FILE TRANSFER IN FCS NETWORKS USING TRANSPORT LAYER MULTISTREAMING*

Sourabh Ladha, Paul D. Amer, Janardhan Iyengar, Armando L. Caro Jr.
Protocol Engineering Lab
Computer and Information Sciences
University of Delaware
{ladha, amer, iyengar, acaro}@cis.udel.edu

ABSTRACT

We identify overheads associated with the current FTP protocol, which uses TCP for transport. We discuss why using TCP to avoid such overheads puts a burden on the application. Unlike TCP, SCTP allows transport layer multistreaming within a single association. We present two modifications to FTP, which use SCTP multistreaming in a "TCP-friendly" manner. Our modifications avoid the identified overheads in the current FTP protocol without introducing complexity at the application. We have implemented these modifications in the FreeBSD environment. Extensive emulations have been carried out to compare the current FTP design with the modifications introduced, in terms of end to end latency. Our results indicate dramatic improvements in transfer time and throughput achieved between two endpoints under certain network conditions.

1. INTRODUCTION

File Transfer Protocol (FTP) [2] is one of the most common protocols for bulk data transfer. Over the years there has been a steady decline in the use of FTP, chiefly attributed to its bulky nature as well as due to inefficiency seen in end to end delay statistics. Several extensions have been proposed to modify FTP [e.g. 6, 8] but none of them aim at reducing file transfer latency. FTP uses TCP [3] for transport. We have identified reasons why modifying FTP in order to reduce latency overheads has been difficult, mainly due to the existing TCP semantics which put a burden on the application by introducing complexity. One of the ill affects of this has been that several FTP implementations aiming at performance enhancement end up using multiple TCP connections to achieve better throughput. This approach has been regarded as "TCP-unfriendly" [9] as it allows the application to gain unfair share of bandwidth at the expense of other network flows and disturbs the network equilibrium. Future Combat Systems (FCS) networks require crucial information to be delivered between endpoints with least observed latency. Keeping this in mind, our research focuses on improving end to end latency and throughput in the event of file retrievals using FTP.

With the introduction of innovative transport protocols, applications have been exposed to a host of new features. We present modifications to FTP using Stream Control Transmission Protocol (SCTP) [11] as transport. SCTP is a standards track transport layer protocol in the IETF. Like TCP, SCTP provides a full duplex, reliable transmission service to the application. SCTP has a rich and unique feature set suitable for a host of applications. This paper focuses on the use of one such feature, multistreaming. SCTP multistreaming allows logical division of an association into streams with each stream having its own delivery mechanism. This decouples data delivery and transmission and prevents Head of the Line Blocking problem. All the streams within a single association share the same congestion and flow control parameters. In this paper we identify how SCTP, and in particular SCTP multistreaming can benefit the application in reducing overheads. Moreover, SCTP's support for transport layer multihoming provides network fault tolerance which is crucial for survivability and persistent on-the-move sessions in FCS networks.

The remainder of the paper is organized as follows. Section 2 quantifies the overheads in current FTP design with a detailed description of each. Some other inefficiencies of FTP have also been noted. Section 3 talks about possible solutions to eliminate the overheads using TCP as the transport and the drawbacks of each of them. Section 4 and 5 talk about the changes introduced in FTP using SCTP multistreaming and a description of how these designs can reduce some of the overheads. Section 6 presents the experimental results and discussion. Section 7 concludes the paper.

*Prepared through collaborative participation in the Communication and Network Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

2. INEFFICIENCIES IN THE FTP PROTOCOL

We quantify the overheads associated with the FTP protocol. The following discussion also describes some of the implicit inefficiencies in the FTP design.

2.1 Separate Data and Control Connections

A. The out-of-band signaling approach in FTP has consequences in terms of end to end latency. The control connection is periodic in nature and typically remains in the slow start phase of TCP congestion control [7]. Thus a loss over the control connection can only be detected by a timeout.

B. Since the data and control are on separate connections hence there is an extra overhead, in terms of data connection setup-teardown, of at least 1.5 Round Trip Time (RTT) (1RTT for setup and 0.5 RTT for teardown). Moreover the number of control blocks the server has to maintain increases two folds.

C. Over the past years there have been considerable discussions on the security issues involved in FTP, attributing to the data connection information (IP address, port number) being send out on the control connection to assist the peer in establishing a data connection. This causes problems for NATs and firewalls which have to monitor and translate addressing information over the control connection [6].

2.2 Non-persistence of the data connection

A. The non persistence of data connection causes connection setup-teardown overheads (at least to the order of 1.5 RTT) each time a file transfer or directory listing request is serviced. [13] talks about how queuing delays can increase the RTT several folds. Thus to improve end to end delays, by avoiding network latency, extra round trips must be minimized. Moreover processing overhead at the end hosts is also added up in the event of allocating resources for each new connection.

B. Repetitive probing of the congestion window (cwnd) for each data connection, particularly repeated slow start phase for every data transfer process. Each connection must begin by probing for the available bandwidth before it can hover around the steady state cwnd. Moreover a loss in the slow start phase could lead to a timeout at the server. Figure 1 graphically shows the nature of this overhead.

C. Extra round trips in exchange of similar commands for each data connection established (at least 1RTT in the event of PORT command and 200 reply).

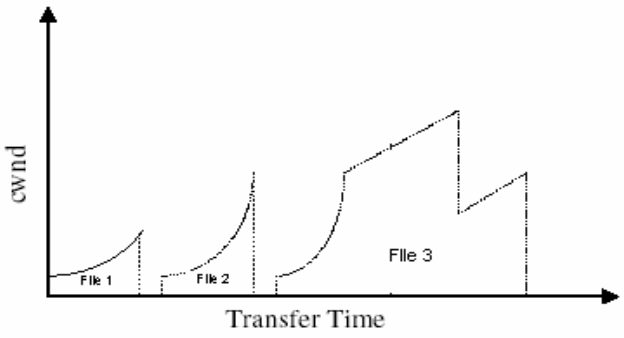

Figure 1: Expected cwnd evolution in the event of multiple file transfers in FTP

D. In the face of multiple small file transfers in an FTP session the server ends up having many connections lying in the TCP TIME-WAIT state. [12] argues that per-connection memory load from TCP can adversely affect connection rate and throughput in the event of multiple connections lying in the time-wait state.

3. POSSIBLE SOLUTIONS AND DRAWBACKS

We describe some of the possible solutions that try to avoid the overheads using TCP as transport. The drawbacks associated with each solution are presented.

A. Single persistent connection for control and data

Drawbacks: TCP provides a byte stream oriented service and does not differentiate between the different types of data it transmits over the same connection. Using the current TCP semantics, this solution will burden the application to insert application layer markers to differentiate between control and data. This carries complexity to the application layer and adds to the processing. Control and file data in an FTP session are logically different types of data. Assigning same connection to them will introduce head of line blocking problem and unnecessary retransmissions of data in the event of a control reply getting lost or delayed in the network.

B. Separate data and control connections with persistent data connection

Drawbacks: Due to the sequential nature of commands over the control connection, the data connection will remain idle in between file transfers in the event of multiple files transfers which will lead to closing of the congestion window and repetitive probe for the available bandwidth. Moreover this approach suffers the overheads listed in section 2.1 of this paper.

C. Separate data and control connections with persistent data connections and command pipelining over the control connection.

Drawbacks: This approach suffers from the overheads listed in section 2.1 of this paper.

Apart from the above "TCP-friendly" solutions, over the years implementations try to achieve better throughput using multiple TCP connections. This approach is not "TCP-friendly" and can adversely affect the network equilibrium. In the face of the drawbacks in the solutions listed and the overheads incurred, SCTP provides a TCP friendly approach which eliminates all the overheads listed in section 2, without complicating the role of the FTP application.

4. USING SCTP MULTISTREAMING IN FTP

The overheads identified are mainly attributed to the fact that FTP uses separate data and control connections. SCTP multistreaming allows us to use streams for control and data within a single association. Our modification uses this feature.

The FTP client establishes an SCTP association with the server. Two outgoing and incoming streams are established. *Stream 0* has been used for exchange of control commands and replies. *Stream 1* has been used as the data stream. In the event of multiple file retrievals issued by the user, the client sends out the requests on *stream 0* and receives the data on *steam 1* for each file in a sequential manner. The figure below shows the retrieval of a single file using this modification. The outgoing stream numbers for all the messages and data have been identified.

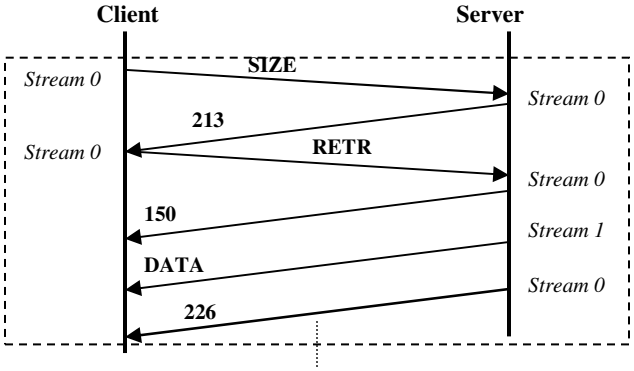

Figure 2: Instance of an FTP session using SCTP multistreaming

This approach has various advantages and avoids most of the overheads described in Section 2 except the following. In the event of multiple file transfers the subsequent file transfers will not be able to utilize the probed available bandwidth. [10] describes that *Max.Burst* must first be applied to recalculate the cwnd before sending out new data chunks. When transferring multiple files the client waits for the entire file to arrive before sending out the next file transfer commands. Since the flow of data transfer from the client to the sender is not maintained between consecutive file transfers, this leads to the closing of the cwnd. The next approach we present avoids this overhead using command pipelining in multiple file transfers.

5. ADDING COMMAND PIPELINING TO THE DESIGN

Even with the single connection for both data and control as seen in the previous section, consecutive file transfers incur the cost of closing of the cwnd as the SIZE, RETR commands for the in sequence file in the event of multiple file transfer is send after all the data for the previous file has been received by the client. This leads to a situation where the server's congestion window closes conforming to Section 6.1 of [10].

We present a solution which allows subsequent transfers to utilize the probed value of congestion window in the event of multiple file transfers. This solution uses command pipelining to ensure the flow of data to be maintained from the server to the client throughout the execution of the command. The only changes that have been made to add command pipelining are on the client side.

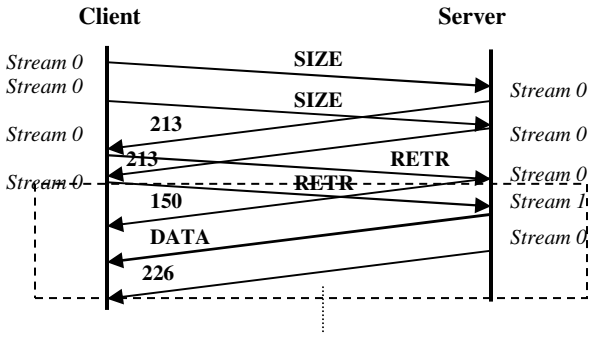

Figure 3: Instance of an FTP session using SCTP multistreaming with command pipelining

The client on parsing the name list of the files sends the SIZE command for each file at the same instance without waiting for the reply from the server. As soon as a reply for the SIZE command for a particular file is received the client sends out the RETR for that file. The client uses simple heuristics to determine whether the data coming in is a reply on the control stream or the file data on data stream. The figure below shows the timeline followed in this solution. The timeline in Figure 3 gives an example of multiple retrievals of two files. This approach overcomes all the drawbacks listed in section 2.

6. EXPERIMENTAL RESULTS

In this section we present results of emulations to measure the effect of end to end latency in file transfers. We report the effect of SCTP multistreaming and command pipelining in FTP.

We have used *netbed* [1] (an outgrowth of Emulab which provides integrated access to experimental networks) for our experiments. Three nodes have been used for each set of experiments with the client and server on two nodes and the third node acting as a router and shaping traffic between the client and the server nodes. The client and server nodes have 850MHz Intel Pentium III processors and are based on the Intel ISP1100 1U server platform. The FreeBSD kernel implementation of SCTP available with the KAME Stack [4] has been used on the client and server nodes. KAME is an evolving and experimental stack mainly targeted for IPv6/IPsec in the BSD based operating systems. An updated snapshot of the stack (KAME snap kit) is released every week. The snap kit of 14th October, 2002 has been installed on the client and server nodes. The router node runs *Dummynet* which simulates a drop tail router with a queue size of 50 packets and specified bandwidth, propagation delay and packet loss ratio. The queuing discipline, particularly the packet loss ratio has been varied to measure the impact on transfer latency.

We implemented the protocol changes by modifying the FTP client and server source code available with the FreeBSD 4.6 distribution. In our experiments we measured the total transfer time using the packet level traces as follows. The starting time was taken as the time the client sends out the first packet to the server following the user *"mget"* command. The end time was chosen as the time the 226 control reply from the server reaches the client for the last transfer. We measured three configurations with varying packet loss ratio: 0, 0.1 and 0.3. The bandwidth and propagation delay specified for the link from the client to the server and vice versa were 1Mb/s and 35ms. The sample size has been chosen such that a 90% confidence level is achieved with an acceptable error not more than one half of a second. The freely available *tcpdump* program (version 3.7.1) has been used to perform packet level traces. SCTP decoding functionality in *tcpdump* was developed in collaboration of two CTA supported labs (UD's Protocol Engineering Lab and Temple University's Netlab).

Figures 4-5 show the transfer times taken for a transfer for various file sizes. Each point in the graphs denotes the mean value of the time taken for multiple file retrievals using the *"mget"* user command in FTP aggregating to 10 files. Due to lack of space we have not been able to show all the experimental results. Our results compare four designs for FTP: (1) the current FTP protocol (over TCP), (2) the current FTP protocol design with SCTP as the transport protocol, (3) modified FTP protocol design which uses multistreamed SCTP, and (4) modified FTP protocol design which uses command pipelining over multistreamed SCTP.

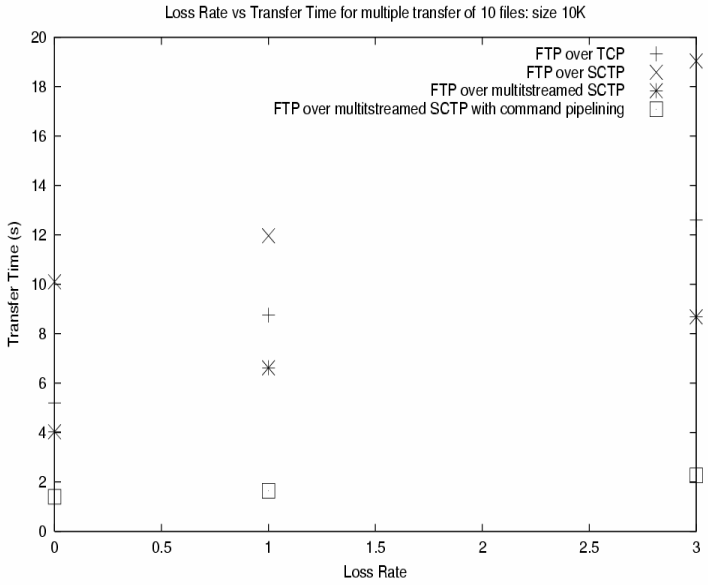

Figure 4: Latency improvements observed for smaller files of size 10K

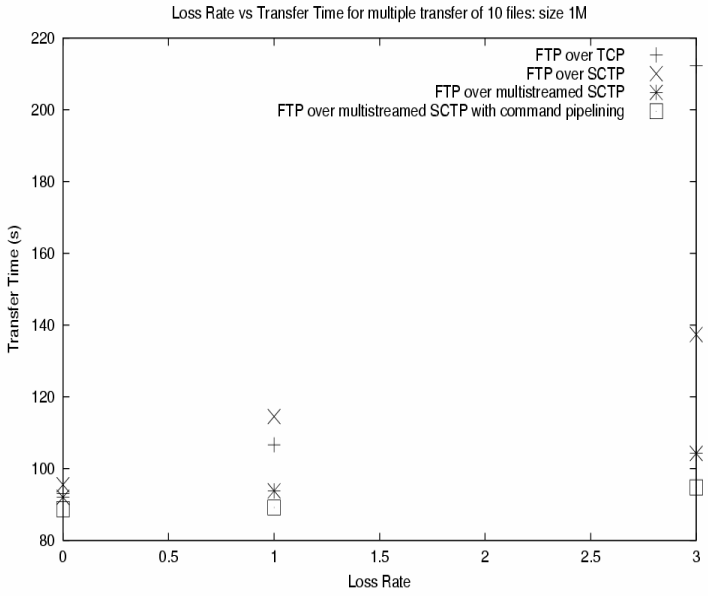

Figure 5: Latency improvements observed for large files of size 1M

As can be seen from the graphs, FTP over multistreamed SCTP with command pipelining outperforms FTP over TCP cutting the latency by more than half for higher loss rates. This matches our theoretical assumptions considering

the reduction of extra round trips and better utilization of available bandwidth. It can be seen form the graph that as the loss rate increases the relative improvement in transfer time increases. Even without the use of command pipelining there is a significant improvement in the end to end delay.

7. CONCLUSION AND FUTURE WORK

We have analyzed several sources of overheads in the FTP protocol which cause added latency in file transfers. We have proposed modifications to FTP that exploits the multistreaming feature of SCTP. Our experimental results confirm that adding SCTP multistreaming to FTP dramatically reduces latency of multiple transfers, uses the available bandwidth more effectively and reduces server load. Also, command pipelining adds additional benefit in improving end to end delay. FTP over multistreamed SCTP also solves a problem that the current FTP protocol faces with Network Address Translators (NAT) and firewalls in transferring IP addresses and port numbers through the control connection [6].

In the future we plan to make the design more comprehensive for the complete FTP specification and extensions added over the years since the release of RFC959. FCS Networks have typically less bandwidth available for communication. Moreover the Bit Error Rate (BER) in FCS Networks is high. We are currently doing experiments to observe the effect of SCTP multistreaming in low bandwidth, high BER channels. One of the weaknesses in our work is that we compare SCTP against New-Reno TCP. Since SCTP uses Selective Acknowledgements (SACK) to perform better loss recovery, this comparison may not be fair. We are currently investigating comparisons involving SACK TCP.

The benefits that SCTP's multistreaming feature provides are not limited to FTP. For example, web transfers using HTTP (Hypertext Transfer Protocol) can also benefit from command pipelining and aggregation of multiple transfers in a single association [13]. Also, SCTP's multistreaming provides a TCP-friendly mechanism for parallel transfers to improving user-perceived latency. Future work is to investigate the benefits of multistreaming in other applications such as web transfers.

ACKNOWLEDGEMENTS

We would like to thank Randall Stewart for providing support for the KAME stack implementation of SCTP. Also, we are grateful to Jay Lepreau and the support staff of Netbed (formerly known as Emulab), the Utah Network Emulation Testbed (which is primarily supported by NSF grant ANI-00-82493 and Cisco Systems) for making their facilities available for our experiments.

DISCLAIMER

REFERENCES

[1] B. White, L. Jay, S. Leigh, R. Robert, G. Shashi, N. Mac, H. Mike, B. Chad, J. Abhijeet. *An integrated experimental environment for distributed systems and networks.* Proceedings of the 5th Symposium on Operating Systems Design and Implementation, December 2002.

[2] J. Postel, J. Reynolds, *File Transfer Protocol (FTP).* RFC 959, Internet Engineering Task Force, October 1985.

[3] J. Postel, *Transmission Control Protocol.* RFC 793, Internet Engineering Task Force, September 81.

[4] KAME Project, www.kame.net.

[5] L. Coene, *Stream Control Transmission Protocol Applicability Statement.* RFC 3257, Internet Engineering Task Force, April, 2002.

[6] M. Allman, S. Ostermann, C. Metz, *FTP extensions for NATS and firewalls.* RFC 2428, Internet Engineering Task Force, September, 98.

[7] M. Allman, V. Paxson, W. Stevens, *TCP Congestion Control*. RFC 2581, Internet Engineering Task Force, April 99.

[8] M. Horowitz, S. Lunt, *FTP Security Extensions.* RFC 2228, Internet Engineering Task Force, October, 1997.

[9] S. Floyd, Fall, K., *Promoting the Use of End-to-End Congestion Control in the Internet*. IEEE/ACM Transactions on Networking, August 1999

[10] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, M. Tuexen, *Stream Control Transmission Protocol (SCTP) Implementers Guide.* Internet Draft draft-ietf-tsvwg-sctpimpguide-07.txt, October, 2002. Work in progress

[11] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, *Stream Control Transmission Protocol.* RFC 2960, Internet Engineering Task Force, October 2000.

[12] T. Faber, Joe Touch, Wei Yue, *The TIME-WAIT state in TCP and Its Effect on Busy Servers*. Proc. Infocom '99.

[13] V. Padmanabhan, J. Mogul, *Improving HTTP latency*.