypt the payload of an IP packet render a split connection gateway useless. Work on how to integrate performance enhancing proxies into the trust infrastructure of a secure network would be valuable. Another issue that could be pursued further is how data flows between split connections should interact. Specifically, consider the case of using a split connection to aid in Web browsing. In this case, packets may trickle in at a slow rate from the terrestrial (server) side of a connection. If the split connection is implemented as a pure byte pipe, then these packets would be sent over the satellite as they arrive at the gateway. However, these types of short packet exchanges are the least efficient because they require frequent use of the backchannel for acknowledgments. It would be much more efficient to bundle and send the whole Web page at once rather than to stream it gradually across a satellite connection. In other words, knowing application-level data boundaries may aid in efficient communication (this is the concept of Application Level Framing). Design of split connnection gateways around this approach may outperform simple connection-splicing approaches as described herein.

Acknowledgments

We thank the following individuals for their contributions to this research. We had several helpful discussions with Sally Floyd regarding TCP fairness, and Sally shared with us some of her unpublished work in this area. Venkat Padmanabhan implemented modified device drivers used in our network experiments. Giao Nguyen provided assistance in developing the ns simulation models, and Bruce Mah and Steve Gribble provided HTTP traffic trace data. Finally, Hari Balakrishnan and Venkat

Padmanababhan provided helpful comments on earlier versions of this work. This work was supported by DARPA contract DAAB07-95-C-D154, by the State of California under the MICRO program, and by Hughes Aircraft Corporation, Metricom, Fuji Xerox, Daimler-Benz, and IBM.

Appendix

Congestion Avoidance and Selective Retransmission Policies for TCP

Our TCP SACK-NewReno implementation obeys standard congestion avoidance policies and rules for selective acknowledgments (SACKs) as specified in [43] and [31], with the following extensions.[9] The following extensions apply whether or not SACK is enabled for a given connection:[10]

1. Initialize a new state variable, *snd_recover*, to the value of *snd_una* upon connection start.
2. Upon receiving three duplicate acknowledgments, if the sequence number acknowledged is greater than or equal to *snd_recover*, then set *snd_recover* equal to *snd_max*, and perform fast retransmit according to [43].
3. If, while in fast recovery phase, a segment acknowledging new data is received and the sequence number acknowledged is greater than or equal to *snd_recover*, then exit fast recovery by setting *snd_cwnd* to either *snd_ssthresh* or the amount of outstanding data in the network plus one segment, whichever is smaller.
4. While in fast recovery phase, if a segment acknowledging new data is received, and the sequence number acknowledged is less than *snd_recover*, if SACK is not enabled for the connection then retransmit the next unacknowledged segment. Additionally, whether or not SACK is enabled, partially deflate the (inflated) *snd_cwnd* by the amount of new data acknowledged, add back one segment to *snd_cwnd*, and call *tcp_output()*.

In addition, if SACK is enabled for a given connection, the following rules apply to retransmissions and new data transmissions during the recovery phase:

5. A given segment is considered "eligible" for retransmission if it has not already been retransmitted and if either three duplicate acknowledgments have arrived for the segment just prior to it or the SACK information implies that the receiver is holding a segment that was sent at least three segments beyond the given segment.
6. While in fast recovery, upon reception of each ACK that does not end the fast recovery phase, the TCP sender first checks whether there are any eligible retransmissions to be sent. If so, one such retransmission is sent. If not, the TCP sender inflates *snd_cwnd* by one segment and attemps to send one or more new segments if permitted by the window.
7. When *snd_max* is greater than *snd_nxt* (e.g., following a TCP timeout), any SACK information received subsequent to the timeout is used to avoid retransmitting data for which the receiver is sending a SACK.

[9]This description assumes a TCP implementation similar in structure to Berkeley-derived TCP implementations.

[10]These first five guidelines are for the TCP NewReno portion of the implementation and have been accepted as (experimental) RFC 2582 within the IETF [16].

References

[1] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. *Internet RFC 2414*, 1998.

[2] M. Allman and D. Glover. Enhancing TCP Over Satellite Channels using Standard Mechanisms. *IETF draft (work in progress): draft-ietf-tcpsat-stand-mech-02.txt*, January 1998.

[3] M. Allman, H. Kruse, and S. Ostermann. An Application-Level Solution to TCP's Satellite Inefficiencies. *Proceedings of 1st Workshop on Satellite-Based Information Systems (WOSBIS '96)*, 1996.

[4] H. Balakrishnan, V. Padmanabhan, E. Amir, and R. Katz. Improving TCP/IP Performance over Wireless Networks. *Proceedings of First ACM/IEEE MobiCom Conference*, November 1995.

[5] H. Balakrishnan, V. Padmanabhan, and R. Katz. The Effects of Asymmetry on TCP Performance. *Proceedings of Third ACM/IEEE MobiCom Conference*, pages 77–89, September 1997.

[6] R. Braden. T/TCP– TCP Extensions for Transactions, Functional Specification. *Internet RFC 1644*, 1994.

[7] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New Techniques for Congestion Avoidance. *Proceedings of ACM SIGCOMM '94*, pages 24–35, October 1994.

[8] L. Brakmo and L. Peterson. Performance problems in BSD4.4. TCP. *ACM Computer Communications Review*, 25(5):69–86, October 1995.

[9] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM Computer Communications Review*, 27(5):19–43, October 1997.

[10] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[11] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[12] R. Fielding, J. Gettys, J. Mogul, H. Prystyk, and T. Berners-Lee. Hypertext Transfer Protocol– HTTP/1.1. *Internet RFC 2068*, 1997.

[13] E. Fitzpatrick. SPACEWAY system summary. *Space Communications*, 13(1):7–23, 1995.

[14] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-way Traffic. *ACM Computer Communications Review*, 21(5):30–47, October 1991.

[15] S. Floyd. A proposed modification to TCP's Window Increase Algorithm. *Unpublished draft, cited for acknowledgement purposes only*, August 1994.

[16] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. *Internet RFC 2582*, 1999.

[17] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[18] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 3(3):115–156, September 1993.

[19] B. Gavish and J. Kalvenes. The impact of satellite altitude on the performance of LEOS based communication systems. *Wireless Networks*, 4(2):199–212, 1998.

[20] S. Gribble and E. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, December 1997.

[21] E. Hashem. Analysis of Random Drop for Gateway Congestion Control. *Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA*, 1989.

[22] J. Heidemann. Performance Interactions Between P-HTTP and TCP Implementations. *ACM Computer Communications Review*, 27(2):65–73, April 1997.

[23] J. Heidemann, K. Obraczka, and J. Touch. Modeling the Performance of HTTP Over Several Transport Protocols. *ACM/IEEE Transactions on Networking*, 5(5):616–630, October 1997.

[24] J. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. *Proceedings of ACM SIGCOMM '96 Conference*, pages 270–280, 1996.

[25] V. Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM '88 Conference*, pages 314–329, 1988.

[26] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. *Internet RFC 1323*, 1992.

[27] T. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997.

[28] T. Lakshman, U. Madhow, and B. Suter. Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance. *Proceedings of INFOCOM '97*, pages 1199–1209, 1997.

[29] D. Lin and R. Morris. Dynamics of Random Early Detection. *Proceedings of ACM Sigcomm '97*, pages 127–37, 1997.

[30] B. Mah. An Empirical Model of HTTP Network Traffic. *Proceedings of INFOCOM '97*, 1997.

[31] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. *Internet RFC 2018*, 1996.

[32] J. Mo and J. Walrand. Fair End-to-End Window-based Congestion Control. *Proceedings of SPIE '98 International Symposium on Voice, Video, and Data Communications*, 1998.

[33] J. Mogul and S. Deering. Path MTU discovery. *Internet RFC 1191*, 1990.

[34] V. Padmanabhan. Addressing the Challenges of Web Data Transport. *Ph.D. Thesis, University of California, Berkeley*, 1998.

[35] V. Padmanabhan and R. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. *Proceedings of IEEE Globecom '98 Internet Mini-Conference*, 1998.

[36] V. Padmanabhan and J. Mogul. Improving HTTP Latency. *Proceedings of the Second International World Wide Web Workshop*, October 1994.

[37] C. Partridge and T. Shepard. TCP Performance over Satellite Links. *IEEE Network*, 11(5):44–49, September 1997.

[38] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. *Proceedings of ACM SIGCOMM '97 Conference*, pages 167–180, 1997.

[39] J. Postel. Transmission Control Protocol. *Internet RFC 793*, 1981.

[40] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. *Proceedings of ACM SIGCOMM '98*, pages 315–23, October 1998.

[41] W. Stevens. *TCP/IP Illustrated, Volume 1*. Addison Wesley, 1994.

[42] W. Stevens. *TCP/IP Illustrated, Volume 3*. Addison Wesley, 1996.

[43] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. *Internet RFC 2001*, 1997.

[44] M. Sturza. The Teledesic Satellite System. *Proceedings of 1994 IEEE National Telesystems Conference*, pages 123–126, 1994.

[45] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Design Considerations for Supporting TCP with Per-flow Queueing. *Proceedings of INFOCOM '98*, pages 299–306, 1998.

[46] J. Touch. TCP Control Block Interdependence. *Internet RFC 2140*, 1997.

[47] G. Wright and W. Stevens. *TCP/IP Illustrated, Volume 2*. Addison Wesley, 1995.

[48] Y. Zhang, D. DeLucia, B. Ryu, and S. Dao. Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential. *Proceedings of INET '97*, June 1997.

[49] Y. Zhang, E. Yan, and S. Dao. A Measurement of TCP over Long-Delay Network. *Proceedings of 6th Int'l Conference on Telecommunication Systems, Modelling, and Analysis*, March 1998.