

SCTP Congestion Control: Initial Simulation Studies

Rob Brennan¹ and Thomas Curran²

Teltec DCU, Dublin 9, Ireland

¹ Contact author: brennanr@teltec.dcu.ie, Ph. +353 1 700 5853, Fax +353 1 700 5092

² currant@eeng.dcu.ie, Ph. +353 1 700 5128, Fax +353 1 700 5092

Abstract

This paper uses simulation studies to explore the behaviour of SCTP congestion control. Several possible variants of SCTP congestion control are identified and each is compared with the behaviour of Reno TCP, a well-known TCP variant. We identify three flaws in the current SCTP congestion control mechanisms. 1. Efficient recovery after a packet loss is only possible if SCTP generates the optional Gap Ack Blocks in SACK messages. 2. The SCTP fast retransmit procedure is vulnerable to being mistakenly triggered multiple times leading to under-utilization of the network during recovery and duplicate retransmissions of the lost packet. 3. The SCTP fast retransmit procedure must wait for half a window of data to be acknowledged before retransmitting a lost packet yielding a slow response to packet loss. We also provide suggested remedies for these flaws. Finally we demonstrate how SCTP congestion control mechanisms can cause a burst of packets after recovery from a packet loss.

1. Introduction

For many years SS.7 has been the dominant bearer of signaling traffic for telecommunications networks, but recently many proprietary solutions for transporting signaling traffic over IP have appeared. This approach promises tighter integration with VoIP solutions and ultimately the possibility of a common core network transporting both signaling and media traffic. Driven by industry interest, and general agreement on the unsuitability of either TCP or UDP, the IETF Signaling Transport (SIGTRAN) group was formed in 1999 to standardize a suitable transport protocol for signaling traffic over IP. Stream Control Transmission Protocol (SCTP) is the result of this work, recently published as RFC 2960 [1] by The Internet Society.

SCTP is the fundamental member of a family of protocols [2] being designed by the SIGTRAN group to allow SS.7 messages to be transported over an IP infrastructure. Furthermore the enhanced capabilities of SCTP when compared with traditional Internet transport protocols such as TCP and UDP may make it attractive as a transport for a wide range of traditional Internet services such as those based on HTTP and SIP.

SCTP is characterized in a number of current internet drafts which discuss its capabilities and applicability to both the signaling domain [3] and more general applications [4]. Essentially, SCTP is a reliable, message-oriented data transport protocol that supports multiple streams within an association and hosts with multiple network addresses (called multi-homing in SCTP). All of these properties were chosen to make SCTP more suitable for signaling transport than TCP. For a more complete overview of SCTP see [5].

This paper seeks to investigate the behavior of SCTP under congestion conditions and provide comparisons with a well-known TCP variant, Reno TCP. SCTP congestion control mechanisms are based upon TCP congestion control principals [6], and the mechanisms are based on the optional SACK extensions to TCP [7]. Modern TCP congestion control principals originate with Van Jacobson in 1988

[8] and they have been refined and investigated by a number of researchers [9,10,11,12] since then. Although many of the leading TCP congestion control researchers participated in the specification of SCTP, to our knowledge no work has yet appeared that specifically deals with SCTP congestion control, although some useful performance comparisons with TCP were included in work done in 2000 by Jungmaier et al. [5].

The following section discusses SCTP congestion control mechanisms, algorithms and options. They are then compared with the TCP mechanisms on which they are largely based. Section 3 discusses the simulation environment and a series of experiments, which have been performed to evaluate similarities and differences between the protocols and SCTP variants. Section 4 presents and analyses the results of the simulation studies. Section 5 discusses future work and planned or possible developments in SCTP and finally some conclusions are presented.

2. SCTP Congestion Control

The basis of SCTP congestion control is an amalgamation of current best practice for TCP implementations with extensions to deal with the multi-homing aspect of SCTP and modifications due to the message rather than stream-based nature of the protocol. The standard specifies an adaptive sliding window control with adapted versions of the well known TCP slow-start, congestion avoidance, fast retransmit and fast recovery mechanisms. In addition recent work on TCP, such as congestion window validation [9], is also incorporated. One currently optional mechanism for TCP, the use of selective acknowledgements (SACKs) [7] to report out of sequence data arriving at the receiver, has been incorporated into SCTP as the baseline for congestion control implementation. This is due to the demonstrated superiority [13] of this mechanism to earlier TCP congestion control options. Although the possibility to support IP Explicit congestion Notification (ECN) [14] has been incorporated into SCTP, this mechanism is optional and (as with TCP) a packet loss is the normal method of congestion indication.

SCTP Associations

Data transfer between two hosts in SCTP takes place in the context of an association. All transfer of data between hosts is encapsulated in SCTP datagrams, which contain a common header and a sequence of structures called chunks. These chunks may be either SCTP control information, such as a SACK chunk, or user data carried in a data chunk. Within the association data chunks belong to a specific stream of data. Normally within a stream data is ordered, although transfer of unordered data is also supported, and ordered data that has arrived out of sequence is buffered until the missing data has arrived.

During association initialization the SCTP endpoints exchange the size of their receiver window (an indication of the space available in their inbound buffer) and the initial transmission sequence number (TSN) of the user data chunks¹ to be exchanged during the association. All data chunks transmitted receive a unique (monotonically increasing) TSN from a 32 bit namespace. Ordered data within a stream also receives a unique stream sequence number (SSN) from a 16 bit namespace. User data is not limited in size as it may be fragmented over multiple data chunks. Individual data chunks may be up to 655634 bytes in length. SCTP should ensure that each SCTP datagram including any chunks fits within one IP datagram and fragments user data into multiple data chunks as necessary.

Monitoring Data Delivery

In order to track the amount of data currently within the network the SCTP sender must associate TSNs and the byte size of the corresponding data chunk. Data chunks arriving at the SCTP receiver are

¹ Each of these forms a distinct user-generated message to be carried by SCTP unlike TCP's stream interface

acknowledged by transmitting a SCTP packet with a SACK control chunk. The SACK chunk carries several pieces of information:

- The Cumulative TSN Ack, which records the highest sequential TSN received.
- The Advertised Receiver Window Credit, which reflects the current capacity of the receiver's inbound buffer.
- A sequence of Gap Ack Blocks, which record any out of sequence TSNs received.
- A sequence of Duplicate TSNs, which record any TSNs for which duplicates have been received.

Data is not considered fully delivered until the cumulative TSN Ack point advances past its TSN. Thus the information in the Gap Ack Blocks corresponds to TCP SACK blocks. Under normal conditions SCTP uses a delayed acknowledgement scheme which sends one SACK for every second incoming packet which contains one or more new data chunks.

In order to perform congestion control SCTP uses a number of internal variables to control the rate at which data is injected into the network. These are as follows, all are recorded in bytes:

- Receiver window size (*rwnd*) – this corresponds to a sender's view of the receiver's incoming buffer space.
- Congestion control window (*cwnd*) – this corresponds to the sender's view of network conditions. The initial value of *cwnd* is less than or equal to twice the path maximum transmission unit (MTU), corresponding to the more aggressive value recently adopted for TCP in [15].
- Slow-start threshold (*ssthresh*) – the sender uses this to distinguish between slow-start and congestion avoidance phases.
- Partial bytes acked – the sender uses this to limit *cwnd* adjustment to once per RTT during congestion avoidance phases.

The receiver window size is always kept on a per association basis, in a multi-homing environment the other values must be recorded on a per destination address basis. The rest of this discussion does not include multi-homing aspects as it is beyond the scope of this paper.

Behaviour

Normal² operation of SCTP follows TCP in allowing a sender to inject $\text{Min}(rwnd, cwnd)$ data into the network. Although in order to protect against lost SACKs when *rwnd* is zero it is permitted to have one packet in flight (if allowed by *cwnd*) to probe for changes in the advertised receiver window credit.

During slow-start (at the start of an association, after a transmission time-out or after a long idle period) *cwnd* is increased exponentially to probe the network for capacity. This occurs while $cwnd \leq ssthresh$. During this phase *cwnd* may only be increased by $\text{Min}(\text{total size of new data acknowledged by a SACK, path MTU})$ and only if the SACK advances the cumulative TSN ack point (i.e. acknowledges new sequential data). This means that duplicate SACKs (ones which do not advance the cumulative TSN Ack point) can only enable new data transmission by reducing the sender's view of how much data is currently in the network (flightsize).

During the congestion avoidance phase ($cwnd > ssthresh$) *cwnd* may be incremented by no more than 1 MTU per RTT if the sender has *cwnd* or more bytes of data outstanding. This produces a linear increase in *cwnd*, up to the limit of the initial *rwnd* exchanged at association initialization.

² Some interesting discussion has occurred on the SIGTRAN mailing list suggesting that some (apparently conformant) implementations in fact allow $\text{Min}(rwnd, cwnd + 1 \text{ path MTU} - 1 \text{ byte})$

Whenever a SCTP sender is not transmitting to a receiver address (both during slow start and congestion avoidance), the *cwnd* is adjusted to $\text{Max}(cwnd/2, 2*MTU)$ per RTO. This protective measure decreases *cwnd* as network conditions are unknown for a destination to which the sender is not transmitting, and is similar to the approach for TCP outlined in [9].

There are two methods of packet loss detection (taken as a congestion notification by these controls) defined in SCTP:

1. Detection of gaps in received TSNs through Gap Ack reports in a SACK. Normally a sender will wait for four consecutive reports before reacting as packet re-ordering could occur in the network. In this case the path variables are reset as follows:

$$ssthresh = \max(cwnd/2, 2*MTU)$$

$$cwnd = ssthresh$$

This will have the effect of putting the sender in slow-start with a reduced *cwnd*. This will force a sender to wait before transmitting new data for either the SACK advancing the cumulative TSN ack point beyond the missing packet (Normally retransmitted through Fast Retransmit, see below) or until enough duplicate SACKs arrive to reduce the flightsize below the modified *cwnd*.

2. Timeout of the retransmission timer (T3-rtx in the specification). In this case the path variables are reset as follows:

$$ssthresh = \text{Max}(cwnd/2, 2*MTU)$$

$$cwnd = 1*MTU$$

This will have the effect of putting the sender in slow start and assure that no more than one packet is in flight until it receives the SACK advancing the cumulative TSN ack point beyond the TSN of the retransmitted packet.

Whenever a SCTP retransmission occurs, the sender bundles as many as possible of the earliest TSNs marked as missing into a single packet and retransmits them. Note that in the case of a Fast Retransmit triggered due to gap reports this will correspond to the earliest TSNs which are reported as missing at least 4 times. For a timer expiry it will be the earliest TSNs which have not yet been acknowledged by the cumulative TSN ack point irrespective of gap ack reports.

SCTP also requires the use of path MTU discovery techniques but these are not discussed here as they are beyond the scope of our simulation studies.

Differences between SCTP and TCP Congestion Control

From the above discussion of SCTP congestion control mechanisms a number of major differences between these mechanisms and the equivalent TCP mechanisms are apparent. However there are also a number of subtle changes in behavior. In this section we try to highlight significant differences, especially those which will be illustrated by our simulation studies in the next section. For the purposes of this section we treat the standard TCP congestion control mechanisms as those laid out in the most recent RFC defining TCP behaviour [6].

There are probably two changes to the control scheme, which represent the most significant departure from standard TCP congestion control mechanisms:

- The direct dependence on the number of bytes acknowledged rather than the number of acknowledgements received to increase the congestion window.
- The implicit dependence on selective acknowledgement (SCTP Gap Ack block, TCP SACK) messages for reporting in the case of packet losses.

The first change should in theory produce a slightly more sophisticated control mechanism and is similar to experimental proposals for TCP proposed by Allman [11]. It also eliminates the need for additional control variables such as the *pipe* variable suggested for SACK TCP [13] to control the amount of data injected into the network during fast recovery as *cwnd* now also controls this. The second major change is in itself a welcome improvement over earlier (pre-SACK TCP) TCP implementations and should ensure greater resilience to packet loss than pre-SACK TCP implementations.

Minor changes to the control mechanisms include:

- Fast retransmission of packets is controlled by *cwnd*. This reduces the speed with which SCTP reacts to packet loss and hence reduces throughput. See the results section for more discussion.
- During recovery from a packet loss certain implementations of the SCTP fast retransmission algorithm are vulnerable to multiple retransmissions of the lost packet and a collapse of *cwnd* down to its minimum value of $2 \cdot \text{pathMTU}$ with a corresponding decrease in throughput. See the results section for more discussion.
- Without gap block generation SCTP has no way of detecting packet loss other than a retransmission time out. See the results section for more discussion.
- No explicit fast recovery phase is required, as no artificial inflation of *cwnd* is required for throughput to be maintained.
- TCP allows a choice of congestion avoidance or slow start behavior when *cwnd* equals *ssthresh*. SCTP mandates slow-start in this case.
- During the congestion avoidance phase *cwnd* may only be increased if the full *cwnd* is currently being used. This restriction is not present in TCP, however current TCP research [11] indicates that it is preferable to the “standard” TCP approach of increasing *cwnd* based on incoming non-duplicate ACKs. This limits *cwnd* in SCTP to be less than or equal to the receiver’s initial advertised window.
- An unlimited number (barring path MTU restrictions) of gap ack blocks are allowed in SCTP. TCP has a maximum of three SACK blocks. This should make SCTP more resilient to acknowledgement loss and better able to deal with multiple (non-sequential) packet loss than TCP.
- The value of *cwnd* decays when packets are no longer being transmitted. This protects against the use of an invalid congestion window during application limited periods as recommended by current TCP research by Handley et al. [9].

3. Simulation Model

All of the simulation results presented in this paper were obtained from an implementation of SCTP for the Opnet Modeler [16] simulation environment. This was combined with standard models for IP. Figure 1 below illustrates the simulated network.



Figure 1: The Simulation Model

The circle G indicates a finite buffer gateway, and the squares indicate sending (S) and receiving (R) hosts. The queue management strategy in the gateway is drop-tail. The links are labeled with their bandwidth capacity and delay. Each simulation contains an additional flow of non-adaptive UDP traffic from the sender to the receiver to generate the desired pattern of packet losses. These flows have limited data to send and are not shown in the figures.

Following the approach of earlier TCP congestion control studies [9,10,11,12,13], the gateway node in these simulations implements a drop-tail congestion control strategy with small buffers. This is not intended as a realistic configuration, just a simple scenario for illustrating SCTP congestion control behaviour.

The SCTP model employed in these simulations does not use production code and is not a detailed model of any particular implementation of SCTP. In fact given the proliferation of TCP implementations and the large number of participants³ in SCTP bake-off sessions it is unlikely that any detailed model could claim to model a “typical” SCTP. Instead generality has been maintained by modelling the most important features of SCTP relevant to the congestion and error control algorithms. All connections have clock granularity limited only by the precision of the simulation clock.

The simulations in this paper all consist of one-way traffic with the transport protocol normally operating in delayed acknowledgement mode i.e. one SACK or ACK for every second data packet received. When packet loss is detected by the receiver it changes to sending one SACK or ACK for every data packet received. Although they are queued at the gateway none of the acknowledgements are significantly delayed or lost. All user data is generated in packets of 1000 bytes. These are transferred in 1000 byte data segments (excluding headers) for TCP and 1000 byte data chunks (excluding headers) for SCTP. For all simulations the initial advertised receiver window was 20,000 bytes (i.e. 20 packets).

Two types of graph of the simulation results are shown. The primary type of graph is plotted at the gateway in a style similar to that employed in [13]. The *x*-axis shows the packet arrival, departure or discard time in seconds. The *y*-axis shows the packet number *mod* 60. Thus a packet arriving and departing the gateway with little delay appears as a single point (■), longer queuing delays produce a resolution of this single point into two colinear points, spaced by the queuing delay. A discard is shown with an “X” rather than a point. Acknowledgements are shown as a colinear point one round trip time from the packet they acknowledge. The second type of graph is only shown for SCTP and illustrates a straightforward plot of the senders congestion window (line) and slow start threshold (◆) against time. These are described in the text accompanying those figures.

SCTP RFC Options and Recommendations

As is usual with any IETF document a number of features of SCTP congestion control have different levels of precedence for conformance to the specification. These are summarized in the table below along with the features included in our model. Only the most important or relevant mandatory parts of the specification are included in the table below as all mandatory features are implemented in our model.

SCTP Control Feature	RFC	Included
Retransmission timer on a per chunk basis	MAY	No ^a
Chunk bundling on transmission	MAY	No ^b
Chunk bundling delay	MAY	No ^b
Delayed SACKs follow RFC2525 section 4.2	SHOULD	Yes
Transmitters are more conservative than congestion control specified in specification	MAY	Yes ^c
Transmitters are more aggressive than congestion control specified in specification	MUST NOT	Yes ^c
SACK reports out of order TSNs in Gap Ack Blocks	SHOULD	Yes ^c
SACK reports duplicate data chunks	SHOULD	Yes
Send a SACK when a packet arrives with both new and duplicate data	MAY	No

³ 23 separate implementations were recorded at the last SCTP bake-off.

chunks		
Determine if SACK loss is occurring from duplicate TSNs reported in a SACK	MAY	No ^d
Receiver notifies sender in timely manner of changes in ability to send data	SHOULD	Yes
Receiver only increments a_rwnd when data is released from receive buffer	SHOULD	Yes
Receiver puts the current value of a_rwnd in every SACK	SHOULD	Yes
Receiver takes into account that sender will not retransmit data chunks that are acked via cumulative TSN ack	SHOULD	Yes
Receiver does not drop data previously acked in a SACK	SHOULD	Yes
Sender uses rules in specification for calculating rwnd from a_rwnd	SHOULD	Yes
New RTT measurements are made no more than once per round-trip	SHOULD	Yes
Slow-start and congestion avoidance algorithms used	MUST	Yes
Initial value of ssthresh equal to a_rwnd	MAY	Yes
Wait for 3 duplicate SACKs before Fast Retransmit	SHOULD	Yes
Retransmitted data is not used for RTT calculation (Karn)	MUST	Yes
^{a)} All of the simulations presented here only include one data chunk per packet. ^{b)} All of the simulations presented here assume that each data chunk represents a maximum size segment so no bundling is possible. ^{c)} The model has the ability to turn on or off this feature to simulate different implementations of SCTP. ^{d)} No procedures to achieve this are included in the current specification, it is marked “for further study”.		

Table 1 : RFC Options and SCTP Model Capabilities

SCTP Variants

In order to investigate the effects of different possible interpretations of the SCTP specification a number of configurations for the SCTP simulation model were possible. The variants used were as follows:

- **“Standard” SCTP** : This variant follows the letter of the SCTP specification exactly and all “MUST” and “SHOULD” features of the specification are supported.
- **No Gap Ack Block SCTP (NoGap-SCTP)**: This variant is the same as Standard SCTP but it does not implement the receiver generation of gap blocks in SACKs.
- **Early Reference Implementation SCTP (RefImpl SCTP)**: This variant is the same as standard SCTP except that it interprets the fast retransmit procedure in the same way as the earlier (pre version 4.0) reference implementation of SCTP [17]. This means that instead of resetting the counter for each TSN gap immediately it reaches 4, it waits until the TSN has been retransmitted. It is to be expected that many early implementations of SCTP follow this reference implementation in its interpretation of the specification.
- **Modified SCTP**: This variant is the same as standard SCTP except that: 1. Once a packet has been retransmitted by the fast retransmit procedure it is marked as ineligible for retransmission again until a transmission time-out has occurred. This more conservative approach is similar to that adopted for various experimental TCP variants [12]. 2. it ignores the value of *cwnd* when performing a retransmission due to the detection of packet loss through gap ack blocks. It is a strictly experimental, more aggressive modification to the SCTP congestion control scheme. It also represents a likely interpretation of the current specification by implementers familiar with TCP as in TCP fast retransmit is not constrained by *cwnd*. The authors, as a result of our simulation studies with the other three variants, suggest these modifications to SCTP congestion control.

4. Results

This section presents the results of running our simulation for three scenarios. The two basic scenarios are “Single Packet Loss” - The effects of a single packet loss early in the slow-start phase of a transfer and “Multiple Packet Loss” - The effects of losing 3 packets early in the slow-start phase of a transfer. For

each scenario we present results for Reno TCP, Standard SCTP, Reference Implementation SCTP, No Gap Ack Block SCTP and Modified SCTP. In all cases SCTP is transfers all data in the association in a single stream of ordered data. This is not an optimal use of SCTP but it makes comparison with a TCP transfer clearer.

Finally in the section “SCTP Bursts” we examine how the loss of a single packet when the congestion window is nearly fully open can produce a burst of packets at the end of the recovery period for certain SCTP variants.

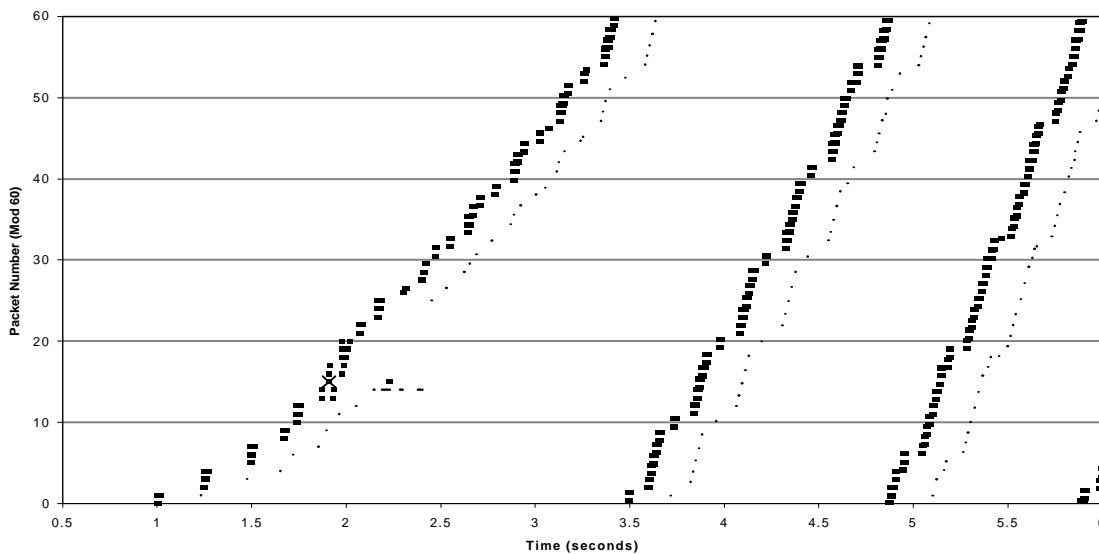
Note that all of the plots showing packet arrivals at the gateway have been adjusted to show the first packet arrival at a time of 1 second for ease of comparison.

These simulation studies demonstrate that there are three flaws in the current congestion control algorithms:

1. Efficient recovery after a packet loss is only possible if SCTP generates the optional Gap Ack Blocks in SACK messages.
2. The SCTP fast retransmit procedure is vulnerable to being mistakenly triggered multiple times leading to under-utilization of the network during recovery and duplicate retransmissions of the lost packet.
3. The SCTP fast retransmit procedure must wait for half a window of data to be acknowledged before retransmitting a lost packet yielding a slow response to packet loss.

We also provide suggested remedies for these flaws which are incorporated into the Modified SCTP variant.

Single Packet Loss



Figure

2: Reno TCP with a single packet loss

Figure 2 shows Reno TCP’s behaviour in this scenario. As shown in earlier studies [13] Reno TCP handles a single packet loss well and recovers smoothly. Unfortunately the performance of the various SCTP variants is more mixed and is covered in more detail in the subsequent paragraphs.

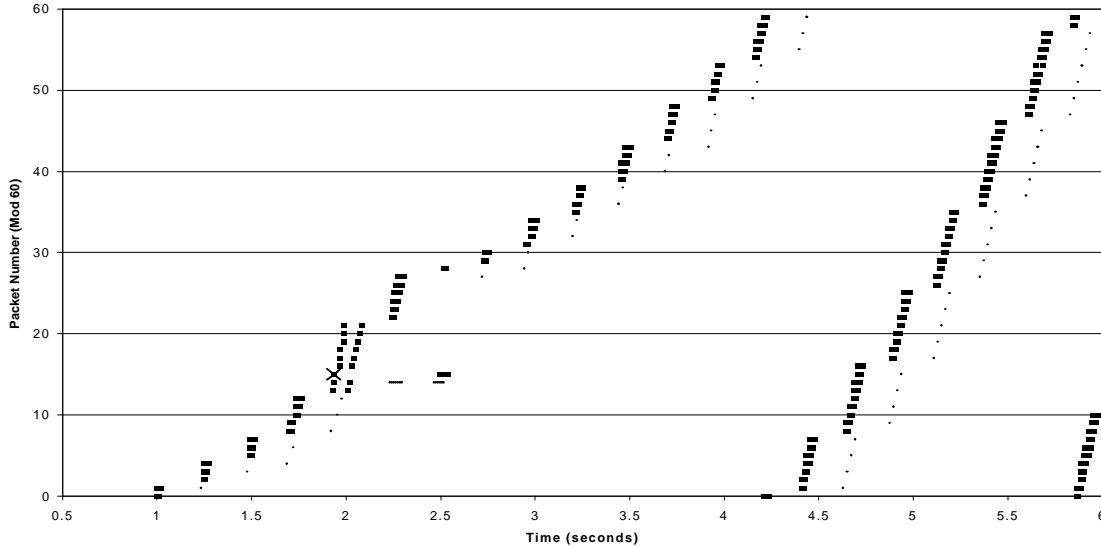


Figure 3: Standard SCTP with one packet loss

The behavior of Standard SCTP after packet 15 is detected as lost by the fast retransmit procedure illustrates three problems in the current specification. These are:

- SCTP must wait for half a window of data to be acknowledged before it can retransmit the packet (it reacts approximately 1 RTT later than TCP in this case).
- Even when retransmission is scheduled, the continuous stream of duplicate SACKs arriving at the sender trigger the fast retransmit algorithm multiple times (3 in this case), each time reducing *ssthresh* and *cwnd*. The effect of this is noticeable after the SACK acknowledging the retransmitted packet arrives as only small amounts of data are transmitted and increase of the *cwnd* must be done with the slower congestion avoidance algorithm. This results in a much slower utilization of the available bandwidth than TCP. After the six seconds shown in the plot SCTP has increased *cwnd* to approximately 14,000 bytes compared with TCP's 15,000.
- Three fast retransmissions of packet 15 are triggered due to the stream of duplicate SACKs arriving at the sender. These occur as after the packet has been retransmitted once (but not yet acknowledged) the current SCTP fast retransmit algorithm requires that subsequent duplicate SACKs trigger fast retransmit again.

The first two flaws in SCTP congestion control are not inherently dangerous, as they are more conservative than equivalent TCP flows. The third problem is more serious as it requires the sender to inject extra, duplicate data into the network at a time of congestion. Of course all three flaws result in poorer performance than TCP. We propose a simple fix for the multiple fast retransmission triggering in the current specification that has commonly been recommended for SACK TCP implementation [13]. The protocol behavior should be changed so that a packet that has triggered the fast retransmit procedure should no longer be eligible for fast retransmit until a retransmission timeout has occurred. It is also possible to fix the first problem by allowing fast retransmit to take place regardless of the value of *cwnd*. This second modification is of a more experimental nature and may be too aggressive when competing with TCP flows or in the face of persistent congestion. Nonetheless, the current SCTP reference implementation has adopted both of our recommendations and we provide simulation results for them.

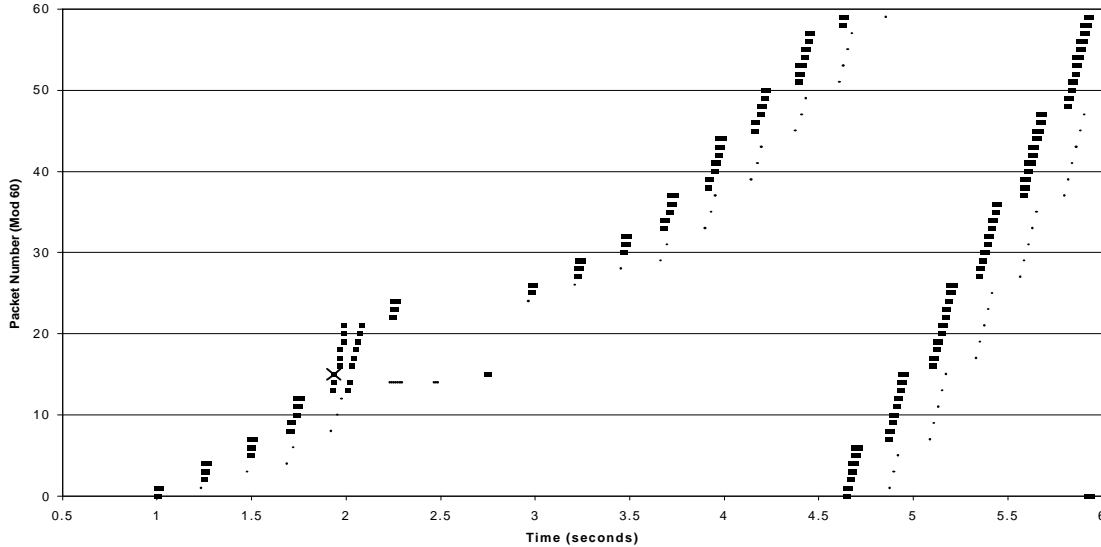


Figure 4: NoGap SCTP with one packet loss

Figure 4 shows the behaviour of NoGap SCTP. This is even more disappointing and illustrates a further flaw in the current specification. Without gap block generation SCTP cannot trigger the fast retransmit procedure and hence must wait for a retransmission timeout to occur. It then uses slow-start to recover from the timeout. The use of slow-start actually gives almost the same throughput as Standard SCTP due to its use of congestion avoidance from a small value of *cwnd*. When compared with TCP it gains less throughput but at least it does not mistakenly inject duplicate data into the network. This is obviously a problem with the specification, if a total reliance is to be placed on Gap Blocks then their generation must be mandatory. It would be possible to implement Reno TCP-style controls in SCTP but these have so far proved to be inferior to SACK TCP [13]. This problem is exacerbated by the lack of a gap ack block negotiation mechanism in the SCTP association initialization process. SACK TCP implementations may use the TCP options field to signal their SACK capability at connection start. Another point to bear in mind is that even if a vendor produced a SCTP implementation which did not generate gap ack blocks but was capable of reacting to duplicate SACKs like Reno TCP they could not guarantee that all hosts they communicate with would also have this capability. In this case the authors recommend that mandatory generation of gap ack blocks for all SCTP implementations is the easiest and most efficient fix.

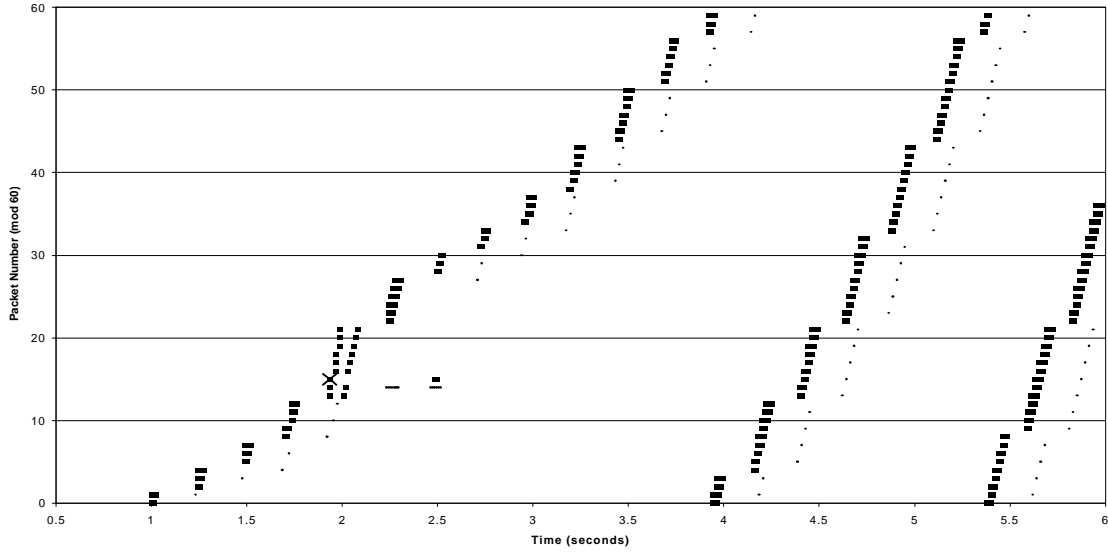


Figure 5: RefImpl SCTP with one packet loss

Figure 5 shows the behaviour of RefImpl SCTP. This performs significantly better than Standard SCTP but is still inferior to Reno TCP. The reason for the changed behaviour is that when packet loss is detected by the fast retransmit procedure, RefImpl SCTP does not reset the gap counter immediately. Thus subsequent duplicate SACKs arriving increase the gap counter significantly beyond 4. The counter is only reset when the lost packet is actually retransmitted. This protects against multiple triggering of the fast retransmit procedure and hence multiple duplicate packet retransmissions and reductions in *ssthresh* and *cwnd*. However as is shown in the next section, “Multiple Packet Loss”, it is still possible for this variant to produce duplicate retransmissions. Overall the flow produced by this variant is not dangerous, as it is less aggressive than competing TCP flows. It is however the authors contention that it does not accurately reflect the current wording of the standard and if it is to become the required behaviour then the standard requires some small changes.

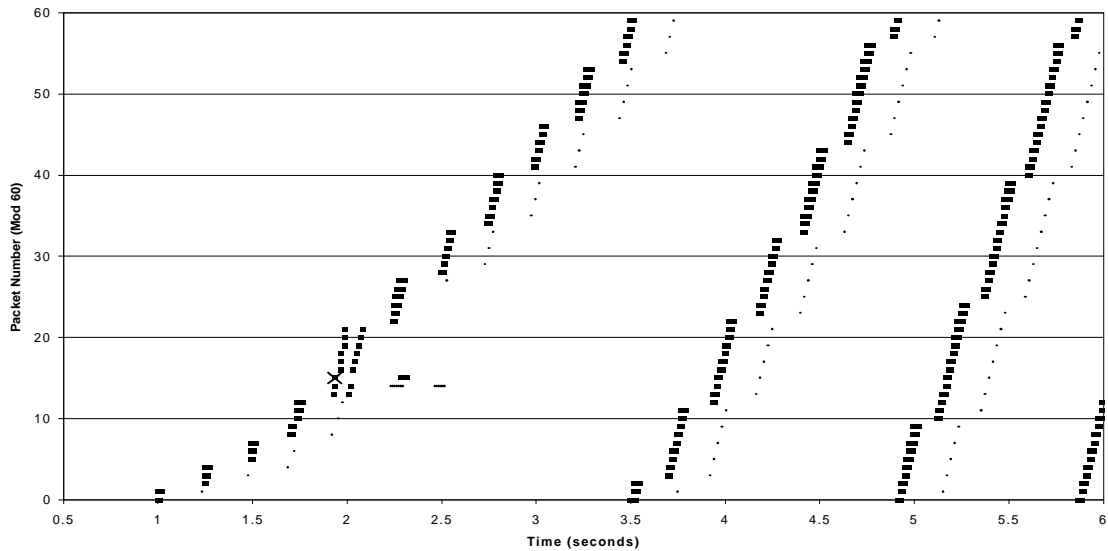


Figure 6: Modified SCTP with one packet loss

Figure 6 shows the behaviour of Modified SCTP. This performs better than any other SCTP variant and is very similar to Reno TCP. By allowing immediate retransmission of the lost packet upon detection (fast retransmit ignores the value of *cwnd*) the recovery process starts as fast as Reno TCP. The introduction of an additional constraint on fast retransmission that packets triggering it can not trigger it again until after a timeout has removed the duplicate retransmission events seen for Standard SCTP. Some slight differences in the figure from Reno TCP can be explained by the TCP model starting to perform some fragmentation of the data segments after *cwnd* is reduced. Our SCTP model does not currently support fragmentation of user data and hence waits for a full datagram to be allowed before transmitting. Additionally our SCTP model uses the more aggressive interpretation of the use of *cwnd* mentioned earlier i.e. if there is even one byte left in *cwnd* it is permitted to send a datagram (up to the size of the path MTU). In general this variant has been shown to perform well in this case and present no danger to competing Reno TCP flows.

Figure 7 shows the behaviour of Reno TCP when 3 packets are dropped within one window of data. As expected from earlier studies [13] Reno TCP reacts very badly to this situation. This is because it has to wait for the retransmission timer to recover from the dropped packets. This is largely due to Reno's fast recovery behaviour and the lack of an acknowledgement mechanism for out of order data received. Both New Reno and SACK TCP implementations recover more gracefully from this situation.

Multiple Packet Loss

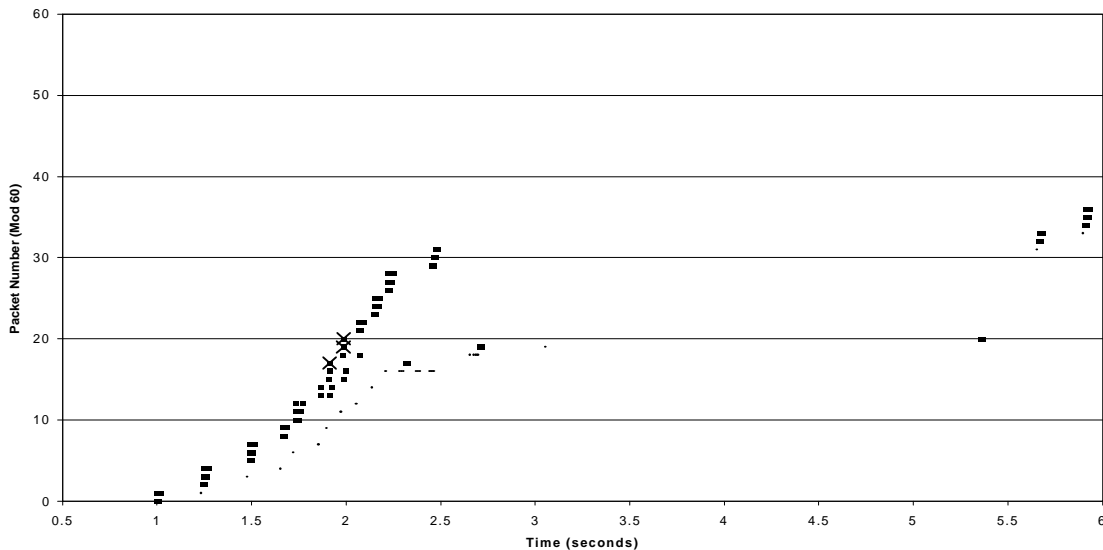


Figure 7: Reno TCP with 3 packet drops

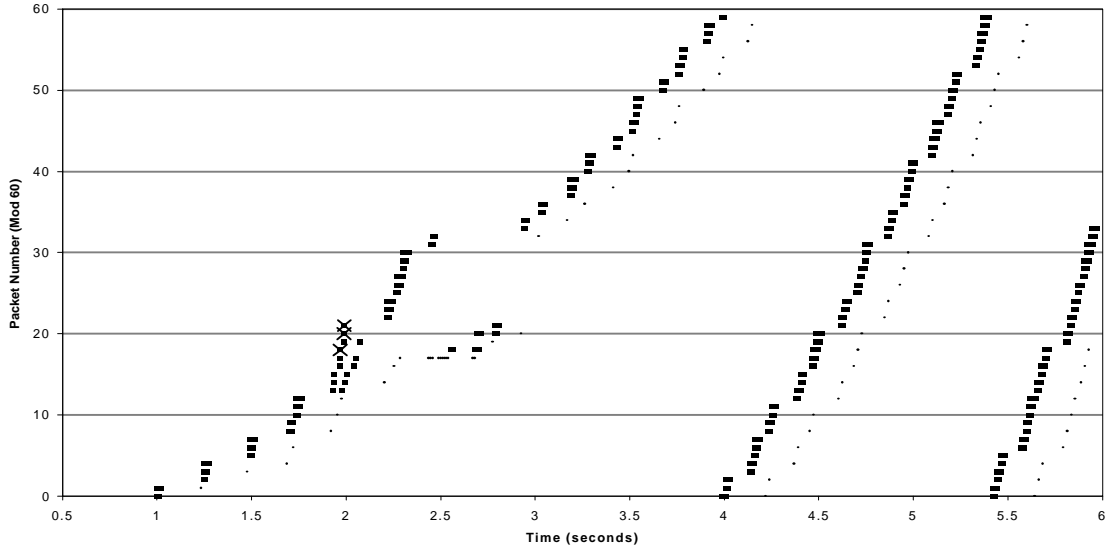


Figure 8: Standard SCTP with 3 packet drops

Figure 8 shows the behaviour of Standard SCTP for this scenario. It recovers much more successfully than Reno TCP as would be expected by comparison with SACK TCP in earlier studies [13]. This is due to the presence of gap ack blocks in the SACK block which report the arrival of out of order data at the receiver. However the recovery is not optimal as the figure shows both multiple retransmissions of data (packet 18 is retransmitted 3 times, packet 20 is retransmitted twice, packet 21 is retransmitted once) and the consequent multiple halving of *cwnd* and *ssthresh* (hence entering a congestion avoidance phase). This produces slower recovery than a SACK TCP implementation and is due to the flaws in the SCTP fast retransmit procedure that were pointed out in the last section. As this recovery is slower than SACK TCP it is not very dangerous but it does include the unnecessary duplicate retransmission of data at a time of congestion. It also under utilizes the link during recovery.

No figure is shown for NoGap SCTP for this scenario as it recovers from any packet loss through a retransmission time out and consequent slow-start of the association (see Figure 4).

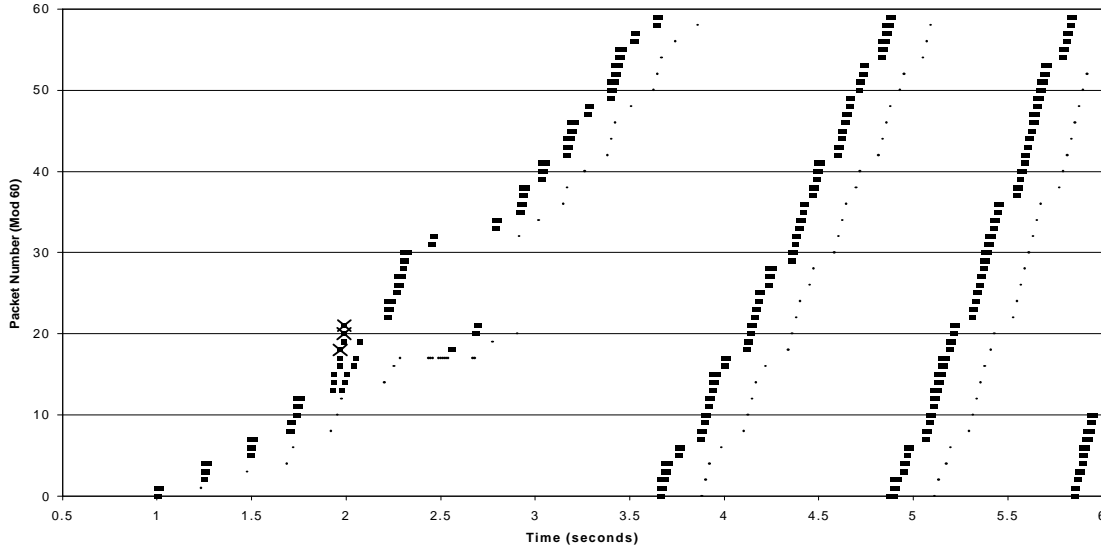


Figure 9: RefImpl SCTP with 3 packet drops

Figure 9 shows the behaviour of RefImpl SCTP for this scenario. As with the results in the last section it performs better than Standard SCTP due to the damping of SCTP's tendency to trigger the fast retransmit procedure multiple times. In this case one additional triggering causes *ssthresh* and *cwnd* to be halved unnecessarily. This causes the recovery to be sub-optimal and throughput is reduced. No duplicate retransmission is demonstrated here due to the restriction of *cwnd* on actual retransmission of data (the acknowledgement for the retransmission arrives before the flightsize reduces sufficiently to allow the packet to be retransmitted a second time). However it is still possible to get duplicate retransmission with this variant as is shown in the next section.

Figure 10 shows the behaviour of Modified SCTP for this scenario. This variant recovers much more successfully than any other SCTP variant or Reno TCP. This is due to the faster retransmission of data (and consequent reduction in the number of duplicate SACKs received) and this variant's restriction on multiple invocation of the fast retransmit procedure. No unnecessary reduction in *ssthresh* and *cwnd* takes place and hence a greater throughput is maintained than RefImpl SCTP.

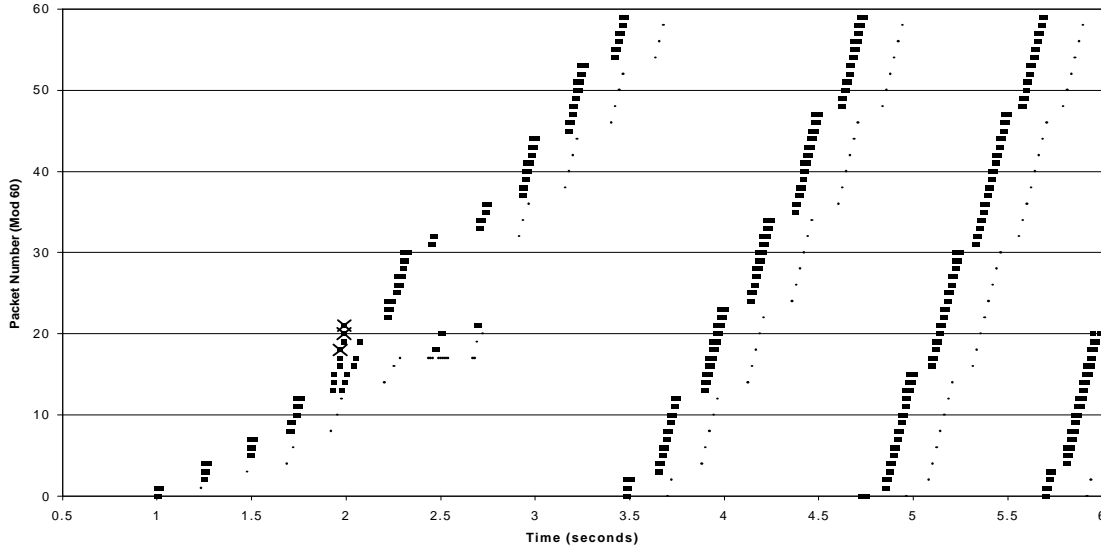


Figure 10: Modified SCTP with 3 packet drops

SCTP Bursts

The earlier scenarios demonstrated Reno TCP and SCTP variant recovery from packet loss early in the slow-start period. The window had only opened to approximately 10,000 bytes before packet loss occurred. In this section we examine the behaviour of various SCTP variants when the window is allowed to open almost to its maximum of 20,000 bytes and packet loss occurs. Without packet loss the gateway buffers are sufficiently large to barely sustain the whole transfer of data (based upon the delay-bandwidth product rule). However the behaviour of some SCTP variants when exposed to packet loss with a larger window is to produce a harmful burst of data which itself can induce one or more packet loss events. In all cases packet number 33 is dropped by the gateway due to the background flow not illustrated in the figures. Neither Reno TCP or NoGap SCTP produce any harmful behavior in this case and hence plots for their behaviour in this scenario are not shown.

Figure 11 shows the behaviour of Standard SCTP in this scenario. No burst is generated by Standard SCTP however its performance is very poor when compared with its recovery from a single packet loss earlier in the slow-start phase (Figure 3). This is due to the additional numbers of duplicate SACKs present in the network when the loss occurs. Packet 33 is retransmitted 4 times with consequent effects on *ssthresh* and *cwnd*. The effect of this single loss is a collapse of *cwnd* and *ssthresh* down to their minimum values of $2 \cdot \text{pathMTU}$. This behaviour demonstrates the increasing impact on performance of packet losses with larger window sizes for Standard SCTP (especially in the case of high bandwidth, long delay links, which are often experienced [12] on the Internet).

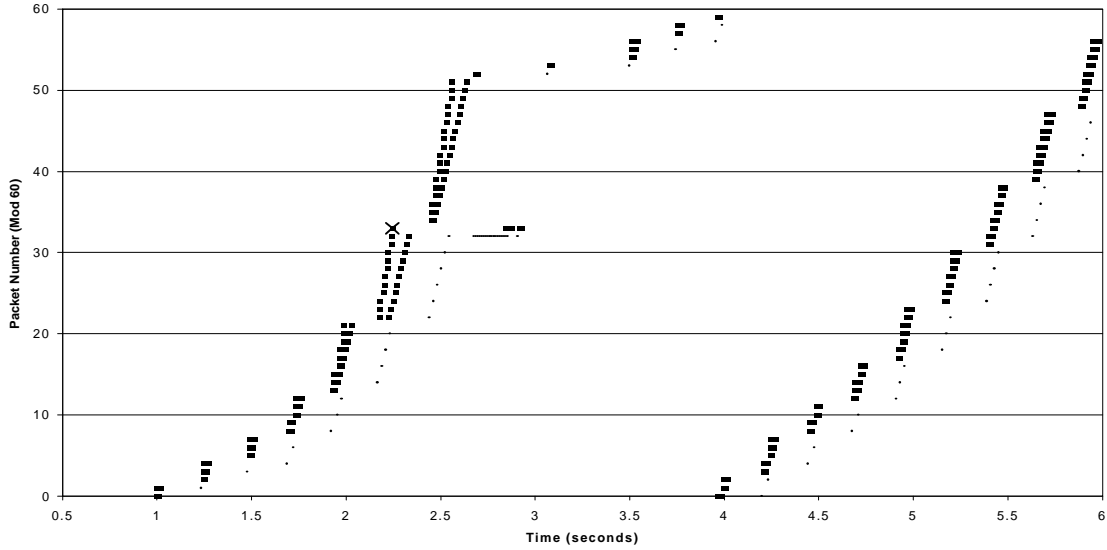


Figure 11: Standard SCTP with 1 drop for a large window

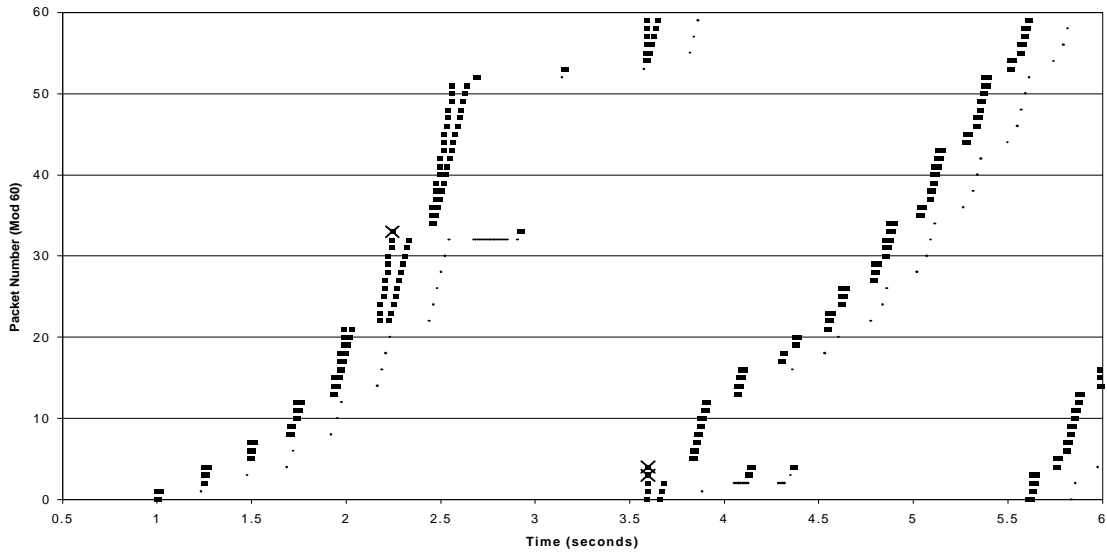


Figure 12: RefImpl SCTP with 1 packet drop and a large window

Figure 12 shows the behaviour of RefImpl SCTP in this scenario. Here we see the generation of a burst of traffic after recovery from the initial packet loss. This is due to the reduced occurrence of fast retransmit with this variant. At the time that packet 33 is lost $cwnd$ is approximately 19,000 bytes. This value is halved upon packet loss detection. By the time detection has occurred almost a full window (20,000 bytes) of data is in flight. This is all buffered at the receiver as it is out of sequence. This reduces the receiver's a_rwnd to approximately 2,000 bytes. Thus when the SACK acknowledging the retransmission is received by the sender it has a very low a_rwnd and this rather than $cwnd$ constrains transmission of new data. As can be seen in the figure only one new packet is sent in response to this SACK. However the SACK also fully acknowledged (moved the cumulative TSN ack point above) all of the out of order data previously received. This reduces the sender's view of the amount of data in the network to

approximately 1000 bytes. When a subsequent SACK arrives (the receiver sends one when it clears its inbound buffers by sending all the previously queued data to the application layer) it will have a full a_rwnd (20,000 bytes) available. The sender is now limited by $\text{Min}(cwnd, a_rwnd - \text{flightsize})$ which is approximately 10,000 bytes, the value of $cwnd$. This produces a burst of packets from the sender, which causes packets at the gateway to be dropped. One interesting thing to note about the recovery of RefImpl SCTP from these secondary losses is that they induce multiple retransmission of packet number 64. This is because even though this variant reduces the number of times fast retransmit is invoked, it still allows multiple triggering of the procedure for a given packet. This of course also unnecessarily reduces $ssthresh$ and $cwnd$ hampering recovery from the secondary losses.

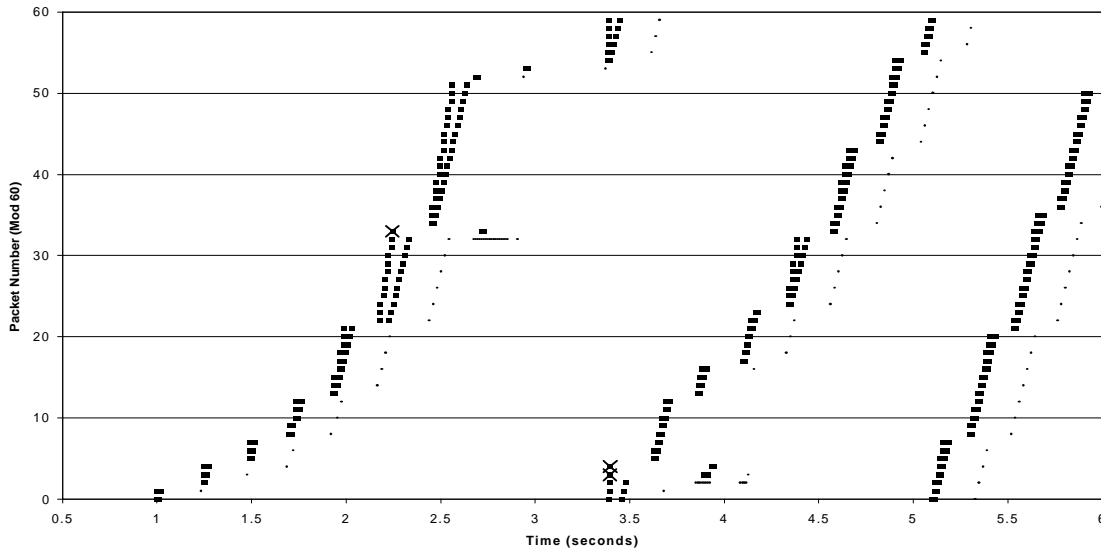


Figure 13: Modified SCTP with 1 drop and large window

Figure 13 shows the behaviour of Modified SCTP for this scenario. This variant behaves similarly to RefImpl SCTP, producing a burst in response to recovery from the initial packet loss. As with RefImpl SCTP, this occurs on reception of the SACK after the recovery SACK. Due to restrictions on multiple retransmissions no duplicate retransmission takes place and recovery is faster than RefImpl SCTP as $cwnd$ has not been unnecessarily halved. This property of Modified-SCTP makes us hesitate from recommending it to the SCTP standardisation process. However similar problems have been described with SACK TCP [13] and perhaps the use of a max_burst control variable (suggested in [13] for SACK TCP) which limited the number of packets SCTP can send in response to a given SACK arrival would be sufficient to control this behaviour. For the moment this is a topic for further research.

5. Future Work

There are many areas of the SCTP specification that have not been tested in the simple scenarios presented here. In particular the effects of multi-homing and the use of multiple streams within an association need investigation. The effects of explicit congestion notification and general SCTP interactions with random early detection (RED) active queuing mechanisms also need to be explored.

From the current work it is obvious that more investigation of SCTP burst behaviour and the resolution of a good control mechanism to prevent this behaviour is necessary. The fast retransmission of packets ignoring the value of $cwnd$ also needs further study as it might lead to overly aggressive behaviour in some scenarios.

6. Conclusions

Although much work continues to be done on TCP congestion control refinements it appears that very little has been published on SCTP. This is a dangerous state of affairs as SCTP proponents continue to position it as a replacement for SS.7 networks, which have well understood behaviour during congestion events. Perhaps this is because many proponents believe that SCTP deployment will take place in a “well engineered” private network isolated from the congestion seen on the Internet. It is the authors’ contention, and the experience of the telecommunications industry, that even in a “well engineered” environment congestion events do occur. Furthermore in order to properly engineer a network it is necessary to understand the underlying responses of protocols to congestion. The work presented here is a first step towards a better understanding of SCTP.

It has been shown that the current SCTP congestion control mechanisms have a number of flaws, particularly with respect to the fast retransmit procedure and the generation of gap ack blocks. It is our recommendation that the generation of gap ack blocks become mandatory in the specification. We have proposed that the fast retransmit algorithm be modified so that a packet that triggers fast retransmit cannot trigger the fast retransmit procedure again until after a transmission timeout event occurs. We have also demonstrated the superior performance and response time of ignoring the size of the congestion window when retransmitting a packet with the fast retransmit procedure. However we recommend for further study the effects of this modification in case it proves too aggressive. Further study is also required on the bursty behaviour of SCTP on recovery from packet loss with larger window sizes.

As a result of the work carried out for this paper, future versions of the SCTP specification will be modified according to our recommendations and the latest release (4.02) of the reference implementation has already been updated to incorporate all of our suggested changes.

Acknowledgements

The authors wish to thank Randall Stewart for his clarifications of the current SCTP specification and his work on the SCTP reference implementation.

References

- [1] R. Stewart, Q. Xie, et al., “RFC 2960: Stream Control Transmission Protocol”, The Internet Society, 2000.
- [2] L. Ong, et al., “RFC 2719: Framework Architecture for Signaling Transport”, The Internet Society, 1999.
- [3] L. Coene et al., “Telephony Signalling Transport over SCTP applicability statement”, work in progress.
- [4] L. Coene et al., “Stream Control Transmission Protocol Applicability Statement”, work in progress.
- [5] A. Jungmaier, M. Schopp, M. Tuxen, “Performance Evaluation of the Stream Control Transmission Protocol”, Proc. IEEE Conference on High Performance Switching and Routing, 2000.
- [6] M. Allman, V. Paxson, W. Stevens, “RFC 2581: TCP Congestion Control”, The Internet Society, 1999.
- [7] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “RFC 1818: TCP Selective Acknowledgement Options”, The Internet Society, 1996.
- [8] Van Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM, 1988.
- [9] M. Handley, J. Padhye, S. Floyd, “TCP Congestion Control Window Validation”, Sep. 1999.
- [10] J. Hoe, “Improving the Start-up behavior of a Congestion Control Scheme for TCP”, In ACM SIGCOMM, 1996.
- [11] M. Allman, “TCP Byte Counting Refinements”, ACM Computer Communication Review, 1999.
- [12] M. Mathis, J. Mahdavi, “Forward Acknowledgement: Refining TCP Congestion Control”, In ACM SIGCOMM, 1996.
- [13] K. Fall, S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, Computer Communications Review, 26(3), 1996.
- [14] K. Ramakrishnan, S. Floyd, “RFC 2481: A Proposal to add Explicit Congestion Notification (ECN) to IP”, The Internet Society, 1999.
- [15] M. Allman, S. Floyd, C. Partridge, “RFC 2414: Increasing TCP’s Initial Window”, The Internet Society, 1998.
- [16] Opnet home page, <http://www.opnet.com>
- [17] SCTP Reference Implementation, <http://www.sctp.refcode.org>, Cisco and Motorola, 1999-2001