Publication III

G. Camarillo, H. Schulzrinne, and R. Kantola. Evaluation of Transport Protocols for the Session Initiation Protocol. *IEEE Network*, Vol. 17, No. 5, Pages 40-46, September 2003.

Evaluation of Transport Protocols for the Session Initiation Protocol

Gonzalo Camarillo, Ericsson
Raimo Kantola, Helsinki University of Technology
Henning Schulzrinne, Columbia University

Abstract

SCTP is a newly developed transport protocol tailored for signaling transport. Whereas in theory SCTP is supposed to achieve a much better performance than TCP and UDP, at present there are no experimental results showing SCTP's real benefits. This article analyzes SCTP's strengths and weaknesses and provides simulation results. We implemented SIP on top of UDP, TCP, and SCTP in the network simulator and compared the three transport protocols under different network conditions.

The limitations present in TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) for transporting signaling traffic led to the design of a new transport protocol within the IETF (Internet Engineering Task Force). The SIGTRAN working group developed the Stream Control Transmission Protocol (SCTP) [1, 2]. SCTP was first intended to transport telephony signaling over an unreliable network such as an IP network [3]. However, the protocol has been designed so that SCTP can be used as a general-purpose message-based transport protocol.

We compare the behavior and performance of SCTP with existing transport protocols. Our simulations show that Head of the Line (HOL) blocking, widely believed to be TCP's major limitation when carrying signaling traffic, does not introduce a significant performance decrease even under heavy load. We also discovered some issues regarding transport-layer fragmentation. In order to perform a fair comparison, we needed an application-layer protocol that could run on top of all the different transport protocols we wanted to analyze, namely SCTP, UDP, and TCP. We chose the Session Initiation Protocol (SIP) [4]. SIP is an application-layer protocol for creating, modifying, and terminating sessions. The basic operation of SIP does not depend on the lower-layer transport protocol, which makes it an ideal choice to compare the performance of different transports.

The SIP specification [4] describes how the protocol operates over TCP [5] and over UDP [6]. TCP provides reliable in-order transfer of bytes, while UDP does not ensure either reliability or in-order delivery. We defined in [7] how SIP can run on top of SCTP. Although there are already implementations of telephony signaling protocols such as ISUP on top of SCTP, we know of no SIP implementations that use SCTP.

The remainder of this article is organized as follows. We introduce SCTP and SIP, respectively. We also introduce the simulation environment. Using simulations, we compare SCTP, UDP, and TCP as transport protocols for SIP. Finally, we outline some conclusions.

Stream Control Transmission Protocol (SCTP)

The Stream Control Transmission Protocol (SCTP) [1] is a transport protocol tailored to transport signaling traffic. SCTP inherits all its congestion and flow control mechanisms from TCP, but includes a number of improvements intended to make it a more efficient signaling transport than TCP.

SCTP Connection Establishment

SCTP is a connection-oriented transport protocol. In SCTP terminology, a connection is referred to as an association. An association is established through a four-way handshake as opposed to the three-way handshake implemented by TCP. Having an additional message in the SCTP handshake, shown in Fig. 1, protects servers against Denial of Service (DoS) attacks with spoofed IP addresses. In TCP, the SYN flood attack exploits the lack of a stateless return-routability confirmation.

The SCTP receiver, after receiving an INIT message, sends back a random number, called a cookie, in an INIT ACK message. This receiver does not allocate any resources for this SCTP association until it receives the same cookie in the COOKIE ECHO message. Thus, resources are allocated only when it is ensured that the party supposedly sending the INIT message is really willing to establish an SCTP association. The COOKIE ACK message simply informs the sender that the COOKIE ECHO massage has been successfully received and that it contained a valid cookie.

The last two messages of the four-way handshake can already carry user data. With this data piggybacking, SCTP has the same connection-establishment delay as TCP, namely one roundtrip time.

Multihoming

To increase robustness, servers are often equipped with multiple network interfaces. Such servers are said to be multihomed. SCTP was designed with multihoming in mind. During association establishment, an SCTP end-point can provide its

0890-8044/03/$17.00 © 2003 IEEE

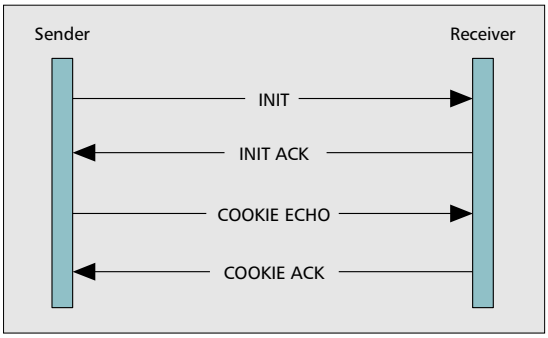

Figure 1. *SCTP four-way handshake.*

peer with a list of IP addresses or host names. One destination address is marked as the primary, and is used by the SCTP end-point to send data to the peer under normal circumstances. Should the primary destination address become unavailable, the end-point starts sending traffic to another address on the list provided by the peer. This feature allows SCTP associations to survive certain network failures that would make a TCP connection collapse.

Message Orientation

Like UDP, SCTP is a message-oriented protocol while TCP is stream-oriented. SCTP delivers messages to the application, whereas TCP delivers a stream of bytes. Message-oriented protocols such as SCTP and UDP simplify application parsers. When these transport protocols deliver a message to the application, it contains a single signaling message. Conversely, in order to extract a signaling message from the stream of bytes received over TCP, it is necessary to implement application-level framing.

Every application-layer message transported by SCTP is referred to as a DATA chunk. SCTP bundles DATA chunks so that a single IP datagram can carry several DATA chunks, increasing network efficiency.

Figure 2 shows the format of an SCTP message when bundling is performed.

Multiple Streams within an Association

SCTP provides multiplexing and demultiplexing of DATA chunks within an association. A single association can contain several streams, where each stream is identified by its stream id. The number of streams in both directions is determined during the four-way handshake (Fig. 1).

Every DATA chunk belongs to one stream; streams are delivered to the application independent of one another. This independent delivery of streams resolves the Head of the Line (HOL) blocking problem present in TCP. HOL blocking occurs when the data associated with multiple logical sessions is interleaved over a single TCP connection. This problem is common in HTTP/1.1, where images and HTML from a single server are retrieved in one TCP connection. If a segment for one of the images gets lost, the delivery of messages for the HTML text has to wait until the image segment has been retransmitted. Thus, a logical session blocks because of loss in another, unrelated one. Similar problems occur if multiple signaling sessions are multiplexed onto a single TCP session between two signaling servers. With SCTP, each application-level session would be assigned its own stream, so that the loss of a packet in one stream does not affect progress in other streams.

An SCTP association can contain several types of streams, each offering a different service. The base SCTP specification [1] defines two services: reliable ordered delivery and reliable

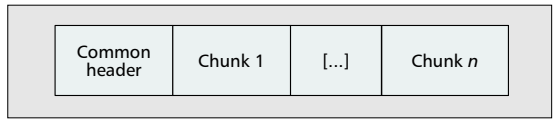

Figure 2. *SCTP message format.*

unordered delivery. Extensions to SCTP [8] describe an unreliable delivery service. In the ordered service, the receiver receives messages in the same order as they were sent by the sender. The unordered service delivers messages to the receiver as soon as they are received from the network, regardless of the order in which the sender produced them. It is important to note that a particular service is provided to each DATA chunk. Therefore, one DATA chunk within a stream might be delivered in order, whereas another DATA chunk within the same stream uses unordered delivery.

Flow and Congestion Control per Association

Even if an association contains several streams, SCTP performs flow and congestion control for the whole association. This allows us to use the behavior of all the traffic within the association as input to the flow and congestion control mechanisms, which are based on the ones used by TCP.

SCTP flow control is based on TCP SACK [9]. SACK is an extension to TCP that uses selective ACKs in addition to the cumulative ACKs. The cumulative ACK acknowledges the reception of all the data within a connection with a sequence number less than a certain number, whereas the selective ACK acknowledges the reception of non-contiguous ranges of sequence numbers.

The cumulative ACK is the main mechanism to detect packet losses in TCP. If a TCP end-point receives packets 1, 2, 3, 5, and 7, it will send ACK (1), ACK (2), ACK (3), ACK (3), and ACK (3) again. When the sender receives ACK (3) multiple times, it knows that packet 4 was lost and thus has to be retransmitted. However, the sender does not know which packets with a higher sequence number than 4 arrived successfully at the peer and which ones were lost as well. If the receiver had used the SACK TCP option it would have returned ACK (3)-SACK (5) and (7). With this information the sender is able to retransmit not only packet 4, but also packet 6. Performance measurements [10] show that TCP SACK recovers better than TCP Reno and New Reno from multiple losses in a single TCP window.

There are only two differences between TCP SACK and SCTP. The 40 bytes available for TCP options can only carry a maximum of four SACK blocks per TCP segment, while SCTP does not impose any limit on the number of SACK chunks in a packet. Secondly, TCP SACK uses the information in SACKs only to retransmit lost segments, while SCTP uses this information to perform both flow and congestion control. SCTP uses SACKed DATA chunks to increase its congestion window, whereas TCP SACK does not. In this way SCTP achieves a slightly higher window growth rate than TCP SACK under congestion when both DATA chunks and SACKs get lost.

SCTP is "TCP-friendly," that is, it uses about the same amount of bandwidth as a TCP connection under the same network conditions. This is not surprising since SCTP and TCP both use the same window-based congestion control. The sender implements a congestion window (`cwnd`) that limits the number of bytes that can be injected into the network at a given point of time. The rate at which `cwnd` grows depends on the state of the connection. If no congestion is detected, the slow start algorithm doubles `cwnd` every time a whole window of data is acknowledged. Under packet loss, the congestion avoidance algorithm increases `cwnd` more slowly, namely, linearly rather than exponentially.

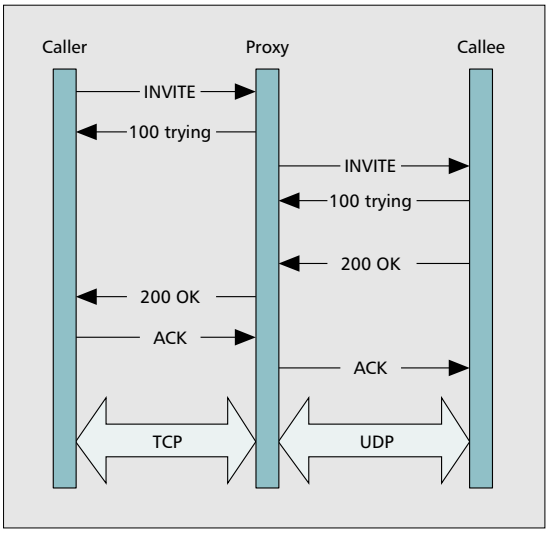

Figure 3. *SIP message exchange.*

Session Initiation Protocol (SIP)

SIP is a text-based application-layer protocol used to establish multimedia sessions. SIP can use any transport protocol. Currently, UDP is most widely used, but most implementations are now supporting both UDP and TCP.

SIP endpoints negotiate the parameters of a particular multimedia session using an offer/answer model. One end-point provides a session description — typically using the Session Description Protocol (SDP) [11] — that contains the network address and port number where it wants to receive media and the media format, described by the voice and video codecs to be used. SIP delivers this offer to the peer, who then responds with its own session description, the answer, in another SIP message. Once both endpoints have exchanged their session descriptions, they can send media to each other. SIP messages carry session descriptions in the same way as email messages carry attachments, using MIME [12].

SIP establishes sessions between user agents. Besides user agents, SIP defines intermediate network entities called SIP proxies. A SIP proxy receives a message from a user agent or from another proxy and relays it to the next SIP hop, which can be yet another proxy or the destination user agent.

Figure 3 shows the typical SIP message exchange to establish a session. The figure contains two user agents, caller and callee, and a proxy. The caller sends an `INVITE` request indicating that the caller is willing to establish a session with the callee. The caller then receives two responses, a "100 Trying" response followed, if the callee answers, by a "200 OK" response. The former indicates that the request has been received; the callee uses the "200 OK" to indicate that she is willing to join this multimedia session. (The callee has a list of error status codes to indicate various failures.) SIP is based on an HTTP-like request/response transaction model and therefore SIP responses follow the same format used in HTTP. Upon reception of the "200 OK" response, the caller sends an `ACK` request to acknowledge receiving this response.

SIP uses two two-way handshakes to establish a session: "`INVITE`-100 Trying" and "200 OK-`ACK`". The first handshake is between neighboring SIP nodes, with each response generated by the next hop. The second handshake is between the request origin and its final destination, i.e., two user agents. The "200 OK" response still traverses all hops, while the `ACK` request may bypass intermediate hops and go straight from caller to callee or proxy servers may ask to be in its path.

In Fig. 3, the proxy responds to the `INVITE` request received from the caller. Therefore, a single transport protocol is used for this type of handshake. In the second handshake, the response is generated by the remote SIP user agent; proxies in the path simply relay the response. Thus, end-to-end handshakes can involve multiple transport protocols. In the example of Fig. 3, the "200 OK-`ACK`" handshake uses two different transport protocols: TCP from caller to proxy and UDP from proxy to callee.

This article focuses on the "`INVITE`-100 Trying" hop-by-hop handshake. We have chosen this handshake because the traffic behavior between two particular nodes only depends on a single transport protocol. Implementing different transport protocols between the same nodes under the same network conditions shows how efficient each transport protocol is. However, in the end-to-end handshake, different transport protocols affect the traffic behavior. A packet drop by UDP, for instance, may trigger an application-layer retransmission over a TCP connection. We will have this situation in Fig. 3 if the `ACK` request is dropped in the UDP leg. The caller would have to resend the same `ACK` over its TCP connection. These interactions between application-layer and transport-layer reliability are outside the scope of this article.

SIP over Unreliable Transports

For unreliable transport protocols, the hop-by-hop handshake is implemented using timeouts and application-layer retransmissions. The client retransmits the `INVITE` until the "100 Trying" response is received. The server retransmits the "100 Trying" each time it receives a retransmission of the `INVITE`. `INVITE` requests are retransmitted first after 500 ms, with the interval doubling after that. Clients typically consider the next hop unreachable if no response has been received after the sixth retransmission.

SIP over Reliable Transports

If the hop-by-hop SIP handshake traverses a reliable transport connection such as TCP, the client sends the `INVITE` over the transport connection and the server returns the "100 Trying" response over the same connection. The transport layer undertakes the task of delivering the message to the next hop, so that no application-layer retransmissions are needed.

SIP over SCTP — SIP over SCTP follows the rules for TCP, so that requests and responses are sent over the same association [7]. We map all SIP traffic onto a single stream using the SCTP unordered service, so that incoming SIP messages are delivered to the application in the same order as they arrive from the network, regardless of the order in which they were sent. This helps avoid the HOL blocking problem present in TCP. (An alternative discussed in [7] is to assign each SIP session its own SCTP stream. This also avoids HOL blocking and may speed up matching requests to state entries, but requires more stream resources.)

Introduction to the Simulations

All our simulations were carried out using the network simulator (ns-2) [13]. Figure 4 shows the network topology we used. Nodes 2 and 3 are buffer-limited droptail routers, so that packets that arrive at the router when the buffer is full are dropped. The other nodes are endpoints, with node 1 being a SIP client and node 4 a SIP server. In our simulations, we study the behavior of the SIP traffic between these two nodes using different transport protocols.

We also studied the behavior of SIP traffic when it is competing with two bulk-data TCP connections. Nodes 5 and 7 send TCP traffic to nodes 6 and 8, respectively. All the traffic generated by these two sources shares the link 2-3 with the SIP traffic generated by node 1. When nodes 1, 5, and 7 generate too much traffic, a FIFO queue in router 2 builds in length. When this queue is full, router 2 drops incoming packets.

The link between routers 2 and 3 introduces a 15 ms propagation delay and has a bandwidth of 1.7 Mb/s. The links between the endpoints and the nearest router introduce a delay of 15 ms and have a bandwidth of 10 Mb/s. We chose the delay values so that the total transmission delay between the SIP endpoints is 45 ms, which corresponds to the transmission delay between a SIP server in the U.S. and another located in Europe. The bandwidth values were chosen so that the only bottleneck in the network is link 2-3. We wanted to have a high link utilization for the traffic patterns employed in our simulations. Based on personal communications with a number of vendors, current high-capacity SIP servers are able to handle approximately 330 requests per second, the equivalent of 1.2 million busy hour call attempts. If we assume that each SIP request is 500 bytes long, the 1.7 Mb/s link 2–3 sees a utilization of about 80 percent.

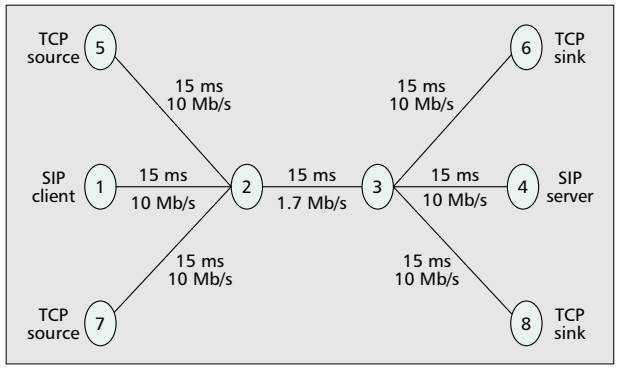

Figure 4. *Network topology.*

Metrics Measured

We measured the instant when the application passed the SIP message to the transport protocol at the sender side and the moment when the transport protocol delivered the message to the application on the receiver side. These two values allow us to calculate the delay introduced by the transport protocol.

Implementation

We added an SCTP implementation to the ns-2 [13] C++ core, based on the existing TCP SACK implementation. We also implemented the application-layer retransmissions used when SIP runs over UDP.

The SIP application in node 1 simulates a SIP proxy that handles SIP signaling related to many sessions. Every time this proxy receives a session establishment request (`INVITE`) it relays it to node 4. We simulated the arrival of these requests to the proxy in node 1 as a Poisson process. This process is a common choice when simulating signaling between traditional telephony switches. It is assumed that session establishment attempts are independent between each other.

Comparison Between Transport Protocols

We distinguish two network scenarios to study SIP traffic: traffic between a user agent and a proxy, and traffic between two proxies. SIP traffic patterns differ significantly in these two scenarios. A user agent typically handles a single session, whereas proxies usually handle many sessions at the same time. (Proxies are often run by organizations or carriers, so that all traffic from one organization to another would traverse the same proxy pair.) Therefore, there is typically much more traffic between two proxies than between a user agent and a proxy. We focus on the proxy-to-proxy scenario because it can take advantage of all the SCTP features. In the user-agent-to-proxy scenario, a connectionless protocol such as UDP avoids the connection-establishment time, thus reducing the call setup delay to one round-trip time as long as messages are lost. However, more work is needed to study interactions between lossy links such as some radio interfaces and transport-layer or application-layer retransmissions.

Connection-Oriented vs. Connectionless Transport Protocols

Connectionless transport protocols such as UDP reduce call setup latency since there is no need to establish a connection before sending the first SIP message. However, for the proxy-to-proxy case, proxies will keep the TCP or SCTP connection to other high-traffic proxies alive, thus avoiding the per-session setup latency. (We assume that most sessions connect to a relatively small number of next-hop proxies. There currently is insufficient operational experience to confirm this hypothesis.)

We show that UDP's late packet loss detection, lack of congestion control mechanisms, and lack of transport-layer fragmentation make UDP unsuitable for proxy-to-proxy communications.

Loss Detection — UDP treats each SIP message without regard to the previous ones. Message loss is always detected by an application-layer timeout. Connection-oriented transport protocols, on the other hand, take advantage of high traffic loads by multiplexing multiple SIP messages together in a single transport connection. When an SCTP end-point detects a gap in the transmission sequence number (TSN) of the incoming DATA chunks, it sends back a SACK for every new DATA chunk received, reporting the gap in the TSNs. The sender, upon reception of a preconfigured number of SACKs (typically, four) indicating a missing TSN assumes that the DATA chunk corresponding to this TSN has been lost. The sender then retransmits the missing DATA chunk without waiting for a timeout. Retransmitting a particular DATA chunk before a timeout occurs is referred to as the fast retransmit algorithm [2] and has been borrowed from TCP. Fast retransmit detects packet losses faster than timeouts as long as four or more messages are sent per round-trip interval and assuming a single packet loss.

We simulated SCTP and UDP traffic under loss conditions. We generated the same traffic pattern to be carried first by UDP and then by SCTP between nodes 1 and 4 of Fig. 4. In both simulations we had the router 2 drop the packet that was sent at t = 12,195 ms.

The x-axis of Fig. 5 shows the instant when a particular SIP message was passed to the transport protocol at the sending side and the y-axis shows the delay from that moment until the message was received by the peer application. In normal conditions the delay introduced by the transport protocol is less than 49 ms (45 ms of propagation delay plus the transmission delay introduced by the routers and the source), as can be seen in Fig. 5. The SIP message that was dropped had a final delay after having been retransmitted of 551 ms in UDP and only 176 ms in SCTP. We used 500 ms as the value for the UDP timer, as described earlier. (Even if we assume a

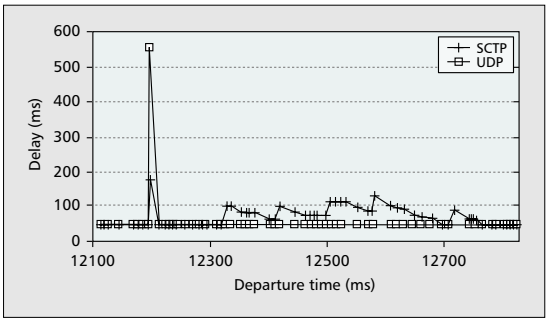

Figure 5. *SCTP vs. UDP.*

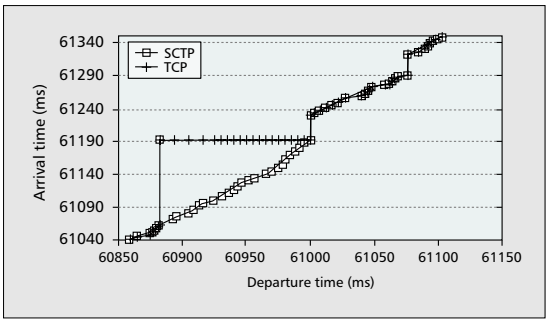

Figure 6. *HOL blocking effects.*

better timer estimation algorithm such as the TCP algorithm, the UDP timer value would still be approximately 300 ms.) This example shows that under conditions of heavy load, connection-oriented transport protocols detect packet loss much faster than timeout-based connectionless transport protocols.

Congestion Control — Figure 5 shows that SCTP delays SIP messages by approximately 125 ms after the loss event, whereas all subsequent messages sent using UDP have the same delay of approximately 49 ms. The SCTP delay penalty is caused by the SCTP congestion control mechanisms. After a SCTP sender detects a packet loss, it reduces its sending window to avoid congesting the network even more. This decrease in SCTP throughput introduces small delays in the SIP messages that follow a packet loss. UDP, on the other hand, continues transmitting at the same rate as before because it lacks congestion control mechanisms. However, this improved delay comes at the cost of increasing congestion and thus likely causing additional timeouts.

Transport-Layer Fragmentation — SIP messages carry a payload that is typically a session description written in SDP [11]. The size of an average SIP `INVITE` request is approximately 500 bytes, with 175 bytes consumed by the session description and 325 bytes consumed by SIP header fields. However, new XML-based session description formats such as SDPng [15] can increase this average size dramatically. Besides, SIP messages sometimes carry large payloads such as, for example, a JPEG picture of the caller or callee. It is foreseen that compression [16] will play an important role in reducing the size of SIP messages, but nothing guarantees that a particular SIP message does not become larger than the path MTU (Maximum Transmission Unit). In that situation, if UDP is used, the UDP packet containing that particular SIP message will be fragmented into several IP datagrams.

SCTP TCP		
Mean delay (ms)	397.4	400.8
Mean ratio	1.0000	1.0085
Variance	72,735.5	72,416.9
C. I. for the difference of means	(–6.95, 0.15)	

Table 1. *Performance with losses due to a buffer-limited router.*

IP fragmentation can lead to loss amplification. If a UDP packet that contains a SIP message is spread across several IP datagrams and one of them is lost, the whole UDP packet needs to be retransmitted. All the fragments that were successfully received are simply discarded, leading to further opportunities for packet loss.

Network address translators (NATs) and firewalls also do not work well with fragmented IP packets. NATs are common for consumer broadband access. When an IP packet is fragmented, the TCP, UDP, or SCTP header of the packet is only in the first fragment. Since the transport header contains the destination and source port numbers, NATs or firewalls will not be able to find this information in the fragments after the first one and are likely to discard the fragment. This is a general NAT limitation, not a shortcoming specific to any particular transport protocol.

IP fragmentation can be avoided by using transport-layer fragmentation. Both TCP and SCTP perform path MTU discovery (PMTUD) [17] using router ICMP responses. For IPv4, the sender sets the DF (Do not fragment) bit in the IPv4 header. IPv6 does not perform router-level fragmentation to begin with. If a router receives a message that exceeds the interface MTU, it discards the packet and returns an ICMP error message to the sender. For TCP, the sender then reduces its TCP segment size to conform to this MTU size. In SCTP, messages are split across multiple DATA chunks. PMTUD is not without practical problems [18].

We have discovered an important SCTP limitation when the path MTU changes. The new, smaller DATA chunks have consecutive TSNs. However, once a DATA chunk is sent, even if a notification from the network arrives indicating that it was too large, SCTP cannot fragment it anymore. Fragmenting it would require the use of contiguous TSNs for all the fragments. However, by the time SCTP notices that the packet was too large, the next TSNs have typically already been assigned to other DATA chunks. These chunks might even have already arrived at the destination. If it cannot assign consecutive TSNs for the split-up DATA chunks, SCTP performs IP fragmentation and sends the DATA chunk using several IP packets. If the peer is behind a NAT, it will never receive the DATA chunk. This problem might cause a whole SCTP association to timeout after trying to retransmit the oversized DATA chunk several times.

Changes in the path MTU can occur due to changes in the network path, but fortunately, these are not frequent. Nevertheless, this situation, although not common, can eventually cause a whole SCTP association to collapse. Thus, we conclude that the TCP byte count behaves better than the message count of SCTP. The lack of mechanisms in UDP to avoid IP fragmentation makes it rather unsuitable as a transport protocol for large signaling messages.

SCTP vs. TCP

In the following sections we compare TCP SACK with SCTP. The differences between SCTP and TCP without SACK can be easily deducted from our results and a comparison between TCP SACK and TCP without SACK [10].

Congestion Control — As noted earlier, SCTP was designed to be "TCP friendly." SCTP uses the same window-based congestion control mechanisms as TCP, namely, slow start, congestion avoidance, fast retransmit, and fast recovery [15]. The sender's congestion window (`cwnd`) limits the number of bytes that can be inserted into the network at any given point in time. If all the packets within a window are received by the receiver, the sender assumes that the current sending data rate is not causing congestion, and therefore the data rate is increased by increasing `cwnd`.

These algorithms assume that `cwnd` is limiting the amount of data that the sender inserts in the network. However, if the application generates data more slowly than `cwnd` allows, the application layer limits the amount of traffic inserted into the network. This scenario is very common for signaling applications since networks handling signaling traffic are usually overdimensioned to ensure low latency and robustness under load spikes.

When the data rate is limited by the application instead of by `cwnd` and no congestion is encountered at that rate, SCTP and TCP make `cwnd` grow dramatically. If a sudden burst of traffic arrives, the application layer will pass a large amount of data to the transport layer at once. This uncontrolled growth of `cwnd` causes all this data to be sent at once, typically causing network congestion that leads to packet losses and transport-layer timeouts.

We saw in some simulations how the window-based congestion control mechanisms used by TCP and SCTP could not cope with sudden traffic bursts under the previously mentioned conditions. Techniques such as those described in [19] may address this issue.

Head of Line Blocking — Earlier we described HOL blocking as one of the best known limitations of TCP when carrying multiplexed application sessions such as proxy-to-proxy signaling. As noted, SCTP was designed to avoid this problem. The undesirable effect of HOL blocking only manifests itself when multiple concurrent SIP sessions send traffic over a single TCP connection. If the transport protocol in use does not have the HOL blocking problem (e.g., SCTP), only those transactions that encounter message loss will show above normal overall delay.

We performed simulations in order to see how significant HOL blocking really is and to measure the performance benefits offered by SCTP. We analyzed the following scenarios:

- Packet losses caused by a buffer-limited router.
- Induced packet losses.
- Packet losses in the presence of competing TCP traffic.

Figure 6 shows an example of HOL blocking. The *x*-axis represents the time when a particular SIP message is passed to the transport layer by the sender, and the *y*-axis indicates when that message is delivered to the receiving application by the transport layer.

This simulation (Fig. 6) shows the behavior of a TCP connection and an SCTP association under the same traffic. The SCTP association avoids HOL blocking, whereas the TCP connection does not. In the experiment, the packet that was sent at t = 60,882 ms by node 1 was lost. Node 1 retransmits this packet and this retransmission finally arrives at node 1 at t = 61,193 ms. Before this message is finally received, node 4 has been receiving new DATA chunks containing new SIP messages from node 1.

The graph shows that while SCTP delivers all the SIP messages received regardless of the packet loss (bottom line in the graph), TCP buffers all this data without passing it to the application (top line). When the retransmission of the lost packet is finally received, TCP passes all the data at once to the application layer. This sudden delivery of several SIP messages is represented by the flat line in the graph. The delay introduced by HOL blocking to a particular packet is the distance between the two lines. When a packet is not affected by HOL blocking, both lines coincide.

	0.2% packet loss		0.3% packet loss	
	SCTP	TCP	SCTP	TCP
Mean delay (ms)	455.9	459.3	840.8	844.8
Relative to SCTP	1.0000	1.0074	1.0000	1.0047
Variance	382,063	382,171	862,894	862,563
C. I. for the difference of means	(–11.58, 4.71)		(–16.18, 8.30)	

Table 2. *Induced packet loss.*

The Tables 1, 2, and 3 show our simulation results. In each simulation, the SIP application at node 1 of Fig. 4 passed 45,000 SIP messages with an exponential interarrival time distribution to the transport layer. We ran the simulation until all the SIP messages were received by the SIP application at node 4. We discarded the delays of the first 1,000 packets in calculating the mean delay, so that the initial slow start did not influence our results. The mean delays contained in the tables are the result of running each simulation once. Future work is needed to perform a larger number of systematic simulations or real measurements to obtain more general results.

We first analyze the scenario in which packet losses are caused by a buffer-limited router. SIP traffic is exchanged between nodes 1 and 4 of Fig. 4. There is no competing traffic and therefore packet losses are only produced by the SCTP association sending too much data at a given moment, overloading the queue of router 1. For this simulation, we chose a data rate that loads the bottleneck link (2–3) to 78 percent. We chose this load because the effect of HOL blocking appears to be maximized at that load. When we increased the application rate above this, the system became unstable, with delays growing dramatically. Lower rates do not generate as many packet losses and thus HOL blocking is less likely to occur.

Table 1 contains the statistical analysis of the delays using both ordered and unordered SCTP. The 95 percent confidence interval for the difference between both means is (–6.95, 0.15). Since this interval spans zero, there is no statistically significant difference between the means at the 95 percent confidence level.

Next, we analyze HOL blocking behavior under randomly induced packet losses. To analyze this scenario, we reduced the average data rate until we had a 50 percent utilization of the bottleneck. This rate does not cause any packet loss in the network, so we had router 2 drop 0.2 percent or 0.3 percent of the packets. Table 2 summarizes the results. Both 95 percent confidence intervals for the difference between the means span zero. Therefore, HOL blocking does not introduce a statistically significant delay in these two scenarios either.

The next scenario we analyze includes competing TCP traffic. We reduced the average data rate at which the application generates SIP messages to 39 percent of the bottleneck capacity and had node 5 transfer a large file to node 6 over a TCP connection at the same time. Our SCTP association competed with that TCP connection in the link between nodes 2 and 3. We then reduced the average data rate down to 28 percent of the bottleneck capacity and established a second file transfer between nodes 7 and 8, making the SCTP association compete with two TCP flows. Table 3 shows the results of both

	One competing TCP flow		Two competing TCP flows	
	SCTP	TCP	SCTP	TCP
Mean delay (ms)	757.2	771.2	589.1	606.9
Relative to SCTP	1.0000	1.0184	1.0000	1.0302
Variance	361,544	361,980	582,583	598,105
C. I.	(–21.92 , –6.06)		(–27.89 , –7.63)	

Table 3. *Competing traffic.*

simulations. This time, SCTP does offer a statistically significant improvement, albeit of no more than three percent.

Not surprisingly, these results show that the effects of HOL blocking grow with the number of packet losses. If a packet loss causes a timeout rather than a fast retransmit, the effect of HOL blocking is greatest because no new data is received from the moment the packet loss is detected until the timeout occurs. HOL blocking keeps the transport from delivering this data to the application, so the longer this period, the bigger the HOL blocking effect is. Fast retransmit, as opposed to timeouts, shortens this period.

Therefore, HOL blocking becomes significant only under heavy network congestion causing many packet losses and many timeouts. However, even SCTP, which avoids HOL blocking, is sometimes not able to provide an acceptable delay under such circumstances. The 95th delay percentiles in the simulations of Table 3 were 1,633 ms and 2,196 ms, respectively. Signal delays of this order of magnitude are found unacceptable by many applications.

Conclusions

Implementing application-layer retransmissions over UDP can be useful when carrying a small amount of signaling traffic, but is inappropriate for heavy traffic loads. For significant signaling loads, the fast retransmit algorithms and their congestion control mechanisms make TCP and SCTP much better choices than UDP.

SCTP has some advantages over TCP. It provides some protection against denial of service attacks, it allows multihomed hosts to use all their available IP interfaces for a particular association, and it delivers complete signaling messages to the application rather than a stream of bytes.

However, our simulations show that the supposedly largest advantage of SCTP over TCP — HOL blocking avoidance — does not provide a substantial performance increase under normal circumstances. Under network conditions suitable for signaling traffic (i.e., moderate traffic loss) the mean delays achieved by SCTP and TCP are not statistically different at the 95 percent confidence level. However, for higher levels of packet loss the effects of HOL blocking increase and SCTP achieves a modestly better delay performance.

While our simulations focus on the mean delay between two adjacent SIP entities (hop-by-hop handshake), operators and regulators are typically interested in end-to-end delay percentiles (e.g., 95 percent of the calls must be connected under two seconds after dialing). Our future work includes studying the number of user sessions affected by HOL in a particular link, as opposed to the change in the average message delay, as our current study measures.

We also found a limitation in the transport-layer fragmentation provided by SCTP. When the path MTU decreases suddenly, SCTP is not able to avoid IP fragmentation. IP fragmentation causes a decrease in transport performance, since successfully received fragments need to be retransmitted if one of the fragments is lost. In addition, port-aware NATs that do not keep track of fragments and that do not perform datagram reassembly may be unable to route fragmented datagrams.

We can conclude that SCTP is somewhat better suited to signaling transport than TCP, providing a slight performance increase from the end user's perspective. In general, the worse the network conditions are, the bigger the performance increase offered by SCTP.

Theoretically, window-based congestion control mechanisms similar to those implemented by TCP and SCTP would not be suitable for signaling transport. These mechanisms were not designed to cope with sudden bursts of traffic when the initial traffic rate is limited by the application rather than by the congestion window. Therefore, future work on congestion control mechanisms for signaling traffic is needed in order to analyze whether or not this is a real issue.

References

[1] R. Stewart *et al.*, "Stream Control Transmission Protocol," RFC 2960, IETF, Oct. 2000.
[2] Q. Xie, R. Stewart, and M. Allman, *Stream Control Transmission Protocol (SCTP)*, Addison-Wesley, Pub Co, 2001.
[3] L. Coene and J. Pastor, "Telephony Signalling Transport over SCTP Applicability Statement," Internet Draft, IETF, Nov. 2002, Work in progress.
[4] J. Rosenberg *et al.*, "SIP: Session Initiation Protocol," RFC 3261, IETF, June 2002.
[5] J. Postel, "Transmission Control Protocol," RFC 793, IETF, Sept. 1981.
[6] J. Postel, "User Datagram Protocol," RFC 768, IETF, Aug. 1980.
[7] J. Rosenberg, H. Schulzrinne, and G. Camarillo, "The Stream Control Transmission Protocol as a Transport for the Session Initiation Protocol," Internet Draft, IETF, July 2002, work in progress.
[8] Q. Xie, R. Stewart, and C. Sharp, "SCTP Unreliable Data Mode Extension," Internet Draft, IETF, April 2001, work in progress.
[9] M. Mathis *et al.*, "TCP Selective Acknowledgment Options," RFC 2018, IETF, Oct. 1996.
[10] K. Fall and S. Floyd, "Simulation-Based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Comp. Commun. Review*, vol. 26, no. 3, July 1996, pp. 5–21.
[11] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, IETF, Apr. 1998.
[12] N. Borenstein and N. Freed, MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies," RFC 1341, IETF, June 1992.
[13] http://www.isi.edu/nsnam/ns/index.html
[14] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, IETF, Apr. 1999.
[15] D. Kutscher, J. Ott, and C. Bormann, "Session Description and Capability Negotiation," Internet Draft, IETF, July 2002, work in progress.
[16] R. Price *et al.*, "Signaling Compression," Internet Draft, IETF, June 2002, work in progress.
[17] J. McCann, S. Deering, and J. Mogul, "Path MTU Discovery for IP version 6," RFC 1981, IETF, Aug. 1996.
[18] K. Lahey, "TCP Problems with Path MTU Discovery," RFC 2923, IETF, Sept. 2000.
[19] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," RFC 2861, IETF, June 2000.

Biographies

GONZALO CAMARILLO (Gonazalo.Camarillo@ericsson.com) received M.Sc. degrees in electrical engineering from the Stockholm (Sweden) Royal Institute of Technology and from Universidad Politecnica de Madrid (Spain). He heads the Advanced Signalling Research Laboratory in Ericsson Finland. His research interests include signaling, multimedia applications, and network security.

RAIMO KANTOLA (Raimo.Kantola@hut.fi) received a M.Sc. in applied mathematics and computer science at the Leningrad Electrotechnical University in 1981 and D.Sc. degree in computer science at the Helsinki University of Technology in 1995. He had a diverse industrial career in R&D, product marketing and research at Nokia prior to joining Helsinki University of Technology as a professor of communications technology in 1996. His current research interests include QoS in the Internet, routing, naming and addressing, and signaling over IP.

HENNING SCHULZRINNE (hgs@cs.columbia.edu) received degrees from Darmstadt (Germany) University of Technology, the University of Cincinnati, and the University of Massachusetts in Amherst. He has held research positions at GMD Fokus, Berlin and Bell Laboratories before joining the faculty of Columbia University, New York. His research interests encompass real-time network services, the Internet, and modeling and performance evaluation.