# Transport Protocols for Internet-Compatible Satellite Networks

Thomas R. Henderson, *Student Member, IEEE*, and Randy H. Katz, *Fellow, IEEE*

*Abstract*— **We address the question of how well end-to-end transport connections perform in a satellite environment composed of one or more satellites in geostationary orbit (GEO) or low-altitude earth orbit (LEO), in which the connection may traverse a portion of the wired Internet. We first summarize the various ways in which latency and asymmetry can impair the performance of the Internet's Transmission Control Protocol (TCP), and discuss extensions to standard TCP that alleviate some of these performance problems. Through analysis, simulation, and experiments, we quantify the performance of state-of-the-art TCP implementations in a satellite environment. A key part of the experimental method is the use of traffic models empirically derived from Internet traffic traces. We identify those TCP implementations that can be expected to perform reasonably well, and those that can suffer serious performance degradation. An important result is that, even with the best satellite-optimized TCP implementations, moderate levels of congestion in the wide-area Internet can seriously degrade performance for satellite connections. For scenarios in which TCP performance is poor, we investigate the potential improvement of using a satellite gateway, proxy, or Web cache to "split" transport connections in a manner transparent to end users. Finally, we describe a new transport protocol for use internally within a satellite network or as part of a split connection. This protocol, which we call the "Satellite Transport Protocol (STP)," is optimized for challenging network impairments such as high latency, asymmetry, and high error rates. Among its chief benefits are up to an order of magnitude reduction in the bandwidth used in the reverse path, as compared to standard TCP, when conducting large file transfers. This is a particularly important attribute for the kind of asymmetric connectivity likely to dominate satellite-based Internet access.**

*Keywords*—**Internet, Transport protocols, satellite communication, TCP.**

## I. Introduction

SEVERAL companies (e.g., Alcatel, Hughes, Teledesic) have recently 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 [1]). 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 paper, 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 discuss extensions to standard TCP that alleviate some of these performance problems. Through analysis, simulation, and experiments described in Sections 4 and 5, 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. We identify scenarios where TCP can be 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. For the cases in which performance is poor, we next investigate in Section 6 the improvements that can be gained by using a transport gateway to "split" the end-to-end connection in a manner transparent to the end user. Finally, in Section 7 we describe a new transport protocol for use within a satellite subnetwork or on the satellite side of a split connection. This protocol, which we call the "Satellite Transport Protocol (STP)," is optimized for challenging network impairments experienced by satellite networks such as high latency, bandwidth and path asymmetry, and high error rates.

The following are our main contributions:

• 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 clean satellite 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.

• 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 [2], used for Web browsing), and evaluate the relative merits of proposed TCP options that reduce the latency of short HTTP connections.

• We describe the design of STP, an adaptation of a reliable ATM link layer protocol known as SSCOP, to provide transport service in a connectionless network environment. Besides being efficient and resilient to loss in the forward direction of data transfer, the chief advantage of this protocol relative to satellite-optimized TCP is a substantial reduction in the bandwidth needed in the reverse channel.

Manuscript received February 15, 1998; revised September 10, 1998. The authors are with the Electrical Engineering and Computer Science Department at the University of California, Berkeley. T. Henderson was also supported by HRL Laboratories, Malibu, CA, for a portion of this work.

E-mail: {tomh, randy}@cs.berkeley.edu.

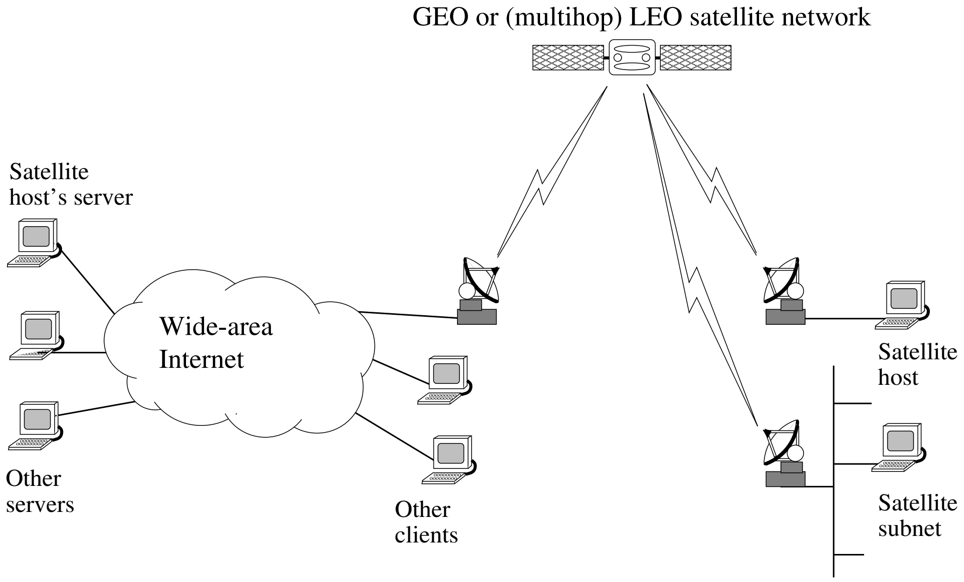



Fig. 1. Example of a future satellite network in which a satellite-based host communicates with a server in the Internet

## II. Transport environment of future satellite systems

Our assumptions about future satellite network characteristics are shaped by projections of future commercial systems (e.g., Teledesic [3], Spaceway [4]). 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 [5].
- **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 [6]. 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, 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.

## III. 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 [7].

### A. *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 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" [8]. 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 [7], 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 [9]:** 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) [10]:** 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) [11]:** 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 [12]:** 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 [13]; Partridge and Shepard also discuss some of these transport improvements [14].

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.

### B. *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

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

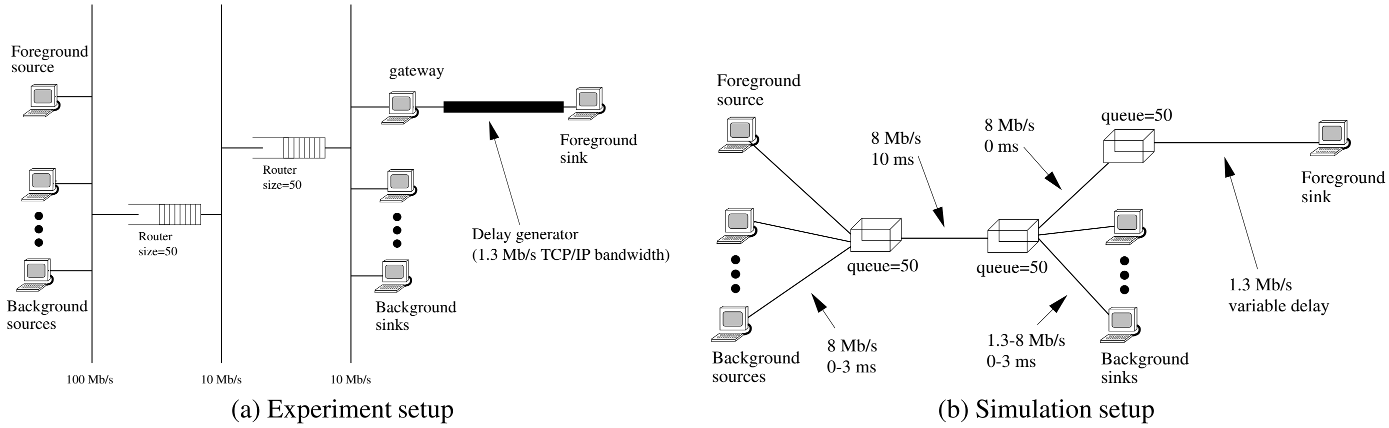



Fig. 2. Experiment and simulation setup.

when slow start terminates prematurely, forcing TCP into the linear window growth phase of congestion avoidance early in the connection [14]. 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 [15]. 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 [16],[17].

• **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 [6]. 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 [6]. 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 [18]. 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 [18]. 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, [19]). 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 $RTT^{\alpha}$, where $\alpha < 2$ [20]. This problem has been observed by several researchers (e.g., [20],[21]), but a viable solution has not yet been proposed, short of modifying network routers to isolate and protect competing flows from one another [22]. Furthermore, bandwidth asymmetry exacerbates the fairness problems by shutting out certain connections for long periods [23]. We have recently investigated changes to the TCP congestion avoidance policy which modestly improve the fairness problem for connections with long RTTs, but we could not completely solve the fairness problem via simple changes to TCP [24]. 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 [25], no such algorithm has been successfully constructed in practice.

## IV. Methodology

### A. 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 [26]), 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 2a illustrates the experimental topology. To generate traffic, we used a combination of the "sock" program [7] 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 [27]. 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 [5]. Our links had no bit errors or variation in propagation delay.

In addition to controlled experiments performed in our local environment, we also describe experiments in Section 6 involving two commercial networks in our wireless networking testbed. We used a network based on a direct broadcast satellite (the Hughes DirecPC system, which covers the contintenal US), and a packet radio network (the Metricom Ricochet system, deployed in the San Francisco Bay area). For DirecPC experiments, we sent data from a computer located at the DirecPC uplink center at Germantown, MD over the satellite link to a multi-homed host on one of our subnets. We used the wide-area Internet to return acknowledgments to the traffic source. To emulate a normal user experience with the DirecPC system, we constrained the return link to be bandwidth limited to 50 Kb/s to simulate a modem connection. Although not a satellite network, the Ricochet network offers a challenging environment for transport connections, including asymmetry and large latencies; we used this network only for testing of the STP protocol as described in Section 7. In these experiments, a wired host at Berkeley communicated with a host on the Ricochet network using the packet radio network in both directions.

### B. *Simulation configuration*

We used the discrete-event network simulator known as $ns^2$ 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 [21]. Our simulation topology, which conformed closely to the experimental setup, is shown in Figure 2b. We used the simulations to verify the experimental data. For brevity, we do not plot our simulation results, which can be found in [28].

[2] http://www-mash.cs.berkeley.edu/ns/

## V. End-to-end TCP performance over satellite networks

In this section, we quantify how well different TCP implementations perform in a satellite environment for two types of workloads: large file transfers, and short Web connections.

### A. *Performance for large file transfers*

TCP is the dominant protocol for file transfers (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:

- **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 identical to TCP Reno except that it avoids false fast retransmissions [29], multiple window reductions in one window of data [30], and constrains the burstiness of the sender upon leaving fast recovery [30].
- **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.

Details of our satellite-optimized SACK-NewReno implementation are provided in the Appendix. 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 in order to introduce low levels of cross traffic (approximately 80kb/s 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 activity, the low amounts of live traffic on the networks

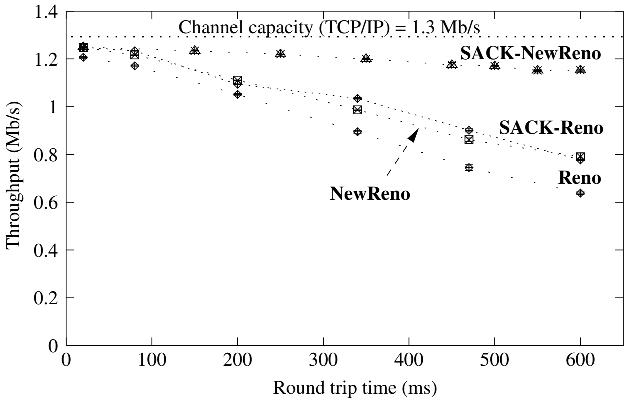



Fig. 3. Throughput performance of TCP SACK NewReno, TCP SACK Reno, TCP NewReno, and TCP Reno over an experimental path with a TCP/IP bandwidth of 1.3 Mb/s and no transmission errors. Data points represent the sample means from 20 independent transfers of 10 MB each. In this and subsequent figures, error bars represent 95% confidence intervals.

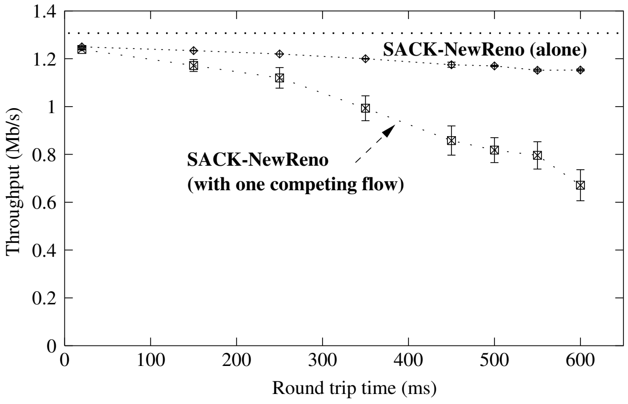



Fig. 4. The effect of a single competing short-delay connection on the satellite connection's throughput. The competing connection was a persistent file transfer using TCP SACK NewReno with a nominal 20ms RTT between a background source and sink in the experimental topology.

and the variable processing delays of the hosts sufficed to add variability to the experiments.

We plot the results of these experiments in Figure 3. In all of our figures, throughput is defined as "application-level" throughput. For low 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 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 reducing its window multiple times followed by taking a retransmission timeout after the first slow start, resulting again in slow window growth.

The above data set is appropriate to model connections entirely within a satellite subnetwork, but does not accurately portray conditions found on the wide-area Internet. In the Internet, competition from many different connections leads to network congestion. 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 4. This is the TCP fairness problem identified earlier. 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. Other recent results illustrate that multiple small-window connections can have the same effect [24]; 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.

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. Finally, the result we would like to emphasize 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. In related work, we have demonstrated how large amounts of wide-area network congestion can nearly shut out GEO satellite connections from obtaining bandwidth on a bottleneck link [24].

## *B. 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.[3] 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 [31]. 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.

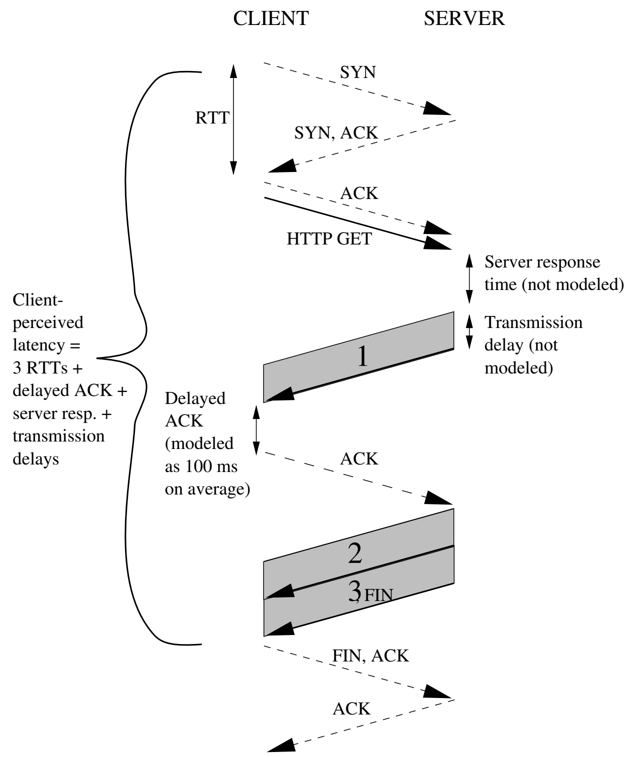



Fig. 5. TCP latency of a 3 segment server reply using standard TCP.

Figure 5, adapted from a similar figure in [32], 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.[4] 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.
- We do not model some of the bugs that have appeared in early HTTP implementations and that are discussed in [32], 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 [2].

With these assumptions in mind, consider Figure 5, 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 [7], 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 [27], comes from a well connected Berkeley subnet. The second set, collected by Gribble in 1997 [31], 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

[3] We will discuss shortly a modification to this approach, known as Persistent-HTTP (P-HTTP), which reuses the same TCP connection for multiple items.

[4] 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 [31].

TABLE I

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.

| | 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** | 0.60 | 1.2 | 1.2 | 1.4 (0.1) | 0.60 | 1.8 | 1.7 | 2.0 (0.1) |
| **TCP with “4KSS”** | 1.2 | 1.2 | 1.4 | 1.6 (0.1) | 1.2 | 1.8 | 1.7 | 1.9 (0.1) |
| **T/TCP with “4KSS”** | 0.60 | 0.60 | 0.80 | 1.0 (0.1) | 0.60 | 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 with “4KSS”** | 0.16 | 0.16 | 0.18 | 0.25 (0.01) | 0.16 | 0.24 | 0.23 | 0.31 (0.01) |
| **T/TCP with “4KSS”** | 0.08 | 0.08 | 0.10 | 0.16 (0.01) | 0.08 | 0.16 | 0.15 | 0.23 (0.01) |

the processing delays in real end systems.

In Table 1, we present the results from an analysis of the data set provided by Mah [27]. 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 on the order of 50%. 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 [31] 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 [33]) recommends that servers and clients adopt the persistent connection and pipelining techniques known as “persistent-HTTP” (P-HTTP) [34]. 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 trade-off in performance for satellite connections. On the one hand, P-HTTP 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 [32]). 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 [35]. 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 [32], 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 [36].

In summary, for connections using GEO satellite links, TCP optimizations such as T/TCP and 4KSS, especially when used together, can offer a 50% improvement in user-perceived latency and in reducing 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.

## VI. 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.

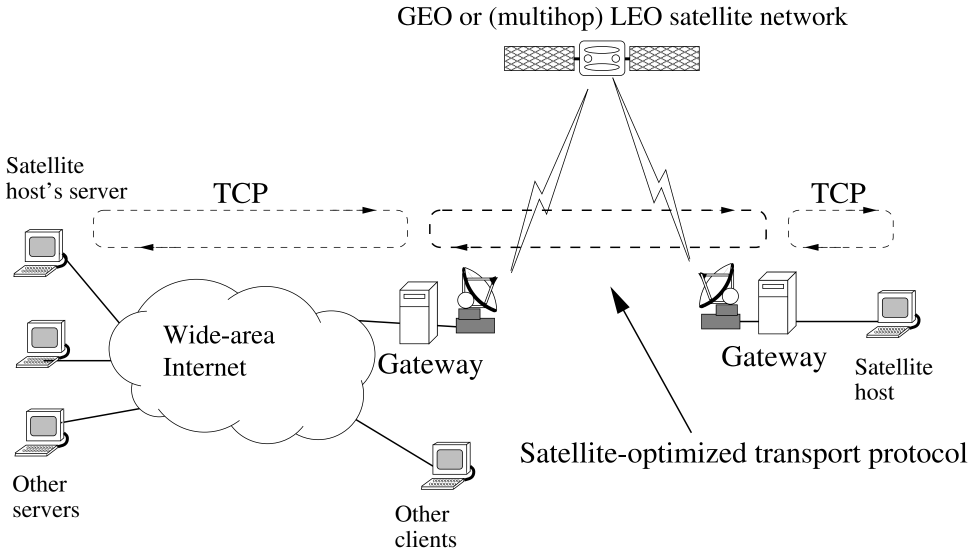



Fig. 6. Future satellite networking topology in which a satellite-based host communicates with a server in the Internet

### A. 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 6 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 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.

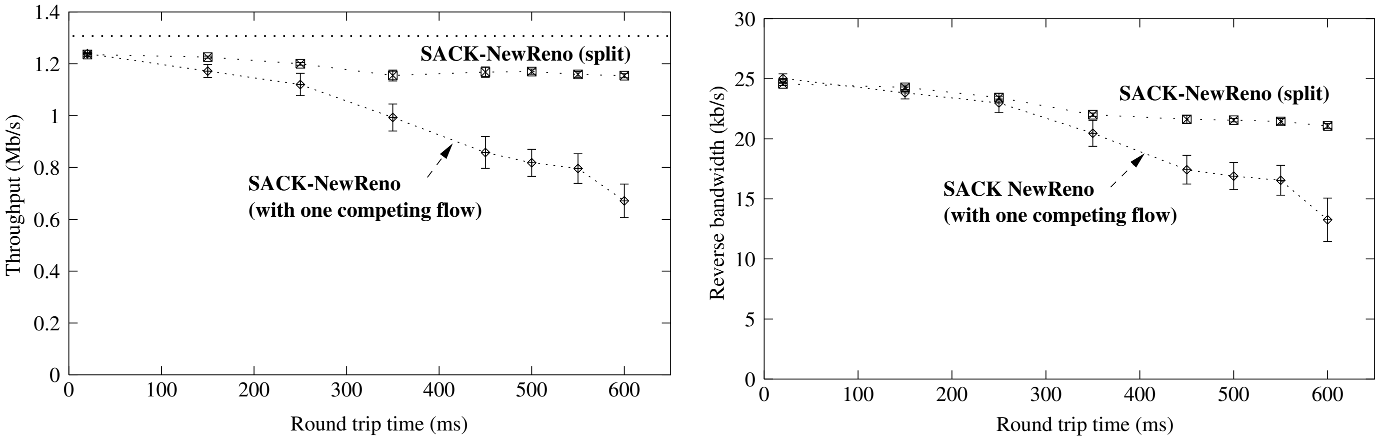
TCP may be split in the following ways:

- **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 [37]. 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) are not trivial. 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 of the transport layer. Although approaches for TCP improvement over local area wireless links, such as Berkeley's "snoop" protocol [38] and "mobile TCP" [39], can preserve end-to-end semantics, it is more difficult to due so in the satellite environment




(a) Forward throughput performance (b) Reverse channel utilization

Fig. 7. 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.

because combating the fairness problem relies on 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 gateway must be included as part of the "trust infrastructure" to operate.

### *B. Split connection performance*

In Figure 7a, we examine 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 4 (a competing short delay connection in the Internet). We replotted the relevant data from Figure 4 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 7b, 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 might be the case in some satellite systems, this sets an upper bound on the forward throughput achievable. 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 [6]) or to use a protocol over the satellite portion of the connection that uses less bandwidth. We investigate the latter possibility in the next section.

## VII. SATELLITE TRANSPORT PROTOCOL

As an alternative to further modifying TCP, we have studied the performance achievable when a protocol specifically optimized for the satellite environment is used in place of TCP. We have developed such a protocol, which we call the *Satellite Transport Protocol (STP)* [40], by making modifications to an existing ATM-based, reliable link layer protocol known as SSCOP [41]. STP can be used in two ways: i) as the satellite portion of a split TCP connection, and ii) as a transport protocol for control and network management traffic within a satellite communications network.

### *A. STP design*

The overall design of STP may be contrasted with that of TCP. Like TCP, STP provides a reliable, byte-oriented streaming data service to applications. We designed STP to offer the same API as does TCP, and to operate over an IP-based network. The transmitter sends variable-length packets to the receiver, storing the packets for potential retransmission until the receiver has acknowledged them. However, STP's automatic repeat request (ARQ) mechanism uses selective negative acknowledgments, rather than the positive acknowledgment method of TCP. Packets, not bytes, are numbered sequentially, and the STP transmitter retransmits only those specific packets that have been explicitly requested by the receiver. Unlike TCP, there are no retransmission timers associated with packets.

One of the main differences between STP and TCP, and one that offers an advantage for asymmetric networks, is the way in which the two protocols acknowledge data. TCP acknowledgments are data-driven; the TCP receiver typically sends an ACK for every other packet received. While this is beneficial for accelerating window growth upon connection startup, it results in a large amount of acknowledgment traffic when windows are large. In STP, the transmitter periodically requests the receiver to acknowledge all data that it has successfully received. Losses detected by the receiver are explicitly negatively acknowledged. The combination of these two strategies leads to low reverse channel bandwidth usage when losses are rare and to speedy recovery in the event of a loss.

STP has four basic packet types for data transfer (we ignore, for now, the additional packet types needed for connection setup and release). The *Sequenced Data (SD)* packet is simply a variable length segment of user data, together with a 24 bit sequence number and a checksum. SD packets which have not yet been acknowledged are stored in a buffer, along with a timestamp indicating the last time that they were sent to the receiver. No control data is included in the SD packets; instead, the transmitter and receiver exchange *POLL* and *STAT(us)* messages. Periodically, the transmitter sends a POLL packet to the receiver. This POLL packet contains a timestamp and the sequence number of the next in-sequence SD packet to be sent. The receiver responds to the POLL by issuing a STAT message which echoes the timestamp, includes the highest in-sequence packet to have been successfully received, and contains a list of all gaps in the sequence number space. The STAT message is similar in concept to a TCP selective acknowledgment, except that the STAT message reports the entire state of the receiver buffer (rather than the three most recent gaps in a SACK). Since each STAT message is a complete report of the state of the receiver, STP is robust to the loss of POLLs or STATs.

The fourth basic packet type is called a *USTAT (unsolicited STAT)* packet. USTATs are data-driven explicit negative acknowledgments, and are used by the receiver to immediately report gaps in the received sequence of packets without waiting for a POLL message to arrive. This allows the POLL and STAT exchange to be run at a low frequency (typically two or three per RTT when the RTT is large). In a network in which sequence integrity is guaranteed or highly likely, a USTAT can be sent upon any reception of a packet numbered beyond the next expected. If resequencing by the network is possible, USTATs can be delayed until there is a high probability that the missing packet was not reordered by the network. However, if the USTAT is sent too early there is only the small penalty of a redundant retransmission. USTATs are the primary form of negative acknowledgment, and STATs recover all second-order losses.

The basic operation of STP can best be illustrated by an example. For simplicity, Figure 8 only illustrates one direction of data transfer and assumes that sequence integrity of transmissions is preserved. In the example, the transmitter sends a series of consecutively numbered packets. After packet (SD) #4 is sent, a POLL packet is sent (due to either the expiration of a POLL timer or a threshold on the number of new packets sent). The POLL tells the receiver that the next message to be sent is #5, so the receiver knows that it should have received packets 0 through 4. In this case, since they have all been received, the receiver returns a STAT packet acknowledging all data up to and including packet #4. After sending the POLL, the transmitter continues with packets 5 through 9. However, packet #7 is lost. The receiver detects this loss upon receipt of packet #8 and immediately requests retransmission of #7 with a USTAT packet. Before this USTAT is received at the transmitter, the transmitter agains sends a POLL packet. Upon reception of the USTAT, the transmitter immediately resends #7, continues on with new data transmission, and then receives a STAT packet again reporting #7 as missing. However, the timestamp in the STAT packet allows the transmitter to determine that the retransmission has not

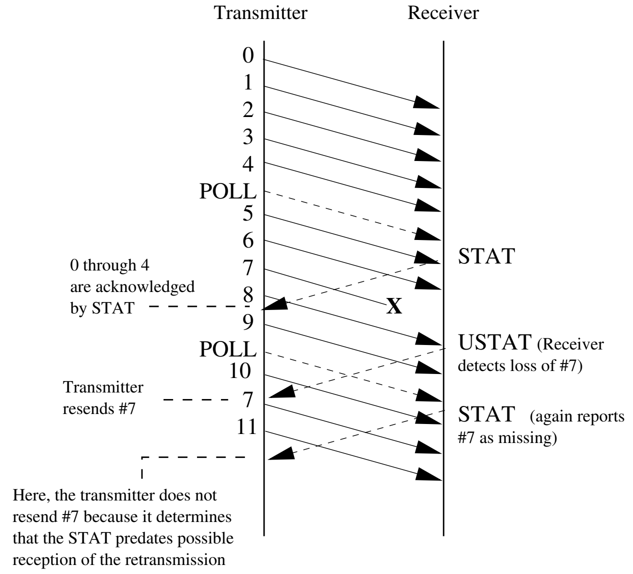


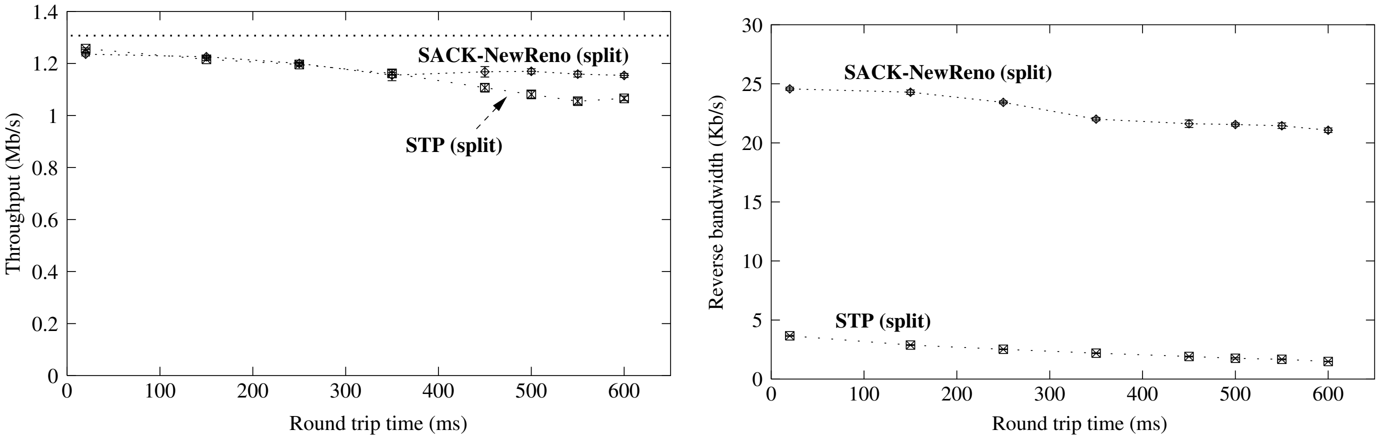

Fig. 8. Example of STP operation.

yet had an opportunity to reach the receiver, thereby avoiding an unnecessary retransmission. If #7 had again been lost, the next STAT message would have stimulated a second retransmission.

### B. *Our protocol modifications*

In the previous subsection, we described the core data transfer mechanism of STP, which is based on the basic operation of the SSCOP protocol. However, SSCOP cannot operate over connectionless networks for a number of reasons. In [40], we have described how STP builds on the basic SSCOP design through several protocol additions. In this section, we highlight three of the most important differences between STP and SSCOP; namely, the addition of a hybrid window/rate congestion control mechanism, a fast connection start that avoids unnecessary handshaking, and the piggybacking of a POLL message on a data segment.

• **Congestion control** The SSCOP specification included no flow or congestion control mechanism. For data transfer in a distributed packet-switched network, some mechanism is needed to adapt to changing network conditions. The TCP congestion control mechanism, in which each connection adjusts its sending rate based upon implicit feedback from the network (the dropping of packets), has two main problems when applied to STP. First, TCP relies on a property known as ACK-clocking: the arrival of an ACK triggers departures of new packets, which helps to smooth out the transmission of packets to a degree of burstiness that the network can accept. In STP, since ACKs (STATs) are only sent periodically, another technique to smoothly send data is required. Second, it is unlikely that congestion control in a satellite network will operate in a completely distributed manner with no bandwidth constraints. The solution that we adopted is based on modifications to TCP's flow control. In particular, we designed a mechanism that adapts to the amount of rate control desired in the network.




(a) Forward throughput performance

(b) Reverse channel utilization

Fig. 9. Comparison between split TCP and split STP. For fair comparison, both TCP and STP used identical congestion control policies.

We start with the basic TCP algorithm and describe operation when there is no network rate control. The protocol maintains a congestion window, which is set to an initial number of segments and which is guaranteed never to exceed the window offered by the receiver. The protocol then undergoes slow start by increasing its congestion window by one packet for each ACKed packet; i.e., it follows rules for TCP slow start. The congestion avoidance algorithm is also similar. However, slow start is never reentered since there are no timeouts. The protocol increases its window or enables new retransmissions only upon receipt of a STAT or USTAT message. Therefore, at every reception of a STAT or USTAT, the transmitter counts how many transmissions are enabled, and schedules them to be sent uniformly over the estimated RTT of the connection. The estimated RTT is computed from the timestamp of a received STAT and the current time, and we perform a low pass filtering across several samples to obtain the *delayed send* timer.

Next, consider the case where a minimum and maximum sending rate are imposed by the network. The technique described above easily generalizes to this case by constraining the allowable values of the timeout interval for the *delayed send* timer. If a hard upper bound exists on the sending rate, retransmissions can be counted among the packets scheduled to be sent. The required granularity of the *delayed send* timer depends on the granularity of the rates enforced by the network and on the access speed of the network.[5] Additionally, the granularity of the timer may be relaxed to reduce the overhead of interrupts in the protocol processing. In our implementation, we used timers with a granularity of 10 ms.

• **Handshake avoidance** SSCOP originally had hooks placed in the protocol specification to allow the standardization of a "fast connection start," but the mechanism was never completed. We added this feature to STP as follows. Data is allowed to be sent in a BEGIN message, in anticipation of a connection acceptance by the peer host. In addition, depending on the initial value of the window (if window control is being used in a network), SD and POLL segments may also be sent before an acknowledgment of the BEGIN message is received. Therefore, both the T/TCP reduced handshaking and policies such as the 4KSS may easily be implemented. Connection sequence numbers help to distinguish different connections in much the same way as in T/TCP.

• **Piggybacked POLL** Finally, a fundamental design principle of SSCOP was the separation of data and control flow. SSCOP was designed for an ATM environment, in which a POLL message fits into a single cell and occupies a small amount of switch buffering. For this reason, POLL messages or ACK information is not piggybacked on SD segments, although the mechanism was seriously considered during SSCOP development. In the Internet, however, most IP routers place buffer limits on the number of packets received, not on the size of such packets, so a POLL segment actually takes up as much buffer space as full data segment. Because of this, we noticed in our initial experiments an effective reduction of usable buffer space along the forward data path. Therefore, we experimented with piggybacking POLL messages on outgoing data segments if both types of segments were scheduled to be sent around the same time. This modification helped greatly, reducing the number of standalone POLL segments by about an order of magnitude, leading to substantial improvement at the small cost of defining an additional packet type. Moreover, piggyback POLLs can be used to efficiently and quickly trigger STAT responses when the windows are too small to justify periodic POLLing.

More complete details and a specification of STP can be found in [42].

### C. STP performance

Figure 9 plots the difference in file-transfer performance between split STP and split TCP (SACK-NewReno) when there is competing short-delay traffic in the wide-area Internet. To permit a fair comparison between the two protocols, we implemented in STP the identical slow start, congestion avoidance, and exponential backoff algorithms found in TCP (the main difference is that STP uses byte counting, rather than ACK counting, to build its congestion window). In practice, depending on the bandwidth management employed in the satellite network, other congestion control mechanisms may perform better. Figure 9 illustrates that STP achieves approximately the same for-

[5] It may be possible to relax the required granularity of this timer if the MAC layer also performs traffic smoothing.

ward throughput as TCP, because the forward throughput is primarily governed by the congestion avoidance policy. For long RTTs, STP's throughput is slightly smaller than TCP's because the STP congestion control mechanism, in smoothing the transmission of new data over the estimated RTT of the connection, effectively makes the control loop longer. We found the bandwidth overhead in the forward direction to be slightly lower for STP than for TCP, since the per-segment overhead reduction in STP data packets more than compensates for the POLL traffic. In the reverse direction, STP uses much less bandwidth than TCP. STP's reverse channel usage linearly decreases with the RTT, since we configured the polling frequency to be 3 times the estimated RTT of the connection. The amount of return bandwidth required is therefore independent of the forward throughput. We have also found STP to be less sensitive to variations in RTT, since the sending of data does not rely on regular reception of ACKs on the reverse path [40].

We also examined the performance of STP versus that of TCP and T/TCP for short transfers. There is an inherent tradeoff between the user-perceived latency of the connection and the amount of bandwidth used to return ACKs. To complete the connection as fast as possible, data must be ACKed regularly and quickly, but this leads to more packets sent on the reverse channel. For long file transfers when the window and buffers are large, data can be ACKed less frequently. In our STP design, when the congestion window was low (below some threshold value), we configured the STP transmitter to send the last packet of every data burst with a piggybacked POLL, and to suppress timer-driven POLL transmissions. When the window grew above the threshold, POLL transmissions were regularly scheduled. This led to frequent STAT messages (one per arrived data burst) at the beginning of connections, but also reduced the relative amount of POLL traffic in the forward direction and kept the latency low. The overall STP behavior is similar to that of T/TCP for short transfers, while for long transfers when the window is large, the reverse channel utilization is greatly reduced. In our experiments, we found that a window threshold value of approximately 10 times the segment size worked well.

Table 2 illustrates the relative performance of TCP, T/TCP, and STP in terms of both the average latency and average number of packets, when driven by a traffic generator based on the HTTP trace distributions of [27]. The data were collected from experiments on a local network in which the device drivers of the hosts were configured to produce a RTT of 600 ms, and STP, T/TCP, and TCP implemented standard TCP congestion avoidance with an initial congestion window size of one. Each table entry is the average latency of 1000 independent runs with the given protocol. We observed that STP's performance was better than TCP's but slightly worse than T/TCP's, both in terms of average latency and average number of packets per connection. The reason that the number of packets required for an STP connection is higher than for T/TCP is because, as discussed above, for small values of the congestion window, the protocol "ACKs" (i.e., sends a STAT) more frequently than every other packet, to reduce latency. However, the reason that STP's latency is not consequently lower than T/TCP's is due to its traffic smoothing mechanism: packets eligible for transmission are not sent immediately but rather paced out over the estimated RTT.

TABLE II
Comparison of TCP, T/TCP, and STP performance for HTTP traffic. The results are averages of 1000 HTTP transfers, where the traffic generated was drawn from an empirical distribution based on traces described in [27].

| | Avg. latency (s) | Avg. packets |
|---|---|---|
| **TCP** | 2.0 | 12.3 |
| **T/TCP** | 1.4 | 7.3 |
| **STP** | 1.5 | 9.1 |

In short, this data illustrates yet another tradeoff in protocol design, this time between smoothing bursty data and reducing latency. For small transfers, STP behavior could be further tuned to more closely approximate T/TCP operation, although we did not experiment with this approach. Empirically, we have observed that Web browsers using STP over GEO-like emulated channels continue to operate with good performance for reverse channels with bandwidth as low as 1 Kb/s, while such a constrained backchannel renders conventional TCP unusable.

In addition to laboratory testing, we experimented with the performance of both TCP SACK-NewReno and STP in commercial networks. As described in Section 4, we used the DirecPC satellite system and Ricochet packet radio networks, both of which are high latency networks with asymmetric paths. The RTT over the DirecPC system and back through the Internet was roughly 375 ms over 12 hops. The base RTTs in the Ricochet system were roughly 350 ms, but because of the deep packet queues in the radio network, latencies could range as high as 15 seconds. In addition, 15 network hops were required between the wireless gateway and the machine at Berkeley. Table 3 provides experimental results from several file transfers over these systems. Both networks rely on the wide-area Internet for at least a portion of the traversed path. For the DirecPC network, the average forward throughput performance for STP is better than that of TCP, and STP also uses less than half of the reverse bandwidth required for TCP. Similarly, STP does better on average in the packet radio network. The packet buffers in this case are very deep, and STP's sending behavior was so smooth that we often observed extremely long queueing delays (15 seconds) built up in the network before STP took a loss due to buffer overflow. This behavior suggests that STP, when used in low bandwidth networks, should back off its window growth upon detection of lengthening RTTs. In addition, the fact that some transport protocols can induce this much queueing delay argues for the deployment of router-based congestion control mechanisms such as Random Early Detection (RED) [43] in packet radio networks [44].

In summary, we have described the design and performance of a satellite-optimized transport protocol which compares favorably with satellite-optimized TCP for certain environments. STP inherently incorporates many of the features that have been proposed or adopted as TCP options for improved satellite performance. STP also allows for the use of rate-based congestion control, and because the reverse bandwidth usage is roughly constant, STP is well matched to satellite networks which allocate fixed amounts of uplink bandwidth to users (such as those using TDMA multiple access). One drawback of using STP with

TABLE III

Results of file transfer experiments over the DirecPC DBS system and Ricochet packet radio network. The throughputs listed are the averages of 25 file transfers. The file sizes were 1 MB for DirecPC and 100 KB for Ricochet.

| | **STP (Kb/s)** | **TCP SACK (Kb/s)** |
|---|---|---|
| **DirecPC fwd.** | 480 | 370 |
| **DirecPC rev.** | 2.8 | 7.6 |
| **Ricochet fwd.** | 28.1 | 27.1 |
| **Ricochet rev.** | 0.6 | 0.9 |

a heterogeneous client population is the requirement that either the end host implement STP or the satellite network interface (such as a set-top box) convert the protocol back to TCP. However, many of the changes proposed as TCP options also require client-side changes; particularly those dealing with TCP asymmetry. Finally, STP can be used internally within a satellite network by applications that are written to use TCP. For example, Web caches in a global satellite network could be interconnected by STP connections.

## VIII. Related work

There have been several efforts aimed at improving transport protocol performance over satellite links. Partridge and Shepard discuss several causes for poor satellite TCP performance in [14]. Regarding TCP modifications for the satellite environment, three recent research efforts stand out. The first is the development of a modified version of TCP known as the *Space Communications Protocol Standards– Transport Protocol (SCPS-TP)* for the general space environment [45]. SCPS-TP proposes a new TCP option which would enable several changes to basic TCP mechanisms, including the following: distinguishing between packet loss and packet errors (to react differently to the two events), using the TCP Vegas [46] congestion avoidance algorithms, identifying link outage events, performing header compression, and using selective negative acknowledgments. However, SCPS-TP does not advocate a particular strategy for handling asymmetric channels, although several possibilities are discussed. A more comprehensive study on the use of TCP over asymmetric channels was recently performed at Berkeley [6], although the motivation for the study was packet radio and wireless cable networks. The authors investigated several techniques for reducing the frequency of ACKs generated by the TCP receiver, by examining both network agent-based solutions that do not require host modifications and solutions involving modifications to the TCP implementation. By combining strategies from SCPS-TP and the Berkeley modifications for asymmetry, it is possible to construct a modified TCP which behaves quite similarly to STP, although it requires implementation changes at both the sender and receiver, or receiver-side gateways. Finally, there currently is a research and standardization effort within the IETF to identify satellite-friendly TCP protocol options for use when traversing a path that includes a satellite link [13].

STP is an outgrowth of the ATM link-layer protocol known as the *Service Specific Connection Oriented Protocol (SSCOP)* [41]. SSCOP itself was primarily a synthesis between two research efforts in the 1980s. Researchers at AT&T developed the "SNR" protocol for high bandwidth-delay product networks [47]; the protocol is named after its inventors Sabnani, Netravali, and Roome. In parallel, COMSAT Laboratories was working on selective-repeat strategies for satellite networks [48]. Standardization proposals based on these efforts were combined to form SSCOP. Some STP protocol mechanisms such as selective negative acknowledgments and USTATs resemble similar mechanisms in the unicast transport protocol for Xpress Transport Protocol (XTP) version 4.0 [49]. Timer-driven acknowledgment mechanisms similar to those in SSCOP date back to 1984 [50]. The error performance of SSCOP was studied in [51].

In the 1980's, a protocol known as NETBLT (NETwork BLock Transfer) was designed for data transfer over satellite IP links [52]. Two main differences exist between STP and NETBLT. First, data transfer in NETBLT is semantically arranged in large fixed block sizes called "buffers." Applications are aware of these data boundaries and pass contiguous buffers to the transfer protocol. This is in contrast to the "stream" data service of TCP and STP, where the application data unit is a single byte. Second, flow control in NETBLT is based on rate control rather than window control, and the parameters of rate control are negotiated during connection setup and periodically throughout the connection (although using rate control as part of congestion control is not specified). STP resembles NETBLT in its use of selective acknowledgments, and in its support of rate control to supplement window control.

## IX. Conclusions

In this paper, 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 latencies (or long LEO paths) is challenging.

ii) 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.

iii) 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 50% or more for GEO links.

iv) We showed that the performance problems due to misconfigured 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.

v) Finally, we designed an alternative transport protocol (STP) for the satellite environment, and were able to achieve up to an order of magnitude reduction in the reverse bandwidth used for large file transfers. For small transfers, we found STP performance to approach that of T/TCP.

Our results have implications for the design of future networks:

- Direct-to-user satellite service providers cannot depend on the many existing TCP implementations in the world to operate with high performance over satellite links. Moreover, even the best of TCP implementations may fall prey to TCP unfairness when operating over congested paths. Therefore, satellite networks designed to provide Internet access should incorporate Web caches and proxies as much as possible, to decouple the satellite subnetwork from the rest of the Internet. A satellite-optimized protocol, either TCP with appropriate enhancements or another protocol such as STP, should be used between the proxy and the satellite user. Furthermore, the use of split-connection gateways for non-Web traffic is also recommended if security and scalability concerns can be resolved. With these accommodations for TCP, even GEO-based satellite networks can provide high performance transport connections.
- The fundamental fairness problem in TCP congestion avoidance may be difficult to rectify solely with algorithms in end hosts [24]. If, however, TCP-friendly buffer management and fair scheduling mechanisms were to be introduced into routers in the Internet (such as discussed in [22]), satellite connections would potentially be among the main beneficiaries.
- STP is a viable alternative to TCP for satellite networks based on datagram packet routing. Not only could STP be implemented as part of a split connection approach, it could also be used internally within a satellite network to support control and network management traffic, especially since the use of STP can be made transparent to applications designed for TCP. STP is especially well suited for networks that impose rate controls on users (such as fixed MAC bandwidth) or that have a very low bandwidth return channel.

## 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 [53] and [10], with the following extensions.[6] The following extensions apply whether or not SACK is enabled for a given connection:

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 [53].
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, 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()*.
5. Do not send more than four segments in response to any acknowledgment.

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

6. 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.
7. 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.
8. 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.

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

## ACKNOWLEDGMENTS

We thank Venkata Padmanabhan for implementing modified device drivers used in our experiments, and for suggesting the slow-start transition from the NewReno recovery phase. We also thank Keith Sklower for implementing drivers for the DirecPC experiments and Giao Nguyen for assistance in developing the $ns$ simulation models. We are grateful to Bruce Mah and Steve Gribble for providing HTTP traffic trace data. Additionally, we thank Hari Balakrishnan and Venkata Padmanabahan for discussion and 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.

## REFERENCES

[1] J. Postel, "Transmission Control Protocol," *Internet RFC 793*, 1981.
[2] W. Stevens, *TCP/IP Illustrated, Volume 3*, Addison Wesley, 1996.
[3] M. Sturza, "The Teledesic Satellite System," *Proceedings of 1994 IEEE National Telesystems Conference*, pp. 123–126, 1994.
[4] E. Fitzpatrick, "SPACEWAY system summary," *Space Communications*, vol. 13, no. 1, pp. 7–23, 1995.
[5] B. Gavish and J. Kalvenes, "The impact of satellite altitude on the performance of LEOS based communication systems," *Wireless Networks*, vol. 4, no. 2, pp. 199–212, 1998.
[6] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance," *Proceedings of Third ACM/IEEE MobiCom Conference*, pp. 77–89, Sept. 1997.
[7] W. Stevens, *TCP/IP Illustrated, Volume 1*, Addison Wesley, 1994.
[8] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of ACM SIGCOMM '88 Conference*, pp. 314–329, 1988.
[9] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *Internet RFC 1323*, 1992.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," *Internet RFC 2018*, 1996.
[11] R. Braden, "T/TCP– TCP Extensions for Transactions, Functional Specification," *Internet RFC 1644*, 1994.
[12] J. Mogul and S. Deering, "Path MTU discovery," *Internet RFC 1191*, 1990.
[13] M. Allman and D. Glover, "Enhancing TCP Over Satellite Channels using Standard Mechanisms," *Internet-Draft (work in progress): draft-ietf-tcpsat-stand-mech-05.txt*, Aug. 1998.
[14] C. Partridge and T. Shepard, "TCP Performance over Satellite Links," *IEEE Network*, vol. 11, no. 5, pp. 44–49, Sept. 1997.
[15] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," *Internet RFC 2414*, 1998.
[16] V. Padmanabhan and R. Katz, "TCP Fast Start: A Technique for Speeding Up Web Transfers," *Proceedings of IEEE Globecom '98 Internet Mini-Conference*, 1998.
[17] J. Touch, "TCP Control Block Interdependence," *Internet RFC 2140*, 1997.
[18] J. Heidemann, "Performance Interactions Between P-HTTP and TCP Implementations," *ACM Computer Communications Review*, vol. 27, no. 2, pp. 65–73, Apr. 1997.
[19] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," *Proceedings of ACM SIGCOMM '97 Conference*, pp. 167–180, 1997.
[20] 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*, vol. 5, no. 3, pp. 336–350, June 1997.
[21] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet Switched Gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115–156, Sept. 1992.
[22] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, "Design Considerations for Supporting TCP with Per-flow Queueing," *Proceedings of INFOCOM '98*, pp. 299–306, 1998.
[23] 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*, pp. 1199–1209, 1997.
[24] T. Henderson, E. Sahouria, S. McCanne, and R. Katz, "On Improving the Fairness of TCP Congestion Avoidance," *Proceedings of IEEE Globecom '98 Conference*, 1998.
[25] 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.
[26] G. Wright and W. Stevens, *TCP/IP Illustrated, Volume 2*, Addison Wesley, 1995.
[27] B. Mah, "An Empirical Model of HTTP Network Traffic," *Proceedings of INFOCOM '97*, 1997.
[28] T. Henderson, "TCP Performance over Satellite Channels," *Technical Report, University of California, Berkeley*, 1999.
[29] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *Proceedings of ACM SIGCOMM '96 Conference*, pp. 270–280, 1996.
[30] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communications Review*, vol. 26, no. 3, pp. 5–21, July 1996.
[31] 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*, Dec. 1997.
[32] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the Performance of HTTP Over Several Transport Protocols," *ACM/IEEE Transactions on Networking*, vol. 5, no. 5, pp. 616–630, Oct. 1997.
[33] R. Fielding, J. Gettys, J. Mogul, H. Prystyk, and T. Berners-Lee, "Hypertext Transfer Protocol– HTTP/1.1," *Internet RFC 2068*, 1997.
[34] V. Padmanabhan and J. Mogul, "Improving HTTP Latency," *Proceedings of the Second International World Wide Web Workshop*, Oct. 1994.
[35] 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.
[36] V. Padmanabhan, "Addressing the Challenges of Web Data Transport," *Ph.D. Thesis, University of California, Berkeley*, 1999.
[37] 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.
[38] H. Balakrishnan, V. Padmanabhan, E. Amir, and R. Katz, "Improving TCP/IP Performance over Wireless Networks," *Proceedings of First ACM/IEEE MobiCom Conference*, Nov. 1995.
[39] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *ACM Computer Communications Review*, vol. 27, no. 5, pp. 19–43, Oct. 1997.
[40] T. Henderson and R. Katz, "Satellite Transport Protocol (STP): An SSCOP-based Transport Protocol for Datagram Satellite Networks," *Proceedings of 2nd Workshop on Satellite-Based Information Systems (WOSBIS '97)*, 1997.
[41] "B-ISDN Signaling ATM Adaptation Layer– Service Specific Connection Oriented Protocol (SSCOP)," *ITU-T Recommendation Q.2110*, 1994.
[42] T. Henderson, "Satellite Transport Protocol Specification," *Technical Report, University of California, Berkeley*, 1999.
[43] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 115–156, Sept. 1993.
[44] S. McCanne, "Private communications," 1997.
[45] E. Travis R. Durst, G. Miller, "TCP Extensions for Space Communications," *Wireless Networks*, vol. 3, no. 5, pp. 389–403, 1997.
[46] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Avoidance," *Proceedings of ACM SIGCOMM '94*, pp. 24–35, Oct. 1994.
[47] A. Netravali, W. Roome, and K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 2010–24, Nov. 1990.
[48] K. Mills, D. Chitre, H. Chong, and A. Agarwal, "A Joint COMSAT/NBS Experiment on Transport Protocol," *Proceedings of 7th International Conference on Digital Satellite Communications (ICDSC)*, May 1986.
[49] "The XTP 4.0 Specification," *XTP Forum*, 1995.
[50] R. Donnan, "Method and System for Retransmitting Incorrectly Received Numbered Frames in a Data Transmission System," *U.S. Patent No. 4439859*, 1984.
[51] T. Henderson, "Design Principles and Performance Analysis of SSCOP: A New ATM Adaptation Layer Protocol," *ACM Computer Communications Review*, vol. 25, no. 2, pp. 47–59, Apr. 1995.
[52] D. Clark, M. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," *Internet RFC 998*, 1987.
[53] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *Internet RFC 2001*, 1997.