Signalling Transport Protocols

Gonzalo Camarillo
Advanced Signalling Research Lab.
Ericsson
Finland
Gonzalo.Camarillo@ericsson.com

Henning Schulzrinne
Department of Computer Science
Columbia University
USA
hgs@cs.columbia.edu

Raimo Kantola
Networking Laboratory
Helsinki University of Technology
Finland
Raimo.Kantola@hut.fi

February 12, 2002

Abstract

SCTP is a newly developed transport protocol tailored for signalling 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 paper 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.

1 Introduction

The limitations present in TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) for transporting signalling 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) [13]. SCTP was first intended to transport telephony signalling over an unreliable network such as an IP network. However, the protocol has been designed so that SCTP can be used as a general-purpose message-based transport protocol.

In this paper, we compare SCTP with already existing transport protocols and analyze which real gains SCTP brings. Interestingly, our simulations show that Head of the Line (HOL) blocking, which is widely believed to be TCP's biggest limitation regarding transport of signalling traffic, does not introduce a significant performance decrease in most of the cases. 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) [12]. SIP is an application-layer protocol for creating, modifying and terminating sessions. SIP basic operation is independent of the lower-layer transport protocol, which makes it an ideal choice to compare the performance of different transports.

The SIP specification [12] describes how the protocol operates over TCP [10] and over UDP [9]. TCP provides reliable in-order transfer of bytes while UDP does not ensure either reliability nor in-order delivery. We defined in [11] how SIP can run on top of SCTP in order to encourage experimentation and to be able to carry out our performance analysis in our simulator. Although there are already implementations of telephony signalling protocols such as ISUP on top of SCTP, no SIP implementations use SCTP at present.

The remainder of this paper is organized as follows. Section 2 and 3 introduce SCTP and SIP respectively. Section 4 introduces the simulation environment. Sections 5 and 6 analyze, based on simulations, advantages and disadvantages of using SCTP, UDP and TCP as transport protocols for SIP and finally section 7 outlines some conclusions.

2 Stream Control Transmission Protocol (SCTP)

The Stream Control Transmission Protocol (SCTP) [13] is a transport protocol tailored to transport signalling traffic. SCTP inherits all its congestion and flow control mechanisms from TCP but includes a number of improvements aimed to make it a more efficient signalling transport than TCP.

2.1 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 figure 1, protects servers against Denial of Service (DoS) attacks performing IP address spoofing, which are possible with the TCP handshake. This kind of attack is known as SYN flooding.

The SCTP receiver, upon reception of an INIT message, sends back a cookie in a 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. This way, resources are allocated 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 inform the sender that the COOKIE ECHO massage has been successfully received and that it contained a valid cookie.

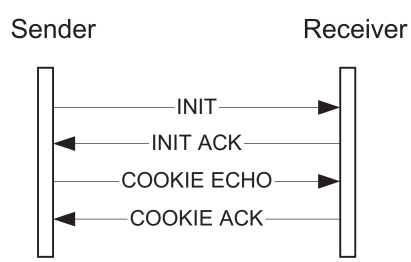

Figure 1: SCTP four-way handshake

The last two messages of the four-way handshake can already carry user data. This data piggybacking makes the association establishment delay the same as the connection establishment delay in TCP; one Round Trip Time (RTT).

2.2 Multihoming

Mutihomed servers have multiple IP interfaces. Multihoming improves the robustness of a server because if one of its IP interfaces fails, the server can still send and receive traffic using another interface.

SCTP was designed with multihoming in mind. During association establishment, an SCTP end point can provide its peer with a list of IP addresses or host names. One destination address is marked as the primary, and it is used by the SCTP end point to send data to the peer under normal circumstances. Should the primary destination address eventually become unavailable, the end point starts sending traffic to another address of the list provided by the peer. This feature allows SCTP associations to survive certain network failures that would make a TCP connection collapse.

2.3 Message Orientation

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 allow implementing simpler parsers. When these transport protocols deliver a message to the application it contains a single signalling message. Conversely, in order to extract a signalling message within 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. This increases network efficiency.

2.4 Multiple Streams within an Association

SCTP provides multiplexing/demultiplexing of DATA chunks within an association. A single association can contain several streams. Each stream is identified by its stream id. During the four-way handshake (Figure 1), the number of streams in both directions is determined.

Every DATA chunk belongs to a particular stream, and every individual stream is delivered to the application independently from the others. This independent delivery of streams resolves the Head of the Line (HOL) blocking problem of TCP. HOL blocking occurs when the signalling associated with multiple sessions is sent over a single TCP connection between two servers. A TCP client sends two singalling messages over the TCP connection: the first message (message A) corresponds to a session between two end users and the second message (message B) to another session between two different end users. If message B arrives successfully to the TCP peer but message A gets lost, TCP will not deliver message B to the application until message A is retransmitted and is finally received. Therefore, the session to which message B belongs is affected by the loss of message A, which belongs to an unrelated session between two different users. When SCTP is used, a loss in a particular stream does not affect the rest of them.

An SCTP association can contain several types of streams. The base SCTP specification [13] defines two services: reliable ordered delivery and reliable unordered delivery. There are extensions [14] that provide 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 the sender produced them. It is important to note that a particular service is provided on a DATA chunk basis. Therefore, one DATA chunk within a stream might be delivered in order whereas another DATA chunk within the same stream uses unordered delivery.

2.5 Flow and Congestion Control per Association

Even if an association contains several streams, SCTP performs flow and congestion control per association. This allows 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 [7]. 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 endpoint 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 got 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 got 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 [3] show that TCP SACK recovers better than TCP Reno and New Reno from multiple losses in a single TCP window.

The only difference between TCP SACK and SCTP is that whereas TCP SACK uses the information in SACKs only to perform flow control (retransmit lost segments), 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. This way SCTP achieves a slightly higher window growth rate than TCP SACK under congestion when both DATA chunks and SACKs get lost.

Both TCP and SCTP use window-based congestion control. The sender implements a congestion window (cwnd) that limits the number of bytes that can be inserted into the network at a certain 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 size every time a whole window of data is acknowledged. Under packet loss, the congestion avoidance algorithm increases cwnd more slowly (i.e., linearly rather than exponentially).

3 Session Initiation Protocol (SIP)

SIP is a text-based application layer protocol used to establish multimedia sessions. SIP is transport independent; although at present UDP is the most widespread transport for SIP there are many implementations of SIP over TCP as well.

SIP end points negotiate the parameters of a particular multimedia session using an offer/answer model. One end point elaborates a session description – typically using the Session Description Protocol (SDP) [4] - that contains the IP address and port number where it wants to receive media and the media format (e.g., voice and video codecs to be used). SIP delivers this session description (the offer) to the peer who then elaborates its own session description (the answer) and sends it back 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 [2].

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

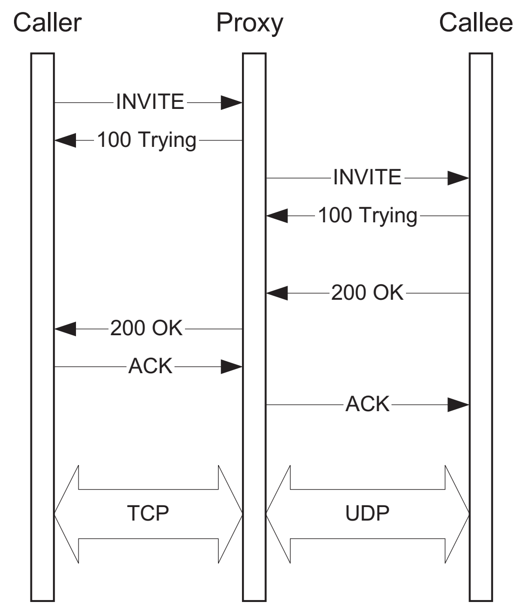

Figure 2: SIP message exchange

Figure 2 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. This request indicates that the caller is willing to establish a session with the callee. The caller then receives two responses: first a ”100 Trying” response and then a ”200 OK” response. The former indicates that some action is being performed in order to process the INVITE request and the latter indicates that the callee is willing to join this multimedia session. SIP is based on HTTP and therefore SIP responses follow the same format used in HTTP. Upon reception of the ”200 OK” response the caller sends an ACK to acknowledge the reception of this response.

Therefore, SIP uses two two-way handshakes to establish a session: ”INVITE-100 Trying” and ”200 OK-ACK”. The first handshake is hop-by-hop. The second is end-to-end. The node that generates the response for the request is different in both handshakes. In a hop-by-hop handshake the next SIP hop in the path generates the response. In figure 2, the proxy responds to the INVITE request received from the caller. Therefore, a single transport protocol is used for this type of handshake. In an end-to-end handshake the response is generated by the remote SIP end point. Proxies in the path simply forward the request and later the response. Thus, end-to-end handshakes can involve multiple transport protocols. In the example of figure 2, the ”200 OK-ACK” handshake uses two different transport protocols: TCP from caller to proxy and UDP from proxy to callee.

This paper 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 is. However, in an 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 figure 2 if the ACK is dropped in the UDP leg. The caller would have to resend the same ACK over its TCP connection. These interactions between transport protocols are outside the scope of this paper.

3.1 SIP over unreliable transports

The hop-by-hop handshake is implemented over unreliable transports using timeouts and application-layer retransmissions. The client retransmits the INVITE until the ”100 Trying” response is received. The server retransmits the ”100 Trying” response upon reception of retransmissions of the INVITE. The timer used for INVITE retransmissions typically begins at 500 ms and doubles every time the INVITE is retransmitted. This produces the following intervals between retransmissions: 1 s, 2 s, 4s, 8s, 16s and 32s. Clients typically consider the next hop unreachable if no response has been received after the 6th retransmission.

3.2 SIP over reliable transports

In the hop-by-hop SIP handshake over a reliable transport 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. No application-layer retransmissions are needed. When TCP is used the INVITE and the response are sent over the same TCP connection.

3.2.1 SIP over SCTP

SIP over SCTP follows the rules above. [11] defines how SIP messages are mapped into SCTP streams. The mapping we use for this analysis consists of sending all the SIP traffic over a single stream (stream ID= 0) and using the SCTP unordered service. This implies 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.

4 Introduction to the simulations

All our simulations were carried out using the network simulator (ns). Figure 3 shows the network topology that we used. Nodes 2 and 3 are buffer-limited droptail routers. The rest of the nodes are endpoints. Node 1 is a SIP client and node 4 is a SIP server. In our simulations we study the behavior of the SIP traffic between these two nodes using different transport protocols.

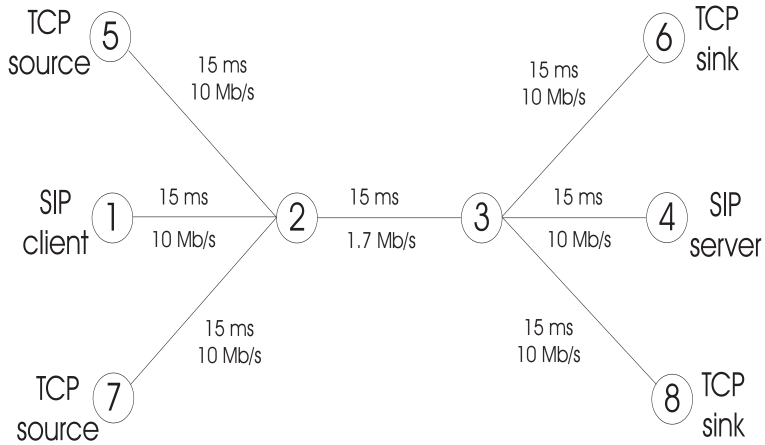

Figure 3: Network Topology

We also simulated the behavior of SIP traffic with competing TCP connections. Node 5 is a TCP traffic source and node 6 is its sink. Node 7 is too a TCP source and node 8 its sink. 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 is built in router 2. When this queue is full router 2 drops incoming packets.

The link between routers 2 and 3 introduces a 15ms delay and has a bandwidth of 1.7 Mb/s. The links the endpoints and a 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 both SIP endpoints is 45 ms. This value corresponds to the transmission delay between a hypothetical SIP server in the US and another located in Europe. The bandwidth values were chosen so that the only bottleneck in the network is the link 2-3. We wanted to have a high link utilisation for the traffic patterns employed in our simulations. Current high capacity SIP servers are able to handle up to 333 requests per second (the equivalent to 1.2 million busy hour call attempts for a telephone switch). Assuming 500 byte SIP requests, a bandwidth of 1.7 Mb/s achieves a link utilisation of 85%.

4.1 Measurable Metrics

For every SIP message in a particular simulation we measured the instant when the application passed the message to the transport protocol at the sender side and the moment when the transport protocol passed the message to the application on the receiver side. These two values allow to calculate the delay introduced by the transport protocol to every packet. We then calculated the statistical variables that describe the delay distribution of a whole particular simulation.

4.2 Implementation

We added an SCTP implementation to the ns C++ core. The network simulator provides a TCP SACK implementation. We took this implementation as a base to build our SCTP agent. The network topology and the traffic patterns produced by the application were defined in tcl/tk scripts. 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 signalling related to many sessions. Every time this proxy receives a session establishment request (INVITE) it forwards it to node 4. We simulated the arrival of these requests to the proxy as a Poisson process. This process is a common choice when simulating signalling between traditional telephony switches. It is assumed that session establishment attempts are independent between each other.

5 Comparison between Transport Protocols

We distinguish between two network scenarios to study SIP traffic:

- Traffic between a user agent and a proxy
- Traffic between two proxies

SIP traffic patterns differ notably in these two scenarios. A user agent typically handles a single session whereas proxies usually handle many sessions in parallel at the same time. 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 reducing significantly the call setup delay. Thus, as long as SIP messages are small enough to avoid IP fragmentation, UDP is a better choice than TCP or SCTP for user agents (section 5.1.3 discusses fragmentation in further detail). Note that a single session does not produce enough traffic to make UDP's lack of congestion control mechanisms a disadvantage.

Therefore, the traffic analyzed in the following sections consists of a large number of SIP transactions between two proxies. An example of this configuration is the traffic between a proxy that acts as the outbound proxy for users within a service provider's domain and the inbound proxy for another domain. Every SIP message sent by a user in the first domain to a user in the second domain traverses both proxies.

5.1 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, this is not an advantage over TCP and SCTP in this scenario, given that proxies handling a high level of traffic have long lived connections between them. All the SIP messages are sent over the same connection, so it is not necessary to establish a new connection every time a SIP message has to be sent to the remote proxy.

Sections 5.1.1, 5.1.2 and 5.1.3 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.

5.1.1 Loss Detection

UDP treat each SIP message separately from the rest. Message loss is always detected by a timeout. Connection oriented transport protocols, on the other hand, take advantage of high traffic loads by bundling multiple SIP messages together in a single transport connection. When an SCTP endpoint 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 hole in the TSNs. The sender, upon reception of a preconfigured number of SACKs (typically 4) indicating a missing TSN assumes that the DATA chunk corresponding to this TSN has been lost. At this point the sender retransmits the missing DATA chunk without waiting for a timeout. This way of retransmitting a particular DATA chunk before a timeout occurs is referred to as fast retransmit algorithm, and it has been borrowed from TCP. Fast retransmit detects packet losses faster than timeouts as long as 4 messages or more are sent per RTT (assuming a single packet loss).

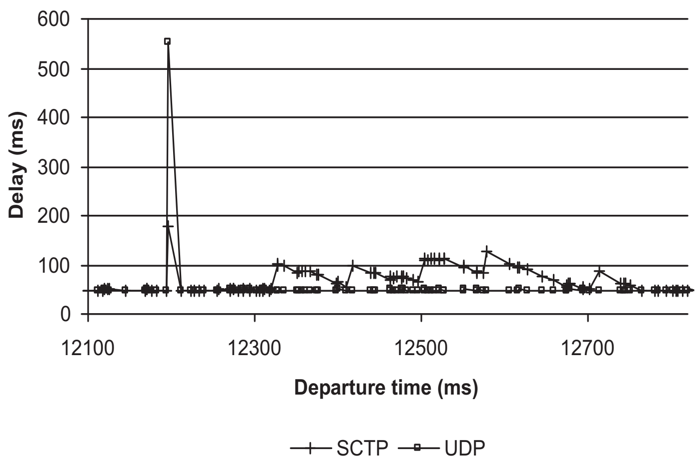

Figure 4: SCTP vs. UDP

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 figure 3. In both simulations we had the router 2 drop the packet that was sent at t=12195 ms.

The abscissa of figure 4 shows the instant when a particular SIP message was passed to the transport protocol at the sending side (in ms) 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 processing delay introduced by the routers and the transport protocol stacks), as can be seen in figure 4. 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. This result shows that under heavy load connection oriented transport protocols detect packet loss much faster than timeout based connectionless transport protocols.

5.1.2 Congestion Control

The tail of the graph of figure 4 shows that SCTP introduces delays up to around 125 ms in SIP messages sent afterwards whereas all subsequent messages sent using UDP have the same delay around 49 ms. This delay is caused by the SCTP congestion control mechanisms. After SCTP 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. In this simulation the packet that got lost was dropped by the filter we installed in the router rather than by a congested router, so subsequent packets suffered a very low delay. However, in a congested network, this aggressive UDP behaviour simply worsens the situation of the network creating extremely high delays.

5.1.3 Transport Layer Fragmentation

SIP messages carry a payload that is typically a session description written in SDP [4]. The size of an average SIP INVITE request is around 500 bytes (175 bytes for the SDP and 325 for SIP header fields). However, new XML-based session description formats such as SDPng [6] 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 [5] 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. In that situation, if UDP is used, the UDP packet containing that particular SIP message will be sent using several IP datagrams.

There are some issues related to IP fragmentation. If a UDP packet that contains a SIP message is sent using several IP datagrams and one of them gets lost, the whole UDP packet will be retransmitted. All the fragments that were successfully received are simply discarded. This leads to a serious performance decrease.

Another important reason why IP fragmentation should be avoided as far as it is possible are Network Address Translators (NATs) and firewalls. When IP fragmentation takes place the TCP, UDP or SCTP header of the packet is in the first fragment. The rest of the fragments do not carry any transport header. Since the transport header contains the destination and source port numbers any NATs or firewall will not be able to find this information in the fragments after the first one. This implies that any IP packet that gets fragmented cannot traverse any NAT or firewall that inspects port numbers. Since some access technologies such as cable modems place every user behind a NAT this issue becomes very important.

IP fragmentation can be avoided using transport layer fragmentation. TCP and SCTP perform path MTU discovery [8]. IPv6 routers do not perform fragmentation and IPv4 traffic sent with the DF (Don't Fragment) bit of the IPv4 header set is not fragmented by routers either. Thus, if TCP or SCTP send a message that is larger than the path MTU, an ICMP notification is received from the network. Upon reception of this notification TCP repacketizes the data. It reduces the segment size and sends fewer bytes of data in each segment. This ensures that IP fragmentation never takes place. In SCTP, however, we have discovered an important limitation when the path MTU changes. If the application passes SCTP a message that is larger than the path MTU SCTP fragments it and sends it in different DATA chunks. These 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 to use contiguous TSNs for all the fragments. However, by the time SCTP notices that the packet was too large, the next TSNs have typically been already assigned to other DATA chunks, that might even have been received already by the peer. SCTP then 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 to retransmit the oversized DATA chunk several times until the association is considered down.

Changes in the path MTU can occur due to changes in the routing topology, but fortunately, they are not frequent. Nevertheless, this situation, although not common, can eventually cause a whole SCTP association to collapse. So, we conclude that the TCP byte count behaves better than the message count of SCTP. UDP's lack of mechanisms to avoid IP fragmentation makes it very unsuitable as a transport protocol for large signalling messages.

5.2 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 [3].

5.2.1 Congestion control

SCTP uses the same window-based congestion control mechanisms as TCP, namely slow start, congestion avoidance, fast retransmit, and fast recovery [1]. The sender's congestion window (cwnd) limits the number of bytes that can be inserted into the network at a certain point of 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 is the one limiting the amount of traffic inserted in the network. This scenario is very common, since networks handling signalling traffic are usually over dimensioned so that they provide more capacity than needed by the application under normal circumstances.

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 occurs, the application layer will pass a large amount of data to the transport layer at once. This uncontrolled growth of cwnd causes that all this data is sent to the network suddenly, typically causing a massive network congestion that leads to massive packet losses and transport layer timeouts.

Our simulations show that the window-based congestion control mechanisms used by TCP and SCTP cannot cope with sudden traffic bursts under the previously mentioned conditions. Future congestion control mechanisms tailored to transport bursty signalling traffic could bring significant performance gains to both SCTP and TCP.

5.2.2 Head of the Line Blocking

Section 2.4 already described HOL blocking and stated that SCTP was designed to avoid this problem, which is probably the most well-known TCP limitation regarding signalling transport. We performed some 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 presence of competing TCP traffic

Figure 5 shows how HOL blocking works. The X-axis represents the time when a particular SIP message is passed to the transport layer by the sender (in ms) and the Y-axis represents when that message is delivered to the receiving application by the transport layer (in ms).

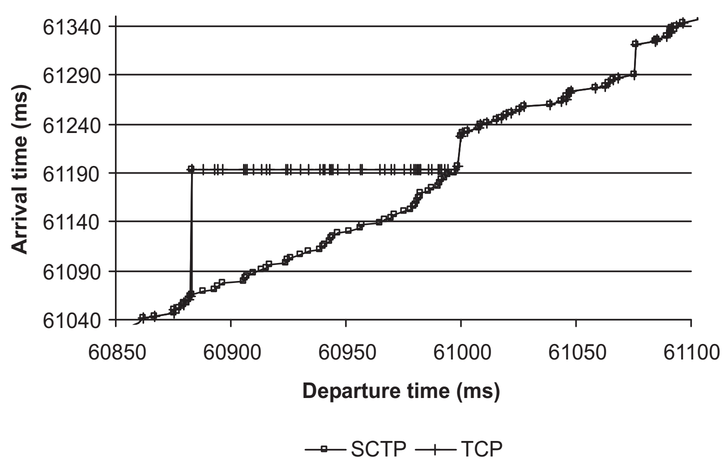

Figure 5: HOL blocking effects

This simulation (Figure 5) shows the behaviour of a TCP connection and an SCTP association under the same traffic: the SCTP association avoids HOL blocking whereas the TCP connection does not.

The packet that was sent at t=60882 ms by node 1 gets lost. Node 1 retransmits this packet and this retransmission finally arrives to node 1 at t=61193 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 lost (bottom line in the graph), TCP buffers all this data without passing it to the application (top line). When the retransmission of the packet lost 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 difference between the two lines. When a packet is not affected by HOL blocking both lines coincide.

We first analyze the scenario where packet losses are caused by a buffer-limited router. SIP traffic is exchanged between nodes 1 and 4 of figure 3. 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 achieves 85% of link utilisation in the bottleneck (link 2-3 in figure 3). We chose this rate because the effect of HOL blocking is maximum. At rates higher than that the throughput achieved by SCTP in this network topology is not enough to cope with the amount of data generated by the application. Delays start growing dramatically and the system becomes unstable. Rates lower than that do not generate as many packet losses and thus the effect of HOL blocking is lower.

Table 1 contains the statistical analysis of the delays using both, ordered SCTP and unordered SCTP.

Table 1: Losses due to a buffer-limited router

	SCTP	TCP
Mean	397.4 ms	400.8 ms
Mean Ratio	1.0000	1.0085
Variance	72735.5	72416.9
C. I. for the Diff of Means	$(-6.95, 0.15)$	

The 95% confidence interval for the difference between both means is (-6.95 , 0.15). Since this interval contains the value 0.0, there is not a statistically significant difference between the means at the 95% confidence level. Therefore, HOL blocking does not introduce enough delay to be able to say at the 95% confidence level that the mean delay is significantly higher.

Now we anaylize HOL blocking behaviour under randomly induced packet losses. To analyze this scenario we reduced the average data rate until we had a 50% utilisation of the bottleneck. This rate does not cause any packet loss in the network so we had router 2 drop 0.2% or 0.3% of the packets. Table 2 contains the results of these simulations.

Table 2: Induced packet loss

	0.2% packet loss		0.3% packet loss	
	SCTP	TCP	SCTP	TCP
Mean	455.9 ms	459.3 ms	840.8 ms	844.8 ms
Mean Ratio	1.0000	1.0074	1.0000	1.0047
Variance	382063	382171	862894	862563
C. I. for the Diff of Means	$(-11.58, 4.71)$		$(-16.18, 8.30)$	

Both 95% confidence intervals for the difference between the means contain the value 0.0. Therefore, in these two scenarios HOL blocking does not introduce a statistically significant delay 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% 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% of the bottleneck's capacity and established a second file transfer between nodes 7 and 8. Now the SCTP association competed with two TCP flows. Table 3 shows the results of both simulations.

Table 3: Competing traffic

	1 competing TCP flow		2 competing TCP flows	
	SCTP	TCP	SCTP	TCP
Mean	757.2 ms	771.2 ms	589.1 ms	606.9 ms
Mean Ratio	1.0000	1.0184	1.0000	1.0302
Variance	361544	361980	582583	598105
C. I. for the Diff of Means	$(-21.92, -6.06)$		$(-27.89, -7.63)$	

Interestingly, this time the confidence intervals for the difference between the means do not include the value 0.0. Therefore, in both scenarios HOL blocking introduces a statistically significant delay at the 95% confidence level.

Not surprisingly, these results show that the effects of HOL blocking grow with the number of packet losses. Besides, if a packet loss causes a timeout rather than a fast retransmit, the effect of HOL blocking is maximum because from the moment the packet loss is detected until the timeout occurs new data is being transmitted. HOL blocking keeps the transport from delivering this data to the application, so the longer this period is the bigger the HOL blocking effect is. Fast retransmit, as opposed to timeouts, shortens this period, which is characterized by having both lines of the graph of figure 5 separated.

Therefore, only under heavy network congestion (many packet losses and many timeouts) HOL blocking becomes significant. However, even SCTP, which avoids HOL blocking, is sometimes not able to provide an acceptable delay under such circumstances. The 95th percentiles in the simulations of table 3 were 1633.4 ms and 2196.4 ms respectively. Signalling related delays of this order of magnitude are found unacceptable by many architectures.

6 Conclusions

Whereas implementing application layer-retransmissions over UDP can be useful to transport a small amount of signalling traffic, such a transport mechanism is not appropriate under a heavy traffic load. The fast retransmit algorithm and their congestion control mechanisms make TCP and SCTP better choices than UDP to transport heavy signalling traffic.

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 delivers complete signalling messages to the application rather than a stream of bytes.

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

We also found an important limitation in the transport layer fragmentation provided by SCTP. When the path MTU decreases suddenly, SCTP is not able to avoid IP fragmentation. This might cause a whole SCTP association to collapse if the traffic traverses port-aware NATs of firewalls.

We can conclude that SCTP is a better signalling transport protocol than TCP that provides 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 is.

Our simulations showed that window-based congestion control mechanisms like the ones TCP and SCTP implement are not suitable for signalling transport. These mechanisms cannot 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 signalling traffic is needed in order to design even more suitable signalling transports than the current SCTP.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. Request for Comments 2581, Internet Engineering Task Force, April 1999.

[2] N. Borenstein and N. Freed. MIME (multipurpose internet mail extensions): Mechanisms for specifying and describing the format of internet message bodies. Request for Comments 1341, Internet Engineering Task Force, June 1992.

[3] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno, and SACK TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[4] M. Handley and V. Jacobson. SDP: session description protocol. Request for Comments 2327, Internet Engineering Task Force, April 1998.

[5] H. Hannu. Signaling compression requirements and assumptions. Internet Draft, Internet Engineering Task Force, November 2001. Work in progress.

[6] D. Kutscher, J. Ott, and C. Bormann. Session description and capability negotiation. Internet Draft, Internet Engineering Task Force, November 2001. Work in progress.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Request for Comments 2018, Internet Engineering Task Force, October 1996.

[8] J. McCann, S. Deering, and J. Mogul. Path MTU discovery for IP version 6. Request for Comments 1981, Internet Engineering Task Force, August 1996.

[9] J. Postel. User datagram protocol. Request for Comments 768, Internet Engineering Task Force, August 1980.

[10] J. Postel. Transmission control protocol. Request for Comments 793, Internet Engineering Task Force, September 1981.

[11] J. Rosenberg, H. Schulzrinne, and G. Camarillo. SCTP as a transport for SIP. Internet Draft, Internet Engineering Task Force, November 2001. Work in progress.

[12] J. Rosenberg, H. Schulzrinne, et al. SIP: Session initiation protocol. Internet Draft, Internet Engineering Task Force, October 2001. Work in progress.

[13] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. Request for Comments 2960, Internet Engineering Task Force, October 2000.

[14] Q. Xie, R. Stewart, and C. Sharp. SCTP unreliable data mode extension. Internet Draft, Internet Engineering Task Force, April 2001. Work in progress.