TCP Extensions for Space Communications

Robert C. Durst[†], Gregory J. Miller[‡], and Eric J. Travis*

[†]The MITRE Corporation [*]Gemini Industries
1820 Dolley Madison Boulevard
McLean, VA 22102

[‡] MCI Telecommunications
2100 Reston Parkway
Reston, VA 20191
durst@mitre.org, gmiller@mci.net, travis@clark.net

Abstract

The space communication environment and mobile and wireless communication environments show many similarities when observed from the perspective of a transport protocol. Both types of environments exhibit loss caused by data corruption and link outage, in addition to congestion-related loss. The constraints imposed by the two environments are also similar — power, weight, and physical volume of equipment are scarce resources. Finally, it is not uncommon for communication channel data rates to be severely limited and highly asymmetric. We are working on solutions to these types of problems for space communication environments, and we believe that these solutions may be applicable to the mobile and wireless community. As part of our work, we have defined and implemented the Space Communications Protocol Standards-Transport Protocol (SCPS-TP), a set of extensions to TCP that address the problems that we have identified. The results of our performance tests, both in the laboratory and on actual satellites, indicate that the SCPS-TP extensions yield significant improvements in throughput over unmodified TCP on error-prone links. Additionally, the SCPS modifications significantly improve performance over links with highly asymmetric data rates.

1. Introduction

A standardized suite of space data communication protocol standards would be beneficial to both the military and civilian space communications user communities. Use of common protocols for data communications will increase interoperability and reduce the cost of space systems, which are often designed and custom-crafted for every mission or set of missions. In developing a standard protocol suite for use in space communications, it is natural to draw from the long-standing, successful, and robust Internet protocols, TCP and IP. However, there are significant differences between the space environment and that of typical terrestrial communications. While TCP/IP has already been deployed with favorable results in a plethora of diverse environments, from low data-rate satellite links to OC-12 fiber optic lines, success in the range of space missions anticipated requires certain extensions to address the constraints imposed by the hostile space environment.

This work was supported by the U. S. Government under Contracts DAAB0796-C-E601, F19628-94-C-0001, and NAS532607.

Working jointly with NASA and the U. S. DoD, we have designed, specified, implemented, and are testing a set of protocols for use in space data communications, known as Space Communications Protocol Standards (SCPS). We are developing four SCPS protocols: a file transfer protocol, a transport protocol, a security protocol, and a network protocol. In this paper, we discuss only the transport protocol, SCPS-TP, which is actually TCP with a set of extensions and modifications to improve operation in the space environment. We believe that many of these enhancements may also prove useful to the more general mobile communications community.

The space environment poses a number of challenges in providing reliable, end-to-end data communication with a tolerable level of service. Large propagation delays, limited bandwidth, losses due to errors, asymmetric link capacities, and intermittent connectivity all conspire to limit TCP's performance severely. In many cases, TCP can cope with a subset of these environmental obstacles, although the performance achieved is far from optimal. The set of TCP extensions that comprise SCPS-TP improves performance in the space and mobile environment. Some of these extensions are changes to the TCP specification, while others are implementation details that do not affect interoperability. We have adopted the RFC 1323 Timestamps and Window Scaling options [JBB92] and propose our own Selective Negative Acknowledgment option. We also adopt the TCP Vegas [BOP94] low-loss congestion control mechanisms; however, we invoke congestion control only when we have evidence that losses are a result of network congestion rather than bit errors. Also, rate control optionally replaces the standard TCP ACK-clocking for operation environments with highly asymmetric bandwidth. Section 2 elaborates on the communications problems encountered in the space environment that motivated our work. Section 3 describes the SCPS TCP extensions that address these problems. Section 4 relates SCPS-TP to other proposed schemes for improving TCP performance over wireless links. Section 5 briefly describes our prototype SCPS-TP implementation. Section 6 presents performance test results of the SCPS-TP protocol operating in both simulated and actual space environments. Finally, section 7 concludes with recommendations and directions for future work.

2. Communications Problems

The space environment in general offers a number of impediments to reliable data communications. We use the term "space" generically to refer to a wide range of environments in which one communication endpoint is on Earth and the other is on a spacecraft. However, we draw parallels to the wireless and cellular environments where applicable because in many ways, the problem domain for wireless communications is similar to that of the generic space environment. The spacecraft that we consider range from Low Earth Orbiting (LEO) satellites to interplanetary deep-space probes. In the remainder of this section, we describe the shared traits of the mobile and space environments that affect data communications along with a description of their specific impact on the TCP protocol. This list includes a comprehensive set of features, any single one of which may not be present in a particular environment or operating scenario.

2.1 Error-Prone Links

In wireless communications in general, but especially in space communications, bit-errors caused by noise are not uncommon. To some degree, forward error correction can compensate for such errors – trading bandwidth for effective data rate, but the space link still is rarely as clean as those of modern terrestrial networks. TCP is designed to handle packet loss by identifying and retransmitting lost segments; however, TCP assumes the source of all packet loss is network congestion. Consequently, TCP invokes congestion control, reduces its congestion window, and in turn, its transmission rate as a result of any packet loss [Jac88]. As has been pointed out previously [Fox89, CI95, BSAK95], this response is inappropriate when faced with loss due to corruption rather than congestion, as is often the case on space and other mobile communications links. TCP's congestion control algorithm works well in dealing with congestion-induced loss, but only results in reduced throughput on uncongested, noisy links without providing any benefits.

2.2 Asymmetric Channels

Communications channels between spacecraft and the ground are frequently asymmetric in terms of both channel capacity and error characteristics. Often the forward link bandwidth (from the spacecraft to the ground) is substantially larger than the return link bandwidth, with ratios of 1000:1 not uncommon. This asymmetry is a result of various engineering tradeoffs (such as power, mass, and volume), as well as the fact that for scientific missions, most of the data originates at the satellite and flows to the ground. The return link is generally used for commanding the spacecraft, not bulk data transfer. While such high asymmetries are not as common in the wireless/cellular environments, reducing the required acknowledgment channel bandwidth is still a desirable goal since it reduces the energy emissions of a mobile unit when it is only receiving data. Other scenarios involving channel asymmetry, although non-mobile, are cable modem networks and direct broadcast satellite networks, which provide high-bandwidth data pipes to homes and which use limited-capacity telephone lines as return channels.

Bandwidth asymmetry can limit TCP's throughput even when the data are flowing on the larger channel and the smaller channel is carrying only acknowledgments. A receiving TCP endpoint generally acknowledges every other segment, which dictates an acknowledgment channel capacity that is proportional to the data channel capacity and is a function of the segment size. With a 1024-byte segment size, TCP throughput is relatively unhindered by a forward-link to return-link bandwidth ratio of less than 50:1; however, throughput performance is limited by the acknowledgment channel capacity at higher ratios.

2.3 Limited Link Capacity

Wireless communications channels tend to offer less available bandwidth than wired networks. In the mobile and space arenas, this problem is coupled with the constraint that power is limited and bit-efficiency is important in terms of the cost of transmitting as well as in terms of link capacity.

There is substantial bit-overhead associated with the TCP protocol, especially when using small segments to increase the probability of successfully transmitting a packet without incurring a bit-error. This overhead, at least 20 bytes of TCP header per packet, can consume a sizable share of a limited-bandwidth channel.

2.4 Intermittent Connectivity

For satellites in non-geostationary orbits, connectivity on a given communications link is usually intermittent. Contact may be interrupted for a number of reasons, including ground station handovers, changing network topology, antenna obscurations, weather, and orbital dynamics. When a satellite transitions from the visibility of one ground-station to another, the behavior is similar to a cellular hand-off between two base stations, although the handover time may be much longer.

In addition, complex, dynamic topologies (i.e., communications among and through large satellite constellations) are likely to exist in future space communication scenarios, as opposed to the simpler space-to-ground model prevalent today. In a dynamically changing network, when an intermediate system, either a fixed base station or a satellite, hands off a mobile user to another intermediate system, connectivity may be temporarily interrupted. While highly dynamic network connectivity is rare in current wireless and cellular networks, the possibility for mobile base-stations exists in rapidly-deployable military or disaster-relief networks, as well as ad-hoc networking of wireless devices. Also, with increasing frequency there are incidences of what can appear to be dynamic network topology in terrestrial networks, such as the Internet, as a result of routing pathologies [Pax96].

Even short-term link outages pose a problem for TCP, resulting in poor throughput in the best case and aborted connections in the worst [CI95]. In the absence of a steady flow of acknowledgments, TCP will invoke congestion control and repeatedly retransmit and back-off its retransmission timer. If connectivity is restored before TCP exceeds its maximum retransmission threshold, it will resume transmission where it left off. However, by the time the link is restored, the retransmission timer may be backed-off so far that minutes elapse before TCP recognizes it. If, on the other hand, TCP reaches its maximum retransmission threshold before connectivity is restored, the connection is aborted.

Another potential effect of a changing network topology is a wide fluctuation of measured round-trip delays due to increases or decreases in path hop-count and propagation distances. TCP can retransmit either too early or too late if its round trip time estimate is sufficiently far from the current actual round trip time. Premature retransmission timeouts result in unnecessary invocations of slow start, reducing throughput.

3. Proposed Solutions

The SCPS-TP protocol consists of standard TCP augmented by a set of extensions and enhancements that consist of both implementation and specification changes. These modifications each respond to requirements derived from the characteristics of the space environment described above. Some of these TCP modifications have been proposed elsewhere in the literature; however, they targeted environments different from the one we study here, for instance, high-speed networks. We describe the list of TCP enhancements that are incorporated into SCPS-TP below. The enhancements are grouped according to the environment-induced problem they address.

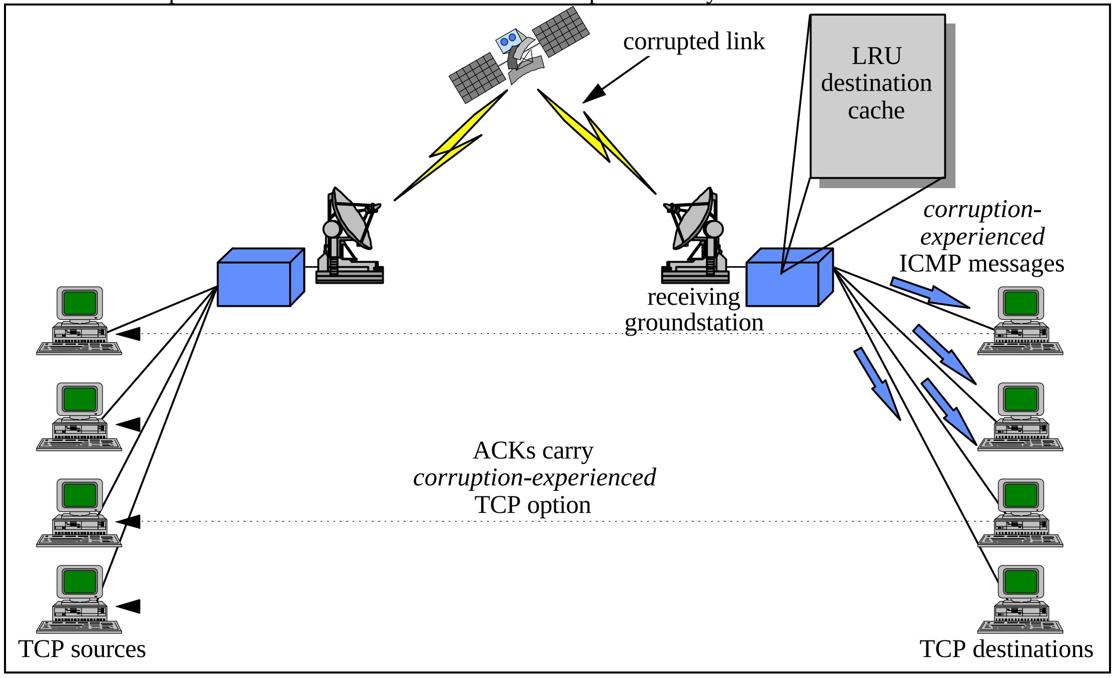

Figure 1: Corruption Signaling.

We note that some of the SCPS-TP extensions affect interoperability with regular TCP. Therefore, the use of the non-standard SCPS-TP options and behaviors is negotiated on connection establishment via a "SCPS-TP capable" option. If the peer TCP endpoint does not return the "SCPS-TP capable" option when it is sent by the connection initiator on the SYN segment, SCPS-TP will behave like regular TCP.

3.1 Coping with Different Sources of Loss

A key to optimizing TCP throughput performance is to identify the source of packet loss and react appropriately. At least three sources of loss exist in the space and mobile environments: network congestion, corruption, and link outage. The appropriate response to each of these types of loss is different, and SCPS-TP implements a distinct response for each. SCPS-TP has two mechanisms for determining the source of packet loss. Like standard TCP, SCPS-TP makes a default assumption regarding the source of loss in the absence of any explicit information. TCP's default assumption is that all loss is caused by congestion; however, SCPS-TP's default is a parameter that can be set by a network manager or an application on a per route basis. In some circumstances, where network bandwidth is carefully managed on private links, congestion is unlikely and it is reasonable for SCPS-TP to assume by default that any loss is due to errors.

The second mechanism for identifying the source of loss is explicit signaling. The destination host or other network elements (routers or groundstations) may send explicit signals to the sending TCP regarding the source of packet loss.

3.1.1 Congestion-Induced Loss

When congestion control is enabled, SCPS-TP uses the TCP Vegas congestion control algorithms [BOP94, DLY95] to minimize loss and facilitate the use of large windows. To achieve optimal throughput performance, TCP depends on having the receive window tuned to a value greater than the network bandwidth-delay product, but small enough to prevent congesting the network [VS94]. The TCP Vegas congestion control algorithms eliminate the need to tune the receive window to serve as an upper bound on the size of the congestion window. TCP Vegas' congestion control bounds itself, self-tunes, and avoids network congestion without overdriving the link to find the saturation point, even when operating with large windows. Window sizes larger than the bandwidth-delay product are desirable when errors are present to decrease the probability of draining the pipe during loss recovery. Also, on space links, where it is expensive to recover from losses because of long delays and limited power, it is especially desirable to find an optimal operating point without repeatedly inducing loss the way standard TCP does. TCP Vegas attempts to satisfy this requirement by increasing its congestion window more slowly than standard TCP and by measuring the achieved throughput gain after each increase to detect the available capacity without incurring loss.

SCPS-TP uses the TCP Vegas variant of the Slow Start algorithm [Jac88], which doubles the congestion window every other round-trip time as opposed to every round-trip like standard TCP. SCPS-TP modifies this algorithm by providing an additional trigger for transitioning from the congestion window's exponential-growth phase into the linear-growth congestion avoidance phase. SCPS-TP enters congestion avoidance based on TCP Vegas' specified technique for sensing the point at which the throughput increase tapers. In addition, SCPS-TP will enter congestion avoidance once the congestion window size reaches the bandwidth-delay product of the network. (Note that an equivalent feature is documented by J. Hoe in [Hoe96].) The network bandwidth-delay product for a connection is a supplied parameter, as is the receive window size in TCP. TCP Vegas' congestion avoidance mechanism continues to measure the achieved throughput and adjust the congestion window size accordingly throughout the course of the connection.

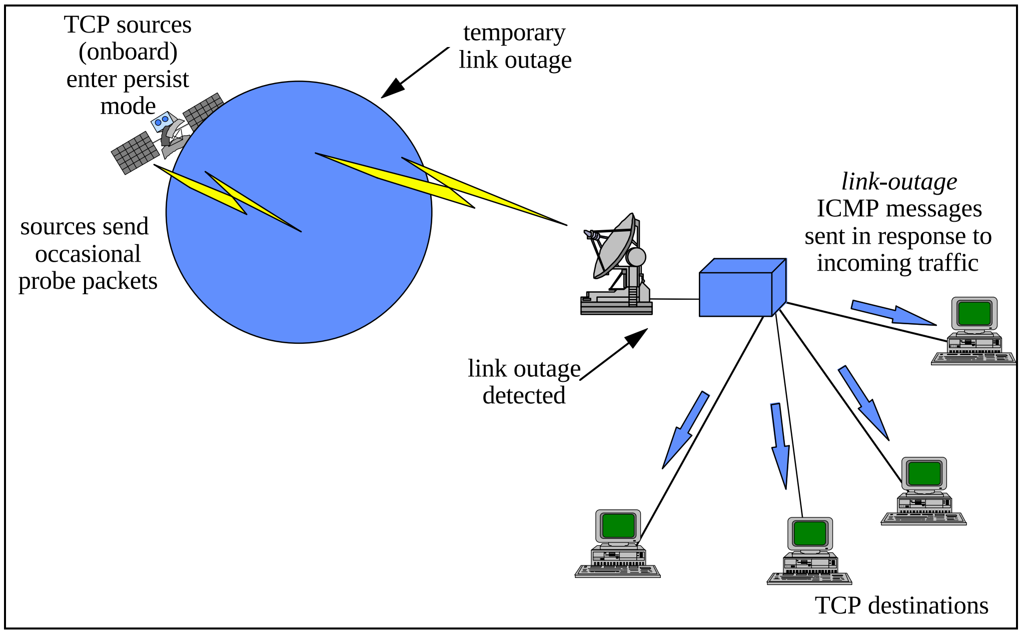
TCP sources (onboard) enter persist mode
temporary link outage
sources send occasional probe packets
link-outage ICMP messages sent in response to incoming traffic
link outage detected
TCP destinations

Figure 2: Link Outage Signaling.

Congestion control is invoked in response to packet loss whenever congestion is set to be the default assumption regarding the source of loss and whenever an Explicit Congestion Notification (ECN) message is received. Any intermediate router or the destination host can send an explicit congestion notification signal to the sending SCPS-TP, either with a Source Quench ICMP message or with an ECN field in a packet header, as described in [Flo94]. The rules for generating ECNs are addressed in [Flo94] as are proposed rules for responding to them. An explicit signal regarding the source of loss will always override SCPS-TP's default assumption.

Note that the use of the Source Quench form of ECN has distinct advantages when the congested router is on the data sender's side of a long-latency hop (such as a satellite link). The Source Quench message reaches the data sender without incurring the two long-latency hops required for acknowledgment-based signals from the destination host to be returned. This reduced delay in congestion feedback helps the data source to respond to the onset of congestion before the node is driven to significant loss.

Note also that TCP Vegas is robust against the possible loss of an ECN message. If the assumed source of data loss is not congestion, loss of an ECN can cause congestion to go unchecked. TCP Vegas does not depend on loss as an indication of congestion, so even when the default source of loss is corruption, TCP Vegas still controls congestion, using whatever ECN messages are received as additional information. Our use of TCP Vegas presents a number of unresolved problems, as we summarize in section 7. However, the fact that it does not depend on loss as a signal of congestion is one of its promising features.

3.1.2 Corruption-Induced Loss

On space links, it is often the case that bit-errors are the primary cause of packet loss, and that congestion is unlikely based on careful scheduling of the link use. In these cases, SCPS-TP disables its default assumption that congestion is the source of loss and does not invoke congestion control in response to packet loss. To keep from over-running the link capacity, SCPS-TP uses an open-loop, token bucket rate control mechanism [Par94] that meters out transmissions at a specified rate. At each endpoint, the allowed rate for each link is a managed parameter that is stored in the globally-accessible routing structure. On a host, the available capacity for a particular link is shared among all SCPS-TP connections using that link.

We identify four phases to corruption handling: identification of onset, signaling, response, and identification of termination. The identification of the onset of corruption on the downlink is performed by the groundstation at the receiving end of the space link. This receiver can request that the link layer pass up information regarding the number of packets received that can be demultiplexed but are in error (i.e., the CRC failed). The receiver maintains a weighted moving average of the number of corrupted packets received and transitions into a *link-corrupted* state when the average exceeds a threshold. The receiving network layer maintains an aged LRU cache of destinations to which it has forwarded packets from the "corruptible" link. When the receiver transitions into the link-corrupted state, it begins sending *corruption-experienced* ICMP messages to the destinations contained in the cache (see Figure 1). The destinations each inform their respective SCPS-TP sources about the corrupted link via a TCP option on the acknowledgment segments. The destination SCPS-TP remains in the link-corrupted state for approximately 2 x RTT unless the state information is refreshed by an ICMP message.

While a *corruption-experienced* option is present on incoming acknowledgment segments, the sending SCPS-TP does not invoke congestion control in response to packet loss. That is, the sender does not reduce the congestion window and does not back-off the retransmission timer. This response to packet loss, which is detected in the usual manner from duplicate acknowledgments and time-outs, continues until an acknowledgment is received without a *corruption-experienced* option, at which time the default response takes over.

The network layer on the receiving side of the corrupted link continues to generate *corruption-experienced* ICMP messages-based on arriving traffic while the moving average indicates that corruption is present on the link.

3.1.3 Link Outage

A link outage is a transient loss of connectivity, resulting from a satellite temporarily passing out of view of the groundstation, a handover in a mobile network with dynamic topological changes, or other short-term interruptions. In our discussion, we assume that when a link is out in one direction, it is out in the other direction as well. We consider the same four aspects in dealing with link outages as we did for corruption. The mechanism for identification of the onset of a link outage is link dependent. In general, a link outage may be identified at the groundstation by loss of carrier lock or the received signal strength falling below a threshold. Once the groundstation (or spacecraft) detects the link outage, it sends a *link-outage* ICMP message to any host on its own side of the severed link from which it receives traffic (see Figure 2). The ICMP message is triggered by incoming traffic and contains the TCP header of the packet that caused the message to be generated. The sending SCPS-TP's response to a link outage signal is to enter persist mode, sending periodic probe packets. SCPS-TP does not repeatedly time-out, retransmit, and back-off the retransmission timer; instead it suspends its timers and ceases transmitting, except for the occasional probe packets. SCPS-TP exits persist state when it receives a *link-restored* ICMP message from the groundstation, or when one of the probes gets through and is acknowledged. When the link is restored, the sending SCPS-TP can infer where in the data stream to resume transmission from the sequence number in the TCP header carried in the original *link-outage* ICMP message.

3.2 Coping with Asymmetric Channels

TCP relies on a stream of acknowledgments to clock out data continuously from the sender. Communication links between spacecraft and the ground are typically configured with a high-rate channel in one direction and a much lower rate channel in the opposite direction: these are typically used for telemetry and commands respectively. If a satellite is reliably transmitting data over the telemetry channel, the volume of TCP acknowledgment traffic generated can easily overrun the reverse channel carrying the acknowledgments. To overcome this problem, SCPS-TP removes the requirement that TCP acknowledge at least every other segment received [Bra89]. SCPS-TP also relaxes the requirement that TCP immediately acknowledge every segment received when an out-of-sequence queue exists [Bra89]. Instead, the SCPS-TP data-receiver delays acknowledgments for a configurable period of time that is related to its estimate of the RTT.[1] We are currently experimenting with acknowledgment frequencies ranging from once per RTT up to 8 per RTT, but have not determined an optimal value yet.

Such drastically delayed acknowledgments impact TCP in several ways. Depending on the particular acknowledgment frequency, retransmission timer value, and window sizes being used, the Fast Retransmit algorithm, through which the sender detects loss by counting duplicate acknowledgment segments [Ste94, Ste96], may essentially be disabled. Also, in the absence of a regular acknowledgment-driven clock, TCP must use another mechanism to meter out data smoothly at the sender. The open-loop rate control mechanism described in section 3.1.2 serves this purpose by regulating the transmission of packets to a specified rate and by limiting the burst sizes. Finally, there is a problem with using a fixed ACK frequency as SCPS-TP does rather than an ACK frequency driven by the data transmission rate like TCP's. When the link is shared by multiple connections and the ACK channel becomes congested, TCP's congestion control will reduce the data transmission rate, and consequently, the acknowledgment rate. In contrast, SCPS-TP will continue to ACK at a fixed rate on a congested ACK channel, even if congestion control reduces the data transmission rate. We are currently studying solutions to address this problem, one of which involves triggering acknowledgments by the receipt of at least some threshold amount of data. In such a case, the delayed ACK timer would still ensure that ACKs are emitted regularly on an interactive data flow that does not send large enough blocks of data to trigger an ACK.

A second technique that aids throughput performance on asymmetric channels is SCPS-TP header compression, which is described below. Header compression can significantly reduce the overhead on the acknowledgment channel and allow higher ACK rates.

3.3 Coping with Limited Link Capacity

SCPS-TP uses two mechanisms to improve performance in bandwidth-constrained environments: SCPS-TP Header Compression and the SCPS-TP Selective Negative Acknowledgment (SNACK) option. These two mechanisms are discussed in the following subsections.

3.3.1 SCPS-TP Header Compression

The primary approach SCPS-TP takes to dealing with limited bandwidth is to use end-to-end TCP header compression. SCPS-TP header compression differs from the TCP/IP header compression scheme that is specified in RFC 1144 [Jac90], which was designed for use on low-speed serial links. RFC 1144 TCP/IP header compression is performed on a hop-by-hop basis at the link layer, where connection state tables are maintained for inbound and outbound TCP/IP connections. The state for each connection consists of the last uncompressed TCP and IP headers sent (outbound) or received (inbound) on that connection. The compressor initializes by allocating a connection identifier for the connection and saving the first TCP and IP headers sent. Subsequent headers are constructed by sending only the changes from the previous headers. For example, the sequence number, acknowledgment number, urgent pointer, and window field are sent as changes to the previous values. At the receiving side, uncompressed headers are created by applying the changes contained in the newly received compressed header to the saved header. Copies of the new headers are then saved in the connection state table as the new compression state, and the uncompressed headers are forwarded to the destination with the data. If a segment is lost or corrupted, an invalid uncompressed header will be created when the incorrect changes are applied to the stored compression state. This invalid header will be detected when the TCP checksum fails, and the packet will then be discarded. In such a case when the compression state becomes invalid, all packets arriving after the lost or corrupted packet will be decompressed improperly, causing them to be discarded. Compression state is resynchronized when the TCP source eventually retransmits the original lost or corrupted packet. Retransmissions are sent uncompressed to facilitate resynchronization.

[1] The SCPS-TP data-receiver can accurately estimate the RTT if the TCP Timestamps option is being used on the connection. Otherwise, calculating the RTT is more difficult. We are exploring mechanisms for estimating the RTT in this case.

RFC 1144 TCP/IP header compression performs well on serial links with short delays and low error rates, such as telephone lines, but it is inadequate for the space or mobile environment for two reasons. Because it is performed at the link layer, the compression algorithm must be restarted whenever connectivity changes, causing all packets in flight to be lost. RFC 1144 header compression is also sensitive to corruption, loss, and misordering since it requires the compressor and decompressor to resynchronize whenever a packet is lost or arrives out of order. Synchronization is expensive, especially when operating with large windows or over long delays, as it causes go-back-*n* behavior. The go-back-*n* behavior is a result of the delta encoding that the compressor uses, which dictates that the decompression of a particular packet depends upon the successful decompression of the preceding packet. When a single packet is lost, corrupted, or misordered, none of the packets that follow it can be decompressed. These invalid packets, up to one bandwidth-delay product's worth, cannot be buffered by the receiver, so they are lost.

We have developed a loss-tolerant TCP header compression scheme that operates end-to-end, at the transport layer, and can tolerate loss and changing connectivity. SCPS-TP header compression reduces the size of TCP headers by approximately 50%, which is a substantial savings for pure acknowledgment segments. It works by summarizing information that is static for the duration of the connection and by omitting information that is not relevant to the segment being sent. For instance, it assigns a connection identifier to replace the port numbers and it omits fields such as the urgent field if the URG flag is not set. The actual compressed SCPS-TP header is variable in length, but always contains a connection identifier, a bit-field indicating what optional fields are present and what flags are set, and a checksum. Unlike RFC 1144 TCP/IP header compression, SCPS-TP header compression does not operate at the link layer, it does not use delta encoding, and it compresses only the TCP header, not the IP header. By operating end-to-end, SCPS-TP header compression avoids the problems caused by changing connectivity, since the endpoints, where the compressor and decompressor reside, never change. SCPS-TP header compression addresses the problems introduced by misordering and packet loss by ensuring that each packet is decompressible independently of preceding packets. This way, TCP's usual resequencing mechanisms can be used to avoid go-back-*n* behavior in the event of loss.

The use of SCPS-TP header compression is negotiated using an option on the uncompressed SYN and SYN/ACK segments that open a connection. Note that when SCPS-TP header compression is being used, a different protocol number than TCP's (105) is used in the IP header to identify the encapsulated transport protocol.

3.3.2 The SCPS-TP SNACK Option

A second extension that improves both utilization of limited bandwidth and loss recovery is the SCPS-TP Selective Negative Acknowledgment (SNACK) option. This option allows the receiver to inform the SCPS-TP sender about one or more holes in the receiver's out-of-sequence queue. Without a selective acknowledgment, TCP can use the ACK number to identify at most a single hole in the receiver's buffer. Using its simple cumulative acknowledgment and the Fast Retransmit algorithm [Ste94], TCP can recover efficiently from a single loss per window. However, because new data must be received for the receiver to advance the ACK number, TCP requires a minimum of one RTT to signal *each* additional hole in the out-of-sequence queue. The SNACK option, which is carried on an acknowledgment segment, identifies multiple holes in the sequence space buffered by the receiver. By providing more information about lost segments more quickly, the SNACK option can hasten recovery and prevent the sender from becoming window-limited, thus allowing the pipe to drain while waiting to learn about lost segments. The ability to transmit continuously in the presence of packet loss is especially important when loss is caused by corruption rather than congestion. In such a case, when it has been deemed that it is appropriate to disable congestion control as the response to loss, SNACK is of particular benefit in keeping the pipe full and allowing transmission to continue at full throttle while recovering from losses.

The SCPS-TP SNACK option draws from both the TCP Selective Acknowledgment (SACK) option [MMFR96] and the TCP Negative Acknowledgment (NAK) option [Fox89], which was proposed in 1989, but never adopted as a standard. Both of these options address the problem described above, in which throughput suffers when TCP experiences more than one segment loss in a window. We briefly describe these options in the following paragraphs, explain why they are unsuitable in the space/mobile environment, and finally describe the SCPS-TP SNACK option.

The SACK option has recently been approved as a Proposed Standard within the IETF as RFC 2018, and is expected to enjoy wide deployment. The new SACK option is a refinement of the one that was originally proposed in RFC 1072 [JB88], but later deferred pending further study. The RFC 1072 SACK option contains advisory information regarding non-contiguous blocks of data that have been received and queued by the receiver. The SACK option does not change the interpretation of the acknowledgment number in the basic TCP header. The intention is that the data-sender will optimize retransmissions based on the additional information provided by the SACK option. The option itself consists of a variable-length list of pairs of 16-bit integers. Each pair defines a block of sequence space that corresponds to data that is being buffered by the receiver. A pair consists of a *relative origin* and a *block size* for each isolated, contiguous block of received data. The relative origin specifies the first sequence number in a block, as an offset from the acknowledgment number in the TCP header, and the block size specifies the size in bytes of the block of data. The RFC 1072 SACK option was never standardized primarily because it is incompatible with the Window Scaling option [JBB92]. Using its 16-bit relative origin field, it is incapable of referencing more than the first 64 kbytes of the potentially large scaled window without losing precision by resorting to a solution that scales the fields in the SACK option.

The RFC 2018 SACK corrects this shortcoming of the RFC 1072 SACK by using full 32-bit sequence numbers to identify the start and end of each SACK block, rather than an offset and a size. Furthermore, the new SACK specification clarifies the sender and receiver behavior with regard to the SACK option, which had been incompletely specified in RFC 1072. The RFC 2018 specification dictates that when the data-receiver decides to send a SACK option, the first SACK block in the list must identify the most-recently-received block of data. The remaining blocks in the list, whose length is dictated by the available space in the TCP header, are SACK blocks that may have already been SACKed. This means that, as duplicate acknowledgment segments carrying the SACK option are sent in response to incoming data segments, each SACK option first specifies the block of data containing the segment that triggered that particular acknowledgment. This way, every received block will be SACKed at least three times, depending on the space available for options, and that the data-sender always gets the most up-to-date information about the state of the receiver's queue. The data-sender, upon receiving a SACK option, simply marks the specified segments as *SACKed*. Later, if a retransmission is deemed necessary through receipt of duplicate acknowledgments, any segments tagged as *SACKed* are skipped over during retransmission. If the retransmit timer expires, all *SACKed* flags are reset in case the receiver has reneged on previously sent SACKs. TCP's loss detection and congestion control algorithms remain unchanged by the use of SACK.

The use of the SACK option poses two difficulties in the space and mobile environment. The SACK option was motivated through work in the high-speed networking arena, and as a result, it is not particularly bit-efficient. In a bandwidth-constrained environment, the use of two 32-bit sequence numbers to specify a single SACK block is costly. In addition, because a TCP header is limited to carrying at most 40 bytes of options, the maximum number of SACK blocks that can be signaled in a single SACK option is three when the TCP Timestamps option [JBB92] is also being used. Such a limitation on the number of SACK blocks per acknowledgment segment may be too restrictive in the space environment. Especially in deep space, window sizes will be large, allowing a large number of segment losses per window. If acknowledgments are sent infrequently because of limited acknowledgment channel bandwidth, it may difficult to signal all of the isolated blocks in the receiver's queue with the ability to specify only three at a time. The second problem with using SACK in an environment with an asymmetric channel is that it relies on the Fast Retransmit algorithm to detect lost segments and trigger retransmission. When the acknowledgment channel bandwidth is highly constrained, SCPS-TP's acknowledgment frequency may be tuned so that the data-sender will never receive enough duplicate acknowledgments to trigger a Fast Retransmit. In such a case, SACK will provide no benefit since the SACK information is cleared when the retransmit timer expires, which could be the only means for detecting loss.

The other TCP option that has been proposed to help improve loss recovery is the NAK option. Unlike SACK, this option was motivated through work in the satellite environment, so it is more bit-efficient than SACK. The NAK is a negative acknowledgment, rather than a positive acknowledgment. It allows the data-receiver to report a block of data that was not received and potentially needs to be retransmitted. Like SACK, the NAK option does not alter the meaning of the regular acknowledgment in the TCP header, but simply provides additional information that can be ignored without affecting TCP's current behavior. The NAK option contains the sequence number of the first byte being NAKed and the number of consecutive maximum-segment-sized blocks being NAKed. When the data-sending TCP receives a NAK, it may elect to retransmit the NAKed segments immediately or it may ignore the NAK information.

The main problem with the NAK option for use in long-delay environments is that it has the ability to signal only a single hole in the sequence space buffered by the receiver. As mentioned above, a more powerful mechanism, capable of specifying multiple holes is desirable in the noisy, long-delay environment to provide the sender with more information about the state of the receiver's potentially large out-of-sequence queue. It is especially important that the acknowledgment mechanism convey as much information as possible in the case when the acknowledgment frequency is significantly reduced, since there are fewer opportunities to send acknowledgment information.

The SCPS-TP SNACK option borrows ideas from both SACK and NAK. The SNACK is a negative acknowledgment, like NAK, but it is capable of specifying a large number of holes in a bit-efficient manner. It is a variable-length option that consists of 5 fields: the kind and length fields required of all TCP options, followed by the mandatory *offset* and *length* fields (each 16 bits long) and an optional variable-length bit-vector. Offset specifies the displacement from the ACK number carried in the regular TCP header to the starting location of the first hole that is being signaled by this particular SNACK option. The length field specifies the size of this hole. Note that this hole is not necessarily the first hole in the receiver's overall out-of-sequence queue. Both the offset and length values are expressed in Maximum Segment Size (MSS) units. The bit-vector then signals zero or more additional holes, also expressed in terms of MSS-sized blocks. The bit-vector maps the sequence space of the receiver's buffer beginning one byte beyond the end of the block specified by offset. Each "0" in the bit-vector signifies that one or more bytes are missing in the corresponding MSS-size block of the receiver's resequencing queue. The length of the bit-vector, which may vary at the SNACK-sender's discretion, is determined from the option length.

While an out-of-sequence queue exists, the data-receiver scans its receive buffer, forming SNACK options and sending them on outgoing ACK segments. By setting the offset field to zero, the data-receiver can NAK a block beginning at the ACK number in the TCP header. Alternatively, by specifying a non-zero offset value, the SNACK option can begin by addressing any arbitrary portion of the sequence space. The latter capability is especially useful when the out-of-sequence queue is large, even in terms of MSS units, and a single SNACK option is unable to reference the entire sequence space because of the limit on the size of the TCP header. In such a case, multiple SNACK options can be sent with each continuing to specify holes in the receive buffer where the last left off.

Upon receipt of a SNACK option, the data-sender immediately retransmits all segments necessary to fill the signaled holes. The SNACK is a request for retransmission. This behavior is similar to that dictated by the RFC-1106 NAK option, in contrast to the SACK recommendation. Because acknowledgments, and hence SNACK options, are sent infrequently, there is limited danger of unnecessarily retransmitting a delayed or misordered segment. Note that because the SNACK option triggers a retransmission, there is no reliance on the Fast Retransmit algorithm to detect loss. This independence from the Fast Retransmit algorithm is important since duplicate ACKs may never be received when operating over asymmetric channels with SCPS-TP's acknowledgment frequency set low. In such a case, the retransmission timer would be the only mechanism available for detecting loss.

3.4 Other Techniques for Coping with Errors

SCPS-TP employs two TCP options specified in RFC 1323 [JBB92] that improve performance in the long-delay, noisy environment. The TCP Timestamps option helps TCP make accurate RTT estimates in the face of loss, when it can be difficult to time the round-trip of particular segments that may be lost and subsequently retransmitted. SCPS-TP also uses the TCP Window Scaling option, which enables window sizes larger than 64 kbytes to be used. In an environment in which loss is caused by corruption, operating with a window that is larger than the bandwidth-delay product is beneficial so that the source is able to continue transmitting new data while recovering from losses. When losses are corruption-induced, SCPS-TP can maintain its transmission rate in the event of loss. A large window will allow the sender to continuously send new data while retransmitting lost segments, even as the left edge of the window does not advance for periods of time.

4. How SCPS-TP Relates To Other Approaches

In the research community, there is substantial work in progress to improve the performance of TCP over wireless links. Good, critical discussion of several approaches that have recently been proposed can be found in [BSAK95] and [BPSK96]. Two of the promising solutions are the Snoop protocol [BSAK95] and Indirect-TCP (I-TCP) [BB94]. It appears that the use of SCPS-TP could compliment either of these approaches; however, more study is needed to understand all of the potential protocol interactions.

The Snoop protocol provides performance improvements by introducing an agent at the groundstation that monitors passing TCP traffic and caches packets until they are acknowledged. The agent is able to detect lost segments through the receipt of duplicate ACKs or by a local time-out. When the agent detects a packet loss on the wireless link, it retransmits the lost packet and intercepts the duplicate ACKs to prevent the TCP sender from invoking congestion control. Hence, losses on the wireless link are transparent to the TCP sender and are handled independently of congestion-related loss elsewhere in the network. SCPS-TP is compatible with this approach; however, the Snoop agent may have difficulty detecting loss based on duplicate ACKs if SCPS-TP's acknowledgment frequency is configured to be low. Additionally, approaches that operate at the link layer, like Snoop, cannot work at all if end-to-end encryption mechanisms that obscure the transport headers are in use.

I-TCP advocates splitting the TCP connection between the two TCP endpoints into separate connections at the groundstation or base station. The groundstation then communicates with the fixed host on the wired network using regular TCP; however, a different protocol, tuned for the wireless environment, can be used to improve performance on the final wireless link to the mobile host. In this way, congestion control on the wired network is separated from error recovery on the wireless link. Again, SCPS-TP is compatible with this approach, and in fact, should be well-suited for use as the protocol on the single-hop wireless link. Additionally, as multi-hop wireless networks evolve, SCPS-TP can address the need for a transport protocol suited to this environment.

5. Prototype Implementation

We have developed a prototype implementation of SCPS-TP that runs as a user process on a Unix workstation. The transport protocol, the application, and a thread scheduler are all compiled and linked together into a single executable program. The protocol engine and the application program are each implemented as threads and the thread scheduler provides context-switching between the two. The SCPS-TP segments are encapsulated in raw IP packets and are sent over an Ethernet on our testbed. We have developed two simple applications for performance testing and debugging, a data-transmitting SCPS-TP client and a data-receiving server.

We also have developed a program, known as Spanner, that runs on a third workstation on the Ethernet testbed and simulates a satellite channel in the laboratory. The client and server workstations are configured to communicate with each other using the Spanner machine as an intermediate router. Spanner accepts packets from the Ethernet and forwards them to their destination at the configured bit-rate after first applying the user-specified delays and bit-errors. Spanner buffers packets to simulate both propagation delay and delay caused by the specified data rate. Spanner randomly corrupts packets to simulate a particular bit-error rate by choosing packets according to a Bernoulli process and simply not forwarding them to the destination. Spanner also is capable of sending *link-outage* ICMP messages and mimicking a link outage by dropping all packets for a user-specified period of time.

Spanner is also capable of emulating a typical Drop-Tail router that has finite buffers. Spanner's buffer capacity can be configured separately in each direction; all packets arriving while the buffer is full are discarded. In future, we plan to give Spanner the ability to detect incipient congestion using Random Early Detection (RED) [FJ93] and to send Explicit Congestion Notification signals.

6. Experimental Results

We have conducted a wide range of experiments in the laboratory to characterize the performance of the SCPS-TP prototype. We have also conducted two experiments using live satellites: a bent-pipe test over a U. S. DoD satellite link and an on-board test, in which we hosted the SCPS-TP prototype aboard a U. K. spacecraft. In this section, we present results obtained from both the live experiments and the laboratory tests. We show a selected subset of our live test results that demonstrate the performance of SCPS-TP in a variety of challenging communications environments. In the laboratory tests, we compare the performance of SCPS-TP with that of regular TCP.

We present laboratory test results in addition to live experiment results for several reasons. The laboratory provides an experimentation environment for conducting performance measurements under controlled conditions and under scenarios that were not available in our live tests. For example, the live experiments offered only an 8:1 bandwidth asymmetry, so we used the testbed to study performance under greater bandwidth disparities. Also, it can be difficult to control or measure the bit-error rate of an actual satellite channel, while in the lab we have full control over the channel delays and errors using Spanner. Finally, we did not have the ability to run standard TCP for comparison in either of the live tests, so we obtained all of our TCP results on the testbed.[2]

6.1 Comparison of SCPS-TP and Regular TCP

We conducted a number of tests in the laboratory to compare the performance of SCPS-TP with that of regular TCP in various simulated space-link environments. In this section, we describe the results of two types of tests: asymmetry tests and corruption tests. We designed each test to provide a fair comparison between the two protocols by choosing network characteristics and protocol configurations that isolate the one environmental feature on which each test focuses (i.e., asymmetry or corruption). In each test, we configured our SCPS-TP implementation appropriately for the environment being addressed and we provisioned

[2] We note that by later duplicating the live tests on the testbed, we validated the laboratory results that were obtained using Spanner. The close agreement between the lab results and the results from the bent-pipe and on-board tests gives us confidence in the validity of our testbed results.

regular TCP with properly-sized socket buffers. As we describe each test, we list the values of the relevant SCPS-TP configuration parameters.

In all tests, we compare the performance of SCPS-TP to that of regular TCP. The implementation of TCP that we use on the testbed is the one incorporated into the SunOS 4.1.3 Unix kernel. This is one of the most widely used TCP implementations and is derived from 4.2BSD and 4.3BSD and [Ste94]. SunOS's TCP implementation does not include the RFC 1323 extensions, and based on a SunOS socket interface limitation, it has a maximum window size of 51,968 bytes. Consequently, we chose to perform all the tests on a network with a bandwidth-delay product well below SunOS's 51 kbyte window size limit to ensure that TCP's throughput did not suffer because the sender was window limited. In each test, we tune TCP's receive widow size to be approximately 10% larger than the network bandwidth-delay product. We also disable the use of the TCP Timestamps option and SCPS-TP Header Compression in our comparison tests so that both SCPS-TP and TCP have roughly the same header overhead (except for the SNACK option).

To use spanner as a router, we configure the test hosts to believe that they are on different IP networks. Many older TCP implementations, including SunOS's, choose to use 512-byte TCP segments when communicating with a destination whose IP address is non-local. As a result, we use segments that carry 512 bytes of user data in all tests, for both TCP and for SCPS-TP. In each run, we transmit 5 million bytes of user data from the client to the server, and we measure the user-data throughput. For regular TCP, we conduct the tests using the popular TTCP benchmarking tool. In conducting the SCPS-TP tests, we use a version of TTCP that we have ported to run over our SCPS-TP protocol prototype. We replicate each run 5 times and average the results.

We configure Spanner to apply the desired data-rates, delays, and bit-errors for each test. We also enable Spanner's Drop-Tail behavior and provision Spanner with one bandwidth-delay product's worth of buffers (lower-bounded by 6 packets) in each direction for all tests.

6.1.1 Asymmetry Tests

In the asymmetry tests, we demonstrate the throughput performance of SCPS-TP and regular TCP in environments with a range of bandwidth asymmetries. Using Spanner, we configured a simulated space link on our Ethernet testbed with a 50 ms one-way propagation delay and a 1.5 Mb/s data-rate in the forward direction. We systematically varied the data-rate of the return channel from 1.5 Mb/s to 1.5 kb/s to produce data-channel bandwidth to ACK-channel bandwidth ratios ranging from 1:1 to 1000:1. Note that the bandwidth-delay product of the network in all cases is well below the 51 kbyte window size limit of SunOS's TCP.

We configured SCPS-TP as described in Table 1, and varied the ACK frequency to be approximately 50% of the available ACK-channel capacity for each test.

Table 1: SCPS-TP Profile for Asymmetry Tests.

Configuration Parameter	Test Setting
Send buffer size	300 kbytes
Receive buffer size	300 kbytes
Congestion control	On
ACK frequency	Varies: 50% of avail.
Rate control	1.5 Mb/s
SNACK option	On, No bit-vector
Window Scaling option	On
TCP Timestamps option	Off
SCPS-TP Header Compression	Off

Figure 3 shows the throughput performance of both SCPS-TP and TCP over a range of data-channel to ACK-channel bandwidth ratios. We see that on a symmetric channel, both protocols obtain nearly the same throughput. However, as the ACK-channel capacity is reduced, TCP's throughput drops rapidly while SCPS-TP's throughput degrades much more slowly. The decay in TCP's throughput appears to be exponential, with the throughput obtained at a 200:1 ratio only about 30% of the throughput on a symmetric channel. In contrast, SCSP-TP's drop in throughput is nearly linear; SCPS-TP's throughput doesn't decrease by 70% until the asymmetry nears 1000:1.

The disappointing fact that SCPS-TP's throughput is worse at a 400:1 ratio than at a 500:1 ratio can be explained as an interaction between the TCP Vegas congestion control mechanism and the acknowledgment strategy. SCPS-TP's reduced acknowledgment frequency needs substantial tuning for each scenario, and when it is not properly tuned, the variation in the ACK delay can be interpreted by TCP Vegas congestion control as network congestion. In this particular experiment, we did not have the ACK frequency tuned properly in the 400:1 case and throughout suffered as a result. Refining the ACK strategy and limiting its impact on the congestion control mechanisms are areas that we are currently investigating. We note that TCP does not suffer from this problem. The throughput performance of TCP degrades smoothly as the degree of asymmetry increases.

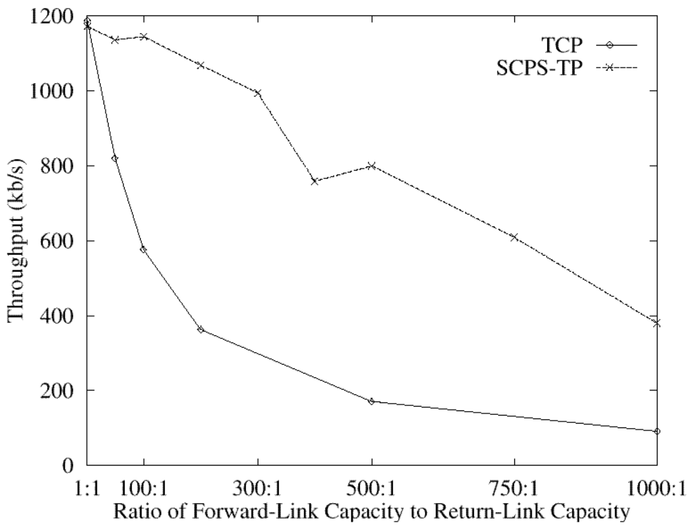

Figure 3: Asymmetric Channel: SCPS-TP vs. TCP.

6.1.2 Corruption Tests

In the corruption tests, we demonstrate the throughput performance of SCPS-TP and regular TCP at a range of bit-error rates. Here, we use Spanner to emulate a symmetric 1.5 Mb/s space link with a set of bit-error rates ranging from 10^{-8} to 10^{-5}. We conducted two independent sets of tests using different one-way propagation delays: 50 ms and 100 ms.

We set SCPS-TP's configuration parameters as listed in Table 2. In these tests, we disabled congestion control for SCPS-TP. This setting is intended to correspond to an implementation whose default assumption regarding the source of loss is corruption, or an implementation that has received an explicit corruption signal. We use a receive window that is several times larger than the network bandwidth-delay product to permit the transmission of new data while recovering from loss. The use of the token-bucket rate control mechanism mentioned earlier allows SCPS-TP to run with large windows without overrunning network capacity.

Table 2: SCPS-TP Profile for Corruption Tests.

Configuration Parameter	Test Setting
Send buffer size	300 kbytes
Receive buffer size	300 kbytes
Congestion control	Off
ACK frequency	1 ACK/69 ms
Rate control	1.425 Mb/s
SNACK option	On, No bit-vector
Window Scaling option	On
TCP Timestamps option	Off
SCPS-TP Header Compression	Off

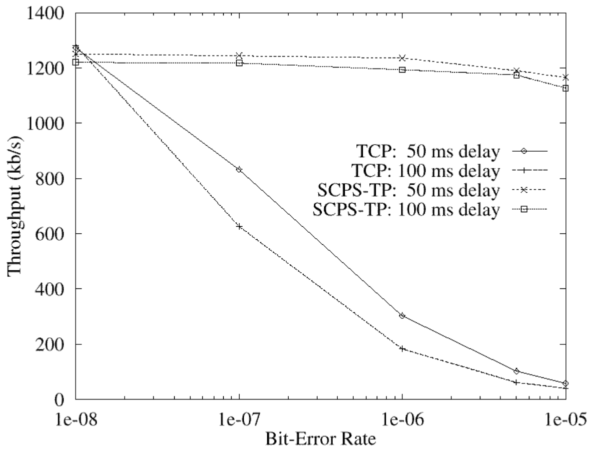

Figure 4: Corrupted Link: SCPS-TP vs. TCP.

Figure 4 shows the throughput performance of SCPS-TP and TCP as a function of bit-error rate at two different delays. We see that SCPS-TP's throughput performance is virtually unaffected by packet loss caused by bit-errors, while TCP's throughput suffers severely as the error-rate increases. In general, SCSP-TP is able to maintain its transmission rate while recovering from losses. Only at bit-error rates worse than 10^{-6} does throughput begin to taper. In the 50 ms delay case, the throughput for SCPS-TP at 10^{-5} is only 7% lower than at 10^{-8}. Meanwhile, TCP throughput is sensitive to bit-errors because TCP invokes a congestion control response to all loss. In addition, TCP learns about segment loss slowly, at most one per round-trip, because it lacks a selective acknowledgment mechanism.

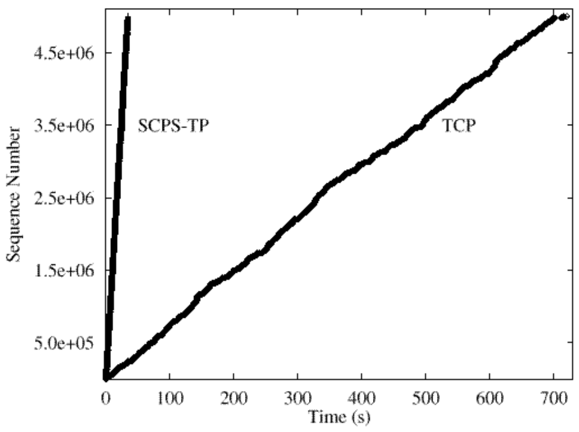

Figure 5: Sequence Numbers vs. Time at 10^{-5} BER.

Figures 5 and 6 contrast the way that SCPS-TP and TCP react to corruption-induced loss. These figures show transmitted sequence numbers as a function of time for both protocols.

Figure 6 is a detailed view of the beginning of the trace shown in Figure 5. The figures show how TCP repeatedly pauses following losses (due to retransmission timeout) and slowly increases its transmission rate until the next loss causes a retransmission timeout. Meanwhile, SCPS-TP transmits at a constant rate, while retransmitting lost segments.

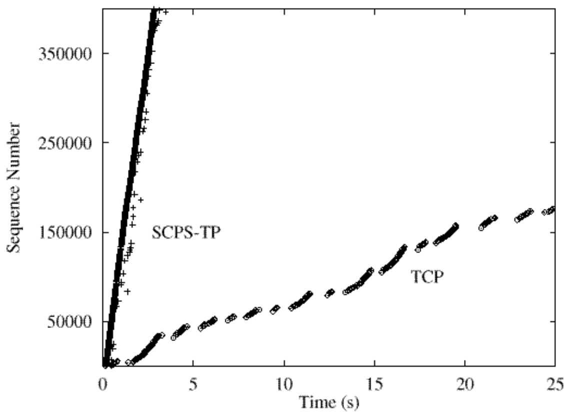

Figure 6: Sequence Numbers vs. Time at 10^{-5} BER (detailed view).

6.2 Bent-Pipe SCPS-TP Experiment

We present a single, summarized set of results from our bent-pipe experiment with SCPS-TP running over a real satellite link. We focus on the throughput performance of a single configuration of the SCPS-TP prototype when faced with a range of bit-error rates. The SCPS-TP prototype was hosted on a pair of Sun workstations (an IPC for the initiator and a Sparc 2 for the responder) that were configured to communicate with each other through their serial ports over the satellite channel. The satellite channel bandwidth was 256 kb/s in the forward direction and 32 kb/s in the reverse, yielding an 8:1 data-channel to ACK-channel ratio. The round-trip propagation delay was 500 ms, which is typical of a geosynchronous satellite hop. The SCPS-TP prototype was configured as shown in Table 3. Because the tests were conducted on a private point-to-point satellite link, it was reasonable to disable congestion control and set corruption to be SCPS-TP's default assumption about the source of loss.

Table 3: SCPS-TP Profile for Bent-Pipe Test.

Configuration Parameter	Test Setting
Send buffer size	71,929 bytes
Receive buffer size	69,881 bytes
Congestion control	Off
ACK frequency	2 ACKs/RTT
Rate control	256 kb/s
SNACK option	On, No bit-vector
Window Scaling option	On
TCP Timestamps option	On
SCPS-TP Header Compression	On

In each test, the SCPS-TP initiator transmitted 6 Mbytes of data over a single SCPS-TP connection to the responder. We conducted tests with a range of packet sizes and measured the user-data throughput in each case. The throughput results of two selected cases are shown in Figure 7. We plot the throughput versus the bit-error rate for 1000-byte and 125-byte segments. Along with the live bent-pipe test data, shown as individual points, the graph depicts the laboratory data for this scenario. We obtained the lab data by duplicating the bent-pipe test conditions on our Ethernet testbed using Spanner to provide the appropriate delay, data-rate, and bit-error rates. The lab data sets are shown as two lines, one for each packet size, that represent the mean of five runs at each of the four bit-error rates: 10^{-8}, 10^{-7}, 10^{-6}, and 10^{-5}.

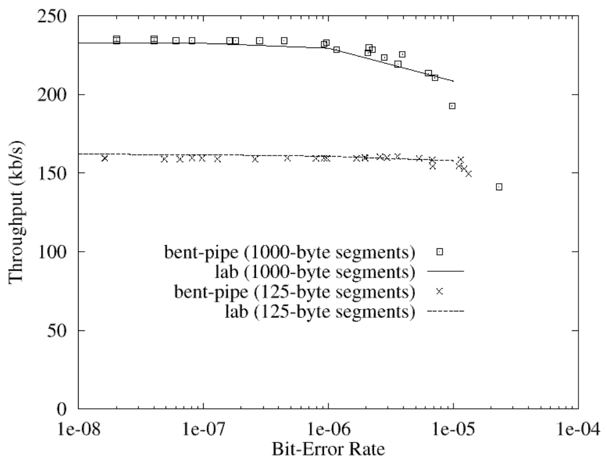

Figure 7: Bent-Pipe Test: Throughput vs. BER.

We make three observations about Figure 7. First, there is close agreement between the live bent-pipe data and the laboratory data. Second, the throughput performance of SCPS-TP is almost flat for bit-error rates better than 10^{-6} with both 1000-byte and 125-byte segments. The throughput degrades by approximately 10% with 1000-byte segments as the bit-error rate moves from 10^{-6} to 10^{-5}; however, the performance with 125-byte segments is reduced only slightly at these bit-error rates. These curves demonstrate SCPS-TP's robustness in dealing with corruption-induced loss. Finally, we see that 1000-byte segments consistently out-perform 125-byte segments until the bit-error rate becomes worse than 10^{-5}, where smaller segments begin to gain an advantage because they are less susceptible to bit-errors.

6.3 On-Board SCPS-TP Experiment

In the on-board SCPS-TP experiments, we had the opportunity to port our prototype to the on-board computer of an orbiting spacecraft and conduct protocol performance measurements. We hosted our code on a British satellite known as the Science and Technology Research Vehicle (STRV). The communications channel bandwidth from the satellite to the ground was 1000 b/s and the return channel data-rate was 125 b/s, again yielding an 8:1 ratio. The maximum transmit unit of the STRV's interface was 90 bytes, making it impossible to experiment with larger packet sizes. The round-trip propagation delay was about 8 seconds because of clocking delays and delays imposed by various components of the ground network between the groundstation and the control center, where the SCPS-TP host was located. We conducted a number of tests on the STRV to characterize the performance of the SCPS-TP protocol and to evaluate the performance improvements offered by the various individual SCPS-TP extensions.

Table 4: SCPS-TP Profile for On-Board Test.

Configuration Parameter	Test Setting
Send buffer size (on-board)	19,712 bytes
Receive buffer size (on the ground)	69,880 bytes
Congestion control	Off
ACK frequency	1 ACK/RTT
Rate control	1 kb/s
Window Scaling option	On
TCP Timestamps option	Off

In Figure 8, we show the results of two independent on-board tests with different protocol configurations. These tests illustrate the impact of SCPS-TP Header Compression and the SNACK option. The first configuration includes these two extensions and the second disables them. The SCPS-TP configuration parameters that are common to the two tests are listed in Table 4. In the live tests, the spacecraft was the data-sender and the ground-based host was the receiver. We repeated the on-board experiments on the testbed and we present both live and laboratory data in Figure 8. The figure plots the SCPS-TP user-data throughput versus bit-error rate for each protocol configuration. The lines on the plot represent the lab data for each protocol configuration and the points are individual runs on the live satellite. As in the bent-pipe test, we see close agreement between the lab and the live test results. The plot shows, as expected, that the SCPS-TP Header Compression and SNACK extensions substantially improve throughput performance when the link experiences bit-errors. By reducing overhead, header compression improves throughput at all bit-error rates since the compression scheme is tolerant of loss. The SNACK option improves error recovery by informing the sender about the loss of potentially multiple segments without having to wait for three duplicate ACKs or a retransmission timer time-out, as is the case when SNACK is disabled.

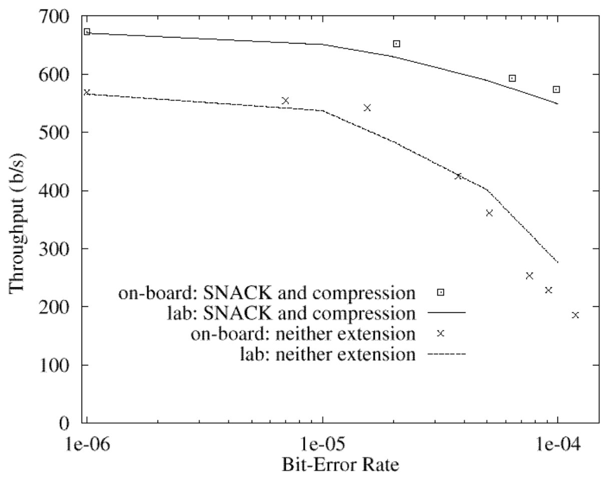

Figure 8: On-Board Test: Throughput vs. BER.

7. Conclusions and Future Work

The space communication environment differs from the terrestrial (wired) communication environment in ways that significantly affect transport protocol performance. While TCP works well in the terrestrial environment, modification is necessary to provide good performance in the satellite environment. One of the primary differences between space communication environments and current terrestrial environments is in the source of data loss. Terrestrial networks primarily experience loss caused by congestion. In contrast, space communication networks exhibit mixed-loss characteristics: losses can result from congestion, corruption, or link outages.

TCP's assumption that virtually all loss is caused by congestion results in severe degradation of performance in error-prone environments. When losses are not caused by congestion, SCPS-TP's throughput remains high by avoiding the congestion-control response and by providing enhanced information about data loss via the SCPS-TP Selective Negative Acknowledgment (SNACK) option.

Improving performance in mixed-loss networks requires a means of identifying the source of data loss and mechanisms to respond appropriately to the different sources of loss. We have identified three primary sources of loss in space communication networks, and are examining existing, albeit not currently widely used, methods for congestion detection and response. We have defined new methods of signaling corruption and link outage, and have developed appropriate responses to these sources of loss.

We have also modified TCP's acknowledgment strategy to accommodate asymmetric channels. SCPS-TP sends ACKs much less frequently than standard TCP does to improve throughput when the ACK channel is highly constrained. This modification has ramifications on TCP's congestion control and loss detection mechanisms. We have adopted the TCP Vegas congestion control mechanisms to help cope with the reduced ACK frequency; however, additional work is still required in this area. We have discovered that TCP Vegas congestion control is very sensitive to changes in the round-trip time. Such changes may be caused by significant ACK delay or by mobility; however, TCP Vegas will interpret them as network congestion. We have also found that our approach of a reduced ACK frequency coupled with TCP Vegas congestion control requires a substantial amount of tuning. In contrast, TCP's congestion control algorithms are quite elegant and robust and they adapt autonomously to changing delays.

In continuing our work with mixed-loss communication environments, we intend to experiment with Random Early Detection (RED) gateways [FJ93] using Explicit Congestion Notification, and to extend the RED gateways to also signal corruption and link outages, based on link quality information from the link layer. We plan to enhance our TCP-Vegas congestion control implementation to provide closed-loop rate control, and to develop adaptive acknowledgment strategies to address the combination of high error-rates and limited acknowledgment channel bandwidth. Additionally, we plan to compare the effectiveness of the SCPS-TP SNACK option to that of the RFC 2018 SACK option [MMFR96], especially in conjunction with a reduced acknowledgment frequency. We are interested in the prospects of improving the TCP Vegas congestion control algorithms by inferring which segments have left the network using the SACK, in a way similar to the Forward Acknowledgment (FACK) algorithm for TCP proposed in [MM96].

Although we are developing our work specifically for the space communication environment, we feel that the extensions to TCP that we are defining will provide similar improvements in TCP performance when operating over mobile and wireless links. We intend to evaluate these extensions in the wireless and mobile environments at a future date.

Acknowledgments

The authors thank the anonymous reviewers for their constructive comments. Sally Floyd and Adrian Hooke each provided comments that improved the quality of this paper. We also thank the many individuals that assisted us in configuring and conducting the live data tests.

References

[Bra89] R. Braden, editor, "Requirements for Internet Hosts - Communication Layers," Request for Comments 1122, IETF, October 1989.

[BB94] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," Technical Report DCS-TR-314, Rutgers University, October 1994.

[BSAK95] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks*, Vol. 1, No. 4, pp. 469-481, December 1995.

[BPSK96] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *Proceedings of ACM SIGCOMM '96*, pp. 256-269, Stanford, CA, August 1996.

[BOP94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Avoidance," *Proceedings of ACM SIGCOMM '94*, pp. 24-35, London, U. K., October 1994.

[CI95] R. Caceres and L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE Journal on Se-*

lected Areas in Communications, Vol. 13, No. 5, pp. 850-857, June 1995.

[DLY95] P. Danzig, Z. Liu, and L. Yan, “An Evaluation of TCP Vegas by Live Emulation,” *Proceedings of ACM SIGMetrics ‘95*, 1995.

[FJ93] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397-413, August 1993.

[Flo94] S. Floyd, “TCP and Explicit Congestion Notification,” *ACM Computer Communications Review*, Vol. 24, No. 5, pp. 8-23, October 1994.

[Fox89] R. Fox, “TCP Big Window and Nak Options,” Request for Comments 1106, IETF, June 1989.

[Hoe96] J. C. Hoe, “Improving the Start-up Behavior of a Congestion Control Scheme for TCP”, *Proceedings of ACM SIGCOMM ’96*, pp. 270-280, Stanford, CA August 1996.

[JB88] V. Jacobson and R. Braden, “TCP Extensions for Long-Delay Paths,” Request for Comments 1072, IETF, October 1988.

[JBB92] V. Jacobson, R. Braden, and D. Borman, “TCP Extensions for High Performance,” Request for Comments 1323, IETF, May 1992.

[Jac88] V. Jacobson, “Congestion Avoidance and Control,” *Proceedings of ACM SIGCOMM ‘88*, pp. 314-329, Stanford, CA, August 1988.

[Jac90] V. Jacobson, “Compressing TCP/IP Headers for Low-Speed Serial Links,” Request for Comments 1144, IETF, February 1990.

[MM96] M. Mathis, J. Mahdavi, “Forward Acknowledgment: Refining TCP Congestion Control,” *Proceedings of ACM SIGCOMM ‘96*, pp. 281-291, Stanford, CA, August 1996.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment Options,” Request for Comments 2018, IETF, October 1996.

[Par94] C. Partridge, *Gigabit Networking*, Addison-Wesley, Reading, MA, 1994.

[Pax96] V. Paxson, “End-to-End Routing Behavior in the Internet,” *Proceedings of ACM SIGCOMM ‘96*, pp. 25-38, Stanford, CA, August 1996.

[Ste94] W. R. Stevens, *TCP/IP Illustrated*, Volume 1, Addison-Wesley, Reading, MA, 1994.

[Ste96] W. R. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,” Internet Draft, draft-stevens-tcpca-spec-01.txt, March 1996.

[VS94] C. Villamizar and C. Song, “High Performance TCP in ANSNET” *ACM Computer Communications Review*, Vol. 24, No. 5, pp. 45-60, October 1994.