IMPROVING FILE TRANSFER IN FCS NETWORKS*

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

ABSTRACT

We compare the performance of two transport protocols, SCTP and the New-Reno variant of TCP, for file transfers in two FCS networking scenarios. We argue why SCTP is better suited for file transfers in a network prone to resource failures. To measure performance, we implemented FTP over SCTP in a FreeBSD environment. Our results indicate for our tested path configurations, (1) using SCTP as the transport for FTP significantly reduces file transfer time, and (2) FTP over SCTP is more robust to losses.

1. INTRODUCTION

Future Combat Systems (FCS) networks require crucial information to be delivered between endpoints with minimal delay. This places three key requirements on file transfers: (1) fault tolerance to resource failures, (2) robustness to loss events, and (3) efficient bandwidth utilization to maximize throughput. Most applications use TCP [11] to provide end-to-end reliability. Unfortunately, TCP does not support fault tolerance at the transport layer. One of the current additions to the suite of transport protocols has been the Stream Control Transmission Protocol (SCTP) [14]. SCTP is a standards track transport layer protocol in the IETF (Internet Engineering Task Force). Like TCP, SCTP provides a full duplex, reliable transmission service to the application. In addition, SCTP also supports transport layer multihoming, a key feature required for network fault tolerance, which is crucial for survivability and persistent on-the-move sessions in FCS networks. Having noted that SCTP multihoming provides for network fault tolerance to resource failures, this paper focuses on the evaluation of performance of SCTP for the other two requirements in an FCS networks setting.

File Transfer Protocol (FTP) [12] is one of the most common protocols for bulk data transfer. FTP uses TCP to provide end-to-end reliability. We have implemented a version of FTP in a FreeBSD environment using SCTP as the transport. A variety of path configurations were used to show the performance benefits of file transfers using SCTP through controlled experiments. We present the results of two such configurations that would be of interest to FCS networks - an Unmanned Air Vehicle (UAV) communication characterized by high bandwidth, and low delay paths; and a geosynchronous satellite communication characterized by low bandwidth, high delay paths. Our results indicate that irrespective of the path configurations, FTP over SCTP outperforms FTP over TCP. Moreover as the loss rate increases on the path, FTP over SCTP proves to be more robust.

This paper only presents the performance implications of using SCTP as the transport for FTP without introducing any changes in the FTP syntax or semantics. A separate body of work currently underway in the Protocol Engineering Lab (PEL) further reduces overhead in FTP using *multistreaming,* another of SCTP's unique transport services. Multistreaming aggregates FTP control and data connections in a single SCTP association and uses command pipelining for multiple file transfers [8].

This paper is organized as follows. Section 2 gives insight into the unique transport services of SCTP. Section 3 outlines the methodology used for experiments. Section 4 presents the results and analysis. Section 5 concludes the paper and presents ongoing and future work in this field.

2. BACKGROUND

SCTP addresses shortcomings of TCP by providing additional transport services to the application. We summarize some of the key features and services that SCTP provides to the Upper Layer Protocol (ULP), which

*Prepared through collaborative participation in the Communication and Network Consortium sponsored by the U.S. Army Research Laboratory under the CTA 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.

give incentive for emerging applications to use SCTP at the transport.

Resistance to blind Denial of Service (DoS) attacks: The connection establishment phase in SCTP authenticates the peers using a cookie mechanism. Thus resources at a receiver's system are not reserved for the incoming association until the peer indicates through the use of cookie that it is a valid endpoint. TCP on the other hand is vulnerable to such attacks.

Selective Acknowledgement (SACK): A TCP sender (without SACK) uses calculated guesses to determine which packets were received correctly at the receiver. TCP-SACK [9] added robustness to this mechanism by having the receiver indicate explicitly through the SACK option fields if it had received a segment out of order. The SACK mechanism in SCTP is derived from TCP, but provides more information and a faster loss recovery. The number of SACK blocks in TCP is limited to three or four. In SCTP there is no limit on the number of such blocks. The congestion control response and loss recovery mechanisms based on the SACK reports is more robust in SCTP than in TCP.

Multihoming: A host is multihomed if it can be addressed by multiple IP addresses [4]. TCP does not support multihoming. Any time either endpoint's IP address becomes inaccessible, perhaps due to interface failure, radio channel interference, or moving out of range, TCP's connection will timeout and abort, thus forcing the application or user to recover. On the other hand, SCTP has a built-in failure detection and recovery system, known as *failover,* which allows associations to dynamically send traffic to an alternate peer IP address when needed without interrupting the ULP. This feature provides network fault tolerance crucial for the performance of FCS networks.

Multistreaming: Multistreaming within an SCTP association separates flows of logically different data within a single association. This separation removes a burden from the application, by allowing it to identify semantically different flows of data, and having the transport layer "manage" these flows (as one would argue should be the responsibility of the transport layer, not the application). Each stream has an independent delivery mechanism, thus allowing SCTP to differentiate between data delivery and reliable data transmission.

3. METHODOLOGY

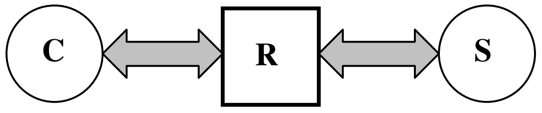
We used controlled experiments to compare the performance of file transfer using TCP and SCTP. Simulations using ns version 2.1b8 [10] were done to verify the experimental results. Through experiments, we were able to capture effects of connection setup-teardown overheads, command exchanges before a file transfer begins, and different data and control connections on the total transfer time. The following discussion explains the methodology used.

Approach: We performed experiments for a varied set of path parameters with the metric for evaluation as the total transfer time observed in file transfers.

- *Bandwidth-Delay Configuration*: We present the results of two configurations: (256Kbit/s, 125ms), (3Mbit/s, 1ms). Both the client to server and server to client paths share common characteristics.

- *Packet Loss Ratio (PLR)*: The PLRs studied were (0, .01, .03, .06, and .1). Each value represents the loss percentage for both the client to server and server to client paths. A uniform probability distribution was used to emulate packet loss.

Experiments: We used *Netbed* [16] (an outgrowth of Emulab), which provides integrated access to experimental networks for our experiments. Three nodes were used for each set of experiments, one for the FTP client (C) and one for the FTP server (S). The third node was used as a router (R) for shaping traffic between the client and the server. Figure 1 shows the experiment topology.



Bandwidth-Delay = {3Mbps-1ms, 256Kbps-125ms}
Queue Size at R = 50 packets
Loss Rates= {0, .01, .03, .06, .1}

Figure 1: Experiment Topology

The client and server nodes were 850MHz Intel Pentium III processors, and based on the Intel ISP1100 1U server platform. All three nodes ran FreeBSD-4.6. The FreeBSD kernel implementation of SCTP available with the KAME Stack [7] was used on the client and server nodes. KAME is an evolving and experimental stack mainly targeted for IPv6/IPsec in BSD based operating systems. An updated snapshot of the stack (KAME snap kit) is released every week. We used the snap kit of 14Oct02. The router node ran *Dummynet* [13], which simulates a drop tail router with a queue size of 50 packets, and specified bandwidth, propagation delay and packet loss ratio. The path parameters as described earlier were varied to measure the impact on transfer time.

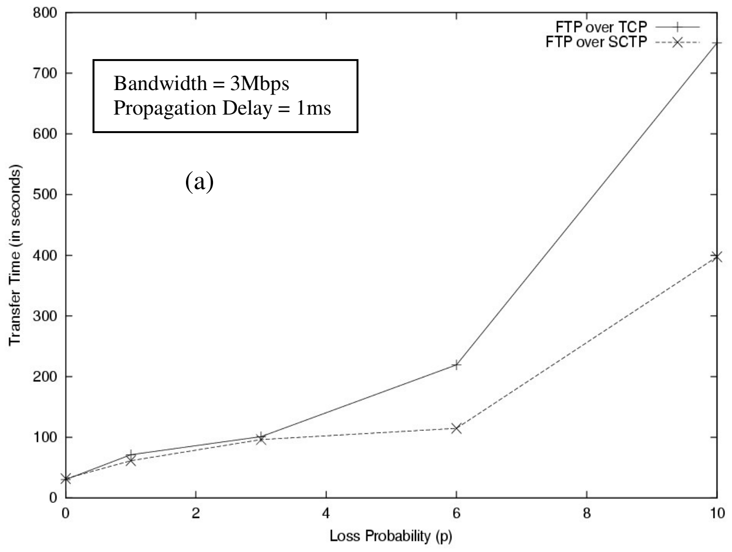
We implemented protocol changes by modifying the FTP client and server source code available with the FreeBSD 4.6 distribution. We measured the total transfer time using packet level traces as follows. The starting time was taken as the time the client sends out the first packet to the server following the user's *"mget"* (*mget* command allows for transfer of multiple files) command. The end time was the time a "226 control reply" from the server reached the client after the last transfer indicating the completion of the *mget* operation. We thus captured the effects of multiple phases of connection setup-teardown and command exchanges involved in transferring multiple files from the server to the client. Each combination of parameters (2 configurations x 5 PLR) was run multiple times to achieve a 90% confidence level for the total transfer time. *Tcpdump* [15] (version 3.7.1) was used to perform packet level traces. SCTP decoding functionality in *tcpdump* was developed in collaboration of PEL and Temple University's Netlab.

4. RESULTS AND DISCUSSION

The results presented in this section represent transfer time taken vs. the loss rate on the path. Since both the forward and reverse paths share common characteristics, the loss of an ACK is as common as the loss of the data packet although the loss of an ACK often has minimal effect since ACKs are cumulative.

Using experiments we have captured the effect of multiple file transfers in FTP. Thus our results include the effect of connection setup-teardown, and command exchanges in FTP. Due to the cookie mechanism, SCTP has one extra "leg" in the connection establishment phase. To be fair in our comparisons, incorporating connection setup was particularly important. We have used the FTP multiple get (*mget*) command to transfer files. The number of files transferred for each experimental run was ten. Thus each run involved eleven connection establishments where the first connection was used for name list transfer and the remaining ten connections were used for sequential file transfers.

Figure 2(a) shows the total transfer time for varying loss rates for a 3Mbps-1ms configuration (UAV-like communication). The TCP variant used in our experiments is TCP New Reno [6]. Each of the ten files transferred was of size 1MB. At lower loss rates, FTP over SCTP takes marginally more time than FTP over TCP. But as the loss rate increases, FTP over SCTP tends to be more robust and recovers better from losses thus reducing transfer time. This result can be seen at 10% loss where FTP over TCP takes approximately 350 seconds more as compared to FTP over SCTP.



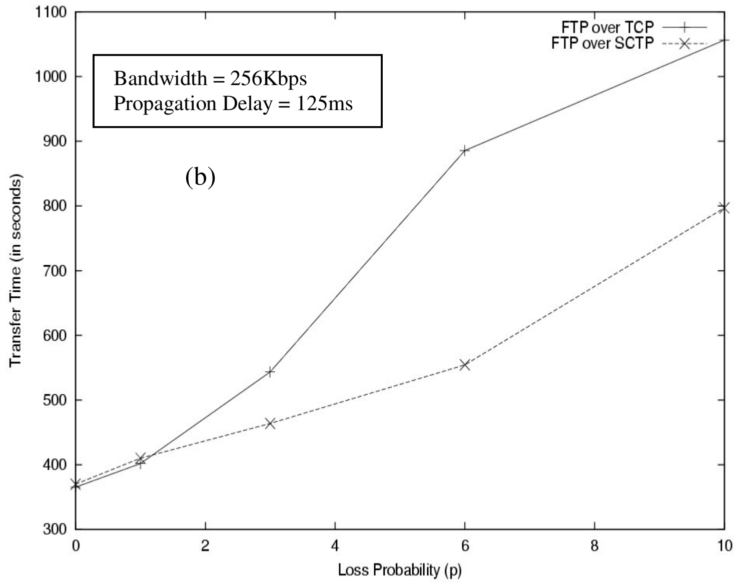


Figure 2: Transfer Time vs. Loss Probability for multiple transfer of 10 files of size 1MB each

Figure 2(b) shows the results for a satellite type communication setting. We see that the performance curve in Figure 2(b) follows closely the pattern of Figure 2(a). Thus for smaller loss rates, the per packet overhead[1] in FTP over SCTP lead to near about same performance, but for higher loss rates, FTP over TCP takes significantly more time than FTP over SCTP.

[1] The current SCTP implementation of the BSD KAME stack does not have extra per-packet overheads as compared to TCP.

Irrespective of the bandwidth-delay configurations, FTP over SCTP performs better than FTP over TCP.

We performed transfers of various file sizes to asses the performance benefits of using SCTP as the transport for not only a bulk data transfer protocol as FTP but also for short flows. We show the transfer results of files of size 200K for similar configurations as shown in the above results. As seen from Figure 3, for lower loss rates SCTP as transport performs closely to TCP but as the loss rate increases SCTP starts to outperforms TCP.

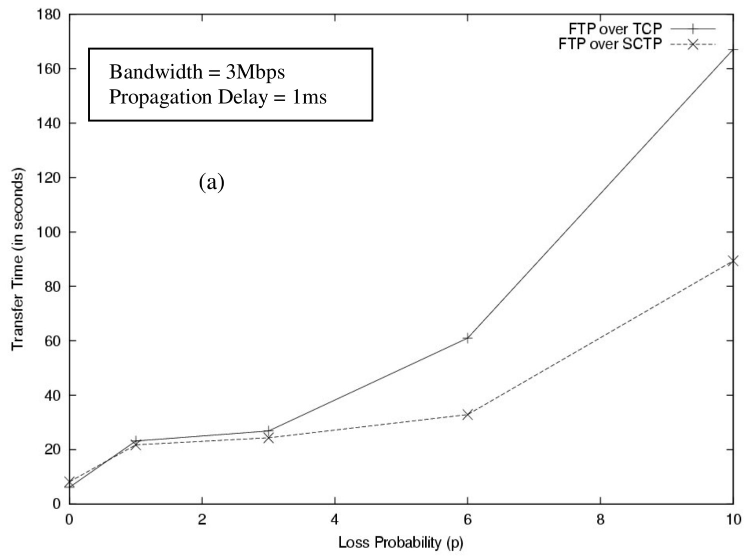
Our experimental results show that:

- Irrespective of configuration, SCTP outperforms TCP in terms of the total time for file transfers.
- SCTP's congestion control mechanisms are more robust than TCP's, while still conforming to the Additive Increase Multiplicative Decrease (AIMD) algorithms recommended for congestion control protocols.
- The improvement in file transfers using SCTP is directly proportional to the number of files transferred.
- The improvement in file transfers using SCTP is directly proportional to the size of the file being transferred.
- More significant gain of using SCTP as the transport is seen at loss rates increase. And the difference in file transfer time using TCP and SCTP is directly proportional to the path loss rate.

5. CONCLUSIONS AND FUTURE WORK

In this paper we show that SCTP proves to be a better transport for FCS-like networks, which are prone to losses and failures. In summary we conclude that:

- Using SCTP as the transport for FTP improves the transfer time and throughput for paths suffering from loss, irrespective of the bandwidth-delay configuration.
- For lower loss rates, the per-packet overhead in SCTP results in marginally lower throughput as compared to TCP.
- SCTP multihoming provides an implicit advantage through network fault tolerance to an FCS networks. Multihoming allows for transparent transfer of files in FTP even when one of the paths becomes inaccessible.



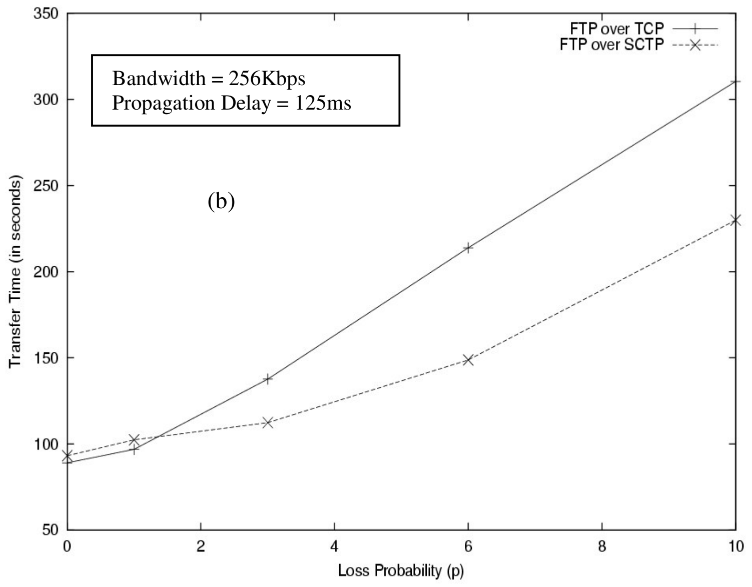


Figure 3: Transfer Time vs. Loss Probability for multiple transfer of 10 files of size 200KB each

We point out certain limitations of the work presented in this paper:

- We have used a uniform loss distribution model for emulating losses on the path. A more realistic scenario would include burst losses or multiple losses in a window. These loss distributions seem a natural extension to this paper.
- One of the weaknesses in our work is that we compare SCTP against New-Reno TCP without SACK. Since SCTP uses Selective Acks (SACK) to perform better loss recovery, this comparison may be unfair. We are currently investigating comparisons involving TCP with SACK.

- A number of recent additions to the TCP congestion control [2, 3] fine-tune TCP's behavior to result in faster recovery from loss events and lesser timeouts. Another extension to our work could be to take such TCP fine tunings into consideration and re-evaluate simulations.

In the process of experimentation we analyzed a number of inefficiencies in the design of FTP. Our current work involves making FTP more efficient by using SCTP multistreaming. Multistreaming involves mapping the existing multiple connection semantics in FTP to SCTP streams, and using a single SCTP association for the entire FTP session. In general, we are working on migrating application protocols to exploit SCTP features to result in a better performance.

ACKNOWLEDGMENTS

We thank Randall Stewart for providing support for the KAME stack implementation of SCTP. 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.

REFERENCES

[1] M. Allman, V. Paxson, W. Stevens, *TCP Congestion Control*. RFC 2581, 4/99.

[2] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, *Early Retransmit for TCP*. draft-allman-tcp-early-rexmt-00.txt (work in progress), 2/03.

[3] M. Allman, H. Balakrishnan, S. Floyd, *Enhancing TCP's Loss Recovery Using Limited Transmit.* RFC 3042, 1/01.

[4] R. Braden. *Requirements for Internet Hosts Communication Layers.* RFC1122, 10/89.

[5] S. Floyd, K. Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*. IEEE/ACM ToN, 8/99.

[6] S. Floyd, T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm.* RFC 2582, 4/99.

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

[8] S. Ladha, P. Amer, *Improving multiple file transfers using SCTP multistreaming*, Univ of DE, CISC TR2003-06, 5/03.

[9] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgment Options.* RFC 2018, 10/96.

[10] UC Berkeley, LBL, USC/ISI, and Xerox Parc. Ns-2, v2.1b8, www.isi.edu/nsnam/ns.

[11] J. Postel, *Transmission Control Protocol (TCP).* RFC 793, 9/81.

[12] J. Postel, J. Reynolds, *File Transfer Protocol (FTP).* RFC 959, 10/85.

[13] L. Rizzo, *Dummynet: A simple approach to the evaluation of network protocols*. ACM Comp Comm Review, 27(1), 1/97.

[14] 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, 10/00.

[15] TCPDUMP public repository, www.tcpdump.org

[16] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, *An Integrated Experimental Environment for Dist'd Systems and Networks*. 5th Symp on OS Design and Implementation, 12/02. Boston.