Multimed Tools Appl (2009) 45:33–60
DOI 10.1007/s11042-009-0286-8

# Improving SCTP retransmission delays for time-dependent thin streams

**Andreas Petlund · Paul Beskow · Jon Pedersen · Espen Søgård Paaby · Carsten Griwodz · Pål Halvorsen**

Published online: 9 May 2009

**Abstract** A large number of network services rely on IP and reliable transport protocols. For applications that provide abundant data for transmission, loss is usually handled satisfactorily, even if the application is latency-sensitive (Wang et al. 2004). For data streams where small packets are sent intermittently, however, applications can occasionally experience extreme latencies (Griwodz and Halvorsen 2006). As it is not uncommon that such *thin-stream* applications are time-dependent, any unnecessarily induced delay can have severe consequences for the service provided. Massively Multiplayer Online Games (MMOGs) are a defining example of *thin streams*. Many MMOGs (like World of Warcraft and Age of Conan) use TCP for the benefits of reliability, in-order delivery and NAT/firewall traversal. It has been shown that TCP has several shortcomings with respect to the latency requirements of thin streams because of the way it handles retransmissions (Griwodz and Halvorsen 2006). As such, an alternative to TCP may be SCTP (Stewart et al. 2000), which was originally developed to meet the requirements of signaling transport. In this paper,

A. Petlund (✉) · P. Beskow · J. Pedersen · E. S. Paaby · C. Griwodz · P. Halvorsen
Simula Research Laboratory, Martin Linges vei 17, 1364 Fornebu, Norway
e-mail: apetlund@ifi.uio.no

P. Beskow
e-mail: paulbb@ifi.uio.no

J. Pedersen
e-mail: jonped@ifi.uio.no

E. S. Paaby
e-mail: espensp@ifi.uio.no

C. Griwodz
e-mail: griff@ifi.uio.no

P. Halvorsen
e-mail: paalh@ifi.uio.no

A. Petlund · P. Beskow · J. Pedersen · E. S. Paaby · C. Griwodz · P. Halvorsen
Department of Informatics, University of Oslo, Gaustadallen 23, 0373 Oslo, Norway

we evaluate the Linux-kernel SCTP implementation in the context of thin streams. To address the identified latency challenges, we propose sender-side only enhancements that reduce the application-layer latency in a manner that is compatible with unmodified receivers. These enhancements can be switched on by applications and are used only when the system identifies the stream as thin. To evaluate the latency performance, we have performed several tests over various real networks and over an emulated network, varying parameters like RTT, packet loss and amount of competing cross traffic. When comparing our modifications with SCTP on Linux and FreeBSD and TCP New Reno, our results show great latency improvements and indicate the need for a separate handling of thin and thick streams.

**Keywords** Latency compensation · Networked games · Transport protocols · Thin streams

## 1 Introduction

Supporting low rate, interactive streams in the Internet introduces huge challenges due to packet loss. Such time-dependent, low latency applications have in recent years greatly increased in number, and some of the most prominent examples are Massively Multiplayer Online Games (MMOGs). Serving more than 16,000,000 subscribers [30], MMOGs produce highly interactive network traffic that are sensitive to latency. Moreover, several of the largest MMOGs (e.g., World of Warcraft) use the Transmission Control Protocol (TCP) as transport layer protocol. Other examples of interactive, latency-sensitive applications include stock trading, thin clients, control systems, remote probe operations and audio conferencing.

Significant characteristics of this kind of applications are their stringent requirement for maintaining a *consistently low latency* in order to provide a good perceived interactive service quality. The rigidity of the former requirement will vary from application to application, but if the maximum tolerable value is consistently exceeded, the quality of experience (QoE) will degrade accordingly. A wide range of applications require latencies below 500 ms, and even the most delay-tolerant of these time-dependent applications suffer heavily when the delay approaches one second [7, 14].

Historically, distributed interactive applications have been developed for use either with transport protocols that could provide per-stream Quality of Service (QoS) guarantees, or with protocols that allowed the sending application to determine the transmission timing, such as the User Datagram Protocol (UDP). However, the QoS protocols have not become widely available, and the use of UDP has been heavily criticized for its lack of congestion control mechanisms. Consequently, many time-dependent and interactive distributed applications today are implemented using reliable transport protocols like TCP, and many applications using UDP, despite criticism, use a reliable transport protocol as a fall back solution, when for example a firewall is blocking UDP. However, the inherent congestion control and retransmission mechanisms of reliable transport protocols introduce severe latency challenges for a large class of time-dependent distributed interactive applications with different traffic characteristics than the targeted, highly optimized bulk transfer scenario. In particular, the data streams are *thin* characterized by small packets and high packet interarrival times. The existing packet loss recovery mechanisms therefore fail to support the required timeliness of packet deliver causing extreme latencies when loss occurs in traffic that exhibits thin stream patterns [12].

An alternative to TCP is the Stream Control Transmission Protocol (SCTP) [26], which is developed to meet the requirements for signaling transport identified by RFC2719 [17], and is currently on the standards track of the IETF. The development of applications which support SCTP has up until now focused on transport of signaling data, as may be expected. The SCTP community is, however, productive and a range of new applications are being tested and developed [22]. Given the requirements for SCTP development, the protocol is a promising alternative for congestion-controlled communication for time-dependent and interactive distributed applications (like MMOGs). This is because the signaling traffic it is designed for shares the thin-stream characteristics. In this study, we have therefore compared SCTP to TCP for thin stream traffic using the SCTP implementation from the Linux kernel (*lksctp* [28]) and FreeBSD (the *Kame* project [27]). The results of our examination show that SCTP does not improve the timeliness of the reliable transport of thin stream data traffic when compared to TCP. This is surprising, considering SCTP's design goals. Furthermore, we have identified the origin of the problem and have devised a strategy to improve support for thin streams in congestion-controlled protocols. We are able to reduce latency for thin streams by modifying timers and fast retransmit policies. To ensure that other traffic patterns are not adversely affected by our modifications, they are only activated in cases where the system detects that a data stream has thin-stream characteristics. The approach is justified by our analysis of thin stream traffic patterns, which show that signaling traffic and other thin streams hardly ever expand the congestion window, and, as such, have very limited impact on the network congestion. The experimental results from different test setups show that the proposed enhancements greatly reduce the retransmission delays of SCTP. This leads to a lower application-layer delay when delivering data, and increases the users' QoE.

## 2 Related work

Latency problems upon recovery of packets for protocols with TCP-like behaviour has been subject to research for many years. Fall et. al. summarise the benefits of different improvements to the fast retransmit mechanism in the paper "Simulation-based comparisons of tahoe, reno, and sack tcp" [10]. TCP Tahoe implemented fast retansmit, thus being able to recover before a timeout was triggered. An improvement was made with TCP Reno, which entered a "Fast Recovery" state when doing a fast retransmit thus avoiding going into slow start. With TCP New Reno, the algorithm retransmits a segment with every received ACK in the "Fast Recovery" state, further improving the algorithm.

The problem of late retransmissions has been addressed before. For example, the optional Early Fast Retransmit (EFR) mechanism[1] exists in SCTP for FreeBSD

[1] This mechanism can be enabled in FreeBSD by using the *net.inet.sctp.early_fast_retran* syscontrol. We have, however, not been able to find any published papers which yields further details of the mechanism's implementation in FreeBSD.

and has been used for tests in this paper. That mechanism is active whenever the congestion window is larger than the number of unacknowledged packets and when there are packets to send. It starts a timer that closely follows Round trip time (RTT) + estimated RTT variance (RTTVAR) for every outgoing packet, and when the timer goes off and the stream is still not using the entire congestion window, it retransmits all packets that could have been acknowledged in the meantime. An EFR timeout does not trigger slow start like a normal timeout, but it reduces the congestion window by one.

In an IETF draft, Allman et al.[2] suggest that measures should be taken to recover lost segments when there are too few unacknowledged packets to trigger Fast Retransmit. They propose Early Retransmit (ER), which should reduce waiting times in four situations: the congestion window is still initially small, it is small because of heavy loss, flow control limits the send window size, or the application has no data to send. The draft proposes to act as follows whenever the number of outstanding segments is smaller than 4: if new data is available, it follows Limited Transmit [1], if there isn't any, it reduces the number of duplicate packets necessary to trigger fast retransmit to as low as 1 depending on the number of unacknowledged segments. It differs from our approach in two ways. The first is the motivation. The second is that Allman et al. try to prevent retransmission timeouts by retransmitting more aggressively, thus keeping the congestion window open even though congestion may be the limiting factor. If their limiting conditions change, they still have higher sending rates available. Our applications are not inhibited by congestion control. We have no motivation to prevent retransmission timeouts in order to keep the congestion window open, but we retransmit early only to reduce application-layer latencies. We are therefore combining the approach with a reduced minimum retransmission timeout ($\mathrm{RTO}_{min}$) to handle the worst-case situation instead of preventing the retransmission timer from firing. ER is less frequently active than EFR, but it is more aggressive when the number of unacknowledged packets it small.

Ekström and Ludwig [9] point out that the retransmission timeout algorithm defined in RFC2988 [19] and used in both TCP and SCTP responds sluggishly to sudden fluctuations in the RTT. This leads to extreme estimated RTO values in some cases. They also point out that the RTTVAR computation does not distinguish between positive and negative variations, and therefore increases the RTO in the case of both RTT increases and decreases. Their proposed algorithm alleviates the consequences of RTT fluctuations and is, as such, a good addition to the main protocol for a range of special cases. Their findings are consistent with our observations made in [20] of high RTO values that are worsened by the SCTP delayed acknowledgement algorithm. While their solution leads to a more stable RTO, it is on average higher than that proposed in RFC2988, which is not desirable for our scenario. Ekström and Ludwig also mention that they should consider a less conservative exponential backoff algorithm, which is one of the mechanisms that we investigated.

Brennan and Curran [5] performed a simulation study for greedy traffic and identified weaknesses in the fast retransmit procedure. However, their modifications would increase delays for thin streams. Problems with carrying time-sensitive data over SCTP were presented by Basto and Freitas [4]. The traffic that they considered was

[2] IETF Draft draft-allman-tcp-early-rexmt-05: Mark Allman, Konstantin Avrachenkov, Urtzi Ayesta, Josh Blanton, "Early Retransmit for TCP and SCTP", June 2007, expired Dec. 2007.

loss-tolerant, and they proposed the use of SCTP's partial reliability extensions [25]. Ladha et al. [16] examined several methods of detecting spurious retransmissions and proposed modifications that would increase throughput but also increase the latency of individual lost packets. Grinnemo and Brunstrom [11] discuss the problem of RTO$_{min}$, and propose a reduction to fulfil the requirements of RFC4166 [8], an RFC on the applicability of SCTP for telephony. The RFC itself discusses problems and solution approaches, and it proposes to choose the path within a multi-homed association that experiences the shortest delay, an approach that may be used as a supplement to other techniques for thin-stream scenarios. The RFC considers both reduction of the RTO$_{min}$ and removal of exponential back-off, but warns that both alternatives have drawbacks. Removing delayed SACK is mentioned without stating any side-effects. This would also be beneficial in our scenario. However, it is a receiver-side change, while we aim exclusively at sender-side changes. Of the discussed options, we choose the removal of the exponential back-off, but instead of doing it arbitrarily, we limit it to situations where fast retransmit is impossible due to lack of unacknowledged packets (i.e. too few packets in flight).

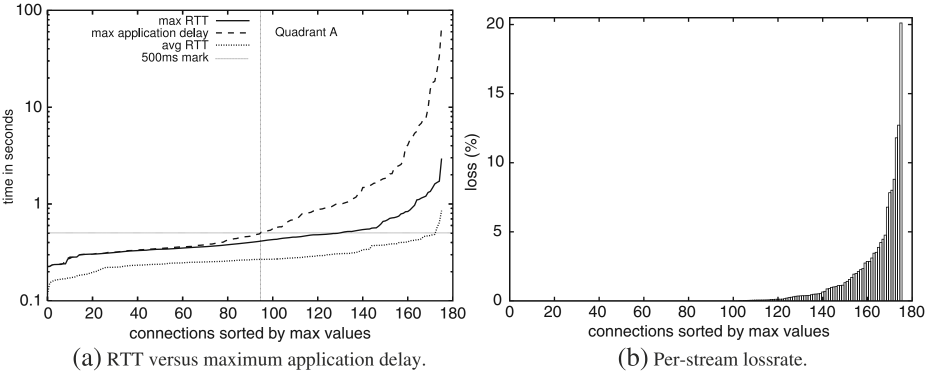
S. Shirmohammadi and N.D. Georganas present a framework with focus on collaboration in virtual environments [24]. They identify several of the *thin-stream* properties that we describe in this paper, and suggest ways of optimising the streams used for this collaboration. This paper points out the benefits of SCTP for virtual environments based on the requirement for timely and reliable (partially reliable) communication. They do not, however, explore ways of reducing latency in SCTP for such applications.

The removal of the exponential back-off can of course result in spurious retransmissions when the RTT changes. The proposed method of TCP Santa Cruz [18] uses TCP timestamps and TCP options to determine the copy of a segment that an acknowledgement belongs to and can therefore provide a better RTT estimate. Since the RTT estimate can distinguish multiple packet losses and sudden increases in actual RTT, TCP Santa Cruz can avoid exponential back-off. The ability of Santa Cruz to consider every ACK in RTT estimation has minor effects in our scenario where hardly any packets are generated. The ability to discover the copy of a packet that an ACK refers to would still be desirable but would require receiver-side changes that we avoid.

The earlier work that has been done in the field of reducing latency upon retransmissions all focus on special cases of *thick* streams where measures can be taken to improve throughput. Our work identifies thin streams as time-critical and latency sensitive. We therefore apply several modifications upon detection of the thin stream, and can thus improve latency for the stream in a manner not yet explored in literature.

## 3 Thin streams and reliable congestion-controlled protocols

Analysis reveals that applications that transmit small amounts of data over the network upon user interaction have some common traits in their network traffic patterns. We call such data streams *thin streams*. Because of the element of interactivity in such applications, they are sensitive to high delays. It is shown that the traffic generated by such applications combined with reliable protocols can experience



(a) RTT versus maximum application delay.

(b) Per-stream lossrate.

**Fig. 1** Per stream latency and loss rate from Anarchy Online server side dump (**a**, **b**)

this kind of unwanted latencies [12]. In this section, we will describe examples of applications that produce such traffic patterns and show how this causes the extreme latencies for reliable protocols.[3] We then, in Section 4, introduce SCTP, which was designed for signaling traffic and which has comparable traffic characteristics to thin streams.

### 3.1 Reliability and congestion control

The design of reliable transport protocols has historically focused on maximising throughput without violating fairness, i.e., mainly aiming for traffic patterns from high-rate download-like applications like file transfers. Because of this, the group of applications that do not use what constitutes their fair share of bandwidth, such as thin stream applications, have been marginalised. A worst-case example of the outcome of this focus can be seen in Fig. 1. The graph shows basic loss- and delay statistics in a one-hour trace from a game-server for *Funcom's* massively multi-player online game (MMOG) *Anarchy Online*. In Fig. 1a, we have drawn a line at 500 ms to show how many streams experienced latencies that would degrade the players' QoE severely [7]. The graph shows that nearly half of the measured streams during this hour of game-play had such latency events. When compared to Fig. 1b, we can see that even connections with a relatively low loss rate suffer such latency events.

The cause for these extreme delays can be found in the combination of certain traffic patters (thin streams) and the reliable transport protocol's mechanisms for retransmission and congestion control. The underlying cause is found in the commonly used method for fast recovery of lost segments called *fast retransmit*. When a segment is lost, the receiver responds by acknowledging the last successfully delivered data segment until it receives the lost one. The sender will, upon receiving the third such duplicate acknowledgement (dupACK), retransmit the segment.

For streams that expand the congestion window and send a lot of data segments for each RTT, this will lead to a much quicker recovery than waiting for the timeout

[3] We have earlier shown that using middlewares adding reliability on top of UDP does not give better results [13].

to trigger. For thin streams, which very often produce less than one packet per RTT, the lack of data to send makes the triggering of fast retransmissions impossible (illustrated in Fig. 2), leading to a situation where nearly all retransmissions are triggered by timeout, i.e., the time before receiving the third dupACK exceeds the RTO.

The additive increase/multiplicative-decrease (AIMD) [2] algorithm is a feedback control algorithm commonly employed in congestion avoidance for TCP, SCTP and other reliable transport protocols. Basically, AIMD represents a linear growth of the congestion window, combined with an exponential reduction when congestion is detected. AIMD resumes normal operation when the flow of feedback (ACKs) from the receiver resumes. Until that happens, the RTO is doubled for each new retransmission (by timeout) of the lost segment. For thin streams, which are non-aggressive and non-greedy, this backoff adds penalties without warrant. Actual real-life consequences are shown in Fig. 1a, where the maximum observed application delay value was 67 s, caused by six consecutive losses of the same segment, handled by timeout retransmission with exponentially increasing RTO.

The combination of not being able to trigger fast retransmissions and suffering from exponential backoffs makes thin-stream applications prone to suffering from high latencies. In the next section, we will show how thin streams tend to be generated by interactive applications that, as such, are sensitive to high delays.

### 3.2 Thin-stream applications

Applications that produce network patterns that have *thin-stream properties*, namely small packets and large packet interarrival times, tend to be interactive and time-dependent. The fact that human interaction is what generates the network traffic makes transmissions sporadic and irregular. Messages in such situations contain often only small position updates or control messages in gaming scenarios and a collection of a few audio samples in voice-over-IP (VoIP) scenarios. Depending on the function, sensor networks also tend to be triggered by natural activity or movement, and will thus display similar characteristics. Table 1 shows a selection of applications

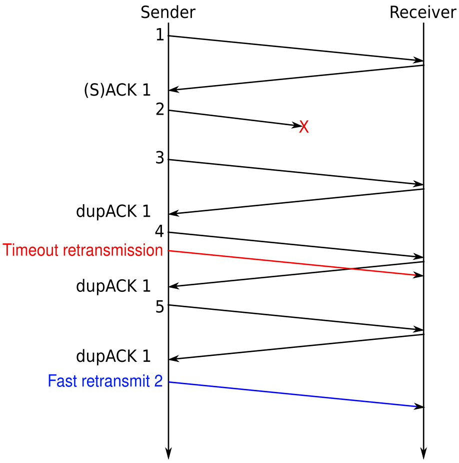


**Fig. 2** Fast retransmit with thin streams

**Table 1** Examples of thin (**thick**) stream packet statistics based on analysis of packet traces

| Application | Payload size (bytes) | | | Packet interarrival time (ms) | | | | | | avg bandwidth requirement | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Percentiles | | | |
| | avg | min | max | avg | med | min | max | 1% | 99% | (pps) | (bps) |
| Casa (sensor network) | 175 | 93 | 572 | 7287 | 307 | 305 | 29898 | 305 | 29898 | 0.137 | 269 |
| Remote desktop(RDP) | 111 | 8 | 1417 | 318 | 159 | 1 | 12254 | 2 | 3892 | 3.145 | 4497 |
| Skype (2 users) | 236 | 14 | 1267 | 34 | 40 | < 1 | 1671 | 4 | 80 | 29.412 | 69K |
| SSH text session | 48 | 16 | 752 | 323 | 159 | < 1 | 76610 | 32 | 3616 | 3.096 | 2825 |
| Anarchy Online | 98 | 8 | 1333 | 632 | 449 | 7 | 17032 | 83 | 4195 | 1.582 | 2168 |
| World of Warcraft | 26 | 6 | 1228 | 314 | 133 | < 1 | 14855 | < 1 | 3785 | 3.185 | 2046 |
| YouTube stream | 1446 | 112 | 1448 | 9 | < 1 | < 1 | 1335 | < 1 | 127 | 111.111 | 1278K |
| HTTP download | 1447 | 64 | 1448 | < 1 | < 1 | < 1 | 186 | < 1 | 8 | > 1000 | 14M |
| FTP download | 1447 | 40 | 1448 | < 1 | < 1 | < 1 | 339 | < 1 | < 1 | > 1000 | 82M |

whose network traffic has been analysed. The identifying element for the thin stream applications, in contrast to thick streams, is that they all have small packet sizes and high interarrival time between the packets, *and the stream keeps those properties throughout its lifetime*.

Windows Remote Desktop using the remote desktop protocol (RDP) is an application used by thin client solutions or for remote control of computers. Analysis of packet traces indicates that this traffic also clearly show thin-stream properties. If second-long delays occur due to retransmissions, this will result in visual delay for the user while performing actions on the remote computer. Another way of working on a remote computer is the common protocol of secure shell (SSH). This is used to create an encrypted connection to a remote computer and control it, either using text console, or by forwarding graphical content. The analysed dump presented in Table 1 is from a session where a text document is edited on the remote computer. We can observe that this stream also displays the thin-stream properties. The interarrival times are very similar to the RDP session, while the packet sizes are even smaller.

As an example of sensor networks we have analysed traffic from the real-time system in the Casa project, which performs research on weather forecasting and warning systems. Here, low-cost networks of Doppler radars are used that operate at short range with the goal of detecting a tornado within 60 s [31]. Control data between the server and a radar is typically small and sent in bursts. A packet trace (see statistics in Table 1) shows that the average packet size from the server is 241 bytes, and a burst of four packets with an interarrival time of about 305 ms is sent every 30 s (the heartbeat interval of the system). To be able to detect a tornado in time, speedy delivery of the control data is essential.

Audio conferencing with real-time delivery of voice data across the network is an example of a class of applications that uses thin data streams and has a strict timeliness requirement due to its interactive nature. Nowadays, audio chat is typically included in virtual environments, and IP telephony is increasingly common. For coding and compression, many VoIP telephone systems use the G.7xx audio compression formats recommended by ITU-T where, for example, G.711 and G.729 have a bandwidth requirement of 64 and 8 Kbps, respectively. The packet size is determined by the packet transmission cycle (typically in the area of a few tens of ms, resulting in packet sizes of around 80 to 320 bytes for G.711). Skype [23] is a wellknown conferencing service, with several million registered users, that communicates on the (best effort) Internet. Table 1 shows statistics analysing a Skype conferencing trace. The small average packet size combined with an interarrival time between packets that averages to 34 ms qualifies it as a thin-stream. To enable satisfactory interaction in audio conferencing applications, ITU-T defines guidelines for the one-way transmission time [14]. These guidelines indicate that users begin to get dissatisfied when the delay exceeds 150–200 ms and that the maximum delay should not exceed 400 ms. This will be hard to achieve over reliable protocols, given the thin-stream properties of the stream.

Finally, World of Warcraft and Anarchy Online are two examples of MMOG games, and we can clearly see from Table 1 that the traffic patterns show thin-stream properties. With respect to user satisfaction, games require tight timeliness, with latency thresholds at approximately 100 ms for first-person shooter (FPS) games, 500 ms for role-playing games (RPG) and 1000 ms for real-time strategy games [7]. Analysis of other game genres (FPS and RPG) shows that they also show similar networking patterns with high interarrival times and small packets, reflecting the human interaction present in the games.

Compared to the *thick* streams shown in Table 1, e.g., streaming a video from YouTube, downloading a document over HTTP from a server in the UK or downloading a CD-image from *uninett.no*, the examples given above are a small selection of applications where the data stream is *thin*. Other examples include control systems, virtual environments (such as virtual shopping malls and museums), augmented reality systems and stock exchange systems. All of these send small packets and have relatively low packet rates. Yet, they are still highly interactive and thus depend on the timely delivery of data.

In summary, the connections in the described scenarios are so thin that 1) they do not trigger fast retransmissions often but retransmit packets mainly due to timeouts and 2) a TCP-style congestion control does not apply, i.e., each stream have too few packets and cannot back off. SCTP is designed for signaling traffic, and as such, should be able to support thin streams with regard to latency. In the next section, we present the basics of SCTP and how it relates to thin streams.

## 4 SCTP

In [12], we showed that TCP-variations in Linux provide poor support for thin-streams. SCTP [26], however, was designed to support signaling traffic in Public Switched Telephone Networks (PSTN), and should therefore be able to provide satisfactory support for such traffic patterns. There is also a range of other services that SCTP aims to provide, which makes it a viable candidate for many different kinds of applications.

Reliability is provided through acknowledged data delivery. The protocol also checks for bit errors and ensures that duplicates are removed. It supports sequenced delivery within multiple streams through one SCTP connection, which is often called an association. SCTP offers the option of bundling several messages in one packet and also supports multi-homing for enhanced fault tolerance. There are also proposed extensions, such as partial reliability [25], that can be used for time-dependent applications. This allows for optional reliability that can enable UDP-like behaviour when requested. A variant of this is the timed reliability option that can invalidate a message in the reader buffer if a given timer has expired.

The messages delivered by SCTP are organised in data units called chunks. There are chunk types for initiation and tear-down of connections, as well as other protocol intrinsics. A salient difference from TCP is that SCTP is message-, not byte-oriented. Instead of retransmitting the previous packet(s) as TCP does, SCTP keeps track of the chunks that have timed out or been reported as lost, and retransmits unacknowledged chunks. This makes the protocol more flexible with regard to packet composition and bundling. It does not mean that retransmitted packets are necessarily identical to the first transmitted packet; just that the chunk(s) scheduled for retransmission is included.

The $\text{RTO}_{min}$ value is set high in SCTP (1000 ms) to avoid spurious retransmissions due to early timeouts. In a thin-stream setting, most retransmissions are caused by timeouts since these applications does not send more than 4 packets every second, and are thus unable to trigger a fast retransmission. This becomes a major factor in increasing latency for retransmitted chunks.

The SCTP specification [26] states that guidelines for delayed ACKs in TCP congestion control [2] should be followed. These guidelines state that an ACK should be delayed until two data packets have arrived, or a maximum of 500 ms have passed. The SCTP specification follows this recommendation and states that a SACK should be generated within 200 ms, and must be generated within 500 ms after reception of a data chunk. This delay is usually implemented with a default value of 200 ms. It is usually possible to customise this value inside the limitations specified by the RFC. However, the delayed SACK algorithm does influence the RTT estimation at the sender, which in turn affects the RTO calculation. For the thin-stream scenario, this means that the timeouts (being the main cause for retransmissions) will have unnecessarily high values that increase the transmission latency. Like TCP, SCTP has a calculated RTO that depends on the measured RTT and RTTVAR. The algorithm for this calculation is vulnerable to changes in the RTTVAR in the sense that both a positive and negative RTTVAR will raise the RTO. The combination of the RTO calculation and delayed SACKs produce too large an RTO for much of the connection's life.

## 5 Enhancements

In [20], we show 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 use reliable transport to fulfil the requirements for signaling traffic that were defined by RFC2719 [17] and avoid error handling by the higher layers. However, advantages such as the maintenance of message boundaries and proactive retransmission by bundling chunks still make SCTP attractive for distributed interactive applications. We would therefore like to introduce variations into SCTP that improve its performance regarding thin stream scenarios.

There are several ideas for enhancing SCTP latency for thin streams that should be evaluated in addition to what is done in the comparison with TCP New Reno [20]. One is to use fewer SACKs to trigger a fast retransmission because the number of

SACKs is so small that they can rarely trigger a fast retransmission. We also consider the suggestions in RFC4166 [8], namely reduction of the RTO$_{min}$ and removal of exponential back-off. The RFC warns that these variations have negative side-effects by increasing the danger of spurious retransmissions and reducing the size of the congestion window. The proposed mechanisms, however, is applied only when the stream is thin and do not aggressively probe for bandwidth.

Next, we address the identification of a stream as thin, and then we describe our enhancements to the SCTP implementation in the Linux kernel (lksctp).

### 5.1 Thin stream detection

For the purpose of our implementation, we define a stream as *thin* when there are so few packets in flight (also often termed in transit) that they cannot trigger a fast retransmission, i.e., there are no unacknowledged packets meaning that the packet rate is too low. When this happens, the only way that the stream can recover from packet loss is to wait for the retransmission timeout (unless packet duplication occurs). We use the very conservative algorithm in Fig. 3 to decide when the stream is thin and thus when to apply the enhancements. Here, *in_flight* is the number of packets in flight (flight size), *pttfr* is the number of packets required to trigger a fast retransmission (3 for Linux 2.6.16) and *lossrate* is the fraction of packets that are detected as lost.

As the figure shows, it relies more or less only on counting transmitted but unacknowledged packets and uses neither packet send times nor additional SACK information to draw further conclusions. However, the dynamic detection algorithm also allows us to (slowly) change the number of packets in flight needed to trigger the thin-stream mechanisms according to the loss rate which could be useful in high (extreme) loss scenarios as described for VoIP data in [21].

SCTP identifies chunks by transmission sequence number (TSN) and bundles them when retransmitting. The number of packets in flight is therefore not available, as it is in TCP. Thus, we added a list that holds the highest TSN for every packet in flight, as well as a packet counter. From the SACK, which acknowledges the highest cumulative TSN, the sender can now know whether or not a packet has left the network. Moreover, lksctp is not able to estimate packet loss, and therefore we implemented an algorithm for estimating packet loss that makes use of the packet-in-flight list to determine whether a packet is lost or not. Then, by looking at the SACKs returned by the receiver, we mark a packet as lost if the highest TSN in a packet corresponds to a gap in the SACK, and following the fast retransmission scheme, the packet is determined to be lost if it is indicated as lost by a SACK on three different occasions.

**Fig. 3** Determining which mechanisms to use with thin stream detection

```
$if\left(in\_flight \leq \frac{pttfr+1}{1-lossrate}\right)\{$   /* thin */
    $apply\ modifications$
$\}\ else\ \{$                                                /* thick */
    $use\ normal\ sctp$
$\}$
```

### 5.2 Modified minimum RTO

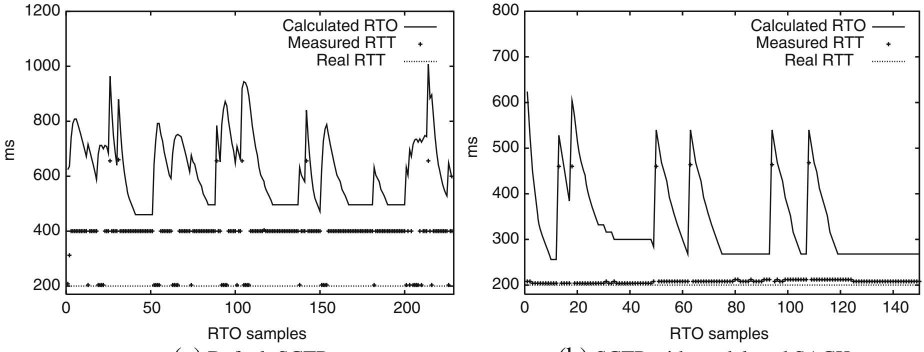
To avoid timeouts occurring too early, which leads to spurious retransmissions and a reduced congestion window, SCTP has a rather high $\text{RTO}_{min}$ value (1000 ms). Nevertheless, in our thin-stream scenario, we can see that almost all retransmissions are due to timeouts. Therefore, we experimented with an $\text{RTO}_{min}$ of 200 ms (equal to the corresponding default value for TCP in Linux).

As a consequence of reducing $\text{RTO}_{min}$, the relative effect of delayed SACKs on the RTO calculation that was described in Section 4 grows, as shown in Fig. 4a. When the receiver-side SACK delay is eliminated, the calculated RTO is greatly reduced due to a lower measured RTT, as shown in Fig. 4b. Thus, although receiver-side enhancements are more difficult to apply in some scenarios (since client machines must be updated), we performed measurements to see the effect on retransmission delay.

SCTP also restarts the retransmission timer with the current RTO when an incoming SACK acknowledges some, but not all, outstanding chunks. The benefit of this approach lies in increasing the probability of achieving a fast retransmit in favor of a slow start, which would be forced implicitly by a timeout retransmission. It has a negative impact in our scenario, where we do not focus on throughput but need to deliver chunks to the application as quickly as possible. We avoid the latency penalty associated with the timer reset by adjusting the expiration time before the timer is restarted. The new expiration time is determined by subtracting the time since the old timer was started from the original timer value. Thus, no more than one RTO will elapse before the oldest unacknowledged chunk is retransmitted by a timeout.

### 5.3 Removal of exponential back-off

If there are too few SACKs to trigger a fast retransmission or no new packets are sent to let the receiver discover loss, retransmissions could be triggered by subsequent timeouts without any intervening fast retransmissions. At this point, an exponential back-off of the retransmission timer is performed, which leads to the retransmission delay increasing exponentially when there are occurrences of multiple loss. However,



(a) Default SCTP.

(b) SCTP without delayed SACKs.

**Fig. 4** Difference between calculated and measured RTO values for thin streams (**a**, **b**)

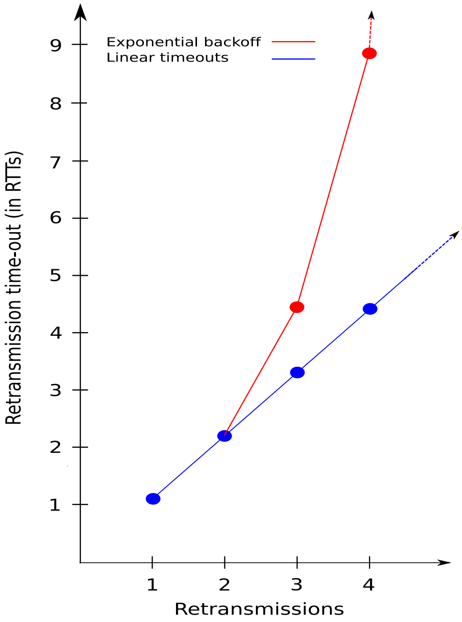


**Fig. 5** Difference between linear timeouts and exponential backoff

because the stream is thin, it can never be aggressive, either. Hence, it is not necessary to use exponential back-off to prevent aggressive probing for bandwidth. We therefore use linear timeouts when a thin stream is detected as shown in Fig. 5.

### 5.4 Modified fast retransmit

Despite the high RTO$_{min}$ value, fast retransmissions hardly ever appear in our thin-stream scenario. The lack of fast retransmissions is because the packet interarrival time is too large (see Table 1) for the three SACKs that are required before the retransmission timer expires to be received. In order to deal with this problem, we allow a fast retransmission to be triggered by the first indication that a chunk is lost, as illustrated in Fig. 6. This is because retransmissions that are triggered by causes other than timeouts are usually preferable with respect to latency. The overhead of this modification will not be serious, because the probability that packets will be reordered in thin streams is low and the amount of data sent is small. The modification implemented may lead to more transmissions in the low-probability case that packets are reordered, but the gain in latency will justify the need to drop occasional spurious retransmissions.

## 6 Experiments and results

A large number of tests have been run in order to determine the effects of the proposed enhancements to SCTP in Linux 2.6.16, especially to the worst case delays that ruin the user experience in our interactive application scenario.

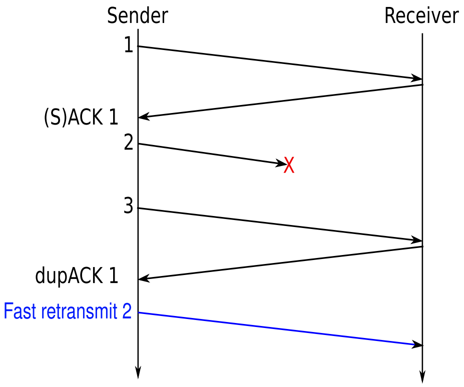


**Fig. 6** Fast retransmission upon first indication of loss

First, a number of lab tests with the setup shown in Fig. 7, using the Linux' traffic control (*tc*) system network emulator (*netem*) and the queueing discipline (*qdisc*), shows the properties and effect of each modification. The emulated network introduced RTTs between 0 and 400 ms (0, 50, 100, 200, 250 and 400 ms). Uniformly distributed loss (1% and 5%) was introduced by the emulator, and bursty, uneven loss patterns (with an average of 5%) by competing web traffic. The packet interarrival time was varied from 50 to 250 ms (50, 100, 150, 200 and 250 ms).

The loss rates were chosen to emulate the losses that can be experienced in real-life scenarios (like illustrated in Fig. 1b). The TCP version used as a reference in the tests was New Reno, which had earlier achieved the best results for this scenario [12]. Each test ran for 2 h and was repeated several times. For different SCTP scenarios, we used exported kernel PROC variables to turn our modifications on and off. Each individual enhancement was evaluated during the lab tests.

Second, we evaluated the mechanisms under more realistic conditions using replayed Anarchy Online game traffic between Oslo and Amherst, Massachusetts. The Anarchy Online trace includes approximately 170 connections (players) from one of the many game regions with statistics as presented in Section 3.2 and Table 1. As the statistics show, all stream are thin, and representative examples of our target scenario. We measured a minimum RTT of 121 ms and loss rates below 0.1%. We performed experiments with all modifications and compared modified lksctp in Linux 2.6.16 with unmodified lksctp on Linux 2.6.22.14 (as well as 2.6.16, which performs

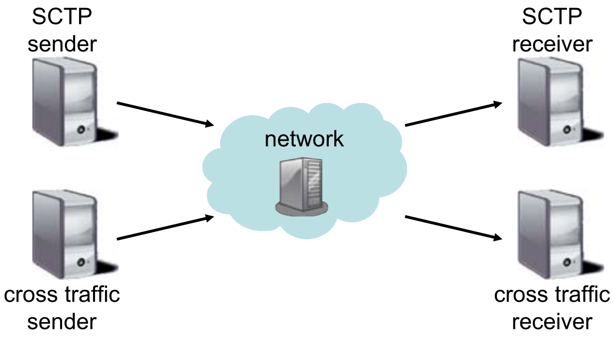


**Fig. 7** Lab test setup using an emulated network

very similarly for the observed situation and is not shown separately) and SCTP in FreeBSD 6.2.

### 6.1 Lab experiments: artificial loss

In this test, we sent SCTP traffic over an emulated network, introducing artificial loss and delays. As a representative example, we present the results of comparing lksctp with our modifications in a thin stream scenario. The different tests all show the same trends where some of the results are summarized in Fig. 8a. When exponential backoff was disabled, we observed a reduction in maximum latencies, especially for 2nd and 3rd retransmission compared to lksctp. The 99 percentile and average latencies were only marginally reduced. With an RTO$_{min}$ of 200 ms, we saw improved average and 99-percentile latencies as well. The results can be explained by the fact that most retransmissions in thin stream scenarios are caused by timeouts. By reducing the RTO, the latency for all these has been lowered. In the test modifying the fast retransmit to be triggered by only one duplicate SACK, we saw that the average and 99-percentile latencies were drastically improved compared to lksctp. Maximum values were still high, caused by exponential backoff. The combined test using all the three modifications showed large improvements both for maximum, 99-percentile and average latencies. Generally, we saw that improvements from the modifications got more pronounced on the 2nd and 3rd retransmission.

We also wanted to compare the results with the de facto choice for reliable transport, namely TCP. For the 1st and 2nd retransmission, TCP performs better than the original lksctp. On the 3rd retransmission, lksctp has a better average value, although the 99 percentiles and maximum latency are still better with TCP. However, our modified lksctp performs better than TCP except for maximum values of the 1st. and 2nd. retransmission. In the third retransmission, however, TCP displays much higher maximum latencies than the modified lksctp. The reason why the difference between TCP and modified SCTP is not larger is that the delayed SACKs introduced

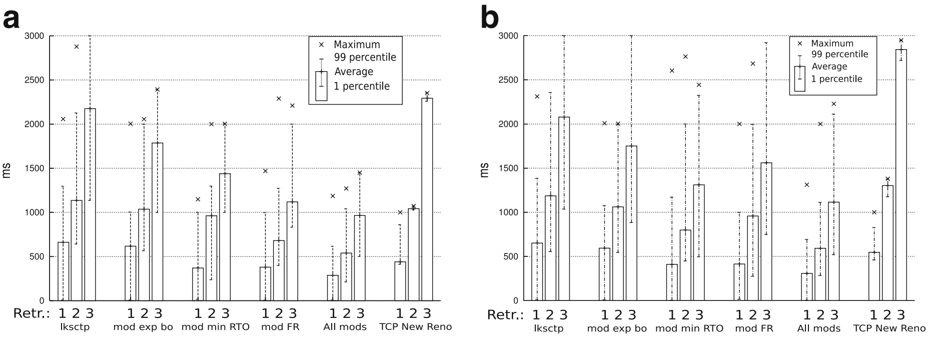


**Fig. 8** Effects of proposed enhancements over an emulated network (RTT=100) sending 4 packets of 100 bytes per second. For the group denoted *lksctp*, standard lksctp with default settings (RTO$_{min}$ =1000 ms) was used. The group denoted *mod exp bo* represents the removed exponential back-off, *mod min rto* is the lowered retransmission timeout modification, *mod FR* is the fast retransmission modification and *All mods* denotes the result of all modifications combined. Delayed ACKs was turned off for all experiments (**a**, **b**)

extra application layer delays for SCTP. TCP will, however, perform worse for each retransmission due to exponential backoff.

### 6.2 Lab experiments: congestion loss

Uniform loss will emulate some network scenarios, but there are many situations where the loss patterns are bursty. The burstiness can increase latency because there is a greater probability that several retransmissions of the same chunk will be lost. Therefore, to compete for resources with a more realistic load, we sent web traffic over the same emulated network to introduce congestion and thereby loss. Since the induced loss was generated by the emulated HTTP traffic, the total loss varied slightly from test to test.

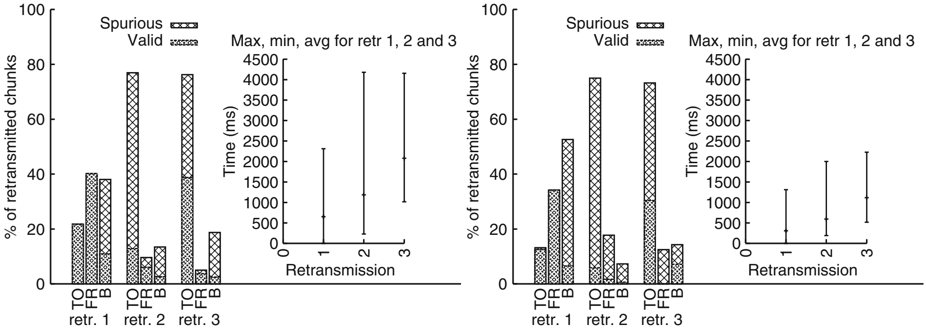
With respect to emulating real Internet HTTP-traffic, a lot of work has been done to define parameters such as file-transfer size and mean interarrival time, as well as the number of concurrent clients [3, 6]. Most studies agree on a heavy-tail distribution to describe the file size [6]. The studies show that there are many small files, and few large ones, but the greater sizes can become almost arbitrarily large. Thus, we used a Pareto distribution with a minimum size of 1000 bytes[4] giving us a mean transfer size of approximately 9200 bytes per connection. Furthermore, we had 81 concurrent web-client programs running, where the number was determined by the number of different delays that one *netem* instance can assign to connections. Each of the client programs started new streams continuously. An exponential distribution with a mean of 655 ms was used to decide the request interarrival time per client process. On the bottleneck, the bandwidth was limited to 10 Mbps with a queue length of 100 packets. Using these settings, we experienced an average packet loss of about 5% in the emulated network.

Using the same representative example as in Section 6.1, the performance of each individual modification in a congested network is shown in Fig. 8b. The results are very similar to the ones presented for artificial loss in Fig. 8a. Each individual modification improves upon lksctp where the relative magnitude of the improvements tends to increase with the number of retransmissions.

Furthermore, Fig. 9 shows the distribution of retransmissions types and the respective experienced latencies. For lksctp, we can see that fast retransmission is the dominant cause of the first retransmission. Bundled chunks are the second most common, but the majority of these are spurious. Timeouts represent a little more than 20%. For the second retransmission, the share of retransmissions due to timeouts increases. These are responsible for around 75% of the retransmissions, and most of these are spurious. The share of spurious retransmissions due to fast retransmissions and bundled chunks is also large. Although the number of samples for the third retransmission is low, the data indicate that timeouts are still dominate.

The results from the experiments comparing a modified SCTP (using all modifications) to lkstcp are summarised in Fig. 9. We can see that the modified SCTP shows a slight reduction in the share of timeouts for the first retransmission compared to lksctp. The percentage of fast retransmissions is also somewhat reduced, with

[4] The maximum size was limited to approximately 64 MB in our cross-traffic environment. If we were to allow arbitrarily large file sizes, given the configured bandwidth limitation, the large files would, over time, dominate the traffic, and the desired effect would be lost.



(a) lksctp.

(b) Modified SCTP.

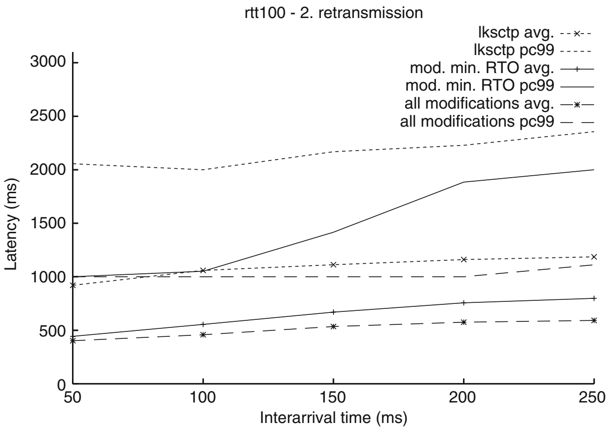
**Fig. 9** Summary of retransmission types and latency results for the (RTT=100) for lksctp and modified SCTP. The *bar* denoted *TO* represents timeouts, *FR* represents fast retransmissions, and *B* are bundled chunks. The *bars* also show what portion of the retransmissions is made up of spurious retransmissions

bundling as the major retransmission factor. For the second retransmission, timeouts still dominate. There is, as expected, an increase in the share of spurious transmissions. The latencies, on the other hand, show that there is a large improvement for all retransmissions. There is also a significant improvement with respect to maximum latency.

To show the effect of the packet interarrival time, Fig. 10 shows average latency and 99th percentiles for the second retransmission. When studying the lksctp results, we can see that the 99th percentiles are far above the other values, but the distance to the 99th percentiles for the minimum RTO modification decreases as the interarrival time increases. However, we can see that the 99th percentile for all modifications is stable and remains below the average value for standard SCTP, i.e., for all tested packet rates between 4 and 20 packets per second.

When we vary the RTT, the results are very much the same. Lowering the RTT reduces the latency only for the smallest interarrival times because timeout retrans-

**Fig. 10** Average latency and 99th percentiles for the tests when the interarrival time increases (RTT = 100)



missions cannot occur before RTO$_{min}$. Increasing the RTT increases retransmission times and the number of retransmissions due to fast retransmit.

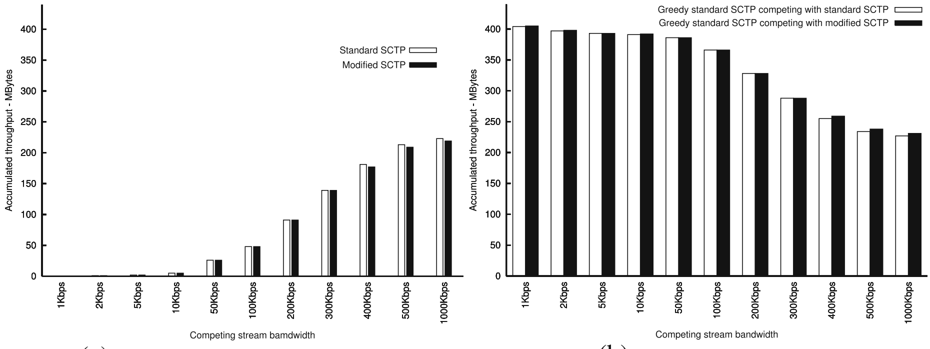
### 6.3 Fairness

A major concern when modifying a transmission protocol like SCTP is whether the principle of fairness for congestion-controlled protocols is preserved. This is especially important in our case, in which more aggressive retransmission measures are implemented. To determine the degree to which the new mechanisms affect fairness, we set up a range of tests where regular SCTP (lksctp) streams competed with modified SCTP. For reference, we also tested two competing lksctp streams. We used the testbed shown in Fig. 7, introduced a 50 ms delay in each direction and limited the bandwidth to 1 Mbps. The streams' achieved throughput was compared as a metric for fairness.

Figure 11a shows the aggregated throughput of the lksctp stream and the modified SCTP stream when trying to achieve different send rates in competition with a greedy lksctp stream. The figure shows no noticeable difference at the "thin-stream" rates. When bit rates increase, and the modifications are no longer active, the regular lksctp actually achieves a little higher throughput than the modified SCTP. This can be explained by small delays in the modified SCTP code that are introduced by the data structures for handling loss and packets in transit. In addition, there are tests to establish whether a stream is thin that are not present in regular lksctp.

In Fig. 11b, the throughput of the greedy streams competing with modified and unmodified SCTP is shown. The graph shows also here that the throughput is nearly identical. As previously explained, the stream competing with the modified SCTP has slightly higher throughput in the 400, 500 and 1000 Kbps experiments. Furthermore, measurements were performed to calculate the average throughput every two seconds to see the short term variations. An example of this is shown in Fig. 12 where only very small differences can be seen between the throughput of the stream that competes with regular lksctp and the stream that competes with the modified SCTP.

The tests indicate that fairness is preserved when a modified SCTP stream competes with an lksctp stream; actually, the stream competing with our modified

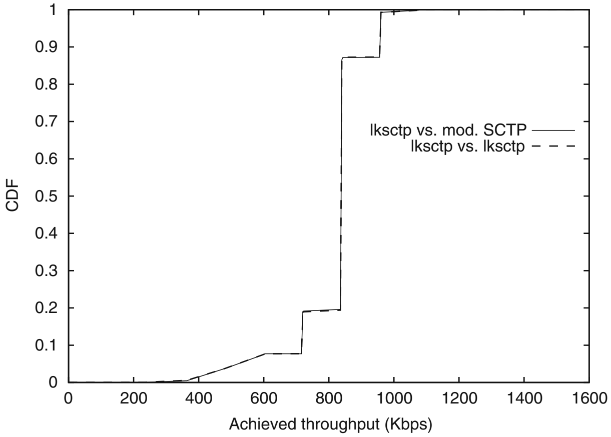


(a) Increasing bandwidth stream throughput.

(b) Greedy stream throughput.

**Fig. 11** Comparison of throughput as an indication of fairness (**a**, **b**)

**Fig. 12** CDF of throughput on 100 ms intervals (connection RTT) for lksctp vs. lksctp and lksctp vs. modified SCTP



lksctp achieves slightly higher aggregated throughput. When few packets are sent per RTT, few resources are consumed whether our modifications are in use or not. When the number of packets per RTT grows, the consumption of resources is almost identical. The reason is that our modifications are switched off when the number of packets in transit exceeds the threshold for thin streams.

## 6.4 Internet tests

To see if our modifications also could improve the latencies observed at the application layer in a realistic, real-world scenario over the Internet, we replayed game traffic from Funcom's massively multiplayer online role playing game Anarchy Online between machines in Oslo and a machine located at the University of Massachusetts (MA, USA). We ran 12-hour tests both from our university network and from three Norwegian ISPs (Get, NextGenTel and Telenor). As can be seen in Figs. 13 and 14, we observed different loss rates and loss patterns that provided different conditions for SCTP, and the results show that the proposed modifications generally improved the application-layer latency, and thus the QoE, when loss occurs and retransmission becomes necessary.

Figure 13 shows the results of replaying the Anarchy Online game traffic between University of Oslo and UMass. In these tests, we compare lksctp and SCTP in FreeBSD, with and without the early fast retransmit (EFR), to our modified SCTP. To get equal network conditions, we had four machines, one for each setup, concurrently sending game traffic to a machine running unmodified lksctp at UMass. We used tcpdump on both sides and calculated the delay between the first transmission of the packet until the packet was received.

Figure 13 shows a cumulative density function (CDF) of the arrival times, i.e., the amount of data (in number of bytes) that has arrived within a given latency (in milliseconds). Large deviations from the average occur only when retransmissions are necessary. In this test, we experienced a packet loss rate below 0.1% which means that the setups perform more or less equally up to a CDF of 0.999. This is also confirmed by the statistics shown in Table 2 which shows that all tests have similar

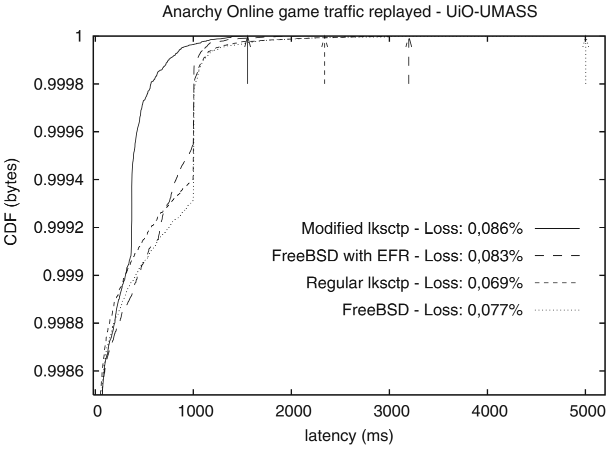


**Fig. 13** CDF for relative arrival time replaying Anarchy Online game traffic

average latencies. As shown in Fig. 1, higher loss-rates can be expected in a game-server setting, and even low loss rates can cause QoE-degrading latency events.

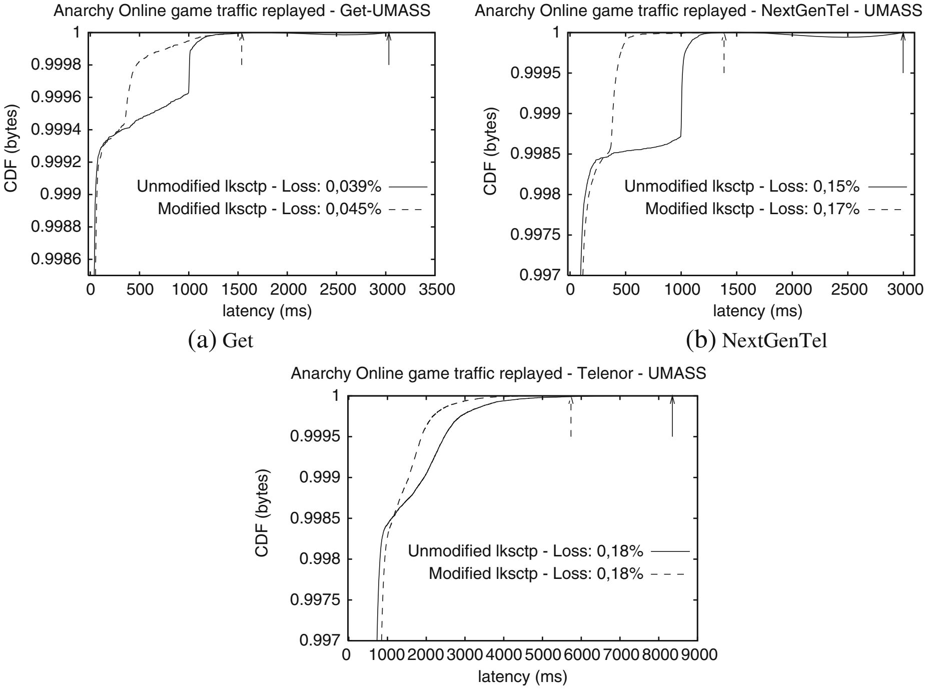
When loss is experienced, the differences is clearly shown. lksctp achieves lower latencies than FreeBSD for a small but relevant number of packets that are retransmitted by fast retransmit. FreeBSD with EFR follows unmodified FreeBSD closely for most situations. It has however clear benefits over both lksctp and unmodified FreeBSD for a relevant number of packets that are early-fast-retransmitted (in the CDF range 0.9992 to 0.9995). That these benefits do not have a larger effect on the CDF is most likely caused by the small number of packets that are concurrently in-flight in our scenario. That inhibits the re-opening of the congestion window when it has collapsed, which in turn prevents EFR from being triggered at all because the condition is that flight size must be smaller than the congestion window size.

Modified lksctp delivers a considerable number of packets with shorter latency, and also looking at the maximum latencies experienced (shown by the arrows in Fig. 13 and in Table 2), we see large improvements. The latency improvement is mainly due to removal of the reset for the retransmission timer after reception of a partial SACK, which forces all other SCTP variations to wait $\mathrm{RTO}_{min}$ before retransmitting lost packets in idle phases of the sender application. Considering that the minimum RTT for the connection was 121 ms, this demonstrates that the modifications can reduce the application-layer latency of a relevant number of lost packets by several RTTs.

As shown earlier (for example in Fig. 9), the latency improvement comes at the cost of a slightly increased bandwidth requirement. Table 2 shows that the modifications increase the number of spurious retransmissions immensely compared

**Table 2** Relative arrival time statistics for about 2.650.000 packets

| | Loss rate (%) | Spurious retransmissions (%) | Average latency (ms) | Maximum latency (ms) |
|---|---|---|---|---|
| mod. lksctp | 0.0855 | 6.708 | 302 | 1725 |
| lksctp | 0.0690 | 0.032 | 304 | 3521 |
| FreeBSD | 0.0765 | 0.006 | 303 | 5326 |
| FreeBSD EFR | 0.0831 | 0.038 | 304 | 2664 |



(a) Get

(b) NextGenTel

(c) Telenor

**Fig. 14** CDF for relative arrival time replaying Anarchy Online game traffic between UMass and three different commercial access networks in Oslo (Norway)

to all the other tested mechanisms. Nevertheless, for the interactive thin-stream applications of our scenario, both the increase in bandwidth and the collapse of the congestion window are negligible disadvantages compared to the latency reduction that can be achieved.

The tests above were performed at our university and may thus not represent the network conditions of a typical user. We validated the results in typical home user settings by running the Internet tests also from three typical access networks provided by Norwegian ISPs. As lksctp and SCTP in FreeBSD (with and without EFR) had similar performance, we compared only modified and unmodified lksctp. The results are shown in Fig. 14. We see the same trends. Our modifications reduce the application-layer latency in case of loss, and as shown by the arrows in the plot, the devastating worst case delays are reduced on order of seconds.

## 6.5 Summary

In summary, our modifications improve the application-layer latency performance for thin streams over the original lksctp and FreeBSD implementations of SCTP (and Linux variants of TCP), regardless of loss pattern and RTT. While the average latency is nearly unchanged, we are able to handle a large number of those problematic cases that are caused by multiple packet losses and that cause severe application-layer delays for interactive applications. The effect is that users of time-dependent applications like interactive games, VoIP or remote desktops experience disruptions much less frequently than before. The improvements, however, come at a price. We see an increased number of retransmissions and more frequently collapsed congestion windows. However, our modifications are meant for applications that mark streams explicitly as being thin, meaning that they require few network resources (small and few packets). For these cases, we consider the extra overhead negligible compared to the reduced latency at the application level where the QoE is increased for the users of the thin stream services.

## 7 Observations and discussion

Changing the retransmission mechanisms of a transport protocol may have consequences for many aspects of fair and reliable networking. In this section, we discuss the choices and alternatives that we have explored and issues that require a more in-depth investigation.

### 7.1 Improving QoE for thin-stream applications

Many applications which are time-dependent greatly suffer from large retransmission delays using existing variations of TCP and SCTP. Our results show that some simple modifications can greatly improve the latency of retransmitted packets. This implies that time-dependent applications (such as those listed in Section 3) can be supported better, which in turn will improve users' experience of the service. Hence, time-dependent thin-stream applications will benefit from our enhancements.

### 7.2 Thin stream detection

To detect when a stream is thin, we use the formula given in Fig. 3, which considers the current number of packets in flight and the loss rate. Modifications are switch on and off depending on this detection. This ensures that only streams with a low packet rate use the enhancements with their more aggressive retransmission behaviour. Thus, if a stream oscillates between thick and thin, the modifications are turned on and off according the the characteristics of the current point in the stream. The Internet tests also show that this works in practise with varying loss and RTT values.

### 7.3 Per-stream enabling of modifications

In the current prototype, we can turn the different mechanisms on and off using exported kernel (/PROC) variables, but this implies that the system administrator must enable the modifications system-wide. To make the use of the protocol modifications more flexible, the code must be extended with per-socket options. The application should be able to turn the modifications on through I/O control function calls (IOCTL).

### 7.4 Fairness

One major issue for modified transport protocols is whether the principle of fairness is preserved. Our modifications of SCTP retransmit more aggressively and consume more resources, depending on packet rate, packet size and loss. However, the results of the small-scale lab experiments presented in Section 6.3 show that fairness is preserved because our modifications are only active when very few packets are in flight. Whenever a stream marked “thin” by the application competes for higher bandwidth, the modifications are switched off. Due to the more frequent collapse of the congestion window, our implementation actually backs off more than unmodified lksctp.

A different fairness issue arises when the number of streams that compete for a bottleneck is so large that their congestion windows are too small to allow fast retransmit. In this case, our modified SCTP would behave more aggressively than unmodified SCTP because of the removed exponential backoff. It would, however, still be less aggressive than a TCP variation that keeps the congestion window open, such as Limited Transmit [1] and Allman et al.’s proposed Early Retransmit. In the future, we want to conduct extensive simulations to map the effect of our proposed transport protocol modifications to different network scenarios.

### 7.5 Linux versus FreeBSD

There exist several implementations of SCTP. To be sure that lksctp does not have any implementation flaws with respect to latency and to have another system to compare with, we also tested SCTP in FreeBSD. In general, our tests in Section 6.4 show that the performance of plain SCTP is more or less equal in both systems. However, FreeBSD has also the EFR latency modification which basically runs an EFR timer (based on the estimated RTT and variance) when there are less packets in flight than the congestion window would permit. Our results show that the EFR modification improves the retransmission delays. Nevertheless, neither of the tested SCTP systems, Linux 2.6.16/2.6.22.14 and FreeBSD 6.2 (with and without EFR), can compete with the latency performance of our proposed modifications. As we also have tested several different implementations on two systems, we believe that our modifications are of general interest for SCTP and not only an lksctp specific improvement.

## 8 Conclusions

We investigated the use of SCTP for thin streams, a type of low-bandwidth stream that is typical for MMOGs and also generated by many other interactive distributed applications. Our investigation started with the assumption that SCTP should perform better than TCP with regard to latency, because it was designed with time-critical signaling traffic in mind. We found that this was not the case and explored, in some detail, the mechanisms that are responsible for the high latencies. Subsequently, we explored changes of SCTP in order to overcome the problem with modifications that require modifications only on the sender side.

We came up with SCTP modifications that reduce application-layer latencies drastically but are paid for with congestion window size reductions and a large increase of spurious retransmissions. Since the modifications are only meant for streams that require low application-layer latency but consume hardly any bandwidth, we looked at a test that disables the modifications for other situations. This thin-stream test checks whether enough packets are in flight to trigger a fast retransmit, and enables the modifications only if this is not the case. The check ensures that the modified SCTP is fair when it competes for bandwidth on a congested link. We suggest further that applications should make the decision of switching our modifications on for individual streams. The reason for this is that the modifications, while very well suited for thin streams that require low latency, are not well-suited for streams with oscillating bandwidth demand that require quick ramp-up of their throughput. These goals are contradictory, and only the application can make the choice.

We have also seen that the computation of the RTO value is highly unstable, and that it remains unstable after our proposed changes. The reason is that the first acknowledgement to arrive for a sample chunk that contributes to the RTO estimation is taken into account without any means of detecting whether it is an acknowledgement of the original transmission. For our situation, where retransmission latency matters, this should definitely be addressed; Karn's algorithm [15] would be a sender-sided remedy, timestamped ACKs [18] or retransmission flags two-sided solutions. We do see benefits in SCTP's aggressive bundling and the 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.

In general, we share other researchers' concern about spurious retransmissions, but such concerns do not apply to our specific scenario. Various proposed fixes increase the average end-to-end message latency in thin streams. Adding Allman et al.'s proposed Early Retransmit would reduce waiting times slightly more when at least two packets per RTT are expected and the RTT is below $\text{RTO}_{min}$, but the first condition is rarely fulfilled in our scenario. The partial ordering and partial reliability extensions to SCTP do, of course, provide other means of overcoming the latency problem. However, we would like to recall that SCTP was designed for signaling and that increased latency is counterproductive in this scenario. Given that it is easy to distinguish between thin and thick streams, we propose to consider the high-bandwidth and low-bandwidth applications of SCTP separately.

## References

1. Allman M, Balakrishnan H, Floyd S (2001) Enhancing TCP's loss recovery using limited transmit. RFC 3042 (proposed standard). http://www.ietf.org/rfc/rfc3042.txt
2. Allman M, Paxson V, Stevens W (1999) TCP congestion control. RFC 2581 (proposed standard). Updated by RFC 3390. http://www.ietf.org/rfc/rfc2581.txt
3. Barford P, Crovella M (1998) Generating representative web workloads for network and server performance evaluation. In: SIGMETRICS '98/PERFORMANCE '98: proceedings of the 1998 ACM SIGMETRICS joint international conference on measurement and modeling of computer systems. ACM, New York, pp 151–160
4. Basto V, Freitas V (2005) SCTP extensions for time sensitive traffic. In: International network conference (INC)
5. Brennan R, Curran T (2001) SCTP congestion control: initial simulation studies. In: Proc. of the international teletraffic congress (ITC 17)

6. Cardoso K, de Rezende J (2002) Http traffic modeling: development and application. http://www.citeseer.ist.psu.edu/556701.html
7. Claypool M, Claypool K (2005) Latency and player actions in online games. Commun ACM 49(11):40–45
8. Coene L, Pastor-Balbas J (2006) Telephony signalling transport over stream control transmission protocol (SCTP) applicability statement. RFC 4166 (informational). http://www.ietf.org/rfc/rfc4166.txt
9. Ekström H, Ludwig R (2004) The peak-hopper: a new end-to-end retransmission timer for reliable unicast transport. In: INFOCOM
10. Fall K, Floyd S (1996) Simulation-based comparisons of tahoe, reno, and sack tcp. Comput Commun Rev 26:5–21
11. Grinnemo K-J, Brunstrom A (2004) Performance of SCTP-controlled failovers in M3UA-based SIGTRAN networks. In: Proc. of the advanced simulation technologies conference (ASTC)
12. Griwodz C, Halvorsen P (2006) The fun of using TCP for an MMORPG. In: International workshop on network and operating system support for digital audio and video (NOSSDAV). ACM, New York, pp 1–7
13. Harcsik S, Petlund A, Griwodz C, Halvorsen P (2007) Latency evaluation of networking mechanisms for game traffic. In: Workshop on network and system support for games (NETGAMES), pp 129–134
14. International Telecommunication Union (ITU-T) (2003) One-way transmission time, ITU-T recommendation G.114
15. Karn P, Partridge C (1988) Improving round-trip time estimates in reliable transport protocols. Comput Commun Rev 17:2–7
16. Ladha S, Baucke S, Ludwig R, Amer PD (2004) On making SCTP robust to spurious retransmissions. ACM Comput Commun Rev 34(2):23–135
17. Ong L, Rytina I, Garcia M, Schwarzbauer H, Coene L, Lin H, Juhasz I, Holdrege M, Sharp C (1999) Framework architecture for signaling transport. RFC 2719 (informational). http://www.ietf.org/rfc/rfc2719.txt
18. Parsa C, Garcia-Luna-Aceves JJ (1999) Improving TCP congestion control over internets with heterogeneous transmission media. In: International conference on network protocols (ICNP), pp 213–221
19. Paxson V, Allman M (2000) Computing TCP's retransmission timer. RFC 2988 (proposed standard). http://www.ietf.org/rfc/rfc2988.txt
20. Pedersen J, Griwodz C, Halvorsen P (2006) Considerations of SCTP retransmission delays for thin streams. In: IEEE conference on local computer networks (LCN), pp 1–12
21. Sat B, Wah BW (2007) Playout scheduling and loss-concealments in voip for optimizing conversational voice communication quality. In: ACM international multimedia conference (ACM MM), pp 137–146
22. SCTP (2006) SCTP information page. http://www.sctp.be/
23. Skype (2008) Skype homepage. http://www.skype.com
24. Shirmohammadi S, Georganas ND (2001) An end-to-end communication architecture for collaborative virtual environments. Comput Networks 35:2–3
25. Stewart R, Ramalho M, Xie Q, Tuexen M, Conrad P (2004) Stream control transmission protocol (SCTP) partial reliability extension. RFC 3758 (proposed standard). http://www.ietf.org/rfc/rfc3758.txt
26. Stewart R, Xie Q, Morneault K, Sharp C, Schwarzbauer H, Taylor T, Rytina I, Kalla M, Zhang L, Paxson V (2000) Stream control transmission protocol. RFC 2960 (proposed standard), updated by RFC 3309. http://www.ietf.org/rfc/rfc2960.txt
27. The KAME project (2006) The KAME project homepage. http://www.kame.net/
28. The lksctp project (2004) The linux kernel stream control transmission protocol (lksctp) project homepage. http://lksctp.sourceforge.net/
29. Wang B, Kurose J, Shenoy P, Towsley D (2004) Multimedia streaming via tcp: an analytic performance study. In: MULTIMEDIA '04: proceedings of the 12th annual ACM international conference on multimedia. ACM, New York, pp 908–915
30. Woodcock BS (2009) An analysis of mmog subscription growth. http://www.mmogchart.com
31. Zink M, Westbrook D, Abdallah S, Horling B, Lakamraju V, Lyons E, Manfredi V, Kurose J, Hondl K (2005) Meteorological command and control: an end-to-end architecture for a hazardous weather detection sensor network. In: Proceedings of the ACM workshop on end-to-end, sense-and-respond systems, applications, and services (EESR), Seattle, pp 37–42. http://mas.cs.umass.edu/paper/385

**Andreas Petlund** received his B.Sc. in informatics in 2003 and his M.Sc. in 2005, both at the University of Oslo. He is currently a third year Ph.D. student at Simula Research Laboratory / University of Oslo. His main research interests include network protocol optimization for time-dependant thin streams, operating systems optimisations and hardware offloading.

**Paul Beskow** received his B.Sc. in Informatics in 2003, and his M.Sc. in 2007, both at the University of Oslo (UiO). His main research interests include system support for low-latency communication including object migration, network protocol optimizations, as well as various TCP protocol optimization issues.

**Jon Pedersen** received his B.Sc. in 2004 and M.Sc. in 2006, both at the University of Oslo. He is currently working as a research scientist at the Norwegian Defence Research Establishment. His main research activities include modelling and simulation of military systems.

**Espen Søgård Paaby** received his B.Sc. in Informatics in 2004, and his M.Sc. in 2006, both at the University of Oslo (UiO). His main research interests include network protocol optimization for time-dependant thin streams, as well as various TCP protocol optimization issues.

**Carsten Griwodz** is a professor at Simula Research Laboratory, Norway and at the Department of Informatics at the University of Oslo. He received his Diploma in Computer Science from the University of Paderborn, Germany, in 1993. In 1997 he joined the multimedia communications lab at Darmstadt University of Technology, Germany, where he obtained his doctoral degree in 2000. His interests lie in the improvement of system support for interactive distributed multimedia, with operating systems and protocol support for on-demand streaming applications and multiplayer games in particular.

**Pål Halvorsen** is an associate professor at Simula Research Laboratory and at the Department of Informatics, University of Oslo, Norway. He is member of the Center for Research-based Innovation "Information Access Disruptions". He received his master and doctoral degree in computer science from the University of Oslo in 1997 and 2001, respectively. His research activities focus mostly on resource utilization and system support for distributed multimedia systems, and in particular, in the area of on-demand streaming applications and multiplayer games.