An Application-Level Solution to TCP's Satellite Inefficiencies[*]

Mark Allman, Hans Kruse, Shawn Ostermann
Ohio University
mallman@cs.ohiou.edu,hkruse1@ohiou.edu,ostermann@cs.ohiou.edu

Abstract

In several experiments using NASA's Advanced Communications Technology Satellite (ACTS), investigators have reported disappointing throughput using the TCP/IP protocol suite over T1 satellite circuits. A detailed analysis of FTP file transfers reveals that the TCP receive window size, the TCP "Slow Start" algorithm, and the TCP acknowledgment mechanism contribute to the observed limits in throughput.

To further explore TCP's limitations over satellite circuits, we developed a modified version of FTP (XFTP) that uses multiple TCP connections. By using multiple TCP connections, we have been able to simulate a large, virtual TCP receive window.

Our experiences with XFTP over both actual satellite circuits and a software satellite emulator show that a utilization of better than 90% is possible. Our results also indicate the benefit of introducing congestion avoidance at the application level.

1 Introduction

In several experiments using NASA's Advanced Communications Technology Satellite (ACTS), investigators have reported disappointing throughput using the TCP/IP[Com95, Pos81, Ste94] protocol suite over 1.536Mbit/second (T1) satellite circuits[Kru95]. A detailed analysis of FTP file transfers reveals that both the TCP window size, and the TCP "Slow Start" algorithm contribute to the observed limits in throughput. Furthermore, in the face of loss, TCP's data recovery (positive acknowledgment) mechanism works poorly over long-delay channels. All of these facts lead to TCP's low throughput across satellite channels.

We are aware of two solutions to this problem. The first solution would be to modify the behavior of TCP to perform better over long-delay channels such as those supported by satellites. This solution would require formal changes to the TCP specifications. The window size limitation in TCP was addressed in RFC 1323[JBB92]. The problem with the TCP acknowledgment mechanism is currently being discussed within an IETF working group[1]. Unfortunately, versions of TCP with larger windows are not yet

[*]This work is sponsored in part by a grant from the NASA Lewis Research Center

[1]An early *selective acknowledgment* scheme was described as part of RFC 1185[JBZ90] in 1990. A more thorough version of this work appeared in RFC 1323[JBB92] in 1992, but the discussion of selective acknowledgments was removed (pending further research). Current work with TCP selective acknowledgments is documented in several internet drafts and studied in Fall and Floyd[FF96] and Mathis and Mahdavi[MM96].

widely available and versions with the modified acknowledgment mechanisms are not likely to be widely available for several years.

A second possible solution to TCP's poor performance over satellite channels involves the use of multiple TCP connections. One advantage to this solution is that it can be implemented entirely at the application level without requiring changes to the TCP protocol. This document describes XFTP[2], a modified version of FTP[PR85] that uses multiple TCP connections and application-level congestion avoidance to achieve high performance file transfers over satellite links. The experimental results presented here were collected using the NASA ACTS network, as well as a satellite circuit emulator built at Ohio University for this project[AO96b].

2 XFTP Overview

To test the validity of a multiple-connection model to increase TCP throughput, we built a modified version of FTP called XFTP. The chief advantage of starting with the existing FTP protocol was that good source code was available and that the FTP protocol is well understood and well documented. One of our design goals was that our modified versions of the FTP client and server software must be backward compatible with standard FTP software. To allow this, we made extensions to the underlying application protocol used by FTP. Changes to FTP have been effected at two levels, the FTP protocol itself and the FTP client user interface.

The enhancements that we made to the FTP client and server application to support multiple-connection file transfer required the addition of a new user command (entered by the user into the client application) as well as new FTP commands (sent by FTP across the control connection).

Our enhanced version of FTP is based on the 4.4BSD Unix source code and has been compiled and run on various computer platforms supporting the Unix operating system. A second XFTP prototype is currently being built as a 32-bit application under Microsoft Windows.

2.1 Why XFTP Uses Multiple Connections

Using multiple TCP connections to transfer data can increase overall throughput for several reasons. The first, and most obvious, is that a given TCP connection has a maximum throughput that can be determined using the formula [Pos81]:

$$throughput_{max} = \frac{receive\ buffer\ size}{round\ trip\ time} \qquad (1)$$

In the case of the satellite connections that we tested using XFTP[3], this yields

$$\begin{aligned} throughput_{max(satellite)} &= \frac{24KBytes}{560ms} \\ &\approx 44000\frac{bytes}{second} \qquad (2) \end{aligned}$$

Another factor that affects satellite throughput is TCP's slow start algorithm. To avoid instantly bombarding the network with the traffic from a new connection, TCP slowly increases the amount of data sent over a new connection

[2]Many modified versions of FTP have been proposed and the name XFTP has been used previously to describe other systems. Throughout this paper, XFTP refers only to the prototype system described herein.

[3]With standard TCP, the receive window size can be as large as 64 KBytes. Typical Unix FTP implementations use a receive window size between 4 KBytes and 24 KBytes. Unless otherwise noted, all of our experiments were conducted using 24 KByte receive windows.

[Jac88]. Because of the long delay over satellite circuits, TCP requires approximately 3.5 seconds (assuming a 500 millisecond round trip time and 512 byte segments) to achieve maximal throughput. Using multiple connections allows XFTP to multiply the maximum throughput of a single TCP connection to use the entire bandwidth of a circuit without increasing the slow start startup costs.

2.2 Changes to the FTP Protocol

The FTP protocol specification defines the messages exchanged between the FTP client and server applications across the control connection. To support multiple TCP connections, we modified the FTP protocol by adding two new message primitives[AO96a]:

MULT
The MULT command serves as a question from the FTP client application to the FTP server application. If the server responds to the MULT command with an error, then the client application knows that the server application does not support multiple connections and the client uses a single TCP connection to transfer data, otherwise the answer includes the maximum number of concurrent TCP connections that the server is willing to accept.

MPRT The MPRT command is analogous to FTP's existing PORT command. With MPRT, the client can select multiple ports, up to a limit imposed by the server and specified by the server as the response to the MULT command. MPRT takes the form:

`MPRT` $\texttt{ipaddrbyte}_1$`, ...` $\texttt{ipaddrbyte}_4$`,\`
$\texttt{port}_1$`, ...` $\texttt{port}_n$

To support multiple connections, we added the new FTP client command MULT, as follows:

MULT [n]
The MULT user command requests multiple-connection transfer mode. Without an argument, the default value is used (currently 4). Note that the actual number of connections used is also limited by maximum values in both the client and the server. We expect that users will normally enable multiple connections using the MULT command with the default number of connections.

2.3 Dividing a File across Multiple Connections

When multiple TCP connections are available to transfer a single file, the file must be divided into chunks that can be sent over the various connections. This process is sometimes called *file striping*, after the commonly-used practice of *disk striping*, in which a single file is stored on multiple physical disks to increase throughput. File striping, in the context of TCP connections, however, must be carefully designed to avoid problems.

A simple, obvious approach to sending a file F across n connections is to divide the file into n blocks, $F_1 ... F_n$. Block F_i is then transferred using connection C_i. This simple approach can perform badly, however, when the TCP connections do not all exhibit the same throughput. Those connections that observe congestion will slow down, whereas connections that are less affected by congestion will speed up and consume more network resources. As a result, if each of n connections transfers the same amount of data, some of the connections will take longer to complete, reducing efficiency.

To send a single file using n connections, XFTP divides the file into m 8k records (where $m \gg n$). The sender of the file (which can be either the FTP client or server) reads the file from local storage one record at a time and sends each record over whichever connection has resources available to accept it, determined using disjunctive wait (select) and non-blocking writes. By dividing a file in this way, XFTP can keep each of the connections busy until the entire file has been transferred, even if the connections do not all transfer the same amount of data. To reassemble the data into the correct order, XFTP prepends a 4-byte sequence number to each record.

3 Results

Our initial experiments with XFTP verified that using multiple connections to transfer a file yields higher throughput than using a single connection [AOK95]. Unfortunately, the throughput improvement was very sensitive to the number of connections, as shown in figure 1, below. This important observation led us to study the behavior of TCP in a satellite environment in much greater detail, as summarized in the sections that follow.

3.1 Multiple Connection Efficiency

To test the overhead of managing multiple TCP connections, we conducted tests using several different settings for the TCP receive window size and the number of data connections. The results are summarized in figure 2, showing throughput as a function of the effective window size for transfers of 5MByte files. In this case, the achievable throughput is clearly dependent only on the effective window size, indicating that the extra overhead of multiple connections is negligible.

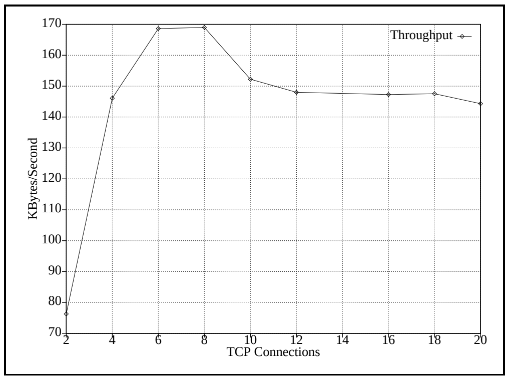

Figure 1: Throughput vs Number of Connections
This figure shows the throughput achieved during an emulated 5 MByte file transfer as a function of the number of connections used.

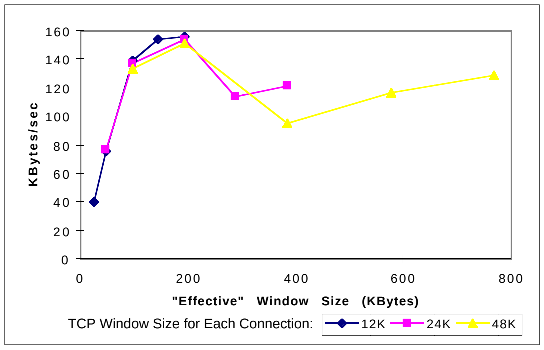

Figure 2: Throughput Improvement vs the Effective Window Size
This figure shows that the throughput achieved is dependent only on the effective window size (number of connections times window size)

3.2 Number of Connections vs Throughput

With the maximum throughput of the T1 satellite circuit being approximately 192,000 bytes/second ($\approx$ 1.5 Mbit/second) and the maximum throughput per TCP connection being approximately 44,000 bytes/second (from equation 2), quick division might lead one to expect maximum throughput with 4 connections[4]. Notice in figure 1, however, that the maximum throughput is achieved using 6 to 8 connections; after that point, throughput drops as more connections are used. These results are consistent with the results using MFTP described in Hahn [Hah94] and Iannucci and Lekashman [IL92].

One reason that XFTP requires more connections than calculated above to achieve maximum throughput is that the round trip time increases with the number of connections[5]. As the amount of data on the satellite circuit increases, the routers that connect the various networks are forced to enqueue more and more of the segments. This queuing behavior delays the TCP segments, effectively increasing the round trip time and decreasing the throughput. Using the NASA ACTS network, we observed round trip times as high as 1.5 seconds, as shown in the graph in figure 3.

To understand the throughput decrease on the right side of figure 1 (as more than 8 connections are used), one must investigate TCP's congestion control mechanism. Routers between the XFTP client and the XFTP server are forced to enqueue more and more data as we increase the number of connections. Because the routers are only able to enqueue a finite amount of data, they are eventually forced to discard incoming TCP segments as the data rate increases. Looking back at figure 1, 8 connections could not generate enough segments per second to cause queue overflow in the routers; at 10 connections, overflow loss begins to occur. Although TCP compensates for the lost data by retransmitting segments that were not acknowledged by the receiver, it also uses segment loss as an indication of congestion (too much data in the network) and decreases the rate at which data is sent into the network[6].

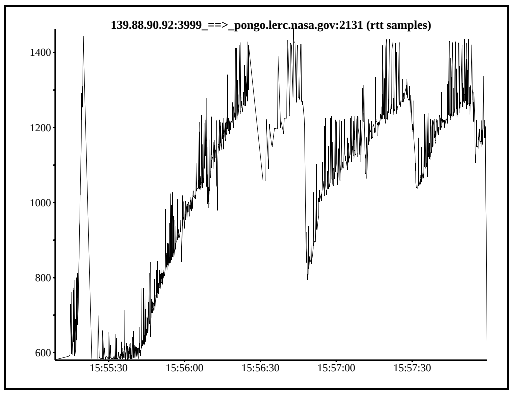

Figure 3: Variations in RTT Over Time
This figure shows the increase in RTT over time for a multiple-connection file transfer using the NASA ACTS network.

The interaction of slow start, multiple TCP connections, and segment loss due to router queue overflow accounts for the decreased throughput. As TCP's slow start algorithm sends more data into the network, intervening

[4] $192,000\frac{bytes}{second} \div \frac{44,000\frac{bytes}{second}}{connection} \approx 4 connections$

[5] TCP *slow start* also accounts for some of the decreased performance. Because slow start is only performed at the beginning of a connection and after some loss events, it is less of a factor in decreased throughput for large file transfers (assuming low segment loss) than is increased round trip times.

[6] TCP's congestion control mechanisms are more complex than explained here. Jacobson [Jac88] provides the original theoretical work and Stevens[Ste94] provides a good overview.

router queues eventually overflow causing the loss of segments belonging to some of the TCP connections. The affected TCP connections initiate congestion avoidance and decrease the rate that they send segments into the network. An example of this behavior can be seen, indirectly, in the previous figure (figure 3) showing RTT over time. Notice that the round trip time quickly grows to approximately 1.5 seconds[7] and then drops back to the base RTT as several of the TCP connections initiate congestion control. This same behavior can be seen four other times in the same figure as the RTT peaks. In these later cases, however, the various TCP connections are no longer increasing their throughput in sync with each other and the resulting segment loss affects fewer connections, resulting in less RTT fall-off after the loss event.

3.3 Application-Level Congestion Avoidance

XFTP uses an adaptive algorithm to control the number of TCP connections being used over time based on changes in the observed round trip time. The user chooses an initial value for the maximum number of connections to use for each file transfer. The XFTP sender (the side sending the file, either the client or the server) uses information gathered by using UDP echo datagrams[8] to determine the current round trip time. Using these round trip time samples, XFTP either increases or decreases the number of TCP connections currently in use. XFTP's application-level congestion control algorithm uses two RTT parameters, α and β[9]. When the observed round trip time exceeds β, XFTP reduces the number of connections in use by half. When the round trip time falls below α, one more connection is added (up to the maximum specified). Using experimentally-obtained values for α and β, one version of XFTP was able to obtain the throughput shown in figure 4. Notice in this figure that, using the new algorithm, throughput continues to increase with the number of connections rather than decreasing as it did with the original version of XFTP.

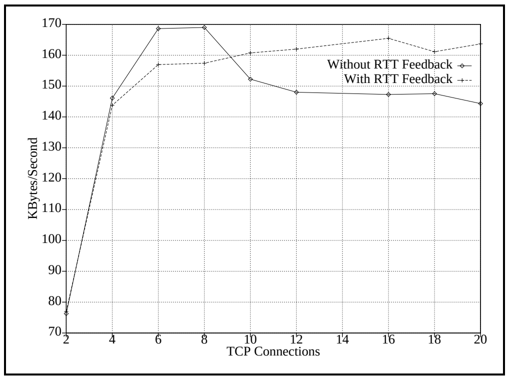

Figure 4: Throughput vs Number of Connections (New Algorithm)
This figure shows the throughput achieved during a 5 MByte file transfer as a function of the number of connections used with both the original version of XFTP and the new version that adapts the number of connections used during the transfer as a response to changes in round trip time.

[7]In our experimental network, router queues were large enough to build up a one second segment delay before dropping segments

[8]The XFTP sender generates a 12-byte UDP datagram containing a 4-byte sequence number and an 8-byte Unix timestamp and sends it to the UDP echo port on the machine that is receiving the file (up to 10 times per RTT).

[9]The idea of using bounding limits for RTT to control TCP behavior is similar to (and was inspired by) the work with TCP Vegas described in Brakmo, O'Malley, and Peterson [BOP94], although the TCP Vegas scheme operates inside the operating system within a single TCP stream.

3.4 Performance Over Links with Errors

The results reported to this point were obtained on an error-free link, where our tests had revealed no difference in throughput between a single large window, and multiple TCP connections creating an "effective" window of the same size. In a brief test using the ACTS network, we purposely degraded the RF performance of an earth station to create a link with bit error rates in the 1×10^{-7} to 1×10^{-5} range. File transfer throughput was evaluated using different numbers of connections to create an effective window of 192 KBytes. Figure 5 shows significantly better throughput when the effective window is created by a large number of connections. We speculate that the multiple connections simulate a type of selective acknowledgment, but further studies of this effect are now underway.

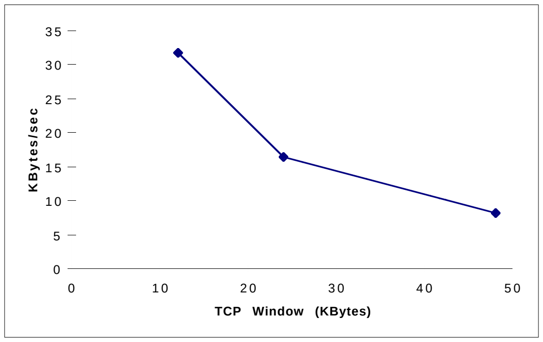

Figure 5: The Effect of Link Errors on Transfer Efficiency
The three points in this graph represent the throughput achieved using a 192 KByte effective window composed of either 16 connections each with window size 12 KBytes, 8 24 KByte connections, or 4 48 KByte connections.

4 Conclusions

The goal of this research was to investigate the previously-reported problems using TCP to transfer data over high-bandwidth, long-delay satellite circuits. Our work with XFTP showed that using multiple connections allowed a user application to achieve high TCP throughput across a satellite circuit. Unfortunately, throughput increases are shown to be very sensitive to the number of TCP connections used.

Experiments over non error-free circuits show that TCP's standard acknowledgment and retransmission mechanisms can adversely affect performance over satellite networks. Unlike terrestrial networks in which link errors are rare (relative to congestion loss), satellite circuits are more susceptible to periodic link errors.

We have also shown that an application-level, adaptive algorithm monitoring changes in the observed round trip time can adapt the number of TCP connections in use at one time to avoid throughput penalties for using too many connections.

Finally, and most importantly, the XFTP application is merely an interim solution to TCP's inefficiencies over satellite circuits. Recent advances in TCP such as increased window sizes, modified congestion control mechanisms, and new (selective) acknowledgment schemes may be able to improve TCP's satellite performance. Unfortunately, these mechanisms are relatively new and haven't been widely studied over satellite links.

Until these new mechanisms are verified and made more widely available, XFTP and the lessons learned from it will be useful in the satellite community. The need for further work in this area is clearly indicated.

References

[AO96a] Mark Allman and Shawn Ostermann. Multiple Data Connection FTP Extensions. Technical report, Ohio University, 1996. (in preparation).

[AO96b] Mark Allman and Shawn Ostermann. One: The Ohio Network Emulator. Technical report, Ohio University, 1996. (in preparation).

[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*, Cleveland, OH, September 1995. NASA Lewis Research Center.

[BOP94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of ACM SIGCOMM '94.*, pages 24–35, August 1994.

[Com95] Douglas E. Comer. *Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture.* Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1995.

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

[Hah94] Jonathan Hahn. MFTP: Recent Enhancements and Performance Measurements. Technical Report RND-94-006, NASA Ames Research Center, NAS Systems Development Branch, June 1994.

[IL92] David J. Iannucci and John Lekashman. MFTP: Virtual TCP Window Scaling Using Multiple Connections. Technical Report RND-92-002, NASA Ames Research Center, NAS Systems Development Branch, January 1992.

[Jac88] V. Jacobson. Congestion Avoidance and Control. In *Proceedings ACM SIGCOMM '88*, pages 314–329. ACM, August 1988.

[JBB92] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[JBZ90] V. Jacobson, R. Braden, and L. Zhang. TCP Extension for High-speed Paths, October 1990. RFC 1185.

[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.

[MM96] Matthew Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *ACM SIGCOMM*, August 1996.

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

[PR85] J. Postel and J. Reynolds. File Transfer Protocol (FTP), October 1985. RFC 959.

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