Considerations of SCTP Retransmission Delays for Thin Streams

Jon Pedersen[1], Carsten Griwodz[1,2] and Pål Halvorsen[1,2]
[1]Dept. of Informatics, University of Oslo, Norway
[2]Simula Research Laboratory, Norway
Email: {jonped, griff, paalh}@ifi.uio.no

Abstract

The popularity of distributed interactive applications has exploded in the last few years. For example, massive multi-player online games have become a fast growing, multi-million industry with a very high user mass supporting hundreds or thousands of concurrent players. Today, such games are usually client-server applications that use TCP for time-dependent communication. Similar multimedia applications also choose TCP frequently. Very thin data streams are sent over each of these TCP connections, which means that they consume very little bandwidth. TCP has several shortcomings with respect to the latency requirements of such thin streams because of its retransmission handling [7].

An alternative to TCP may be SCTP [13] which was developed to answer the requirements for signaling transport. SCTP has subsequently also been considered more appropriate than TCP for congestion controlled streaming of other time-dependent data. Important reasons are its maintenance of packet boundaries and partial reliability.

In this paper, we evaluate the performance of the Linux SCTP implementation for thin streams. Like others before, we identify latency challenges. We also propose some enhancements for reducing the latency compared to the original Linux implementation. We argue for separate handling of thin and thick data streams in SCTP.

Keywords: *SCTP, thin streams, latency reduction*

1 Introduction

Distributed interactive applications have historically either been developed for use with transport protocols that can provide per-stream QoS guarantees, or with protocols that at least allow the sender of the application to determine timing of the data transmission. The QoS protocols for the first approach have not become widely available. The use of UDP, a protocol that provides for the second approach, has been widely criticized for its lack of congestion control mechanisms. This is perhaps the strongest criticism although all the required mechanisms can be implemented on top of UDP at the applications level, and although the acclaimed application layer framing principle [3] seems to encourage its use for certain multimedia applications that can benefit from increased control over their traffic.

Although it is currently an established opinion that bandwidth is no longer a limiting factor in the Internet, congestion control remains in demand also for streaming applications. One reason is that access networks are not yet free from bandwidth problems. For this and other reasons, distributed interactive applications are today built over TCP.

In parallel, alternative congestion controlled protocols are being developed. One of them, the stream control transmission protocol (SCTP) [13], is on the standards track of the IETF. SCTP was developed to answer the requirements for signaling transport identified by RFC2719 [10]. It offers services like acknowledged error-free non-duplicated transfer of user data, data fragmentation to conform to discovered path MTU size, packet boundary maintainance, sequenced delivery of user messages within multiple streams, options for order-of-arrival delivery of individual user messages, and optional bundling of multiple user messages into a single SCTP packet. An extension allows for partial reliability. Several of these services are valuable in a scenario with time-dependent data, and SCTP may be considered more appropriate for congestion controlled streaming than TCP. However, TCP and SCTP implementations are being developed with throughput-bound applications in mind, not for latency-critical, low bandwidth streams.

RFC2719 includes the following requirement concerning latency for network management messages in section 4.2.1: "MTP Level 3 peer-to-peer procedures require response within 500 to 1200 ms. This value includes round trip time and processing at the remote end. Failure to meet this limitation will result in the initiation of error procedures for specific timers, e.g., timer T4 of ITU-T Recommendation Q.704". Since SCTP is meant to address this challenge, among others, we wondered how well this requirement is

total time	total #bytes	total #packets	average packet size	bandwidth requirement
1855 s	44974424	372711	120 bytes	0.194 Mbps

Table 1. Anarchy Online packet statistics (New Reno with FACK, 100 ms delay, 1% loss)

fulfilled and whether it means that the protocol is an appropriate choice for very low data rate (thin) streams such as games traffic as well.

There are several kinds of games, but hardly any consume bandwidth greedily. Some of them require latencies below 150 ms to work appropriately which is difficult to achieve with TCP [7], and if SCTP fulfills the requirements of RFC2719, it may also work well for such games. We have therefore taken a look at the 2.6.15 Linux SCTP implementation *lksctp*. Initial results from a master thesis [11] that we present in this paper show that SCTP has severe problems with very low bandwidth streams as they are generated by games. This was unexpected because we would not expect signaling traffic to have a high volume, and SCTP was after all designed to address the requirements of signaling traffic. We have thereafter considered possible remedies for the existing latency problems.

We are not the first ones to notice the latency problems of SCTP. Brennan and Curran [2] performed a simulation study for greedy traffic and identified weaknesses in fast retransmit procedure. Their modifications would seem to increase delays for thin streams, however. Problems with carrying time sensitive data over SCTP were presented by Basto and Freitas [1]. The traffic that they considered was loss-tolerant, and they proposed the use of SCTP's partial reliability extensions [12]. Ladha et al. [8] examine several approaches to detect spurious retransmissions and propose fixes that would increase throughput but also increase the latency of individual lost packets.

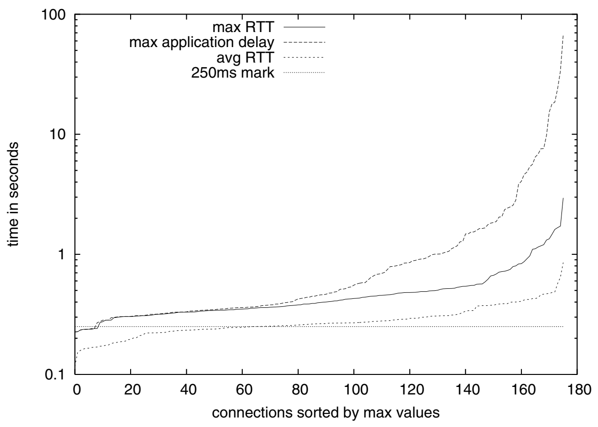
Grinnemo and Brunstrom [6] discuss the problem of minRTO, among other things, and propose a reduction to fulfill the requirements of RFC4166 [5], an RFC on the applicability of SCTP for telephony. The RFC itself discusses problems and solution approaches in its sections 3.2.1 and 3.3. Section 3.3 proposes to choose the path within a multihomed association that experiences the shortest delay, an approach that is impractical for our scenario. Section 3.2.1 considers both reductions of the minRTO and the removal of exponential back-off, but warns of both alternatives. Removing delayed SACK is mentioned without stating any side-effects. In our scenario, however, it is unreliable to rely on client-side changes. On the contrary, we propose the removal of exponential back-off under the condition that a connection consumes very little bandwidth, and we propose the general reduction of minRTO.

2. Transport protocols and thin streams

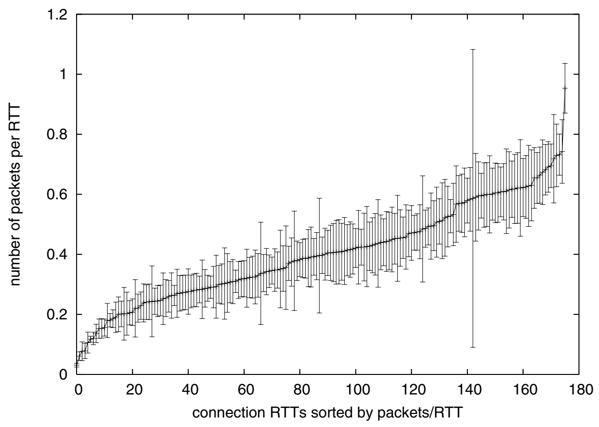
SCTP, like TCP, is designed for carrying elastic traffic from greedy sources. This implies that the mechanisms assume that a sender will always try to send data as quickly as the flow control and congestion control mechanisms of the transport protocol permit. Also most protocol variations for TCP are targeted at high-rate applications like high-quality video streaming or content download. However, several applications have very *thin* data streams, i.e., they consume very little bandwidth. They may still have stringent latency requirements. Mechanisms in the transport protocol that rely on a permanently filled pipe may fail in this situation.

To look at an example for such traffic, we have analyzed a one hour packet trace, containing all packets from one of a few hundred game regions in Funcom's popular MMORPG Anarchy Online [7]. Each machine in the centralized server cluster manages one or more regions, i.e., no region spans several machines. Furthermore, Anarchy Online keeps point-to-point, default (New Reno) TCP connections open, from one or more Linux servers to every client. Figure 1 and table 1 show some statistics gathered from the packet traces, and in figure 1(a), we see that the average RTT is somewhat above 250 ms with variations up to one second, i.e., these RTTs make the game playable [4]. However, looking at the maximum application delay (time for receiving a successful acknowledgment) which may include several retransmissions as a result of packet loss, we have extreme worst case delays of up 67 (!) seconds (6 retransmissions). Obviously, we cannot distinguish between lost packets and lost acknowledgments in this server-sided trace, but we can see the potential for several second-long delays in delivering packets to the application on the client side. Furthermore, figure 1(b) shows that, on average, less than one packet is sent per RTT. Combined with the measured RTTs, we see that the number of packets per second is low, below 4 packets per second. Additionally, each packet contains only small game events like position updates etc, and looking at table 1, we see that each packet is small (about 120 bytes on average). Thus, considering that each stream has very *few* packets per second and that each packet is very *small*, this demonstrates how thin the individual streams are. We further observe that it is not the loss rate itself that is unacceptable (it is slightly below 1 %), but the occasional huge delays when multiple retransmissions are needed.

Our results shows that it is important to distinguish between thick and thin streams, and we define the terms as follows. If the situation that the application has less data to send than supported by the transport protocol is an exception during the lifetime of a connection or if it does not occur at all, we consider the transported stream *thick*. At the opposite end of the option space are streams that hardly generate any traffic at all, and we called these streams *thin*.



(a) RTT versus maximum application delay



(b) Packets per RTT with standard deviation

Figure 1. Anarchy Online packet trace analysis

For this paper, we define the term thin formally and consider a stream *thin* whenever four or fewer packets of the connection are in flight (concurrently on the wire). This means that a stream can be temporarily thin.

In summary, the connections are so thin that 1) they hardly ever trigger fast retransmissions but mainly retransmit due to timeout and 2) TCP's congestion control does not apply, i.e., the TCP stream does not back off. We have seen earlier [7] that such streams are generally suffering badly from loss in TCP when multiple losses occur. Given the application area that SCTP is designed for, we have investigated whether SCTP performs better for thin streams.

3 Comparison of SCTP and TCP New Reno performance

To compare the performance of Linux's SCTP implementation *lksctp* and Linux's implementation of TCP New Reno for thin streams, we have compared their performance in several experiments. The thin streams in these experiments had a bandwidth of 400 bytes/second that were sent in four write operations of 100 bytes each. The round-trip time was varied. We then used emulation to investigate the end-to-end delays that were experienced in case of random packet loss. We consider it reasonable to investigate random packet loss instead of using cross-traffic in this scenario because the streams are so thin that the protocol does not react to congestion anyway. The application generates packets at a rate so low that the minimum packet rate of the transport protocol is only exceeded for very high RTTs. Transient congestion that is induced by cross-traffic will typically not persist for long enough that the evaluated connection can detect it. Persistent congestion will appear identical to random packet loss to the connection. For lower RTTs, the connection will not back off when congestion is detected because it is already sending at the minimal rate.

In SCTP, every 100 byte write operation creates a chunk. Every chunk has a sequence number (TSN) and can either be sent individually or bundled with other chunks into a packet. SCTP has two ways to trigger a retransmission of a chunk; after a retransmission timeout with a minimum retransmission timeout (minRTO) of 1000 ms and after the retrieval of 4 packets containing selective acknowledgments (SACKs) that indicate a loss of the chunk and trigger a retransmission of the chunk. It TCP it is not guaranteed that the boundaries of the write operations are maintained, but completeness and order of the transfer is. Sequence numbers indicate the reception of a connection's data up to a number of bytes. Also TCP has two ways to trigger a retransmission timeout after a minRTO of approximately 200 ms and after the retrieval of at least 3 packets containing acknowledgments that indicate the loss of a byte range and trigger entering in a fast retransmit phase. These values show immediately that a single, random loss would be handled more slowly by SCTP under most circumstances. The details are not quite as straight-forward, however, and warrant a closer look.

3.1 Duplicate acknowledgments

The conditions that must be met to trigger a retransmission by arriving acknowledgments for thin streams are stricter for SCTP than for TCP. Whereas TCP requires 3 duplicate ACKs to initiate a retransmission, SCTP requires 4 duplicate SACKs. For thick streams, this is a reasonable choice. This number can actually be higher for TCP, as Linux' TCP implements the Eifel algorithm [9], which is not standardized yet for SCTP. In any case, the Eifel algorithm can only increase the number of dupACKs required to initiate retransmission. It is meant to reduce unnecessary retransmissions, but not primarily to avoid the redundancy, but to avoid the reduction of *cwnd* that is associated with

RTT (ms)	Type	Number	Frequency	Min (ms)	Max (ms)	Avg (ms)
0	Retransmission timeout	282	76.2 %	999.1	1256.6	1005.5
	Fast retransmit	24	6.5 %	1024.4	1280.4	1088.4
	Reported lost and bundled	34	9.2 %	464.0	744.0	592.7
	Unsolicited bundling	30	8.1 %	231.8	744.0	274.7
100	Retransmission timeout	275	43.0 %	1039.9	1612.1	1049.8
	Fast retransmit	23	3.6 %	1126.5	1386.2	1173.1
	Reported lost and bundled	27	4.2 %	460.0	1356.1	689.3
	Unsolicited bundling	314	49.1 %	15.3	532.0	51.2
200	Retransmission timeout	266	40.1 %	996.2	1460.1	1144.6
	Fast retransmit	35	5.3 %	1228.4	1740.7	1274.2
	Reported lost and bundled	24	3.6 %	487.9	976.0	780.7
	Unsolicited bundling	338	51.0 %	28.0	888.0	172.8
400	Retransmission timeout	242	27.9 %	1343.0	1660.1	1352.0
	Fast retransmit	31	3.6 %	1427.2	1943.6	1496.2
	Reported lost and bundled	26	3.0 %	780.0	1430.1	1011.1
	Unsolicited bundling	567	65.5 %	11.8	832.0	213.4

Table 2. SCTP Cumulative retransmission statistics, first retransmission

a timeout. SCTP's approach to the problem is to wait for a higher number of duplicate SACKs by default. For thin streams that are oblivious to congestion control, neither approach is useful.

It is important to note that SCTP does not implement fast retransmit semantics. Once fast retransmit has been initiated, all following ACKs that report the segment as missing are ignored until either an ACK for the retransmitted segment is received or a retransmission timeout expires. This is not the case for SCTP. Here, every fourth duplicate SACK triggers a retransmission of the same chunk. For thick streams, this means that the sending rate is halved successively several times. On the other hand, it is not forced to enter slow start when a chunk is lost a second time as TCP must. Additionally, SCTP can perform new RTT measurements and calculate a new RTO. Instead of staying in exponential back-off after a retransmission timeout, the timer can be restarted by a SACK that arrives late. This allows a recalculation of the RTO value, the retransmission timer is reset to the new RTO value, and the exponential back-off is collapsed. The effect of this collapse can be seen in figure 2, where it becomes clear that retransmission timeouts do not exhibit the exponential growth that is seen in TCP.

3.2 Retransmission timeouts

The conditions that must be met to trigger a retransmission timeout differ for SCTP and TCP as well. By the definition of the minRTO as 1000 ms for SCTP and 200 ms for TCP it is quite clear that the condition triggers much later for SCTP than for TCP. When streams are thin, SCTP does thus have bigger problems than TCP. Almost all retransmis-

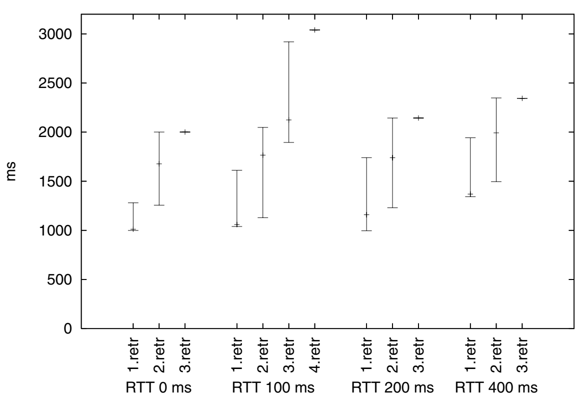


Figure 2. SCTP retransmissions by timeout

sions are triggered by a retransmission timeout unless it is delayed by late restarts and 4 SACKs have the chance to trigger a fast retransmit before the timer expires. With a minRTO of 1000 ms, this leads to average retransmission delays above 1000 ms. Even in TCP New Reno, most retransmissions are triggered by timeout as 3 duplicate ACKs can not be sent within 200 ms. The various combinations of SACK, DSACK and FACK mechanisms do not make any difference as too few packets are in flight. Aside from the larger minRTO, SCTP does not take late restarts of the retransmission timer into account when calculating a new RTT value, while TCP New Reno does. The development of TCP's RTO values is therefore more stable than SCTP's, which tends to overestimate the RTO value and thereby delay the expirations of the retransmission timer.

Another interesting observation that can be made from the figure is the effect of SCTP's use of delayed acknowledgments. When no loss is detected, SCTP uses delayed acknowledgments by default. In case of thick streams, this reduces the overhead of reverse traffic by sending only one acknowledgment for two received packets until loss is detected. However, the delay of the acknowledgment is necessarily limited by a timer, which in the case of *lksctp* is fixed to 200 ms. Since the average packet inter-arrival time in case of thin streams such as our game traffic example is higher than 200 ms, most packets are eventually acknowledged but the acknowledgments are usually 200 ms late. Since SCTP is unaware of the reason for the delay, the 200 ms delay is added to the estimated RTT, such that the retransmission timeout calculation is based on a measurement that is 200 ms too high. Using our thin streams, this occurrence is only broken under two conditions. The first condition arises in case of a packet loss, which triggers immediate sending of a SACK chunk by the receiver. If this occurs, the RTO gets closer to the RTT. The second is the loss of a packet with a SACK chunk. When this happens, the RTO value grows, because the SCTP implementation considers the first arriving SACK that acknowledges the sample packet. The sample is not even discarded when the arriving SACK chunk acknowledges data that was sent later than the sample packet, and as a result, the RTO value grows.

3.3 Bundling

The picture becomes less clear when *lksctp*'s aggressive use of chunk bundling for thin streams is considered. SCTP bundles the earliest outstanding chunks after a retransmission timeout as long as there is room in the packet according to the Path MTU. As messages in thin streams are typically small, several messages can be bundled into one packet in SCTP. A similar effect occurs in TCP, where message boundaries are not preserved, and a triggered retransmission sends up to one PathMTU worth of data if sufficiently much data is in flight. If SACK reports indicate several gaps with intermediate correctly arrived sequences of chunks, SCTP is in contrast to TCP able to bundle and retransmit the missing chunks in a single packet without additional overhead. As TCP is byte-oriented, a retransmitted sequence of bytes must be contiguous.

In the thin stream scenario, we care very little about spurious retransmissions because our main concern is the end-to-end latency of individual messages. If these spurious retransmissions happens in the form of bundling, we consider this advantageous. Bundling does not increase the number of packets in the network, unnecessary retransmissions of chunks consume little additional bandwidth and do not increase routing costs at all. Looking at table 2, we see that the number of retransmissions due to unsolicited chunk bundling is growing considerably with the RTT of a connection. We can see that at an RTT of 400 ms, 65.5 % of all retransmissions are due to unsolicited chunk bundling. Nearly all of these retransmissions are spurious. If this would happen with greedy traffic, it would obviously be highly undesirable. In the thin stream scenario, we do not have to conclude this automatically. At the cost of larger packets, we can observe a linear decrease in the number of retransmissions that are triggered by timeouts. Obviously, the aggressive chunk bundling hides some losses completely from the receiving side. This is good considering improved average message latency for these chunks. On the other hand, acknowledgments for those packets whose loss is hidden are taken into account in the computation of the RTO value, which is then growing.

3.4 Summary

Since SCTP is designed to address the demands of time-critical and probably thin signaling traffic, we would have expected that *lksctp* would handle packet loss in this scenario better than TCP. Actually, we saw that *lksctp* performs worse than TCP.

Considering SCTP's high minRTO, we would expect that dupACKs are the fastest trigger for retransmissions in thin as well as thick streams. However, for the game traffic that we have investigated, this is not the case. The packet rate is so low that an expiring minRTO timer is still the most frequent reason for retransmissions. In spite of this considerable overestimation of the RTT, the packet rate of game streams is still so low that expiration of the RTO timer is the most frequent reason for retransmissions.

Unsolicited retransmission by chunk bundling can improve the average end-to-end latency slightly. A two-fold price is paid for this but it may not be too high in case of thin streams. One is the waste of bandwidth, the other is the increase of the retransmission timeout for those few chunks that are selected for the RTO computation and that also are lost and recovered by unsolicited retransmission by bundling.

4. Enhancements

The investigation in section 3 shows us that the *lksctp* implementation in the Linux kernel is currently not better suited for games traffic than the TCP implementation. We have also seen that the implementation in its current state is not able to fulfills the requirements for signaling traffic that were defined by RFC2719 [10]. However, the advantages like maintenance of message boundaries and proactive retransmission by bundling of chunks, still make SCTP attractive for distributed interactive application. We would therefore like to introduce variations into SCTP which solve or improve some of its problems for thin streams.

There are several ideas for enhancing SCTP for thin streams that should be evaluated after the comparison with TCP New Reno. For example, since the number of SACKs are small and rarely have the chance to trigger a fast retransmit, a way to make use of fewer SACKs to trigger a fast retransmit should be considered. Furthermore, as the packets are relatively small, ways to put more data into one packet should be considered. Additionally, we consider the ideas that were mentioned in RFC4166 [5], reduction of the minRTO and removal of exponential back-off. The RFC warns that these variations have negative side-effects by increasing the danger of spurious retransmissions and reduction of the congestion window. We expect that this would not be a relevant problem in our particular scenario of thin streams. We would therefore like to apply these approaches, but only in case of thin streams. To do this, we are first facing the problem of identifying a stream as thin.

4.1 Thin stream detection

We define a stream as thin when fewer packets are in flight or have been reported missing than are necessary to trigger a fast retransmission. This is a conservative condition that prevents the stream from any other means of recovery from packet loss than waiting for the retransmission timeout (unless packet duplication occurs). For SCTP these are 4 packets, but the implementation does not easily yield the number of packets in flight. SCTP assigns sequence numbers to chunks and not to packets. As SCTP is message-oriented. It will, in contrast to TCP, not hold packets in the transmission queue and retransmit the same packet. Instead, SCTP manages chunks that are identified by a transmission sequence number (TSN) and bundles them as it sees fit when retransmitting. Thus, we additionally need means of keeping track of the number of packets in flight in order to determine whether a stream is thin or not.

We solve this issue by maintaining a list that holds the highest TSN for every packet that is in flight, as well as a counter of packets. If a packet is successfully received in the absence of loss, the highest TSN in the packet corresponds to the cumulative TSN of a SACK that is generated to acknowledge it. In presence of loss, the sender will find gap ack blocks in the SACKs that it receives. From the arrival of the SACK, the sender can conclude that a packet has left the network. The missing packets are still counted as in flight as they could arrive out of order.

The packets in flight handling must be performed for every association, i.e., the number of packets in flight is the same independent of the transport destination address in a multi-homed association. The reason for this is that the transmission queue and the cumulative TSN are shared between all destination transport addresses. Thus, it is the association that takes care of SACKs, independently of the destination address that the chunks are sent to. The association manages the transmission queue and if necessary retransmits chunks to another transport destination address.

We can at all times use the counter for packets in flight as an estimate for the number of packets that actually are in flight, and we can determine whether the thin stream variations or the original SCTP mechanisms should be applied. Thus, we can now differentiate between stream types, and we have therefore evaluated the following enhancements targeted for thin streams.

4.2 Modified retransmission timer

As timeouts are a frequent cause of retransmission in the thin stream scenario, we first look at approaches for reducing the retransmission latency related to the timeout value.

4.2.1 Minimum RTO

To avoid too early timeouts, SCTP has a rather high minimum retransmission timeout value (1000 ms). Nevertheless, in the thin stream scenario, our tests show that almost all retransmissions are due to timeouts. Therefore, we experimented with a minRTO of 200 ms equal to Linux TCP's minRTO value.

However, when the minimum RTO is reduced, the relative effect of delayed SACKs on the RTO calculation that was described in section 3.2 grows, as shown in figure 3(a). When removing the receiver side SACK timer, the RTO calculation is greatly reduced according to a lower measured RTT as shown in figure 3(b). Thus, although receiver side enhancements are harder to apply in a real system (as a huge amount of client machines must be updated), we performed measurements to see the effect on retransmission delay.

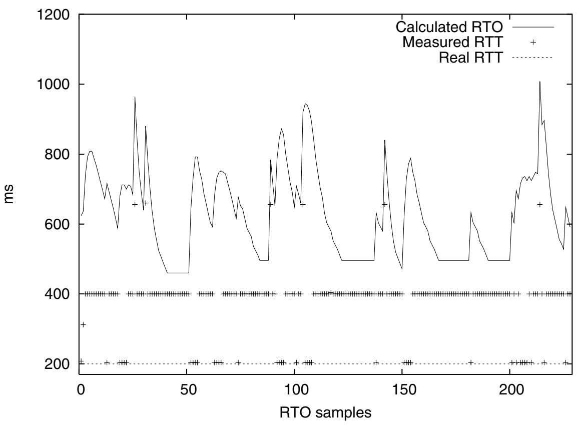
In addition, SCTP performs a retransmission timer restart with the current RTO at the time an incoming SACK acknowledges some, but not all outstanding chunks. In thin streams, late arrivals of such SACKs will occur often, and has not only the positive effects described in section 3.2. A late timer restarts also the first retransmission timer, increasing the average latency. We avoid this by adjusting the new expiration time before the timer is restarted, by subtracting the time since the old timer was started from the new one. Thus, no more than one RTO will elapse before the oldest unacknowledged chunk is retransmitted by a timeout.

4.2.2 Removal of exponential back-off

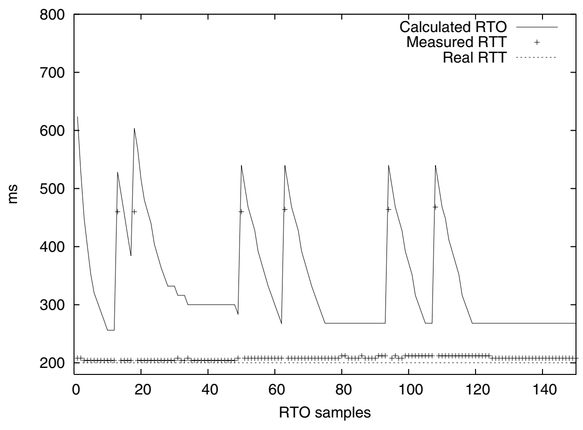
If there are too few SACKs to trigger a fast retransmit or no new packets are sent to let the receiver discover loss, retransmissions could be triggered by subsequent retransmission timeouts without any intervening fast retransmits. At this point, an exponential back-off of the retransmission timer is performed, leading to an exponential growth of the retransmission delay when there are occurrences of multiple loss. However, as the stream is so thin, the loss is not necessary an indication of heavy congestion. Therefore, we have experimented with a removal of the exponential back-offs when the stream is thin.

4.3 Modified fast retransmit

Despite the high minRTO value, fast retransmissions hardly ever appear in the thin stream scenario because not enough packets are sent to get the 4 SACKs needed before the retransmission timer expires. As retransmissions triggered by other causes than timeouts usually are preferable, the probability of reordered packets in thin streams is low and the amount of data sent is small, we allow a fast retransmit to be triggered by the first indication that a chunk is lost. This means that if the stream is thin, then a fast retransmit should be triggered by only one SACK. This may lead to more packets and reordering, but the receiver will probably still improve total performance by dropping a spurious retransmission than let the sender wait for several SACKs to discover possible reorderings.



(a) RTO calculated by SCTP



(b) RTO calculated by SCTP without delayed SACKs

Figure 3. RTO values for thin streams

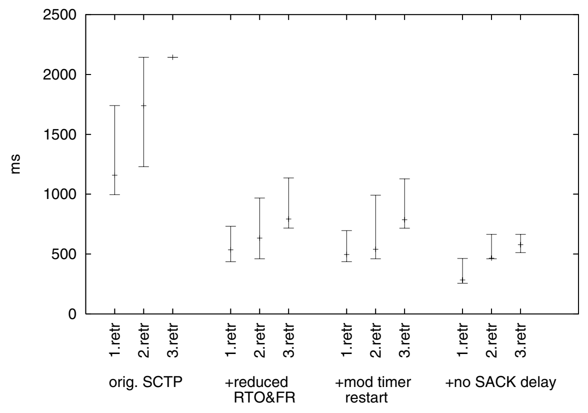


Figure 4. Effects of proposed enhancements

Test Scenario	Type	Number	Min	Max	Avg
orig. SCTP	Timeout	266	996.2	1460.1	1144.6
	Fast retransmit	35	1228.4	1740.7	1274.2
+reduced RTO+FR	Timeout	197	435.8	732.0	626.7
	Fast retransmit	284	460.3	731.2	472.3
+mod timer restart	Timeout	331	436.0	696.0	525.0
	Fast retransmit	288	460.2	464.3	462.3
+no SACK delay	Timeout	633	255.7	448.0	282.9
	Fast retransmit	1	463.0	463.0	463.0

Table 3. First retransmission, RTT = 200

4.4 Modified chunk bundling

SCTP uses as described in section 3.3 an aggressive bundling strategy in case of timeouts. However, in case of fast retransmits, only chunks marked for fast retransmit is sent. Therefore, to always allow SCTP to take advantage of free space in a packet, we have changed this to apply aggressive bundling also for fast retransmit. Similarly, if room in a packet, we bundle not acknowledged chunks with new chunks.

4.5 Evaluation

To see the effects of the proposed thin stream enhancements, we ran tests using the same test setup as in the previous section. The test results (see figure 4 and table 3) are compared against the test results of the original SCTP protocol.

In the first experiment, we combined a minRTO reduction and the modified fast retransmit (denoted *+reduced RTO+FR*). The first observation is that compared to the original SCTP implementation, the number of fast retransmits has drastically increased and the number of retransmissions triggered by timeouts is reduced. We also see a large reduction in the average retransmission latency - for any number of retransmissions. The next test additionally included the timer restart modification (denoted *+mod timer restart*). We now see that the delay is further reduced, but we have a few extra retransmissions. In the last experiment, we also removed the SACK timer delay (denoted *+no SACK delay*). Our results show that this has a huge impact, together with the new minRTO, on the latency, but it will also trigger several spurious retransmission (and it require receiver side modifications).

In summary, all our enhancements improve upon the original lksctp implementation with respect to retransmission delay in a thin stream scenario. Note also that it is not only the average latency that is reduced, but the large maximum peeks are also greatly reduced. This means that in a time dependent application like an interactive online game, the perceived gaming experience will be greatly improved. However, all these enhancements do not come for free. We do see an increased number of retransmissions, but as these streams require so little amount of resources in the first place (small and few packets), an extra overhead of a few more packets is negligible compared to the improved service at the application level.

5. Conclusions

We investigated the use of SCTP for thin streams. Our investigation started with the assumption that SCTP should exhibit considerably reduced latencies for thin streams than TCP, since it was designed with time-critical signaling traffic in mind. We found that this was not the case, and explored in some details the mechanisms that are responsible for the high latencies. Subsequently, we explored modifications of SCTP in order to overcome the problem.

We noticed that one highly effective way of improving SCTP performance requires the removal of delayed acknowledgments in combination with a reduction of the minimal retransmission timeout whenever a stream is thin. The benefit is achieved indirectly, as retransmissions for thin streams will not be triggered by duplicate acknowledgments unless other changes are made as well. The change allows the protocol to retransmit packets 1000 ms earlier than with the original minRTO, and the protocol does not pay the 200 ms penalty in the RTO computation any more that is due to delayed acknowledgments. We defined a simple test that indicates whether a stream is thin, such that the changes are only applied in this situation, thereby limiting the overhead of the improvement to the case that can most benefit from it.

We have also seen that the computation of the RTO value is highly unstable, and it remains unstable after out proposed changes. The reason is that the first arriving acknowledgment for a sample chunk that contributes to the RTO estimation is taken into account without any means of detection whether it is an acknowledgment of the original transmission. For our situation where retransmission latency matters, this should definitely be addressed; timestamps or a flag that indicates first transmission may be remedies. We do see benefits in SCTP's aggressive bundling and unsolicited retransmission of chunks when latencies are high and we consider an even more aggressive variation, but the instability of the RTO value should be addressed in conjunction with these changes.

Other authors' concern about spurious retransmissions is shared by us in general, but not for our specific scenario. Various proposed fixes would increase the average end-to-end message latency in thin streams. The partial ordering and partial reliability extensions to SCTP provide the means to overcome the latency problem, of course. But we would like to recall that SCTP was designed for signaling, and that latency increases are counterproductive in this scenario. Since a distinction of thin and thick streams is easily possible, we propose to consider the high-bandwidth and low-bandwidth applications of SCTP separately.

Thus, even though SCTP has many useful mechanisms for timely delivery of data, they fail in particular when an application sends only the most essential, latency critical data to avoid being blocked by congestion. For those low-rate signaling streams, the proposed enhancements will provide good means for reduced retransmission delays, while not affecting high bandwidth streams at all.

References

[1] V. Basto and V. Freitas. SCTP extensions for time sensitive traffic. In *Proceedings of the International Network Conference (INC)*, Samos Island, Greece, July 2005.

[2] R. Brennan and T. Curran. SCTP congestion control: Initial simulation studies. In *Proc. of the International Teletraffic Congress (ITC 17)*, Salvador de Bahia, Brazil, Sept. 2001.

[3] D. D. Clark and D. L. Tenenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 200–208. ACM Press, 1990.

[4] M. Claypool. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks*, 49(1):52–70, Sept. 2005.

[5] L. Coene and J. Pastor-Balbas. Telephony Signalling Transport over Stream Control Transmission Protocol (SCTP) Applicability Statement. RFC 4166 (Informational), Feb. 2006.

[6] K.-J. Grinnemo and A. Brunstrom. Performance of SCTP-controlled failovers in M3UA-based SIGTRAN networks. In *Proc. of the Advanced Simulation Technologies Conference (ASTC)*, Arlington, VA, USA, Apr. 2004.

[7] C. Griwodz and P. Halvorsen. The fun of using TCP for an MMORPG. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Newport, RI, USA, May 2006. ACM Press.

[8] S. Ladha, S. Baucke, R. Ludwig, and P. D. Amer. On making SCTP robust to spurious retransmissions. *ACM Computer Communication Review*, 34(2):123–135, 2004.

[9] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. RFC 3522 (Experimental), Apr. 2003.

[10] L. Ong, I. Rytina, M. Garcia, H. Schwarzbauer, L. Coene, H. Lin, I. Juhasz, M. Holdrege, and C. Sharp. Framework Architecture for Signaling Transport. RFC 2719 (Informational), Oct. 1999.

[11] J. Pedersen. Evaluation of SCTP retransmission delays. Master's thesis, Department of Informatics, University of Oslo, Oslo, Norway, May 2006.

[12] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758 (Proposed Standard), May 2004.

[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. RFC 2960 (Proposed Standard), Oct. 2000. Updated by RFC 3309.