The Impact of SCTP on SIP Server Scalability and Performance

Kumiko Ono, Henning Schulzrinne
Department of Computer Science
Columbia University
Email: {kumiko, hgs}@cs.columbia.edu

Abstract*— The Stream Control Transmission Protocol (SCTP) is a relatively recent transport protocol, offering features beyond TCP. Although SCTP is an alternative transport protocol for the Session Initiation Protocol (SIP), we do not know how SCTP features influence SIP server scalability and performance. To estimate this, we measured the scalability and performance of two servers, an echo server and a simplified SIP server on Linux, for both SCTP and TCP. Our measurements found that using SCTP does not significantly affect data transfer latency. However, the number of sustainable associations drops to 17-21% or to 50% of the TCP value if we adjust the acceptable gap size of packet reordering.

I. Introduction

The Stream Control Transmission Protocol (SCTP)[1] was originally designed for carrying telephony signaling protocol, Signaling Systems No.7 over IP. Similar to TCP, SCTP is reliable and connection-oriented, but it is message-oriented like UDP, and has additional features such as multi-streaming and multi-homing. SCTP is defined as an alternative transport protocol to UDP or TCP for the Session Initiation Protocol (SIP)[2][3], which is a major Internet telephony signaling protocol. Understanding how SCTP affects scalability and performance is important for building SIP servers and designing large voice over IP (VoIP) systems.

Depending on its role in a VoIP network, a SIP proxy server may connect to a large number of user agents or to a, typically smaller, number of other proxy servers, as shown in Figure 1. Proxy servers maintaining connections to user agents are often called edge proxy servers. Therefore, if a connection-oriented transport protocol is used, the edge proxy server is required to manage a large number of concurrent connections, making server scalability as important as request throughput and data transfer latency.

Even though SCTP is not as commonly used as TCP or UDP, it has been implemented as a kernel module in Linux, so that it can be easily used between user agents and a proxy server as well as between proxy servers. The broad deployment of SCTP would require a server with enough scalability to accommodate a large number of user agents. We present how SCTP affects scalability and the performance of an echo server and a SIP front-end server that only implements the message handling parts of a SIP proxy server. From our measurements, we estimate the effect of choosing SCTP as a transport protocol for SIP.

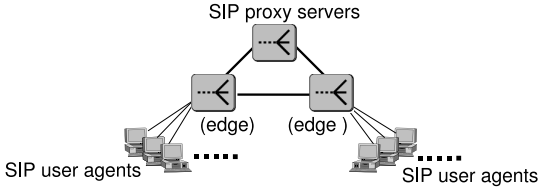

Fig. 1. Two types of connections at SIP proxy servers

The remainder of this paper is organized as follows. Section II discusses which SCTP features benefit SIP. Section III describes our measurement objectives and environment. Section IV compares socket memory usage between SCTP and TCP, suggesting how to reduce it with SCTP. Section V compares the number of sustainable connections and Section VI compares data transfer latency for an echo server between SCTP and TCP. Section VII compares data transfer latency for a simplified SIP server among SCTP, TCP and UDP. We conclude with a discussion of the influence of SCTP on SIP server scalability and performance in Section VIII.

II. SCTP Features

The following SCTP features potentially benefit SIP applications and server scalability, respectively.

A. Beneficial Features for SIP Applications

1) Piggyback setup in four-way handshake: SCTP establishes a connection with a four-way handshake using INIT, INIT-ACK, COOKIE-ECHO, and COOKIE-ACK messages, as shown in Figure 2. Clearly, this four-way handshake requires two round trip times (RTTs), which is one more than that for the three-way handshake in TCP. However, the piggyback setup option allows to bundle user data into the COOKIE-ECHO message, as shown in Figure 3, so that it reduces the RTTs combined in the handshake and sending user data to the same as for TCP. In a SIP session using non-persistent connection, where a new connection is established, larger RTTs in the handshake causes a longer setup delay. Thus, piggyback setup is expected to mitigate this setup delay.

2) Message orientation: Similar to UDP, SCTP preserves message boundaries. Applications can determine if the original message is fully delivered at a receive buffer through the socket API, while they need to parse received messages over TCP using the Content-Length header in SIP. However, message parsing is necessary for SIP applications. Thus, we suspect that this message orientation has a negligible benefit.

3) Message exceeding MTU size: Similar to TCP, SCTP supports the delivery of message exceeding Maximum Transfer Unit (MTU) size by segmentation. Since some networks or services require SIP extension headers or signatures, the message size of a SIP request may grow beyond the Ethernet MTU of 1,500 bytes.

B. Features Improving Server Scalability

Two features of SCTP, one-to-many style sockets and multi-streaming, potentially help server scale by reducing memory usage and increasing throughput.

SCTP provides one-to-one and one-to-many style socket interfaces. These two interfaces differ in representing *associations*, which mean connections in SCTP.

While a one-to-one style socket can represent a single association, a one-to-many style socket can represent multiple associations, similar to UDP, where a single socket can be used to receive messages from multiple clients. Figure 4 shows that a server using a one-to-many style socket can receive messages from different associations at a single listening socket without invoking the `accept()` system call to create new sockets. Thus, using a one-to-many style socket can drastically reduce the number of sockets at a server.

Additionally, a client using a one-to-many style socket can utilize piggyback setup to reduce the setup delay of the four-way handshake described in Section II-A. Figure 3 shows that a client using a one-to-many style socket invokes the `sendmsg()` system call to send a message without calling the `connect()` system call.

Thus, we can expect to benefit from a one-to-many style socket for both server and client, although the socket style can be set independently to each other. At the same time, however, using a one-to-many style socket potentially decreases server request throughput. By sharing a single socket buffer, the server receives and sends messages for all associations and de-multiplexes the messages by four tuples: source and destination IP addresses and ports. Although this is similar to that of UDP, sent messages are kept longer than for UDP, since they cannot be removed until the SCTP ACK has been received. Therefore, the send buffer may be exhausted at high request rates. We will evaluate the effects of this one-to-many style socket from perspectives of the number of sustainable associations and of request throughput.

Another feature, multi-streaming, can minimize head-of-line (HOL) blocking, as evaluated by Camarillo, Kantola and Schulzrinne[4]. The HOL blocking occurs in TCP when a segment is lost and a subsequent segment arrives out of order. The receiving application needs to wait for the lost segment to read the arrived segment. In SCTP, however, by breaking multi-session streams into separate streams, the HOL blocking can be minimized, even though it occurs in the same stream. This multi-streaming feature is effective between proxy servers, where multiple SIP sessions can share an association. We can expect this multi-streaming feature to improve throughput in a congested network, but do not discuss this feature here.

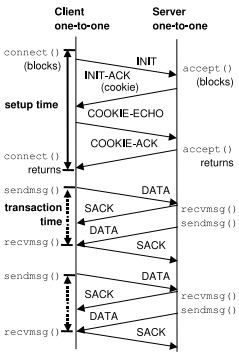

Fig. 2. Message exchanges using one-to-one style sockets for both a server and a client

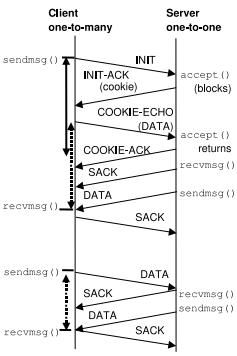

Fig. 3. Message exchanges using a one-to-many style socket for a client

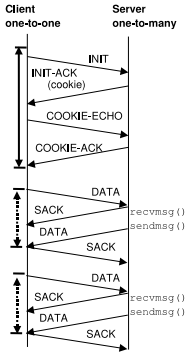

Fig. 4. Message exchanges using a one-to-many style socket for a server

III. Measurements

We measured the effects of three SCTP features, namely, a one-to-many style socket, piggyback setup and message orientation. To evaluate the effect of a one-to-many style socket, we first identified the data structures of the two styles of SCTP sockets, then we compared memory usage among these two styles of SCTP sockets and TCP sockets. Then, to evaluate its effect on server scalability, we compared the number of sustainable associations for an echo server between two SCTP socket styles. To evaluate its effect on server performance, we compared the setup and transaction times. By comparing the setup time, we identified the effect of piggyback setup. By comparing the transaction time, we identified the effect of the SCTP message orientation using a SIP front-end server, which simply receives a SIP request and responds with a 200 OK response without any substantial SIP operation.

A. Measurement Environment

We use two servers, an echo server and a SIP front-end server using a single process and single thread for simplicity. The servers under test (SUT) for both run on a dedicated host with Pentium IV 3 GHz 32-bit dual-core CPU and 4 GB of memory. The SUT runs Linux 2.6.23 configured with the default virtual memory split of 1G/3G, where the kernel space is 1 GB and the user space 3 GB. When the server needs to wait for events on more than 1,024 sockets, it uses the `epoll()` system call. For the echo clients or SIP user agents, we use six hosts with Pentium IV 3 GHz 32-bit CPUs and 1 GB of memory running Redhat Linux 2.6.9. These hosts communicate over a 100 Mb/s Ethernet connection at light load. The round trip time measured by the `ping` command is roughly 0.1 ms.

IV. SCTP Data Structures

We first identified the data structures for establishing and maintaining an SCTP socket and calculated the memory usage. Then, we evaluated server scalability by measuring the number of sustainable associations at the SUT.

TABLE I
ITEMIZED MEMORY USAGE FOR AN SCTP AND A TCP SOCKET

Data structure		Slab cache			
Name	Size (B)	Name	Size (B)	# of objects SCTP	# of objects TCP
dentry	128	dentry	128	1	1
file	136	filp	192	1	1
inode	328	sock_inode	384	1	1
socket	40	_cache			
sock(tcp_socket)	1,096	TCP	1,152	0	1
sock(sctp_socket)	772	SCTP	896	1	0
eppoll_entry	36	eventpoll_pwq	36	1	1
epitem	80	eventpoll_epi	128	1	1
sctp_endpoint	176	size-256	256	1	0
sctp_bind_addr	40	size-64	64	1	0
Subtotal for a socket (bytes)				2,044	2,020
sctp_association	5,120	size-8192	8,192	1 or n[a]	0
sctp_transport	284	size-512	512	1 or n[a]	0
sctp_ssnmap	60	size-64	64	1 or m[b]	0
Subtotal for an association (bytes)				8,768	0
Total memory usage (bytes)				10,812	2,020

[a] one-to-many [b] multi-streaming

A. Comparison of the Data Size of SCTP and TCP Sockets

Table I itemizes the sizes of socket-related data structures and those allocated from slab cache objects as long as an association or a connection remains open at a server. The slab cache is implemented for frequent allocations and deallocations of data in Linux. SCTP requires larger protocol-specific data including the 5,120 byte *sctp_association* which stores parameters in Transmission Control Block (TCB), while TCP requires only the 1,096 byte *tcp_socket*. Table I also shows that the amount of memory using a one-to-many style socket significantly increases as a function of the number of associations, while that using multi-streams increases slightly as a function of the number of streams.

Furthermore, several SCTP-specific data objects, including the *sctp_association*, are allocated from general purpose slab objects, which suffer from internal fragmentation because of power of two sized objects. As a result, maintaining an SCTP socket consumes 10,812 bytes, approximately five times of the amount needed for a TCP socket, even in the simplest case, that is, with a one-to-one style socket and a single stream. The dominant data is association-related data, which consumes approximately 80% of the total memory usage. Even when using a one-to-many style socket, a server needs to allocate the same number of associations with when using one-to-one style sockets. Therefore, we cannot expect a drastic reduction of memory usage by using a one-to-many style socket, although we expected that in Section II-B. Section IV-B will investigate how to reduce association-related data size in order to increase this reducing effect by using a one-to-many style socket.

B. Memory-conscious Usage of Association Data

To cut down the memory footprint of association-related data, it is crucial to reduce the size of the dominant *sctp_association* data. Although the size of this data structure is 5,120 bytes, it grows to 8,192 bytes when it is allocated from a general purpose slab cache object. The size of a general purpose slab cache object is power of two, while the size of a specific purpose slab cache object does not have to be so. This means that defining SCTP-specific slab cache objects can avoid internal fragmentation. As a result, the slab cache size for *sctp_association* data drops to 5,120 bytes or more, which are additionally consumed by the object alignment.

Furthermore, we can cut down the size of *sctp_association* data by reducing the size of the dominant sub-member of the *sctp_association* structure, *map[sctp_tsnmap_storage_size(SCTP_TSN_MAP_SIZE)]*, which accounts for 4,096 bytes. This is a byte mapping array, namely, TSN (Transmission Sequence Number) map, each byte of which indicates the number of chunks for each TSN to trace received TSNs for unordered data delivery. The TSN map size, i.e., the `SCTP_TSN_MAP_SIZE`, is set to allow a default gap of 2,048 segments between the cumulative ACK and the highest TSN. This mapping array is twice the size of the TSN map to allow an overflow map. Therefore, if we do not need to handle such a large gap, we can reduce the TSN map size. Figure 5 shows how memory usage for a socket can decrease by by defining SCTP-specific slab cache objects for the *sctp_association* data, and by adjusting the TSN map size. If we adjust it to 512, the mapping array would decrease to 1,024 bytes, and the *sctp_association* would decrease to 2,048 bytes. This reduced data could then be allocated from a 2,048 byte slab object. As a result, the total amount of memory for an SCTP socket could be reduced to less than half, 4,668 bytes, although the amount is still more than twice the size of a TCP socket. If we do not need to support any reordering packets in a network, which disables the selective ACK mechanism though, we could adjust the TSN map size to zero. The slab object size for the *sctp_association* would then decrease to 1,024 bytes. As a result, the total amount of memory could be reduced to 3,664 bytes which is still around twice the size of a TCP socket.

In a SIP session between a user agent and a proxy server, SIP messages do not have to allow such a large gap, since a SIP request requires the SIP response before sending a new SIP request. However, in a SIP session between proxy servers, a number of aggregated SIP messages are transmitted over an association. For this situation, the SIP servers need to allow a certain gap depending on the traffic model between proxy servers.

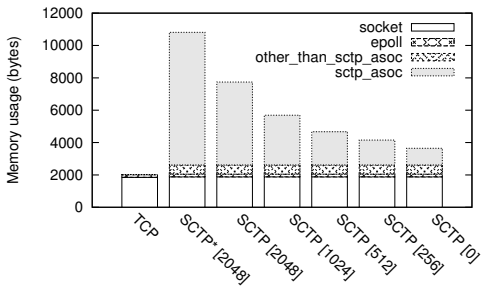

Fig. 5. Memory usage for a socket as a function of `SCTP_TSN_MAP_SIZE`

TABLE II
NUMBERS OF SUSTAINABLE ASSOCIATIONS AND MEMORY USAGE PER ASSOCIATION FOR SCTP AND TCP

Socket style at server	SCTP		TCP
	One-to-one	One-to-many	
Number of assoc.	74,680	90,607	419,019
Ratio	0.17	0.21	1.00
Memory per assoc. (KB)	11.12	8.90	2.05

V. NUMBER OF SUSTAINABLE ASSOCIATIONS

As a metric of server scalability, we measured the number of sustainable associations for the echo server. After the echo clients request new associations, they send a SIP INVITE message, receive a copy and maintain the associations. This measurement was performed without the tuning proposed in Section IV-B. To allow a large number of concurrent associations, we raised an upper limit of the number of file descriptors per process to 60,000 for the clients and to 1,000,000 for the SUT. In addition, to increase concurrent associations for each client, we expanded the range of ephemeral local port to 10,000 - 65,535. For simplicity, we disabled the SCTP heartbeat mechanism, which probes the reachability of remote associations every 30 seconds by default.

A. The Effect of One-to-many Style Sockets

Table II compares the numbers of sustainable associations or connections between SCTP and TCP. The numbers are measured just before the system yields an error, out of memory for sockets. From our measurements, the number of sustainable associations of SCTP is only 17-21% that of TCP. Although a server using a one-to-many style socket can increase the number of concurrent associations by about 15,000 compared to using one-to-one style sockets, this improvement makes barely a difference in the comparison with TCP.

Table II also compares the memory usage per association by measuring the memory usage for the slab cache objects. The memory usage agrees with our analysis in Section IV-A. Thus, if we adjust the TSN map size depending on the requirement for handling unordered data delivery, we could improve the number of sustainable associations up to approximately 50% of that of TCP.

VI. DATA TRANSFER LATENCY FOR AN ECHO SERVER

As a metric of server performance, we measured data transfer latency for the echo server to compare the setup and transaction times of SCTP and TCP to identify how using a single socket buffer in a one-to-many style socket affects and how much piggybacking data in the initial handshake reduces them. For this measurement, we set "no delay" option and increased the socket buffer size to 4 MB for the echo server and clients to eliminate unnecessary delay and errors, in addition to the configuration in Section V.

The setup time of an association is the elapsed time from the instant that a client invokes the `connect()` system call to returning from it. The transaction time is the elapsed time from the instant that a client invokes the `sendmsg()` system call to send a 1,550 byte INVITE request to its invoking the `recvmsg()` system call to receive a copy. The echo server receives the requests at 2,500 requests/second and accumulates the SCTP associations or TCP connections until their number reaches 50,000.

TABLE III
SETUP AND TRANSACTION TIMES FOR SCTP AND TCP

Socket style at server	SCTP			TCP
	One-to-one		One-to-many	
Setup style	Regular	Piggyback		
Setup (ms)	0.34	0.84	0.38-170.91 [0.34][a]	0.17
Transaction (ms)	0.54		0.65-34.14 [0.53][a]	0.48
Total (ms)	0.88	0.84	1.03-205.05 [0.87][a]	0.65

[a] the results after replacing search algorithm

A. The Effect of One-to-many Style Sockets

Table III compares the setup and transaction times of the two SCTP socket styles and TCP for the echo server. Significantly, the setup and transaction times for the one-to-many style sockets grows linearly with the number of associations, while those times using the SCTP one-to-one style sockets and using TCP remain constant.

To investigate the reason of the linear increase with the number of associations using a one-to-many style socket, we traced the kernel source code, and found that when receiving INIT and COOKIE_ECHO messages, a linear search is used to look up a matching association by endpoint. Such search always fails. Therefore, the one-to-many style socket, where an endpoint links to multiple associations, increases the setup time as a function of the number of associations, while one-to-one style sockets, where each endpoint links to a single association, do not increase the setup time. Also, when sending a message, the `sctp_sendmsg()` function in kernel calls a lookup function which performs a linear search. Thus, the transaction time increases linearly. However, the increase is not so drastic as that in the setup time. Unlike to the setup time, this association search is always successfully performed and takes time depending where the matching association is stored in the list of associations. Since the linear search clearly causes the cost of using a one-to-many style socket, both setup and transaction times could be improved by replacing it with a hash table lookup. After applying a patch to replace the search algorithm, we re-measured the setup and transaction times. The values in brackets in Table III indicate that the setup and transaction times using a one-to-many style socket remains constant at 0.34 ms and 0.53 ms, respectively, similar to that using one-to-one style sockets.

Although we suspect negative impact of using a one-to-many socket on server request throughput in Section II-B, this improved implementation did not show any impact of sharing a single socket butter by a large number of associations, after expanding the size of socket buffers.

B. The Effect of Piggyback Setup

With the piggyback setup mechanism, we can expect to reduce the total number of messages and the combined time for the setup and the transaction, not the setup time itself. Table III compares the setup, the transaction and the combined times among two SCTP setup styles and TCP. This table indicates

TABLE IV
ELAPSED TIMES BETWEEN MESSAGES FOR SCTP AND TCP

SCTP			TCP	
Messages	Regular setup (ms)	Piggyback setup (ms)	(ms)	Messages
s:INIT	0.00	0.00	0.00	s:SYN
r:INIT-ACK	0.14	0.14	0.09	r:SYN, ACK
s:COOKIE-ECHO	0.01	0.01	0.01	s:ACK
r:COOKIE-ACK	0.13	0.23		
s:DATA	0.00	N/A[a]	0.00	s:DATA
r:DATA	0.37	0.26[b]	0.34	r:DATA

[a] DATA is piggybacked with COOKIE-ECHO. [b] This indicates the elapsed time between receiving COOKIE-ACK and DATA.

that using piggyback setup reduces the combined time only by 0.04 ms in our local area network.

To investigate the cost of the piggyback setup, we monitored the elapsed time for each RTT at an echo client using the `tcpdump` program. Table IV shows that the elapsed time between sending COOKIE-ECHO and receiving COOKIE-ACK grows by 0.1 ms beyond that of the SCTP regular setup. Therefore, in spite of reducing the elapsed time between receiving COOKIE-ACK and receiving a copied user message by 0.11 ms, the overall effect of using piggyback setup is slight: 0.01 ms at the network layer. Table IV also indicates that the setup with processing signed cookies is originally expensive. As long as SCTP processes signed cookies to protect against a INIT flooding attack, the SCTP setup is more expensive than that for TCP, which processes no cookies by default. Although the effect of using piggyback setup is slight in our measurement environment, the effect of reducing one RTT would be larger in a wide area network.

VII. DATA TRANSFER LATENCY FOR A SIP FRONT-END SERVER

We used a SIP front-end server that only implements the message handling parts of a SIP proxy server, to identify the effect of the message orientation, which is presumably a small effect as described in Section II-A.

A. The Effect of Message Orientation

When calling the `recvmsg()` system call, we can determine whether or not the received message is fully delivered by checking the message flag. If the full message is delivered, the message flag is set to `MSG_EOR`. Figure 6 compares the setup and transaction times among the three SCTP cases, TCP and UDP, for the SIP front-end server. As we suspected, we cannot see any effect of the message orientation in the transaction times. We measured that parsing a SIP message is not a heavy task, since the typical message size is approximately 1,550 bytes. If a SIP message conveys a bulk data as much as for a file transfer application, the message orientation could be effective. Also, message parsing is required for SIP operations regardless of the transport protocol. Therefore, the message orientation feature has little effect on a SIP server.

VIII. CONCLUSION

We have shown how using SCTP impacts on server scalability and performance by evaluating the effect of the three

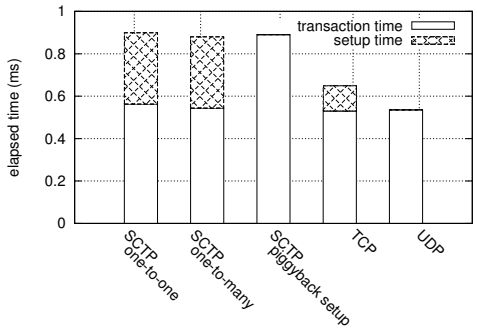

Fig. 6. Setup and transaction times for SIP front-end server

SCTP features, namely, one-to-many style sockets, piggyback setup and message orientation. We suspected that using SCTP has negative impact on server scalability, since an SCTP association, which corresponds to a TCP connection, requires significantly more storage to support the additional features. Then, we expected using a one-to-many style socket enables to increase the number of sustainable associations by reducing the number of sockets. Our measurement results indicates that the echo server scalability decreases to one fifth of that using TCP, because of a large amount of association-related data. By reducing the capacity for accepting packet reordering, we could mitigate the decrease to a half. Yet, the scalability gap between SCTP and TCP remains significant. This gap surely limits the usability of SCTP for edge servers. On the other hand, data transfer latency, such as the setup and transaction times, does not significantly differ between using one-to-one style sockets for SCTP and using TCP. By using the piggyback setup, the combined time of the setup and transaction can slightly decrease in our measurement environment, but it is still longer than for TCP, since handling a signed cookie is expensive. However, the effect of reducing a RTT by piggyback setup would be larger in a wide area network. Regarding the message orientation, we failed to identify the effect using the SIP front-end server.

Therefore, we recommend to use a one-to-many style socket and piggyback setup. However, we had to fix the current lookup function for one-to-many style sockets of SCTP, in order to limit the latency for a scalable server. Since the SCTP kernel implementation is far less mature than the TCP implementation, we suspect that there is a significant room for improvement.

ACKNOWLEDGEMENT

This work was supported by NTT Corporation. The authors would like to thank Vlad Yasevich for providing the patch.

REFERENCES

[1] R. Stewart. Stream Control Transmission Protocol. RFC 4960, IETF, September 2007.
[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, IETF, June 2002.
[3] J. Rosenberg, H. Schulzrinne, and G. Camarillo. The Stream Control Transmission Procotol (SCTP) as a Transport for the Session Initiation Protocol (SIP). RFC 4168, IETF, October 2005.
[4] G. Camarillo, R. Kantola, and H. Schulzrinne. Evaluation of Transport Protocols for the Session Initiation Protocol. In *IEEE Network*, September 2003.