SCTP versus TCP: Comparing the Performance of Transport Protocols for Web Traffic

Rajesh Rajamani, Sumit Kumar, Nikhil Gupta
Computer Sciences Department,
University of Wisconsin-Madison
{raj,sumit,nikhil}@cs.wisc.edu

May 13, 2002

Abstract

The HyperText Transfer Protocol (HTTP) is one of the most widely used protocols on the World Wide Web today. Typically, clients request documents from web servers and display it to the user after the requested document has been fetched. As the user base of this simple, but effective protocol has expanded, users have come to expect quicker responses.
Stream Control Transfer Protocol (SCTP) is a reliable, message-oriented transport protocol that was designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks. SCTP separates the notion of data transmission streams from that of an association, which is functionally equivalent to that of a TCP-style connection. This allows multiple streams within the same association. The data transmission over these streams is fully ordered. However, data from different streams are partially ordered over the association and this can reduce the delay caused by head-of-line blocking problems.
We hypothesize that SCTP could be an effective transport protocol for web traffic and present the results of our comparison of SCTP and TCP.

1 Introduction

The World Wide Web is one of the primary reasons which made the Internet accessible and popular. The Web servers and browsers speak the Hypertext Transfer Protocol (HTTP), which uses TCP as the transport protocol. The Internet that we know today is a huge network of networks that is still growing at a very fast pace. Though the benefits of bringing information to everybody's doorstep or living room, are hardly in question, the size and decentralized nature of the Internet make it susceptible to partitions and outages. The average user, quite oblivious of this fact, wants quick responses. A lot of research has gone into reducing the latency as perceived by the user [6, 3, 7].

Browsers using the original version of HTTP (1.0) [12] opened a new TCP connection for retrieving each data item from the server. This was particularly inefficient while fetching multiple documents from the same server, because of the two extra Round-Trip Times (RTTs) incurred in setting up a new TCP connection between the client and the server [10, 6]. The newer version (HTTP 1.1) allows persistent connections and the browser can use one or more of these to retrieve all data items from a particular server. This, together with caching at various levels, has greatly reduced the latency perceived by the user.

In this paper, we present a comparison of SCTP and TCP as the transport protocol for HTTP. We begin by presenting our motivation for using an alternate protocol for HTTP in the next section. In section 3, we give an overview of SCTP and Section 4 describes our server architecture. In secton 5, we present our hypothesis, which is that SCTP is better suited for HTTP traffic and our reasoning behind this. In section 6, we describe our experimental setup, followed by results in section 7. We provide pointers to related work in section 8 and present our conclusions in section 9.

2 Motivation

For networked applications, the choice of transport protocol is as important as the choice of algorithms and data structures at the application level. Today most applications use either the Transmission Control Protocol (TCP) [9] or the User Datagram Protocol(UDP) [8]. Applications that need a reliable in-order delivery of the bytes

sent by its peer use TCP, whereas ones that can tolerate a certain degree of loss prefer UDP, primarily because UDP provides speedier delivery of packets. Most applications prefer TCP over UDP and applications using TCP include file transfer applications, electronic mail and the world-wide web. UDP is used by streaming audio/video applications for which timely delivery is more important than reliability.

In TCP, the notion of a byte stream and a connection are equivalent. Every TCP connection has two endpoints, whereas UDP is connectionless and allows applications to send messages to one or many peers. There are also many applications that mark message boundaries over the TCP byte stream, because they need reliable message-oriented transport. The Stream Control Transfer Protocol (SCTP) is a new transport protocol which provides a message-oriented, reliable transport. It is similar to other transport protocols like TCP and UDP and is designed to hide the abstractions of the network layers from applications.

In some cases, TCP either does not provide the exact functionality needed by the application or provides more functionality than is needed. In the first case, the application needs to do extra work to be able to use TCP, while in the latter, the extra functionality of TCP might be an overhead. For instance, many applications need reliable message delivery, but TCP is a byte stream oriented protocol. Message-oriented applications achieve their required functionality by delineating the TCP stream into messages. Additionally, TCP provides both strict ordering and reliabilty, but many applications may not need both. In this case, the applications will incur an overhead in using TCP.

HTTP is a message-oriented protocol, where every message has to conform to the RFC 1945 specifications [10]. When multiple embedded files (embedded gifs/jpgs, etc) are being transferred using HTTP, we desire that each of the files be reliably transferred; however, ordered delivery of these files is not a requirement. In fact, we would like to display all the embedded files in the shortest time possible. Most browsers try to achieve this objective by opening multiple connections to the server and dividing the GET requests for these embedded files over them. This allows the browsers to render as many embedded files as can be fetched over the multiple connections, at the same time.

Clearly, there is a mismatch between the requirements of HTTP and the functionality provided by TCP. SCTP [11] is a new transport protocol that provides reliable message delivery and also does not impose a strict ordering. We hypothesize that SCTP would be better-suited for HTTP traffic and help reduce the user-perceived latency and also improve the throughput.

3 SCTP Overview

SCTP [11] is a reliable transport protocol operating on top of a potentially unreliable connectionless packet service such as IP. It was originally designed to be a general purpose transport protocol for message oriented applicatons, as is needed for the transportation of signalling data. It provides acknowledged, error-free, non-duplicated transfer of messages through the use of checksums, sequence numbers and selective retransmission mechanism. SCTP is a transport layer protocol and its services are at the same layer as TCP and UDP. The format of an SCTP packet is shown in Figure1.

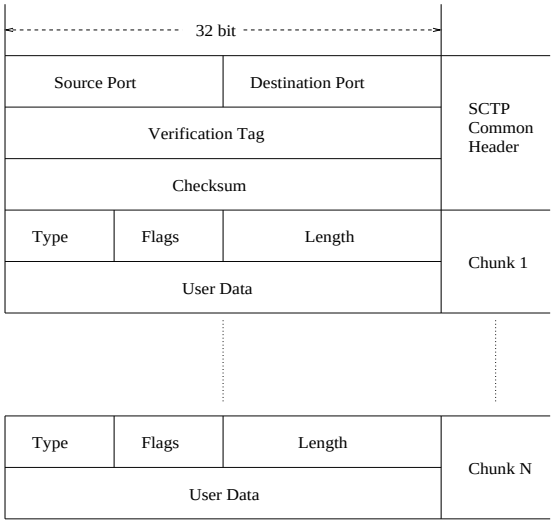

Figure 1: **SCTP Packet format**

Each SCTP packet can contain multiple chunks, which may be either data chunks or control chunks. The first 12 bytes of the packet contain the common header. For identifying an association SCTP uses a Source Port/Destination Port pair along with a 32 bit Verification Tag. The common header also contains a 32 bit checksum (Adler-32 algorithm) for protecting the data against transmission errors. Each chunk has a Type field to differentiate between data chunk and various control chunks, Flags field (which contains chunk specific flags) and the Length field to denote the length of the chunk. The value field contains the acutal payload of the chunk.

Instead of the three phase connection setup for TCP, the initialization of an association is completed after the exchange of four messages. The passive side of the association does not allocate resources for the association until the third of these messages has arrived and been val-

idated. This helps to address the issues of Denial of Service attacks to an extent. The association can be terminated either gracefully ensuring that no data in transit is lost, or it can be aborted which might lead to lost data. SCTP operates at two levels:

- Within an association the reliable transfer of datagrams is achieved by using checksum, a sequence number and a selective retransimission mechanism. Every correctly received data chunk is then handed over to a second level.

- This second level is responsible for realizing the partial ordering of datagrams. That is, order is maintained within each stream, but not amongst the different streams.

Each of the data chunks is numbered with a Transport Sequence Number(TSN) to enable detection of loss and duplication of data packets. The acknowledgements sent by the receiver are based on these sequence numbers. The assignment of the datagram to one of the streams within an association is done by the user. Each of the datagrams is assigned a Stream Sequence Number(SSN) by SCTP. This sequence number is used to realize reliable delivery of datagrams at the receiver. The information about the number of streams to be set up within an association is exchanged between the peers at the time of association setup.

Another important difference between SCTP and TCP is the support for multi-homed nodes in SCTP, i.e. nodes which can be reached using more than one IP addresses. If the nodes and the interconnection network are configured in such a way that the the data from one node to another travels on physically different paths if different destination IP addresses are used, the association can become tolerant against physical network failures. The information about multiple addresses is exchanged at the time of association setup. One of the addresses is selected as the primary path over which the datagrams are transmitted by default. However, retransimissions can be done on one of the available paths.

SCTP uses an end-to-end window based flow and congestion control mechanism similar to the one that is used in TCP [5]. The receiver specifies a receive window size and returns its current size with all the SACK chunks. The sender maintains a congestion window to control the amount of unacknowledged data in flight. The acknowledgements contain a Cumulative TSN Ack, that indicates all the data that has been successfully reassembled at the receiver's side. The Gap Blocks indicate the segments of data chunks that have arrived with some data chunks missing in between. If four SACK chunks have reported gaps with the same data chunk missing, the retransmission is done via the Fast Retransmit mechanism.

The congestion control mechanisms for SCTP have been derived from [5] with some modifications made for multihoming. For each destination address (i.e. each possible path), a discrete set of flow and congestion control parameters is kept, so that from the point of view of the network, an SCTP association with a number of paths behaves similarly as the same number of TCP connections. Each path may either be in Slow Start or Congestion Avoidance mode. For successfully delivered and acknowledged data the congestion window (CWND) is steadily increased and once it exceeds a certain boundary (Slow Start Threshold, SSTHRESH), the mode changes from Slow Start to Congestion Avoidance. Generally, in Slow Start, the CWND is increased faster (roughly one MTU per received SACK chunk), and in Congestion Avoidance mode, it is increased by roughly one MTU per Round Trip Time (RTT). A timeout causes a new Slow Start with CWND=MTU and a Fast Retransimit sets CWND=SSTHRESH. Both of these also cause SSTHRESH to be cut down drastically.

The use of congestion control and flow control mechanisms similar to TCP ensures that SCTP can be introduced without problems in networks where TCP is in widespread use, as has been studied in [1].

4 Server Architecture

In this section we will explain the server architecture that we used for our experiments. The protocol used for file transfer in all the cases is HTTP 1.1 [10] with persistent connections and chunked transfer encoding.

The architecture for single file transfer is similar for both TCP and SCTP and is shown in Figure 2. The client establishes a connection with the server and sends the HTTP GET request. The server receives the request and forks a child process for handling the request. The child process processes the request and if the request can be satisfied, the child sends the requested file, otherwise it sends an appropriate error message.

The server architecture for multiple file transfer (i.e. embedded files) for TCP is shown in Figure 3. As before, the client establishes a connection with the server. It then sends the request for the first file (file 0) to the server. The server forks a child process and the child process either sends the requested file or an error message. On receiving the first file successfully, the client makes requests for a specific number of files (i.e the embedded files). The requests for all the files are sent by the client in a pipelined fashion. The child process on the server

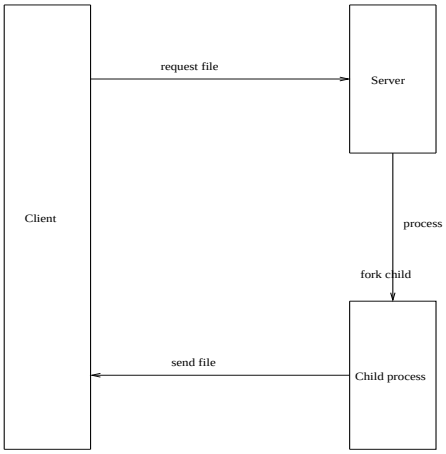

Figure 2: **Single File Transfer - TCP and SCTP**

side receives these requests and sends the requested files one after another. The pipelining of requests by the client reduces latency, as the server does not have to wait to receive a new request after sending one of the embedded files - the request would have already arrived while the first transfer was in progress. The transfer of files is done using a persistent connection, i.e. all the subsequent files are transferred on a single TCP connection.

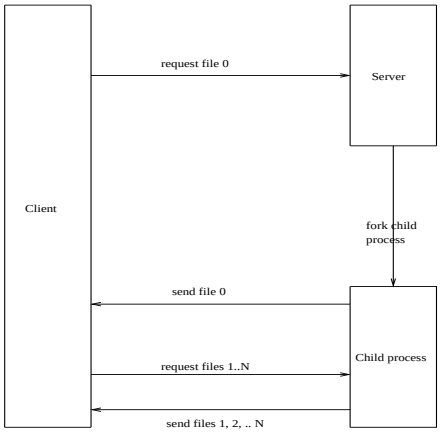

Figure 3: **Embedded Files Transfer - TCP**

The server architecture for multiple file transfer for SCTP is significantly different from TCP and is shown in Figure 4. The client establishes a connection with the server, sends the request for the first file and the server forks a child process to handle the request. The child process now sends the requested file on the first stream (i.e. stream 0). The client process on receiving the first file sends the requests for subsequent files. The child process on the server side receives these requests and instead of sending all the files one after another as in the case of TCP, sends these files in parallel on different streams. Thus, all the files are being transferred simultaneously and the loss of packets for a particular stream would not affect the other streams. The number of streams to be established in a single connection is negotiated between the client and the server at the time of association setup. If the number of streams is less than the number of files to be sent, some or all of the streams will need to be used more than once.

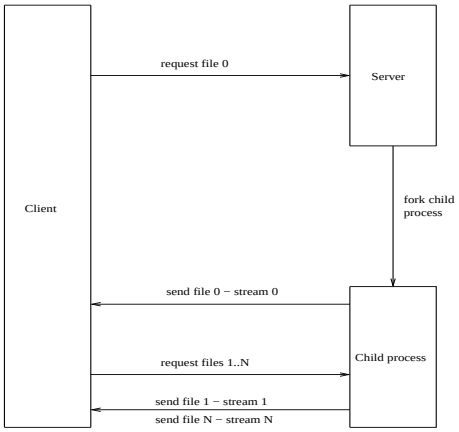

Figure 4: **Embedded Files Transfer - SCTP**

5 Hypothesis

As mentioned earlier, there is a *semantic gap* between TCP and HTTP. SCTP provides services that, we think, are closer to the requirements of the HTTP protocol. Though SCTP is a message-oriented protocol like UDP, it is closer to TCP in terms of some of the services it provides. Like TCP, SCTP provides ordered delivery of messages of the same stream and has a congestion control and flow control mechanism built in. It also uses a 32-bit checksum to detect corrupted packets. The major difference between TCP and SCTP is that, while TCP associates every connection with a byte stream, an SCTP association can have multiple streams in it. SCTP guaran-

tees ordered delivery of messages within each stream, but the messages sent over the association are only partially ordered, since all the streams are multiplexed over a single association. This feature can alleviate the head of line blocking problem present in a TCP byte stream. SCTP also provides multi-homing capability as a fault-tolerance measure.

We reason that over lossy links, SCTP would improve both latency and throughput. We define *latency* as the elapsed clock time between when a particular document is requested and when it is received fully and *throughput* as the elapsed clock time between when a document is requested and when the document and all embedded files referred by it are available at the receiver. We expect an improvement in the latency of each file, because while in TCP the head of line blocking would prevent packets belonging to other documents being delivered to the receiving application, SCTP would deliver packets sent over streams that did not suffer losses. Note that any packet sent over physical media can be lost or corrupted. It is only the transport protocol that sees (over a period of time) streams with gaps in the sequence numbers received, which imply lost or corrupted packets and streams which receive all packets without a gap in the sequence numbers and hence are considered loss free. Loss-free streams may suffer losses in any subsequent period; lossy streams may continue to suffer losses or be loss-free in any subsequent period.

Throughput would improve because the multiple streams in SCTP share the send side kernel buffers. The kernel buffers data that has been sent but unacknowledged and also unsent data. If only some of the streams suffer losses, more kernel buffers would be freed (corresponding to the streams that did not suffer losses) and hence the application would be able send data at a faster rate. TCP would block the application from sending more data whenever there is a loss and it runs out of kernel buffer space.

6 Experimental setup

We used the BSD kernel implementation of TCP and SCTP for performing all our tests. The kernels were built from the head of the *kame* CVS repository *http://www.kame.net*. We had access to three BSD systems, one of which acted as a router and ran *dummynet*. All traffic between the other two systems were sent through this router. *Dummynet* [4] allows the user to control the bandwidth, loss rate, propagation delay and other parameters between two endpoints. For our experiments, we used a constant RTT of 80 milliseconds between the sender and the receiver. This value is an approximation of the RTT between the east coast and west coast of continental United States.

We varied the loss rates between 0% and 25% as shown in the performance results. For the bandwidth metric, we chose four values - 40Kbps, 400Kbps, 3Mbps and 10Mbps. These values were obtained by taking the average of the results of ten speed tests available at *http://bandwidthplace.com/speedtest/*. The aforementioned test tries to evaluate the peak bandwidth attainable using the HTTP protocol over links that use a dial-up modem (56Kbps), cable modem, wireless LAN or a 100Mbps ethernet to connect to the network.

7 Results

7.1 Single file transfer test

We first ran a test to compare the performance of SCTP with that of TCP under no loss. This was done to make sure that the overheads involved in marking message boundaries and chunks does not make SCTP unviable as compared to TCP.

Figure 5 shows the *latency* of a file transfer between the server and client. The line representing SCTP follows that of TCP. The difference may be due to the overhead incurred by SCTP in framing messages. TCP being a byte stream protocol tries to buffer data in the kernel in order to send out Maximum Transmission Unit (MTU) sized packets. SCTP can also put more than one *data chunk* [11] in a packet, provided both the chunks can be accomodated in the packet. Since our client and server used a message size of 1024 bytes and the maximum data payload in a single SCTP packet is 1408 bytes, we did not see SCTP marshalling more than one message into a single packet. As a result, SCTP sends nearly twice as many packets as TCP for the largest sized file shown in the figure below. This, together with the fact that the kernel implementation of SCTP is only 7 months old and has not been optimized for performance, may provide an explanation to the difference between the two protocols.

We performed the same single file transfer test with bandwidth values of 400 Kbps, 3 Mbps and 10 Mbps. Due to space constraints, we show the plot of only the 10 Mbps test. The interested reader is referred to *http://www.cs.wisc.edu/∼raj/sctp* where all plots are archived. As shown in figure 6, the results of this test are similar to the earlier figure, in that SCTP's curve follows that of TCP. TCP is able to better utilize the higher bandwidth and we believe this plot brings out the unoptimized nature of the SCTP implementation. The anomalies in this graph are due to the variance in the queuing delays

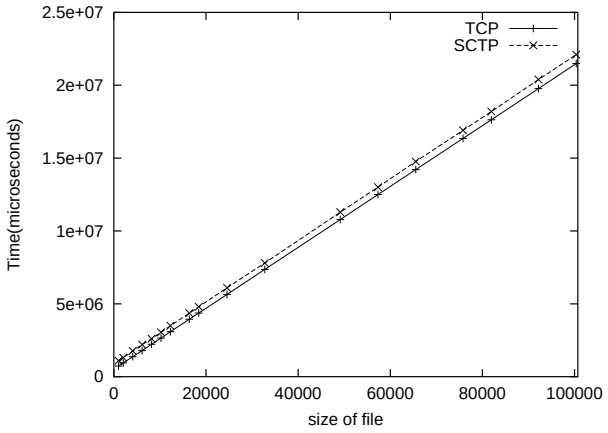

Figure 5: **Time taken to transfer a single file from server to client. Dummynet was used to restrict the bandwidth to 40 Kbps.**

introduced by Dummynet. Each point in figures 5 and 6 are the average values of five samples. We believe taking more samples would have reduced the variance.

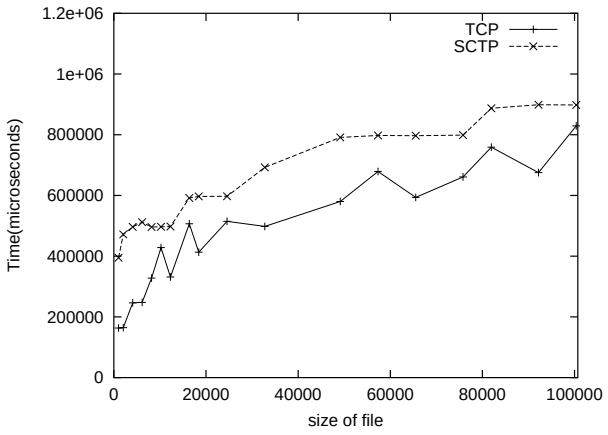

Figure 6: **Time taken to transfer single file from server to client. This test was performed with our highest bandwidth - 10 Mbps.**

7.2 Multiple files transfer tests

This test was designed to be simulate the behavior of a typical web browser. Most websites have a certain number of embedded links which are fetched by the browser automatically. Our client first requests a document and after receiving it, pipelines *n* requests. The *n* requests represent fetching embedded links. For our tests, the size of the first file was 50KB and the client then requested seven files *(n=7)* each of which were 4KB.

Table 1 shows the *latency* of getting these eight files from the server while using TCP and SCTP. Each data point is the average value of twenty data samples. The loss percentages indicated are the loss metric used in each direction, so the cumulative loss is a little less than twice the value indicated. Under no loss conditions, each file is available at the client earlier when using TCP. However, when there is loss, we see that using SCTP reduces the latency.

Figure 7, which plots time taken to fetch all eight files against different values of loss supports our hypothesis that SCTP can help improve the *throughput*. This can be explained by the fact that more send side kernel buffers are freed by SCTP, as not all streams may suffer loss during a window of time. The variance for loss values 10% and 15% is too high and therefore, should not be taken into account. We believe taking more samples would have reduced the variance.

8 Related Work

HTTP performance has been studied very extensively. Given its popularity and widespread use, this comes as no surprise. One of the most influential works is that of Padmanabhan and Mogul [6], in which the authors present an analysis of the sources of latency in the HTTP 1.0 protocol and suggest a set of improvements. This work put forth the ideas of *persistent connections* and *pipelining*, which have since been incorporated into the HTTP 1.1 protocol. [2] presents a comparison of the HTTP 1.0 and HTTP 1.1 protocols. Others have worked on various aspects of caching, which is a widely used mechanism to reduce the latency [3, 7].

Because SCTP is a relatively new protocol, it has not been studied very much. [1] presents an evaluation of the performance of SCTP in a wide area network, especially when competing with TCP. The authors conclude that SCTP is *TCP-friendly* and its introduction does not degrade the performance of existing protocols. However, the effects of SCTP on application-level protocols is not treated in this work. Our work tries to address this issue and is closest to [6].

Table 1: Latency of each file in multiple file transfer test, B/w=10Mbps and all times in microseconds

Protocol	Loss	File 1	File 2	File 3	File 4	File 5	File 6	File 7	File 8
TCP	0%	679395	768656	3873273	3910344	3942401	4243416	4273269	4708352
SCTP	0%	802931	888264	4468128	4507111	4607180	4834083	4878979	4887073
TCP	1%	4930718	5595977	29598318	31047085	31924275	33460332	34333089	38222168
SCTP	1%	4299919	4775626	24132252	24536217	25106516	26678879	27143072	29628740
TCP	2%	5983277	6725730	35361679	37232434	38509110	40681890	42568811	45179090
SCTP	2%	5506176	6098121	31539786	32164926	32692426	33117396	33981264	41551992

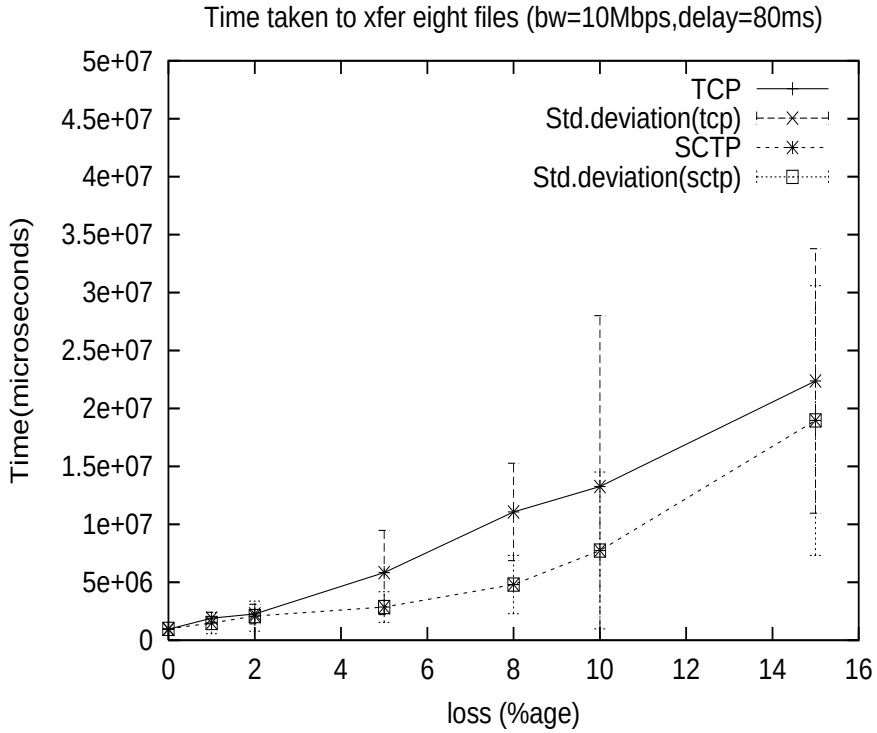

Figure 7: **Time taken to transfer multiple files from server to client. Dummynet was used to restrict the bandwidth to 10 Mbps.**

9 Conclusions and Future work

In this paper, we present our comparison of the effects of using TCP and SCTP as the transport protocol for web traffic. The results support our hypothesis that SCTP can help reduce the *latency* and improve *throughput*. This, together with some other features of SCTP like multi-homing and better protection against Denial of Service attacks, make it a very attractive choice for future web traffic. Though the focus is on web traffic, we feel the results bring out the deleterious effect of head of line blocking that applications using TCP suffer from, because TCP couples the delivery mechanism and reliability. By separating these two important issues and also by providing a reliable message-oriented transport, SCTP provides developers with both flexibility and efficiency. It will be interesting to know the mix of applications that use TCP as a byte stream and those that use TCP as a reliable transport for messages that are delineated by the application level protocol.

There are some significant weaknesses in our work which we plan to address in the near future. Our comparison pits SCTP against TCP reno (default mode of TCP on BSD). Since, SCTP uses Selective Acknowledgements(SACK), we believe this is not a very fair comparison. SACK was introduced to improve TCP's performance when multiple packets were lost from one window of data. Our results for high loss rates, when such an event is likely to occur shows high variance and we recommend that the reader ignore them. We would also like to compare the performance of multiple TCP streams against multiple associations of SCTP. A future study will have to address all these issues to conclusively prove that SCTP offers an advantage over TCP.

10 Acknowledgements

We would like to thank Prof.Paul Barford for providing guidance and support for this work. Our many thanks to Randy Stewart for his valuable inputs and help in debugging the BSD kernel implementation of SCTP. Doug Thain, Joel Sommers, De Byrd, Jim Gast and the CSL helped us sort out various problems.

References

[1] A.Jungmaier, M.Schopp, et al. Performance Evaluation of the Stream Control Transmission Protocol. In *Proceedings of the IEEE Conference on High Performance Switching and Routing*, June 2000.

[2] P. Barford and M. Crovella. A performance evaluation of hyper text transfer protocols. In *Measurement and Modeling of Computer Systems*, pages 188–197, 1999.

[3] L.Fan, P.Cao, et al. Summary cache: A scalable wide-area cache sharing protocol. In *Proceedings of ACM SIGCOMM Conference*, pages 254–265, Sept. 1998.

[4] L.Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):31–41, 1997.

[5] M.Allman, V.Paxson, and W.Stevens. TCP Congestion Control. RFC 2581, 1999.

[6] V. Padmanabhan and J. Mogul. Improving HTTP Latency, Dec. 1995.

[7] P.Cao and C.Liu. Maintaining strong cache consistency in the world-wide web. In *17th International Conf. on Distributed Computing Systems*, pages 12–21, May 1997.

[8] J. Postel. User Datagram Protocol. RFC 768, Aug. 1980.

[9] J. Postel. Transmission Control Protocol. RFC 793, Sept. 1981.

[10] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, and T.Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2068, Jan. 1997.

[11] R.Stewart, Q.Xie, et al. Stream Control Transmission Protocol. RFC 2960, Oct. 2000.

[12] T.Berners-Lee, R.Fielding, and H.Frystyk. Hyptertext Transfer Protocol - HTTP/1.0. RFC 1945, May 1996.