A History of the Improvement of Internet Protocols Over Satellites Using ACTS[*]

Mark Allman
NASA GRC/BBN Technologies
mallman@grc.nasa.gov

Hans Kruse
Ohio University
hkruse1@ohiou.edu

Shawn Ostermann
Ohio University
ostermann@cs.ohiou.edu

Abstract

This paper outlines the main results of a number of ACTS experiments on the efficacy of using standard Internet protocols over long-delay satellite channels. These experiments have been jointly conducted by NASA's Glenn Research Center and Ohio University over the last six years. The focus of our investigations has been the impact of long-delay networks with non-zero bit-error rates on the performance of the suite of Internet protocols. In particular, we have focused on the most widely used transport protocol, the Transmission Control Protocol (TCP), as well as several application layer protocols. This paper presents our main results, as well as references to more verbose discussions of our experiments.

1 Introduction

The work presented in this paper started in 1994 as a series of experiments to determine the impact of a geosynchronous satellite link in a network path on the standard TCP/IP Internet suite of protocols [Ste94]. Our investigations are important for several reasons. First, commercial satellite companies would like to deliver Internet services to consumers and institutions in remote areas of the world not covered by good terrestrial connectivity (e.g., Hughes DirecPC). Our investigations have helped to define and identify the extensions to the Internet protocol suite that are beneficial to delivering Internet content over network paths containing long-delay satellite channels. In addition, NASA is interested in possibly employing off-the-shelf Internet protocols to meet its near-Earth communication needs. Therefore, our experiments focus on improving standard Internet protocols in ways that are both safe in all network environments and beneficial to long-delay networks.

We utilized NASA's Advanced Communication Technology Satellite (ACTS) to conduct our experiments. We used VSAT ground stations and data rates between roughly 0.75 Mbps and 1.5 Mbps (i.e., between half and full T1 rate)in all our experiments. While these tests were conducted at relatively modest data rates, the results scale with the available bandwidth (as shown in [IBF+99]). Generally, our experiments were conducted with a sender at NASA's Glenn Research Center and a receiver at Ohio University (or vice versa). However, several of our experiments were performed with a loopback circuit, such that the sender and receiver were located in the same location.

The bulk of our experiments focus on the Transmission Control Protocol (TCP) [Pos81]. TCP is the Internet's most used transport protocol. TCP provides reliable, in-order transmission of data to applications. In addition, TCP provides end-to-end congestion control mechanisms that attempt to protect the network against *congestion collapse* (a state when the network is very busy, but little useful work is being done) [FF99]. Additionally, we have explored several application layer protocols that utilize TCP.

This paper is organized as follows. Section 2 outlines our early work in determining the problems with using standard Internet protocols over ACTS. Section 3 discusses an application layer mitigation to TCP's shortcomings over long-delay networks. Next, Section 4 outlines our experiences using standardized solutions to mitigate TCP's performance problems over ACTS. Section 5 discusses two experimental mechanisms introduced into TCP and the impact of these extensions on performance. Section 6 outlines our investigation of the performance of HTTP, the application layer protocol used on the World-Wide Web. Section 7 discusses our investigation of using a realistic traffic mix across a network path containing an ACTS satellite circuit. Section 8 outlines our experiments into TCP performance over circuits with non-zero bit-error rates. Finally, Section 9 gives our conclusions

[*]This paper appears in the proceedings of the ACTS Conference 2000, May 2000.

and outlines future work in this area.

2 Problems with TCP/IP Over ACTS

Our early work [Kru95] illustrates two main causes of performance degradation in TCP file transfers. First, in long transfers the advertised window supported by off-the-shelf TCP stacks is inadequate. The throughput (or bandwidth attained) for long-lived TCP transfers is given by the formula in equation 1 [Pos81], where W is the *advertised window size*, B is the bandwidth of the network link and RTT is the *round-trip time* between the data sender and the data receiver.

$$W = B \cdot RTT \tag{1}$$

The advertised window is the largest amount of data that can be buffered by the receiver. Therefore, the advertised window represents the largest amount of data a TCP sender can transmit before receiving an acknowledgment (ACK) from the receiver. As B and/or RTT grow, W must be increased accordingly. However, TCP places a limit on W by only allocating 16 bits of header space for the value. Thus, the advertised window can be no more than 64 KB[1]. The effect of this limit is that TCP cannot fully utilize the bandwidth of a network path with a large *delay-bandwidth product*. In addition, many TCP stacks use advertised window sizes much less than 64 KB by default. For instance, the hosts used in our early experiments [Kru95] utilized advertised window sizes of 24 KB. Therefore, the maximum throughput of a transfer over ACTS was approximately 44,000 bytes/second regardless of the amount of capacity available over the satellite circuit.

The second problem noted in [Kru95] pertains to short transfers. Our experiments illustrate that TCP's slow start algorithm [Jac88, APS99] was the cause of the performance degradation. The slow start algorithm is part of TCP's congestion control mechanism. The algorithm introduces a *congestion window* (*cwnd*), which is the sending TCP's measure of the current capacity of the network. Slow start begins conservatively, by initializing *cwnd* to 1 segment. For each ACK received, *cwnd* is increased by 1 segment, providing an exponential increase in the sending rate. The slow start algorithm terminates when loss is detected (assumed to indicate network congestion) or *cwnd* reaches the advertised window size. For long transfers, this slow probing of the network to determine the capacity is a small percentage of the transfer time and therefore does not have a large negative impact on performance. However, for short transfers, TCP is never able to fully utilize the capacity of the network path. For instance, a 2 segment transfer will take 2 RTTs (or more than 1 second) after TCP's three-way handshake is completed even if the network capacity to transmit both segments was available when the transfer started.

[1]For the first sets of experiments we did not consider TCP's optional window scaling mechanism [JBB92], which allows for advertised windows larger than 64 KB, due to the lack of implementations of the mechanism. Later experiments did utilize these TCP extensions, as outlined in section 4.

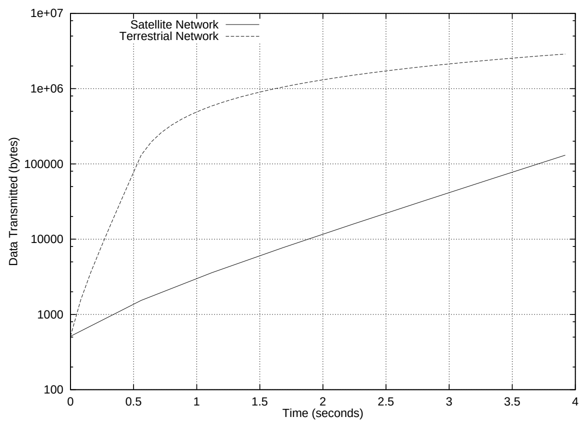


Figure 1: Data transferred as a function of time over satellite and terrestrial network paths.

Figure 1 from [All97] illustrates the low utilization of a satellite network during slow start, as compared to a network with a terrestrial delay (80 ms in this model). Just before 4 seconds into the transfer over the satellite link the slow start phase completes. During that same amount of time, the terrestrial network is able to transfer 22 times the amount of data as is sent over the satellite link! After slow start, both networks send the same number of bytes/second, but obviously the slow start phase hurts the performance of the long-delay connection much more than the shorter-delay terrestrial network connection.

3 An Experimental Application Layer Mitigation

The above problems led to the development of an application-level tool to enhance the efficiency of data transfers. We extended the the File Transfer Protocol (FTP) [PR85] to use multiple TCP connections to transfer a given file, rather than one connection as specified in [PR85]. This multiplied TCP's aggressiveness by the number of TCP connections being utilized. The syntax and semantics of the extensions to FTP are outlined in [AO97]. The ACTS experiments involving *xftp* are outlined in [AOK95, AKO96, All97].

Figure 2 shows the throughput of a 5 MB transfer as a function of the number of parallel data connections used to transfer the file over an ACTS T1 link. Each connection used an advertised (maximum) window of 24 KB which yields throughput of approximately 44,000 bytes/second, as outlined above. Therefore, we would predict that 4 connections would be required to fully utilize the capacity of the channel (approximately 192,000 bytes/second). However, the best performance is obtained when using 6–8 data connections. We believe it takes more than four connections to reach optimal performance due to segment overhead, as well as lingering slow start effects. When using 6–8 connections we achieve nearly optimal throughput when all protocol overhead is taken into account. Using more than 8 connections leads to sub-optimal performance (but, still much better than using a single connection). This drop in throughput is caused by segment losses due to increased congestion from competing TCP flows. Part of TCP's congestion control mechanism calls for a reduction in *cwnd* when a loss is detected, as the loss is assumed to indicate network congestion. As soon as *xftp* starts over-running router buffer queues, thus losing segments, some of the connections reduce their sending rate, so the time required for the entire transfer increases.

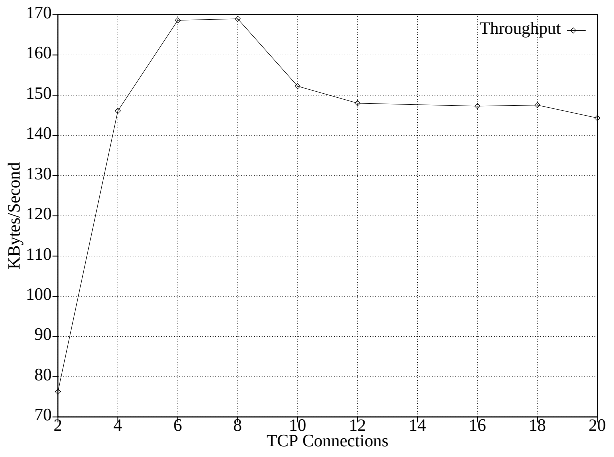


Figure 2: Performance of *xftp* as a function of the number of parallel TCP connections employed over an ACTS T1 circuit.

The following are some of our key findings from our *xftp* ACTS experiments:

- Large advertised windows are required. As predicted by the experiments outlined in the previous section, using a larger *effective window size* (i.e., the sum of the advertised window sizes across all connections used by *xftp*) allows full utilization of the available capacity for long-lived data transfers.

- Larger initial congestion window sizes help. Using N connections in parallel speeds up slow start by using an effective initial *cwnd* of N segments. This cuts several RTTs off the transfer time and could be especially useful for short transfers.

- The throughput of the transfer is sensative to the number of connections employed. Using too few connections results in an effective advertised window less than the delay-bandwidth product and thus an underutilization of the capacity. Using too many connections leads to loss on the channel and a reduction in sending rate due to network congestion. Finding a general mechanism to choose the proper number of connections during the data transfer proved difficult [AKO96].

- The multiple TCP connections acted much like a "selective acknowledgment" (SACK) mechanism. In other words, *xftp*'s loss recovery is more efficient than the standard TCP loss recovery [APS99] because it was spread across many connections that each keep track of their own sequence space. Standard TCP can effectively recover from one lost segment per RTT [FF96]. Therefore, *xftp* can effectively recover from roughly N losses per RTT (assuming N parallel connections).

Finally we note that using multiple parallel TCP connections is not "friendly" to the network in general because each indication of network congestion reduces *cwnd* by less than the reduction would be if one connection were used [FF99]. Therefore, while *xftp* is a valuable tool in learning about network dynamics it *is not recommended for general purpose use*.

4 Standard Solutions

During our investigations, the Internet Engineering Task Force (IETF) standardized options to TCP to mitigate some of the problems outlined above. RFC 1323 [JBB92] introduced an option for TCP to advertise windows much larger than 64 KB. Meanwhile, RFC 2018 [MMFR96] introduced a *selective acknowledgment* (SACK) option to TCP. Using the SACK option, receivers can inform senders exactly which segments have arrived, rather than relying on TCP's cumulative acknowledgment. This allows a TCP sender to efficiently recover from multiple lost segments without reverting to using a costly *retransmission timeout* to determine which segments need to be resent [FF96].

We conducted a series of ACTS experiments using these two new TCP options [AHKO97, Hay97]. Figure 3 shows the throughput for a number of different variants of TCP as a function of transfer size. We used a half-T1 ACTS link for these experiments. The *xftp* experiments use 4 parallel connections. First, we turn our attention to the two experiments run using effective advertised window sizes of the delay-bandwidth product (which produces no network congestion and therefore no segment

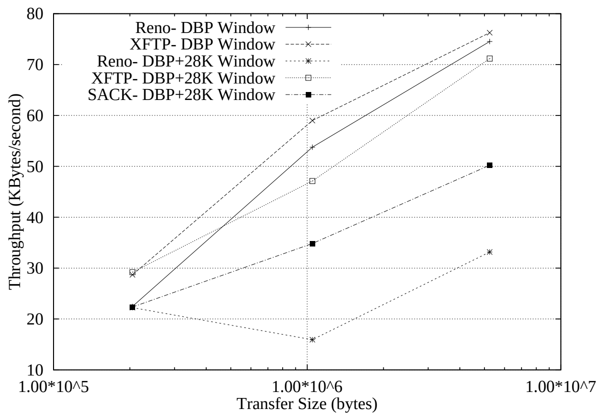


Figure 3: Throughput of various versions of TCP as a function of transfer size.

loss). In this case, *xftp* slightly outperforms the one connection Reno transfer. The amount by which the throughput differs between the transfers gets smaller as the transfers grow longer. This indicates that the difference is due to the *xftp* transfer using a larger initial *cwnd*.

The lower three lines on the plot represent experiments with a larger than necessary advertised window. The increased advertised window leads to dropped segments due to buffer overflow in a router in the middle of the network path. Standard Reno TCP performs the worst in these experiments. As shown, using TCP with the SACK option drastically increases throughput. Using *xftp* provides still better throughput. However, *xftp* has a more aggressive response to network congestion than a single TCP connection. When one loss occurs on the set of parallel connections only one of the four TCP connections reduces its *cwnd* by half, leading to an overall reduction of an eighth in response to a single congestion indication (rather than the standard reduction of one half) in this experiment. The more aggressive response to congestion used by *xftp* explains the throughput benefit shown in the plot.

The following is a summary of our conclusions from this set of ACTS experiments:

- When the network is uncongested, TCP's large window extensions (RFC 1323 [JBB92]) provide nearly the same behavior as *xftp*, modulo the larger initial *cwnd* utilized by *xftp*.

- TCP's SACK option provides drastic throughput improvements in the face of network congestion.

- The results of these experiments alluded to the fact that the throughput of a transfer was quite sensative to the advertised window chosen. Hayes [Hay97] emulated our ACTS setup and shows the disastrous effects that choosing the wrong advertised window size can have on performance.

The ACTS experiments outlined in this section were influential to the IETF's TCP Over Satellite Working Group as RFC 2488 [AGS99] was prepared. This RFC outlines the standard IETF mechanisms that should be used by hosts transfering data over network paths containing satellite links.

5 Experimental TCP Mitigations

Our next short set of ACTS experiments involved investigating ways to mitigate the underutilization of the network during the slow start phase of a TCP transfer. The first mechanism we studied was using a larger initial *cwnd*, as suggested by the experiments outlined in the last section.

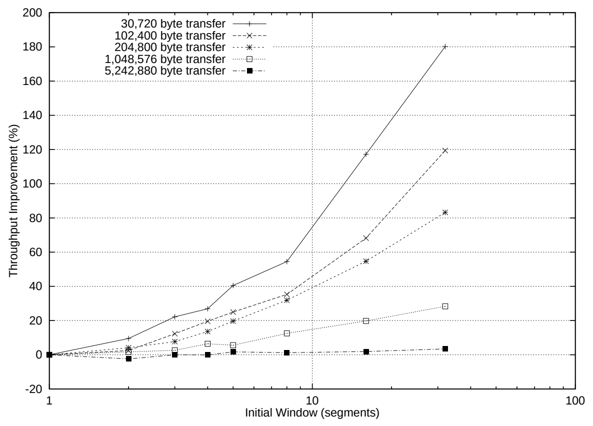


Figure 4: Throughput improvement as a function of initial *cwnd* size.

Figure 4 from [All97] shows throughput improvement as a function of the initial *cwnd* size for various transfer sizes. As shown, the throughput increases as the initial value of *cwnd* is increased. The impact is especially significant for short transfers. The impact for the longer transfers is much less due to the relatively short amount of time spent using slow start when compared to the total time required to transfer the file.

These experiments, along with several additional investigations [AHO98, PN98, SP98], influenced the IETF's decision to make the use of a larger initial *cwnd* a sanctioned experimental mechanism [AFP98].

Our second set of experiments involved a slightly modified algorithm for increasing *cwnd* during slow start. As outlined in section 2, *cwnd* is increased by 1 segment for each ACK received during slow start. Many TCP receivers employ the *delayed acknowledgment* algorithm [Bra89, APS99]. That is, receivers are allowed to refrain from sending an ACK for each incoming segment. However, an ACK must be sent for every second full-sized segment received. Furthermore, an ACK can not be delayed for more than 500 ms. By reducing the number of ACKs sent to the data originator, the receiver is slowing the growth of *cwnd*. We introduced an algorithm called *byte counting* which allows the sender to increase *cwnd* based on the number of new segments acknowledged by each incoming ACK, rather than on the number of ACKs received.

File Size	Throughput Improvement (%)
30 KB	9.4
100 KB	16.9
200 KB	15.3
1 MB	8.5
5 MB	9.5

Table 1: Throughput improvement when using byte counting rather than ACK counting to increase *cwnd*.

Table 1 shows the performance improvement of using byte counting as opposed to traditional ACK counting [All97]. As shown, the improvement for short transfers is better than for long transfers (even though the improvement is good for long transfers, as well). This shows that byte counting is important in slow start, but is also important during congestion avoidance (the phase whereby TCP probes for additional network capacity by increasing *cwnd* linearly).

Byte counting has been adopted by the IETF as a proposed standard during the congestion avoidance phase of TCP connections [APS99]. Further refinements to byte counting have been suggested since the above ACTS experiments [All98, All99]. Our hope is to develop an experimental document within the IETF to allow some form of byte counting during slow start in addition to its already sanctioned use during congestion avoidance.

6 HTTP Experiments

The next set of ACTS experiments we conducted employed the HyperText Transfer Protocol (HTTP) [BLFN96, FGM+97], the application layer protocol used for World-Wide Web (WWW) transfers. HTTP uses TCP for reliable transport of its data. Two versions of HTTP have been defined and are in widespread use on the Internet. HTTP/1.0 [BLFN96] transfers a single WWW "object" (HTML document, image file, etc.) per TCP connection. Oftentimes, WWW browsers open multiple HTTP/1.0 connections simultaneously to decrease the time required to transfer all objects necessary to render a web page. HTTP/1.1 [FGM+97] allows a TCP connection to be re-used for transfering multiple WWW objects[2]. In addition, HTTP/1.1 provides a "pipelining" mechanism, whereby a WWW browser can request any number of objects as soon as possible, rather than waiting until the previous object has been transfered to request the next object.

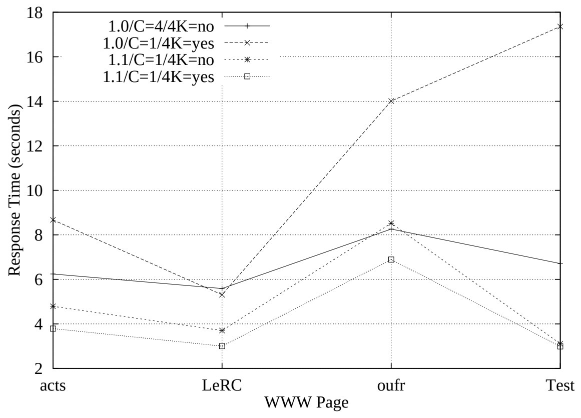


Figure 5: Comparison of HTTP variants.

Figure 5 shows the results of our ACTS experiments with both versions of HTTP. The labels along the *x*-axis represent different WWW pages. The WWW pages used in our study have differing characteristics (number of objects, size of objects, etc.). See [KAGT98, KAGT00] for a description of the page characteristics. Each line on the plot is labeled with three settings used for the particular experiment, as follows.

1. The version of HTTP used ("1.0" or "1.1").
2. The number of parallel TCP connections employed to transfer the WWW objects ("C= x" where x is the number of connections used).
3. Whether the underlying TCP stack used a larger initial *cwnd*, per the proposal outlined in [AFP98] ("4K= z" where z is "yes" when using a larger initial *cwnd* or "no" when using the standard initial *cwnd*).

The following are the key results from our study of HTTP transfers over ACTS.

- HTTP/1.1 generally outperforms HTTP/1.0, even when HTTP/1.0 is used in conjunction with multiple simultaneous TCP connections.

[2]HTTP/1.0 also has a "keepalive" option for using persistent connections. Use of this option in HTTP/1.0 implementations is limited and the mechanism is equivalent to the base HTTP/1.1 persistent connection mechanism. Therefore, we do not present any results using HTTP/1.0 with keepalives, as our experiments indicated the HTTP/1.1 (without pipelining) case is roughly equivalent.

- When using only one TCP connection, HTTP/1.0 performs quite badly, even when using a larger initial congestion window. This happens because each object must endure TCP's slow start phase. When using a single connection with HTTP/1.1, the effects of slow start are diminished because the TCP connection is reused a number of times. Therefore, the small objects that make up the WWW page are combined to behave more like a bulk transfer and therefore improve network utilization (as discussed in the previous sections).

- As outlined in the previous section, using a larger initial value for the congestion window improves performance for short transfers (which are characteristic of WWW traffic).

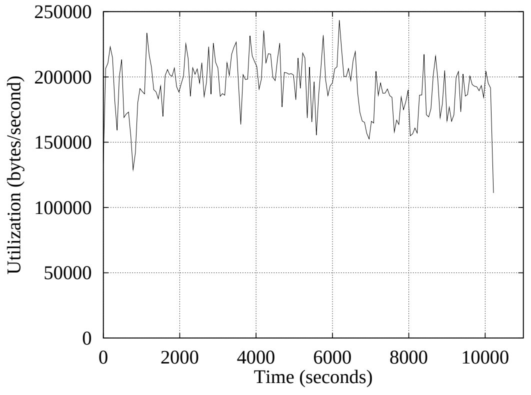
- Kruse [KAGT00] defines a model for HTTP transfers that accurately predicts the transfer time of web pages of various size.

These experiments aided the IETF in deciding to make the use of a larger initial value for *cwnd* an experimental mechanism [AFP98]. In addition, these experiments highlight the importance of carefully designing application protocols such that the interactions between the application and the underlying transport do not hinder performance.

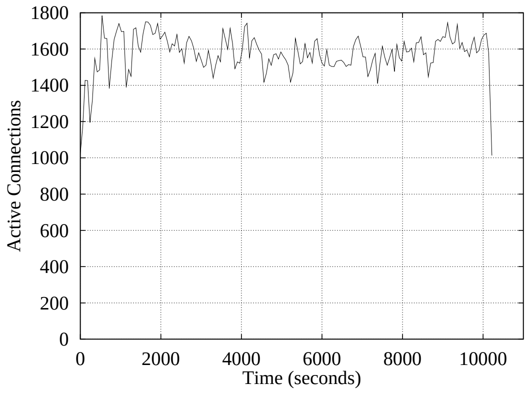
7 Representative Network Traffic

Up to this point our experiments have involved a single file transfer over an otherwise unloaded network path. In our next set of ACTS experiments, we strive to assess the ability of a realistic group of TCP transfers to utilize the available bandwidth across a network path containing a satellite channel [KAG$^+$99]. As shown in the previous sections, short TCP transfers can underutilize the available bandwidth when no competing traffic is present. However, our previous experiments have not assessed the ability of a group of TCP connections to utilize the full capacity of a long-delay network path. We developed a traffic generator called *trafgen* [Hel98], based on *tcplib* [DJ91] for these experiments. First, we take a packet-level trace of network traffic from a production network (e.g., the network connecting NASA GRC to the Internet). The trace is then analyzed using *tcptrace* [Ost97] for traffic characteristics. Finally, these characteristics are imported into *trafgen*, which then generates a realistic mix of TCP connections based on the particular production network that produced the original trace.

Figure 6 shows the results of a *trafgen* experiment over a T1 ACTS circuit between NASA GRC and Ohio University. As illustrated, the network is fully utilized in many instances, while a large number of TCP connections (or



(a) Aggregate throughput.



(b) Number of active connections.

Figure 6: Behavior of a realistic mix of traffic as a function of time over an ACTS T1 circuit.

users) is easily supported. This indicates that a representative group of TCP connections can utilize the available bandwidth. While the long RTT may increase the transfer time of some individual TCP transfers (when compared to the same transfer over a network with a shorter RTT), it does not prevent the sum of the transfers from fully utilizing the satellite channel.

8 The Impact of Bit-Errors

The final experiment we conducted over ACTS attempts to quantify the impact of non-zero bit-error rates (BER) on TCP performance. An outline of this experiment and some preliminary results are given in [KOA00]. These experiments were conducted by adjusting the Earth-station at Ohio University such that it did not track the inclinedorbit ACTS satellite. As the satellite moved with respect to the dish, the BERs observed varied. We ran long-lived (1 hour) TCP flows through the network during this time and measured the bit-error rate using an out-of-band channel. Further details can be found in [KOA00]. The TCP stack employed in this set of experiments used a 512 KB advertised window (via the high performance TCP options outlined in section 4). This allows the network path to determine the performance of a TCP connection, rather than having the performance dictated by a limit on the sending or receiving host (this situation simulates socket buffer autotuning [SMM98]). In addition, the stack employed the TCP SACK option with the rate-halving algorithm [MSML99].

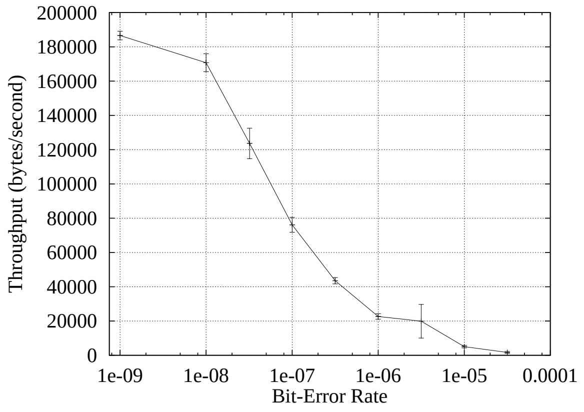


Figure 7: Throughput as a function of bit-error rate.

Figure 7 shows the throughput obtained by a TCP connection as a function of the bit-error rate of the satellite channel with 90% confidence intervals. The figure shows that with no bit-errors (denoted on the plot as 1e-09) the TCP connection is able to fully utilize the T1 capacity of the satellite channel. However, as expected, as the BER increases the throughput obtained by TCP decreases. The root of this problem is the fact that TCP cannot determine why a particular segment was dropped. Therefore, in an effort to behave conservatively, TCP interprets all segment loss as an indication of network congestion and reduces *cwnd* accordingly. Therefore, when a segment is lost due to corruption, TCP mistakenly decreases the sending rate. Research into protocol mechanisms that allow TCP to determine the true cause of a segment loss is ongoing. RFC 2760 [ADG$^+$00] contains a discussion of several of these mechanisms. Our results are consistent with analytical models of TCP performance that show throughput is indirectly proportional to the loss rate [MSMO97, PFTK98].

9 Conclusions and Future Work

Over the last six years, our ACTS experiments have shed light on the performance of the Internet protocol suite over networks containing long-delay links. Table 2 gives a summary of each of our experiments, the papers written about the experiments and the IETF standards influenced by our results. The following are the key results from our experiments.

- TCP can fully utilize the capacity of a satellite link when transfering large amounts of data.
- Short transfers often underutilize the capacity of the network, especially in long-delay environments. While we have introduced mechanisms that may mitigate this problem, more research in this area would be useful.
- Application layer protocols can have a large influence on the performance of a data transfer. For instance, using better application level mechanisms drastically decreased the transfer time required to load WWW pages. Careful attention to the design of future application protocols is required to avoid poor interactions between the transport and application layers.
- A realistic mix of network traffic can fully utilize the available bandwidth in a satellite network.
- As the BER of a channel is increased the TCP throughput decreases. Future research is needed into ways to distinguish between congestion-based segment loss and corruption-based segment loss.

These key results have been influential in several Internet Engineering Task Force Working Groups. In particular, the results aided the TCP Over Satellite WG in producing RFC 2488 [AGS99] that describes which standard TCP mechanisms should be used when transfering data over satellite channels and RFC 2760 [ADG$^+$00] which describes some of the open research topics in this area. Additionally, our ACTS experiments helped the IETF decide to increase the initial value of *cwnd* to 2 segments in RFC 2581 [APS99] and more experimentally to 3–4 segments in RFC 2414 [AFP98].

Acknowledgments

We would like to thank all our colleagues at NASA GRC and Ohio University, whose help was invaluable during our experiments. We would like to especially thank Bob Bauer, Kul Bhasin, Dan Glover, Jim Griner, Chris Hayes, Eric Helvey, Will Ivancic, Paul Mallasch, Cindy Tran and Mike Zernic. In addition, these experiments simply would

Experiment	**Outcome**	**Papers**	**Standards Contributions**
Preliminary FTP Experiments	Larger effective advertised windows are needed. Slow start decreases performance for short transfers.	[Kru95]	RFC 2488 [AGS99]
xftp Experiments	While throughput improves when using multiple parallel connections, choosing the right number of connections is difficult.	[AOK95] [AKO96] [All97]	RFC 2760 [ADG$^+$00]
High Performance TCP Extensions	Large windows help performance but lead to a higher probability of dropping multiple packets from a window of data and thus causing a drastic reduction in the transmission rate.	[AHKO97]	RFC 2488 [AGS99]
SACK Experiments	The SACK option significantly improves throughput throughput over satellite channels.	[AHKO97] [Hay97]	RFC 2488 [AGS99]
Larger Initial *cwnd* Experiments	Using a larger initial *cwnd* improves throughput, especially for short transfers.	[All97]	RFC 2414 [AFP98] RFC 2581 [APS99]
Byte Counting Experiments	Using a modified *cwnd* increase algorithm increases throughput, especially for short transfers	[All97]	RFC 2581 [APS99]
HTTP Experiments	Using old versions of HTTP increases WWW response time significantly. Using HTTP/1.1 with pipelining provides significant benefits over satellite links.	[KAGT98] [KAGT00]	
Experiments with a Realistic Traffic Mix	The Internet protocol suite is able to fully utilize the capacity provided by satellite channels when a representative traffic load is used.	[Hel98] [KAG$^+$99]	
Bit-Error Rate Tests	As the BER increases the throughput obtained by TCP decreases due to the mistaken assumption that lost segments indicate network congestion.	[KOA00]	

Table 2: Summary of key results.

not have been possible without a great deal of help from the ACTS operations team. We especially thank Terry Bell, John Diamond, Peter Harbath and Paul McMasters. Finally, we'd like to thank the many colleagues with whom we have discussed our experiments for their comments and suggestions over the years. While there are too many people to list, the input we have received from the research community has been invaluable. Our thanks to all!

References

[ADG$^+$00] Mark Allman, Spencer Dawkins, Dan Glover, Jim Griner, John Heidemann, Tom Henderson, Hans Kruse, Shawn Ostermann, Keith Scott, Jeff Semke, Joe Touch, and Diepchi Tran. Ongoing TCP Research Related to Satellites, February 2000. RFC 2760.

[AFP98] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's Initial Window, September 1998. RFC 2414.

[AGS99] Mark Allman, Dan Glover, and Luis Sanchez. Enhancing TCP Over Satellite Channels Using Standard Mechanisms, January 1999. RFC 2488, BCP 28.

[AHKO97] Mark Allman, Chris Hayes, Hans Kruse, and Shawn Ostermann. TCP Performance Over Satellite Links. In *Proceedings of the 5th International Conference on Telecommunication Systems,* pages 456–469, March 1997.

[AHO98] Mark Allman, Chris Hayes, and Shawn Ostermann. An Evaluation of TCP with Larger Initial Windows. *Computer Communication Review*, 28(3), July 1998.

[AKO96] Mark Allman, Hans Kruse, and Shawn Ostermann. An Application-Level Solution to TCP's Satellite Inefficiencies. In *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS),* November 1996.

[All97] Mark Allman. Improving TCP Performance Over Satellite Channels. Master's thesis, Ohio University, June 1997.

[All98] Mark Allman. On the Generation and Use of TCP Acknowledgments. *Computer Communication Review*, 28(5), October 1998.

[All99] Mark Allman. TCP Byte Counting Refinements. *Computer Communication Review*, 29(3), July 1999.

[AO97] Mark Allman and Shawn Ostermann. Multiple Data Connection FTP Extensions. Technical Report TR-19971, Ohio University Computer Science, February 1997.

[AOK95] Mark Allman, Shawn Ostermann, and Hans Kruse. Data Transfer Efficiency Over Satellite Circuits Using a Multi-Socket Extension to the File Transfer Protocol (FTP). In *Proceedings of the ACTS Results Conference*. NASA Lewis Research Center, September 1995.

[APS99] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP Congestion Control, April 1999. RFC 2581.

[BLFN96] Tim Berners-Lee, R. Fielding, and H. Nielsen. Hypertext Transfer Protocol – HTTP/1.0, May 1996. RFC 1945.

[Bra89] Robert Braden. Requirements for Internet Hosts – Communication Layers, October 1989. RFC 1122.

[DJ91] Peter Danzig and Sugih Jamin. tcplib: A Library of TCP/IP Traffic Characteristics. Technical Report CS-SYS-91-01, University of Southern California, October 1991.

[FF96] Kevin Fall and Sally Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, 26(3), July 1996.

[FF99] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(6), August 1999.

[FGM+97] R. Fielding, Jim Gettys, Jeffrey C. Mogul, H. Frystyk, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, January 1997. RFC 2068.

[Hay97] Chris Hayes. Analyzing the Performance of New TCP Extensions Over Satellite Links. Master's thesis, Ohio University, August 1997.

[Hel98] Eric Helvey. Trafgen: An Efficient Approach to Statistically Accurate Artificial Network Traffic Generation. Master's thesis, Ohio University, June 1998.

[IBF+99] William Ivancic, David Brooks, Brian Frantz, Doug Hoder, Dan Shell, and David Beering. NASA's Broadband Satellite Network Research. *IEEE Communications Magazine*, July 1999.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[JBB92] Van Jacobson, Robert Braden, and David Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[KAG+99] Hans Kruse, Mark Allman, Jim Griner, Shawn Ostermann, and Eric Helvey. Satellite Network Performance Measurements Using Simulated Multi-User Internet Traffic. In *Proceedings of the Seventh International Conference on Telecommunication Systems*, March 1999.

[KAGT98] Hans Kruse, Mark Allman, Jim Griner, and Diepchi Tran. HTTP Page Transfer Rates Over Geo-Stationary Satellite Links. In *Proceedings of the Sixth International Conference on Telecommunication Systems*, March 1998.

[KAGT00] Hans Kruse, Mark Allman, Jim Griner, and Diepchi Tran. Experimentation and Modeling of HTTP Over Satellite Channels. *International Journal of Satellite Communication*, 2000. To appear.

[KOA00] Hans Kruse, Shawn Ostermann, and Mark Allman. On the Performance of TCP-based Data Transfers on a Faded Ka-Band Satellite Link. In *Proceedings of the 6th Ka-Band Utilization Conference*, June 2000.

[Kru95] Hans Kruse. Performance of Common Data Communications Protocols Over Long Delay Links: An Experimental Examination. In *3rd International Conference on Telecommunication Systems Modeling and Design*, 1995.

[MMFR96] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. RFC 2018.

[MSML99] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Kevin Lahey. The Rate-Halving Algorithm for TCP Congestion Control, August 1999. Internet-Draft draft-mathis-tcp-ratehalving-00.txt (work in progress).

[MSMO97] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27(3), July 1997.

[Ost97] Shawn Ostermann. *tcptrace*, 1997. Available from http://jarok.cs.ohiou.edu/.

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, September 1998.

[PN98] Kedarnath Poduri and Kathleen Nichols. Simulation Studies of Increased Initial TCP Window Size, September 1998. RFC 2415.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

[PR85] Jon Postel and Joyce Reynolds. File Tranfer Protocol (FTP), October 1985. RFC 959.

[SMM98] Jeff Semke, Jamshid Mahdavi, and Matt Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM*, September 1998.

[SP98] Tim Shepard and Craig Partridge. When TCP Starts Up With Four Packets Into Only Three Buffers, September 1998. RFC 2416.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated Volume I: The Protocols.* Addison-Wesley, 1994.