Chapter 5

Satellite Transport Protocol

In the previous chapter, we illustrated the performance advantage gained by splitting a TCP connection at a gateway within an access satellite network. Given such a split connection, however, there is no requirement to actually use TCP within the satellite network. Moreover, for connections completely internal to a satellite network, transport protocols other than TCP are possible. In this chapter, we describe a new transport protocol optimized for asymmetric satellite access networks. The main difference between this protocol and TCP is in the way data is acknowledged. The designers of TCP chose to use a steady stream of data acknowledgments as a pacing mechanism to clock out new data; the implicit design tradeoff was to simplify the protocol implementation at the cost of extra traffic in the network. For broadband geostationary satellite systems, bandwidth is at a premium– therefore, our choice is to reduce the traffic load on the backchannel as much as possible, at the possible expense of more complicated implementations. Fortunately, we can avoid significant increases in complexity by changing the basic data transfer mechanism of the protocol. Our protocol, which we dubbed the "Satellite Transport Protocol" (STP), outperforms TCP in satellite environments characterized by high bit error ratios, asymmetry, or widely varying round trip times. 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. This chapter describes the overall protocol design, followed by simulation and experimental results.

5.1 Design Goals

In Section 2.1.3, we introduced the background material relating to the development of STP. In summary, STP is an outgrowth of the ATM-based link layer known as SSCOP. While SSCOP was mainly intended for networks with large bandwidth-delay products such as broadband wide-area ATM networks, many of the same design principles of that protocol help greatly in the satellite environment. In this section, we describe our design requirements for an optimized satellite transport protocol and discuss how we modified the core protocol mechanism of SSCOP to develop STP.

5.1.1 Basic Design

To understand STP, it is perhaps easiest to contrast its operation 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.

Packet Types

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 that 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

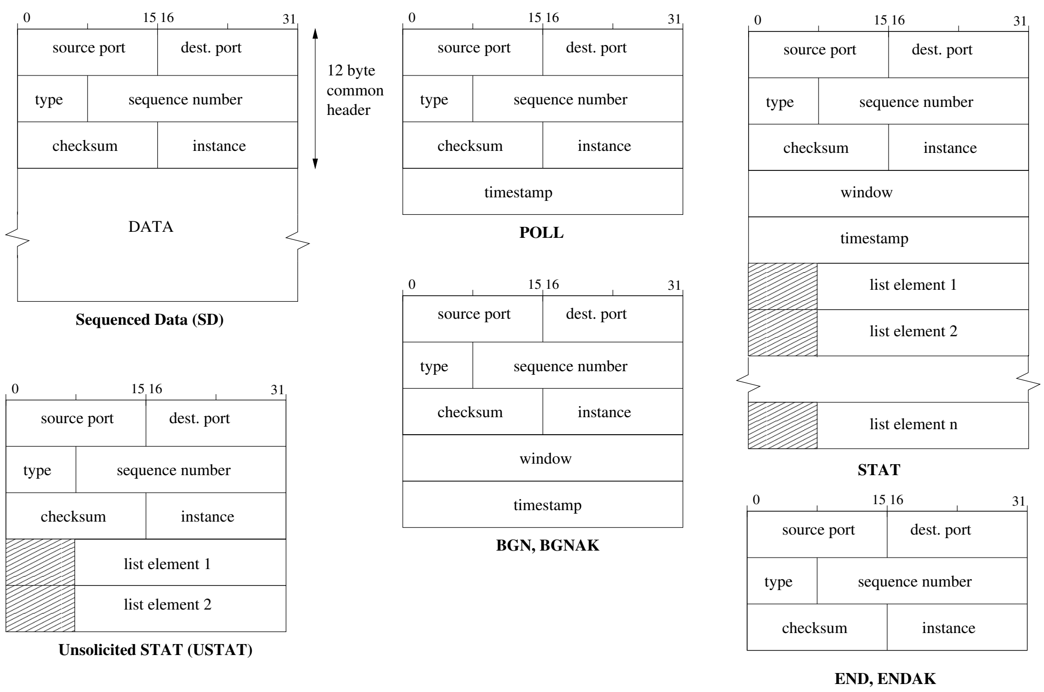

Figure 5.1: Illustration of the main packet formats in STP.

penalty of a redundant retransmission. USTATs are the primary form of negative acknowledgment, and STATs recover all second-order losses.

Figure 5.1 illustrates the key packet types used by STP. The fields are byte-aligned and the data portion of packets is 32-bit aligned. Above, we discussed the use of the SD, POLL, STAT, and USTAT PDUs. The BGN and BGNAK are used to open a connection, and the END and ENDAK are used to close a connection. Each packet contains a 12-byte common header that includes the source and destination port numbers (as in TCP), a type field, a 24-bit sequence number, a 16-bit checksum, and an instance number (to distinguish between different connections that may happen to use the same port numbers). Certain packets are permitted to be concatenated for the purpose of conserving the number of packets transmitted; for example, an SD and a POLL packet may be concatenated, in which case the type field is the logical "OR" of the SD and POLL values, and the POLL timestamp precedes the data.

Bulk Data Transfer Operation

The basic operation of STP can best be illustrated by an example. For simplicity, Figure 5.2 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

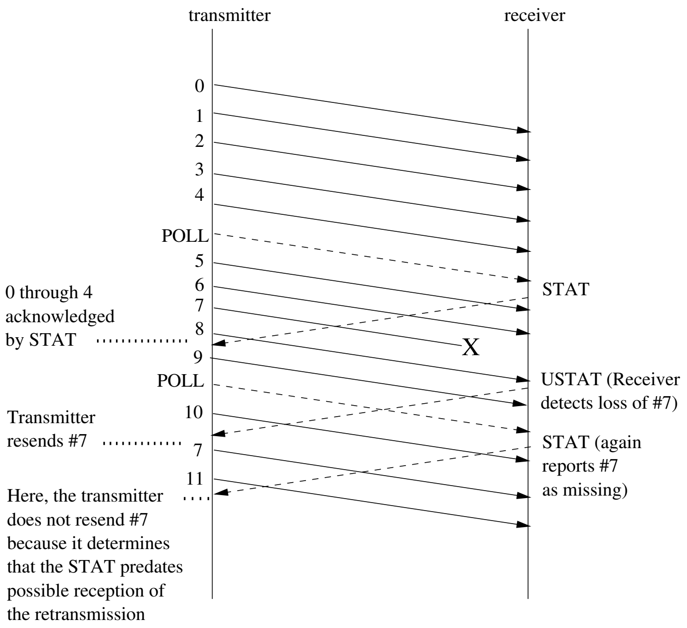

Figure 5.2: Basic STP operation for bulk data transfer.

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 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.

The key to the performance of the protocol is the frequency with which the transmitter polls the receiver. If the bit error rate is high or the sender and/or receiver are using small windows (either due to small socket buffer sizes or a congestion window that has not yet opened up to a large value), it is advantageous to poll frequently, perhaps three or more times per round trip time (to most quickly recover from losses and open up the window). However, if the above conditions are not met, then it is safe to poll once per round trip time or less. This is because, under low loss conditions, the *USTAT* message provides the first-order recovery mechanism for losses. By polling infrequently, the best savings on the bandwidth usage on the return channel can be realized.

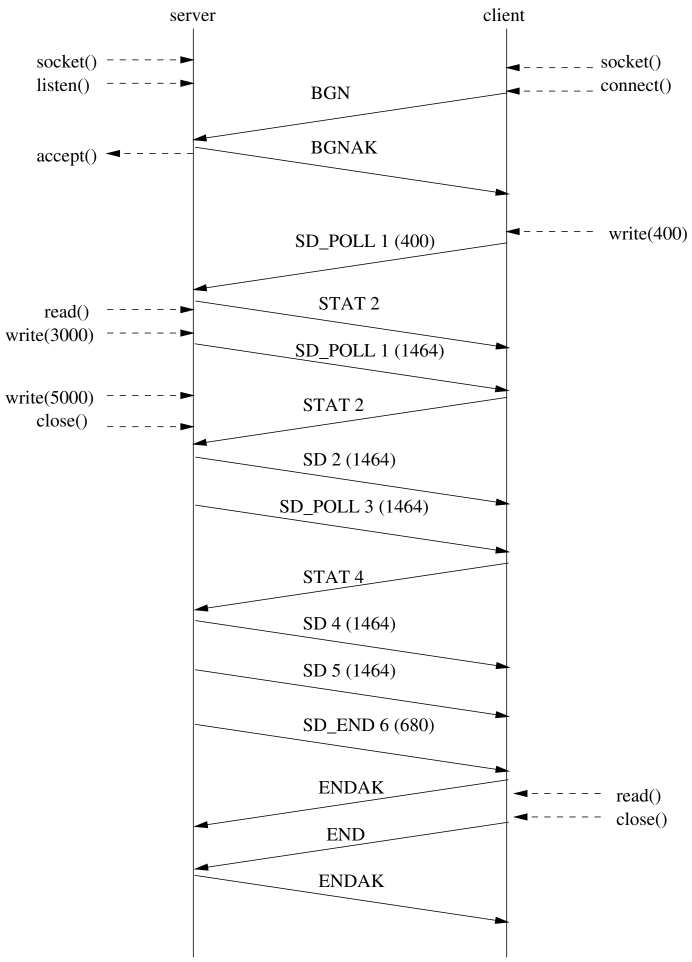

Figure 5.3: Basic STP operation for short transactions.

Short Data Transactions

Figure 5.3 illustrates a hypothetical data transaction in which the client (the initiator of the connection) writes 400 bytes of data to a server and receives an 8000 byte response. The example illustrates typical system calls that would be used by the application. The connection uses TCP-like window control, so that the congestion window builds by one packet for each packet acknowledged. The server is listening on a particular port, and the client connects to that port by issuing a `connect()` system call, which causes STP to send a BGN packet. This exchange of BGN and BGNAK coordinates the sequence numbers to be used by both sides and establishes the window sizes in each direction. The client then writes 400 bytes to the socket, which stimulates an SD to be sent to the other side. In this case, the client is configured to POLL with the first burst of data, so the actual packet sent is a concatenation of an SD with a POLL. The server responds to the POLL by issuing a STAT, reporting the next sequence number (#2) that the server expects

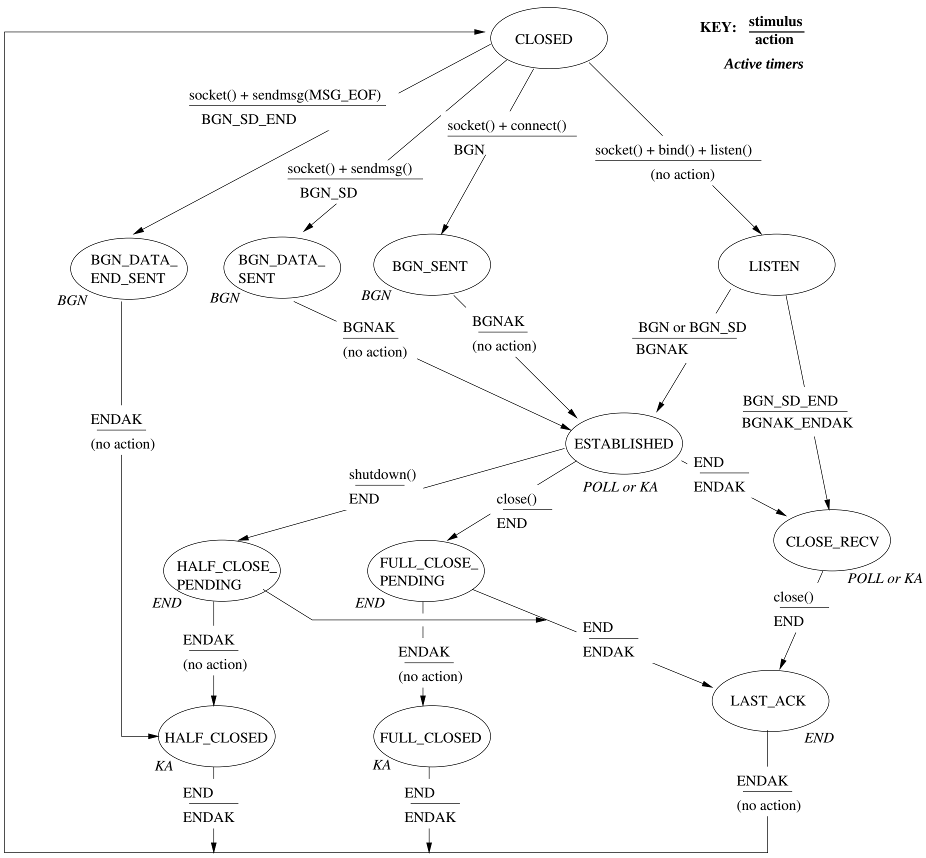

Figure 5.4: Overview of STP state transitions.

to receive. The server then writes 8000 bytes of data to the socket, and it requires six packets to complete the transfer. Upon successful completion of the data transfer, both sides exchange END messages to close down their respective halves of the connection.

State Machine

Figure 5.4 illustrates the state machine associated with typical STP connections, as well as the common stimuli and responses that effect the transitions (packets are labeled in capital letters, while socket calls are listed in small case). This diagram illustrates a number of additional packet types (BGN, BGNAK, END, ENDAK) used for connection control. In much the same way that a TCP segment can be overloaded to perform more than one function (for example, a TCP *SYN* flag may be combined with a TCP *ACK* flag), these STP connection control packets can be combined with data (e.g., BGN_SD packet for fast connection opening) or with other connection control packets (e.g., BGNAK_ENDAK). Using the basic client-server model of data communications, a connection is instantiated by one side (server) putting a socket into the *LISTEN* state, and the other side (client) initiating the connection by sending a BGN packet. In STP, since we allow the protocol to skip the initial handshake waiting for a BGNAK, we define a number of states (BGN_SENT, BGN_DATA_SENT, and BGN_DATA_END_SENT) that capture the state of the client before it receives a BGNAK. The normal data transfer state is the *ESTABLISHED* state. Applications can either close both directions of communications by issuing a `close()` system call, or can shutdown the write side of the connection by issuing a `shutdown()` system call. The remainder of the states illustrated involve the closing of the connection by both sides.

Also listed in Figure 5.4 are the timers active in each state. STP has four main timers: the *BGN* and *END* timers used to guarantee receipt of the respective packet types of that name, and the *POLL* and *KEEPALIVE* timers, active during data transfer. If data is outstanding, the *POLL* timer will be running; otherwise, the *KEEPALIVE* timer will (very infrequently) poll the peer to ensure that the connection is still up.

5.1.2 Our Protocol Modifications

In Section 5.1.1 above, we described the core data transfer mechanism of STP, which follows closely the basic operation of the SSCOP protocol. However, SSCOP cannot operate over connectionless networks for a number of reasons. In [59], 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.

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.[1] 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, which corresponds to a timer "tick" in BSD-derived systems. There exists a tradeoff between the granularity of the delayed send timer and the smoothness with which data is submitted to the network; as the data transfer rate grows, more and more packets are emitted at once when this timer expires. In our experiments, the speed of our network interface cards was more of a limiting factor than this timer granularity.

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 with a BGN packet, 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 BGN packet 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. Although the use of T/TCP connection handshaking in the wide-area Internet is deprecated due to denial-of-service concerns, in a network where the satellite service provider controls both of the endpoints, such concerns are mitigated.

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, such as the initial period of data transfer on a congestion-controlled connection.

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

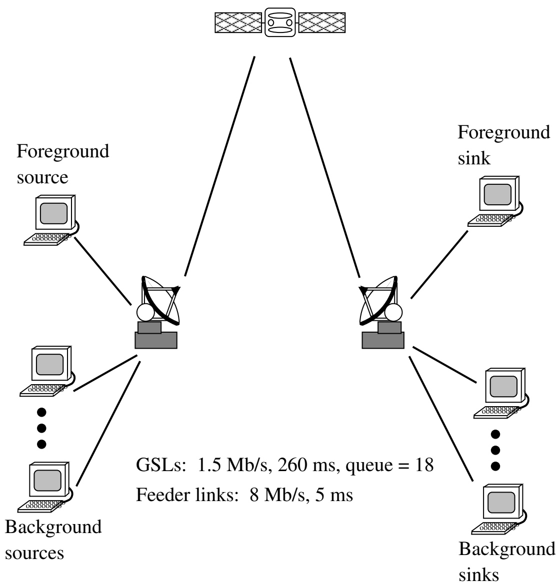

Figure 5.5: Simulation configuration for GEO topology. Simulations involved measuring the performance of file transfers between "foreground" hosts against background Web-like traffic generated by "background" hosts.

5.2 Simulation Results

We implemented the data transfer procedures of STP in the UCB/LBNL network simulator *ns*, described in Chapter 3. We were interested in comparing the performance of STP and TCP by examining the performance of persistent connections (i.e., long file transfers) over simulated satellite network topologies. In this section, we describe our simulations, present the results of a comparison of STP and TCP performance with respect to bulk data transfers, and present performance results for STP connections in a high BER environment.

5.2.1 Simulation Configuration

Topologies

Figures 5.5 and 5.6 illustrates the two simulated topologies with which we experimented. The first topology is configured to emulate a 1.5 Mb/s channel over a geosynchronous (GEO) satellite channel with a one way delay of 260 ms. Counting the propagation delays of the feeder links, the total RTT experienced by a user is 532 ms, excluding queueing and transmission delays. The queue sizes were set to approximately 10 percent of the outgoing line rate (a commonly used rule-of-thumb for queue sizes in practice). The second topology illustrates a hypothetical low-earth-orbit (LEO) configuration. The access links to the satellite have a one-way propagation delay of 5 ms and a bit rate of 2 Mb/s, and the intersatellite links have a propagation delay of 10 ms and a bit rate of 100Mb/s. The topology is similar to transcontinental connections across proposed broadband LEO

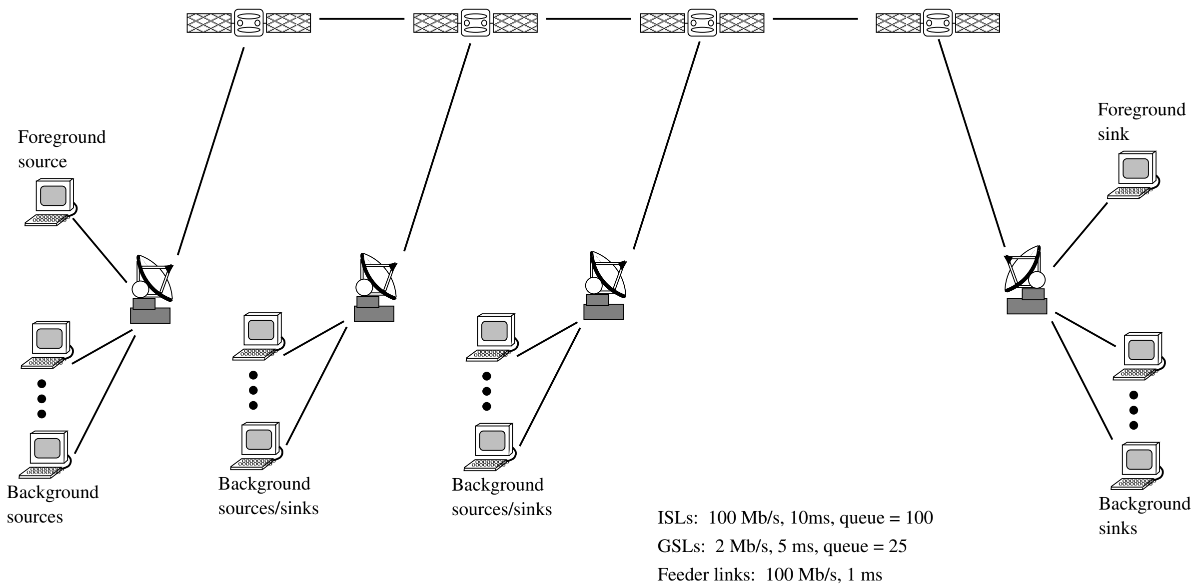

Figure 5.6: Simulation configuration for LEO topology. Simulations involved measuring the performance of file transfers between "foreground" hosts, across four LEO satellite hops, against background Web-like traffic injected at various points in the topology.

systems.

TCP Configuration

We tested two TCP variants: TCP with selective acknowledgments (TCP SACK) and TCP Reno, which corresponds to a current reference TCP implementation. For the TCP simulation configuration, we used *ns* defaults for all parameters except for the offered window size, which we opened up to a large value to avoid artificially constraining the sender, and the coarse timeout interval of 500 ms, which is set by default in *ns* to the non-standard value of 100 ms. Since delayed acknowledgments are standard practice in current implementations (to reduce the reverse channel bandwidth), we configured the TCP sinks to send delayed acknowledgments (typically, an ACK is sent for every two segments). For STP, we set the threshold on duplicate acknowledgments before a USTAT is sent to three (as in TCP), and we configured the STP sender to send roughly three polls per RTT, since such a polling interval has been found to offer high performance for SSCOP [58].

Congestion Control

To permit a fair comparison of the basic protocol performance, we implemented a window-based congestion control policy in STP identical to that of TCP; namely, additive window increases of one packet per RTT and a multiplicative decrease by one half during the congestion avoidance phase, and slow start. For both the GEO and the LEO topologies, we added background Web-like traffic which occupied, on average, about 10% of the bottleneck links. This traffic emulated actual WWW usage based on empirical distributions of actual traffic traces. The purpose was not to heavily congest the network (the greedy behavior of the congestion avoidance mechanisms of the foreground connections guaranteed that congestion would frequently occur) but to add variability to the simulation runs to break up any phase effects. In the GEO topology, the background traffic was sent in the same direction as the foreground transfer, creating an occasional bottleneck at the queue at the ingress of the satellite network. In the LEO topology, the background traffic was sourced from multiple ground stations in different spot beams, creating an occasional bottleneck at either the ingress or the egress of the satellite network. In some situations, discussed below, we balanced the traffic load in each direction by creating a persistent TCP SACK connection in the reverse direction along with the same amount of simulated WWW traffic, so as to cause periodic congestion losses in the reverse path as well.

5.2.2 Bulk Transfer Performance of STP

Tables 5.1 and 5.2 present the results of an average of 200 simulation runs, each 60 seconds long, over the GEO and LEO topologies, respectively. We chose 200 runs of each configuration in order to get the small confidence intervals listed in the tables. We compared the performance of STP, TCP SACK, and TCP Reno, first with foreground traffic only and then with bidirectional traffic. Since no bit errors were introduced on the links, all losses were due to congestive losses induced by the congestion avoidance mechanisms. We used a fixed packet size of 1000 bytes (including TCP/IP or STP/IP overhead), corresponding to the *ns* default. The performance is compared in terms of forward throughput achieved ("goodput"), forward bandwidth efficiency (ratio of goodput to total TCP/IP data transferred in the forward direction) and reverse channel bandwidth usage. We observed the following:

- TCP SACK and STP both significantly outperform TCP Reno in the forward direction, especially over a long delay path. This is because (as described in Section 4.2.3 above) both TCP SACK and STP are able to recover multiple losses in a window's worth of data within a single RTT.
- Both TCP SACK and TCP Reno use much more bandwidth than STP on the reverse channel for returning ACKs. For all of the simulations averaged together, STP required roughly 5 Kb/s, while TCP Reno needed 17 Kb/s and TCP SACK used 21 Kb/s.
- For one way traffic, STP outperforms TCP SACK in the GEO case, in terms of throughput in the forward direction. This is largely because of TCP SACK's delayed acknowledgments, which cause the congestion window to open up more slowly than if every segment were acknowledged. If the TCP receiver were to acknowledge every segment, TCP would slightly outperform STP since STP acknowledgments are slightly delayed relative to the times that packets arrive. Doing so, however, would double the usage of the reverse channel bandwidth, which is already high with delayed acknowledgments. Another benefit to STP in a high bandwidth-delay product environment [2] with high losses is that it reports the complete state of the receiver with every STAT.

[2] "Bandwidth-delay product" refers to the total number of bits that can be "in the pipe" during one round trip time of the connection.

	Throughput (Kb/s)	95% conf. int. (+/-)	Fwd. efficiency	95% conf. int. (+/-)	Reverse bw. (Kb/s)	95% conf. int. (+/-)
One way traffic, GEO						
STP	672.3	20.8	0.938	0.001	2.2	0.03
TCP SACK	594.9	20.5	0.945	0.001	13.2	0.3
TCP Reno	296.5	15.2	0.928	0.003	6.7	0.3
Two way traffic, GEO						
STP	595.7	20.2	0.933	0.008	2.0	0.03
TCP SACK	388.4	13.9	0.939	0.001	8.9	0.3
TCP Reno	259.2	12.0	0.922	0.003	6.0	0.2

Table 5.1: Performance comparison of STP, TCP SACK, and TCP Reno over the simulated GEO topology.

- In the LEO case, TCP SACK slightly outperforms STP in the forward direction. We believe that this is due to STP slowing down its sending rate in response to a queueing backlog (which increases its RTT). This is actually a nice self-regulating property of the congestion control algorithm, since the sending rate is inversely proportional to the measured RTT of the connection.

- STP outperforms TCP SACK to a greater extent when there is two way traffic in the system. We believe that this is due to the effect of ACK compression, which disrupts TCP's self clocking behavior. STP is relatively insensitive to these effects, and in fact it seems that much of the reduction in its performance is due to the presence of the reverse (TCP SACK) connection's acknowledgments in its forward path buffers, effectively decreasing its usable buffer space.

- TCP SACK and STP have comparable performance in terms of forward bandwidth efficiency. Although STP has less per-packet overhead, the overhead of the three POLL messages per RTT must be amortized across data; therefore, the efficiency improves as the throughput improves.

- The difference in performance between the protocols was reduced in the LEO case as compared to the GEO case. There is less of an advantage in using TCP SACK instead of TCP Reno when the RTT is smaller.

In summary, when using the standard TCP flow control policies in identical simulated satellite environments, STP generally outperformed TCP SACK and TCP Reno in terms of throughput while using much less bandwidth in the reverse channel. We did not explore possible improvements via further fine-tuning of the protocols.

5.2.3 STP Performance in a High BER Environment

We next examined the performance of STP in a rate-controlled environment in which the transmitter was only constrained by a maximum sending rate, but in which the BER was varied

	Throughput (Kb/s)	95% conf. int. (+/-)	Fwd. efficiency	95% conf. int. (+/-)	Reverse bw. (Kb/s)	95% conf. int. (+/-)
One way traffic, LEO						
STP	1668.2	13.8	0.961	0.001	8.5	0.05
TCP SACK	1715.1	14.5	0.957	0.001	37.4	0.3
TCP Reno	1552.9	11.6	0.957	0.002	33.5	0.2
Two way traffic, LEO						
STP	1440.5	13.8	0.960	0.001	8.0	0.04
TCP SACK	1154.2	12.6	0.958	0.001	25.5	0.3
TCP Reno	975.4	15.5	0.957	0.001	21.3	0.3

Table 5.2: Performance comparison of STP, TCP SACK, and TCP Reno over the simulated LEO topology.

from 10^{-4} to 10^{-7}. The modification to STP's flow control to permit this operation is simple. The delayed send timer can be regularly scheduled to expire at the rate at which packets are allowed into the network. If retransmissions are queued, they are sent with highest priority in the scheduled slot; otherwise, a new data packet is sent. We did not impose rate control on the traffic in the reverse channel. Figure 5.7 illustrates results for a 1 Mb/s connection (1 Mb/s available transport bandwidth), again using the GEO topology shown in Figure 5.5, but for which bit errors were randomly inserted on the link. Again, we configured the IP packet sizes to be 1000 bytes for data traffic. Since the STP/IP overhead per packet is 32 bytes, the usable throughput is constrained to be 968 Kb/s at best. Figure 5.7 illustrates that the selective retransmission mechanism provides high efficiency even as the BER degrades substantially, and that the reverse channel bandwidth also rises as the BER increases (due to the increased use of the *USTAT* message), as shown in Figure 5.8. As the BER degrades further, good performance can be maintained by using smaller packets (since with a BER of 10^{-4}, the packet loss rate is 55% with 1000 byte packets). We did not compare STP with TCP in this case since TCP has no facility for explicit rate control.

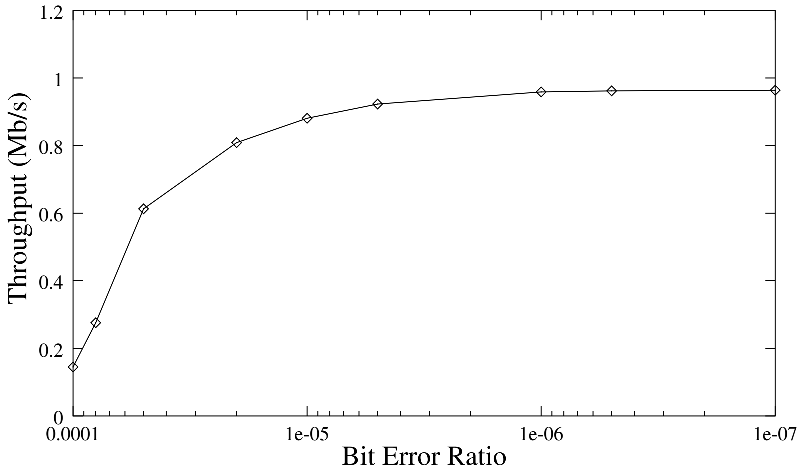

Figure 5.7: Simulation results of the forward throughput performance of STP on a 1 Mb/s channel with a variable BER.

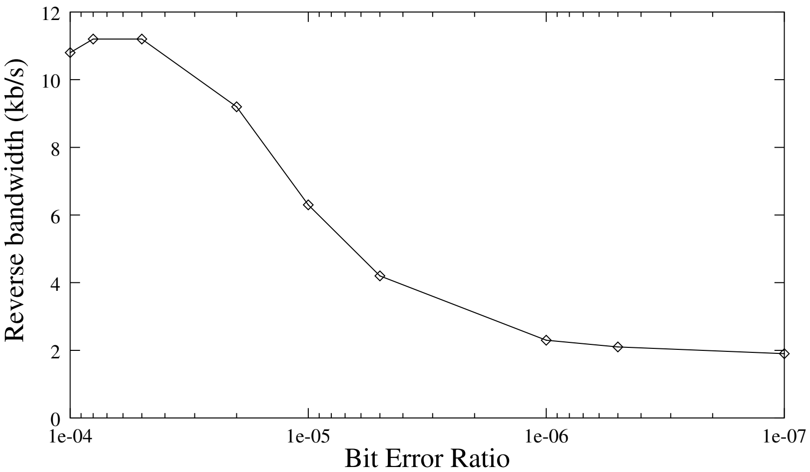

Figure 5.8: Reverse channel bandwidth requred for a large STP file transfer as a function of BER (simulation results corresponding to Figure 5.7).

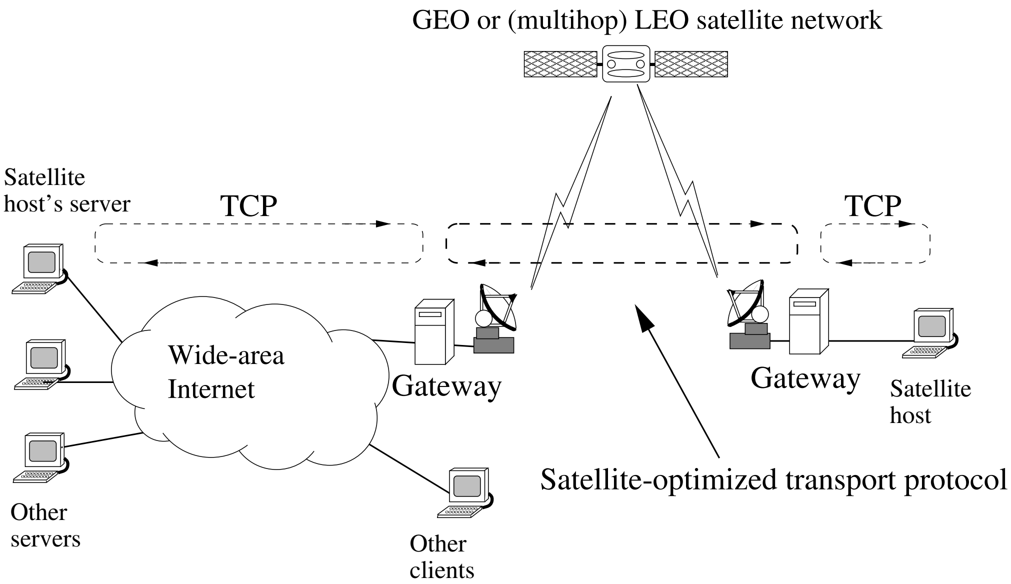

Figure 5.9: Future satellite networking topology in which a satellite-based host communicates with a server in the Internet through a satellite protocol gateway.

5.3 Experimental Results

In the previous chapter (Sections 4.2 and 4.3), we described a series of network experiments aimed at characterizing the performance of end-to-end and split TCP connections in a satellite environment. In this section, we continue that series of experiments with a look at the performance of STP under similar conditions. The reader is referred back to those sections for a description of the methodology used in those and the following experiments. This overall progression from analysis to simulation to experiment is part of our basic research methodology introduced in Chapter 3.

Figure 5.10 plots the difference in file-transfer performance between split STP and split TCP (SACK plus NewReno) when there is competing short-delay traffic in the wide-area Internet. For reference, we reproduce Figure 5.9 above, previously illustrated as Figure 4.22 in Chapter 4, as a description of the networking topology used in these experiments. 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. The TCP data is reproduced from Figure 4.23, discussed previously in Section 4.3. Figure 5.10 illustrates that STP achieves approximately the same forward throughput as TCP, because the forward throughput is primarily governed by the congestion avoidance policy. Again we found that 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 direc-

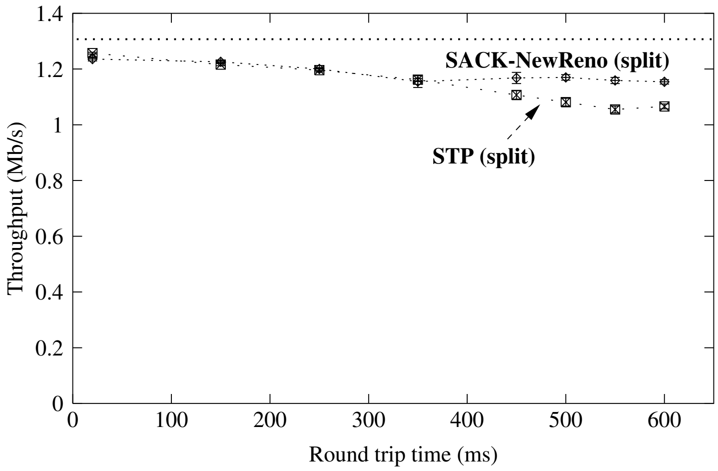

Figure 5.10: Comparison of forward throughput performance of split TCP and split STP. For fair comparison, both TCP and STP used identical congestion control policies.

tion 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, illustrated in Figure 5.11, 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 three times the estimated RTT of the connection. Under this configuration, therefore, the amount of return bandwidth required is roughly independent of the forward throughput. In both Figures 5.10 and 5.11, 95% confidence intervals are plotted as error bars on the data, although the error bars are difficult to notice because they are very small.

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 5.3 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 [79]. The data were collected from experiments on a local network in

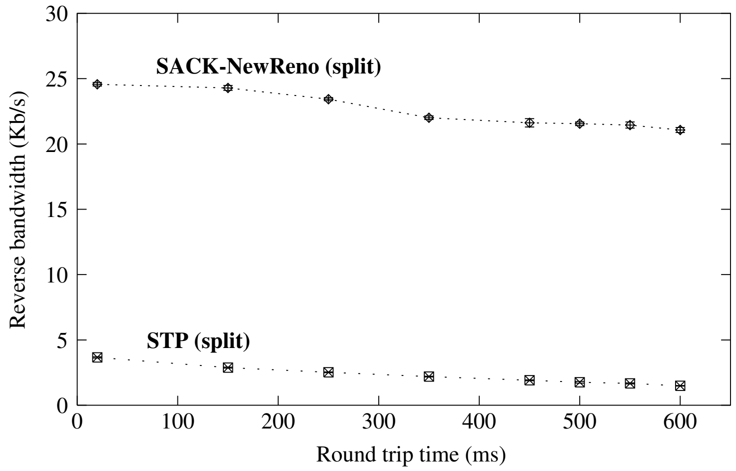

Figure 5.11: Comparison of reverse channel usage of split TCP and split STP, for the forward transfers illustrated in Figure 5.10.

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. 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. For these experiments, we we used the DirecPC satellite system and Ricochet packet radio networks (introduced in Section 3.3), 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 5.4 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.

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

Table 5.3: 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].

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

Table 5.4: 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.

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) [44] in packet radio networks [85].

5.4 Summary

In this chapter we have described the design and performance of STP, a satellite-optimized transport protocol that 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 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 satellite-friendly TCP options also require client-side changes; particularly those dealing with TCP asymmetry. Finally, since STP provides the same reliable byte-stream service as does TCP, STP can be used internally within a satellite network by applications that are written to use TCP.

We experimented with simulation models and UNIX kernel implementations of STP. A key requirement of our tests was that the protocol performance be measured in an environment containing other competing connections sharing portions of the same network path. Under the same window-based congestion control policy as used in TCP, we found that STP data transfers could obtain roughly the same forward throughput as similar TCP transfers while using up to an order of magnitude fewer bytes in the reverse direction; the difference was most pronounced for long file transfers. For short Web-like transfers, we found that STP could achieve a performance better than conventional TCP and approaching that of TCP for Transactions. Our simulation results also highlighted that STP is less sensitive to congestion on the reverse path, and illustrated good throughput performance in environments characterized by BERs as low as 10^{-4}.

This chapter concludes our investigation of transport protocol issues in a GEO satellite environment. In the next chapter, we turn our attention to the problem of designing packet routing protocols for LEO satellite networks.