SCTP and TCP Variants:
Congestion Control Under Multiple Losses*

Armando L. Caro Jr., Keyur Shah, Janardhan R. Iyengar, Paul D. Amer

Protocol Engineering Lab

Computer and Information Sciences

University of Delaware

{acaro, shah, iyengar, amer}@cis.udel.edu

Randall R. Stewart

Cisco Systems Inc.

rrs@cisco.com

Abstract

We characterize an inefficiency in the current specification of SCTP's congestion control, which degrades performance (more than necessary to be "TCP-friendly") when there are multiple packet losses in a single window. We present an SCTP variant, called *New-Reno SCTP*, which introduces three modifications. First, a Fast Recovery mechanism, similar to that of New-Reno TCP, is included to avoid multiple congestion window (cwnd) reductions in a single round-trip time. Second, we introduce a new policy which restricts the cwnd from being increased during Fast Recovery, thus ensuring that the newly introduced Fast Recovery mechanism maintains conservative behavior. Third, we modify SCTP's HTNA (Highest TSN Newly Acked) algorithm to ensure that Fast Retransmits are not unnecessarily delayed. We show that New-Reno SCTP performs better, and still conforms to AIMD principles. Also, we compare these two variants of SCTP with New-Reno TCP and SACK TCP under five different loss scenarios. Our results show that New-Reno SCTP performs significantly better than New-Reno TCP, maintains conservative behavior similar to SACK TCP, and is as robust as SACK TCP to multiple losses in a window.

1 Introduction

Until now, there have been two general purpose transport protocols widely used for applications over IP networks: UDP and TCP. Each provides a set of services that cater to certain classes of applications. However, the services provided by TCP and UDP are disjoint, and together do not ideally satisfy the needs of all network applications. The Stream Control Transmission Protocol (SCTP), designed to bridge the gap between UDP and TCP, addresses shortcomings of both.

SCTP was originally developed to carry telephony signaling messages over IP networks for telecommunications. With continued work, SCTP evolved into a general purpose transport protocol that includes advanced delivery options. Similar to TCP, SCTP provides a reliable full-duplex connection, called an *association*. However, within an SCTP association, *multistreaming* allows for independent delivery among streams, which reduces the risk of head-of-line blocking among application objects. SCTP supports *multihoming* to provide redundancy at the path level, thus increasing association survivability in the case of a network path failure. Finally, SCTP's four-way handshake for association establishment makes it resistant to SYN attacks, and hence increases overall security.

SCTP employs congestion control algorithms similar to those used in TCP. Although SCTP's congestion control mechanisms are expected to be more robust to loss and provide better performance, we discovered an inefficiency in SCTP's current mechanisms (which we refer to as *Original SCTP*[1]). Original SCTP suffers when there are multiple packet losses in a single window. Under these conditions, Original SCTP improperly reduces the *congestion window (cwnd)* for each detected loss.

We introduce an SCTP variant, called *New-Reno SCTP*, which incorporates into SCTP a Fast Recovery mechanism similar to that in New-Reno TCP. We show that New-Reno SCTP performs better than Original SCTP and TCP variants, and importantly, continues to conform to AIMD [4, 8] principles. To illustrate the advantages of New-Reno SCTP, we present simulations with detailed explanations comparing Original SCTP and New-Reno SCTP under two different loss scenarios. We also compare these two variants of SCTP with New-Reno TCP and SACK TCP under five different loss scenarios.

In Section 2 we describe the congestion control and packet retransmission algorithms for Original SCTP and New-Reno SCTP. Section 3 explains the details of the simulation environment used to produce our results. Section 4 provides detailed simulation results for Original SCTP and New-Reno SCTP under two different loss scenarios. In Section 5, we compare the performance of the two SCTP variants with New-Reno TCP and SACK TCP. Important features of these two TCP congestion control approaches are highlighted in these comparisons, but we assume the reader is familiar with RFC2581 [1] concepts, New-Reno TCP [5, 6], and SACK TCP [5, 11]. Section 6 concludes the paper.

---

[1] We refer to Original SCTP as specified in RFC2960 [15] and version 7 of the SCTP Implementer's Guide [13].

2

2 SCTP Variants

2.1 Original SCTP

SCTP is defined in RFC2960 [15] with certain changes and additions included in the SCTP Implementer's Guide [13]. SCTP uses a selective ack scheme similar to SACK TCP. SCTP's congestion control algorithms are based on RFC2581 [1], but include SACK-based mechanisms for better performance. Similar to TCP, SCTP uses three control variables: *rwnd*, *cwnd*, and *ssthresh*. However, unlike TCP, SCTP's cwnd reflects how much data can be sent, but not which data to send; thus, the cwnd in SCTP truly windows the amount of data that may be sent. SCTP also introduces an additional variable, *partial_bytes_acked (pba)*, to calculate cwnd growth during the Congestion Avoidance phase.[2]

SCTP performs Slow Start when cwnd $\leq$ ssthresh. During Slow Start, the cwnd is increased only if two conditions hold true: (1) the incoming ack advances the *cumulative ack point (cumack)*, and (2) the full cwnd was in use before the ack arrived. If so, the cwnd is incremented by the minimum of newly acked data and the MTU, where "newly acked" data includes any data not previously acked. If an ack is a duplicate ack (i.e., the ack does not advance the cumack, but includes selective acks), then any newly acked data in the selective acks reduces the amount of outstanding data. Hence, the unchanged value of cwnd can allow new data to be sent.

SCTP performs Congestion Avoidance when cwnd $>$ ssthresh. During Congestion Avoidance, the goal is to increase the cwnd by one MTU every RTT. SCTP uses pba to facilitate this mechanism. Initially, pba is set to zero. When a SACK that advances the cumack arrives, pba is incremented by the number of bytes newly acked in the SACK, as determined by the cumulative and selective ack feedback. The cwnd is increased by one MTU when an ack arrives and the following two conditions hold true: (1) pba $\geq$ cwnd, and (2) the full cwnd was in use before the ack arrived. In addition to incrementing the cwnd, pba is reset to pba $-$ cwnd. Whenever all of the data sent by the sender has been acked by the receiver, pba is reset to zero.

As in all variants of TCP, SCTP uses two mechanisms to detect loss: Fast Retransmit and Retransmission Timeout. SCTP's Fast Retransmit algorithm is slightly different from TCP's. First, SCTP's Fast Retransmit is triggered by four "missing reports" (via SACKs) instead of three "duplicate acks" as with TCP. Second, the missing reports are counted following the *HTNA* (Highest TSN Newly Acked) algorithm [13]. This algorithm handles stray packets due to reordering as follows. For each incoming ack, the highest *TSN* (Transmission Sequence Number)[3] being newly acked becomes the frame of reference for incrementing missing reports. Only TSNs prior to this reference point can have their missing report incremented with the current ack.

To avoid bursts, SCTP uses a Congestion Window Validation algorithm similar to the one described in [7]. Any time the sender has new data to send, the maxburst parameter (recommended to be 4)

---

[2]Since SCTP supports multihoming, the sender maintains cwnd, ssthresh, and pba per destination. For simplicity, we assume single-homed hosts in the rest of this paper.

[3]TSNs in SCTP are analogous to sequence numbers in TCP, with the exception that TSNs are not assigned per byte, but per data chunk. A data chunk is an individual unit of data in SCTP [15].

is first applied as follows [13]:

```
if((outstanding + maxburst*MTU) < cwnd)
      cwnd = outstanding + maxburst*MTU
```

This algorithm automatically handles scenarios where the sender's cwnd is stale due to idle or application-limited behavior which does not use the sender's full cwnd. The sender does not need to periodically adjust the cwnd for such scenarios as originally suggested in RFC2960 [15].

2.2 New-Reno SCTP

Original SCTP does not include a Fast Recovery mechanism, as found in Reno TCP and later TCP variants. According to RFC2960 [15], "because cwnd in SCTP indirectly bounds the number of outstanding TSN's, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size". The specification is correct that Fast Recovery is not needed to clock out new data packets while the sender is recovering from a Fast Retransmit. However, Original SCTP suffers when there are multiple losses in a single window of data. Each loss reduces the cwnd by half, as experienced by Reno TCP [5].[4] SCTP should avoid performing multiple cwnd reductions per window, as do New-Reno TCP and SACK TCP.

In this paper, we introduce an SCTP variant, called New-Reno SCTP, which defines a new state variable (*recover*). Analogous to the variable recover in New-Reno TCP, New-Reno SCTP stores the highest outstanding TSN to mark the end of the Fast Recovery period (which is entered on a Fast Retransmit). When a future ack acknowledges all TSNs up to and including the recover TSN, Fast Recovery is exited. During the Fast Recovery period, subsequently detected lost packets are Fast Retransmitted without reduction of the cwnd.

Original SCTP increases the cwnd for any ack that introduces a new cumack, including *partial acks*. Although the concept of a partial ack does not exist in Original SCTP, with the introduction of a recovery period, a partial ack in New-Reno SCTP is defined as in New-Reno TCP. New-Reno SCTP introduces a new policy which avoids increasing the cwnd for partial acks. Since Original SCTP had a cwnd reduction for each loss (even if they occured in the same window), this policy was not necessary. But with New-Reno SCTP, this policy is necessary to avoid increasing cwnd incorrectly during multiple loss scenarios (see Section 4.2).

SCTP's Fast Recovery mechanism differs from New-Reno TCP and SACK TCP in that subsequent losses in the same window continue to be triggered for Fast Retransmit by four missing reports. While in Fast Recovery, New-Reno TCP and SACK TCP trigger subsequent Fast Retransmits by a single missing report (or partial ack in the case of New-Reno TCP). The assumption is that once a loss is detected, missing reports in the same window are more likely due to loss rather

---

[4]Original SCTP does not suffer as badly as Reno TCP in the face of multiple losses in a window due to its SACK mechanism and different cwnd semantics. Reno TCP almost always suffers a retransmission timeout when there are three or more packet drops [5].

than reordering. However, in scenarios where loss is combined with reordering, Fast Retransmits triggered by a single missing report may cause spurious retransmissions. Hence, SCTP's Fast Recovery is more conservative, but is more robust to reordering scenarios.

New-Reno SCTP also introduces a modified HTNA algorithm for Fast Retransmit. The original HTNA algorithm sometimes delays the triggering of Fast Retransmit under multiple losses in a window (see Section 4.2). The retransmission of the first lost packet triggers an ack which advances the cumack, but does not ack the remaining lost packets. Upon the arrival of this ack, the sender recognizes the retransmitted TSN as the highest TSN newly acked at this time. The remaining lost packets have higher TSNs; thus, according to the current HTNA algorithm, the remaining lost packets will not have their missing reports incremented. However, this ack should increment the missing reports, because this ack does not correspond to a reordered packet; instead, it corresponds to a retransmitted packet. The modified HTNA algorithm solves this problem by ensuring that any time the sender is in Fast Recovery, an ack that advances the cumack is always used to increment missing reports.

In summary, New-Reno SCTP differs from Original SCTP in three ways. First, a Fast Recovery mechanism, similar to that of New-Reno TCP, is included to avoid multiple cwnd reductions in a single round-trip time. Second, we restrict the cwnd from being increased during Fast Recovery, thus ensuring that the newly introduced Fast Recovery mechanism maintains conservative behavior. Third, we modify SCTP's HTNA (Highest TSN Newly Acked) algorithm to ensure that Fast Retransmits are not unnecessarily delayed.

3   Simulations

All our simulations were done using ns-2 [2], which supports the variants of TCP in this paper. The SCTP module for ns-2, however, was developed by the Protocol Engineering Lab at the University of Delaware and is available as a third party module [3]. The simulations presented were produced using the same scripts used by Fall and Floyd in [5] with modifications to include the two SCTP variants.
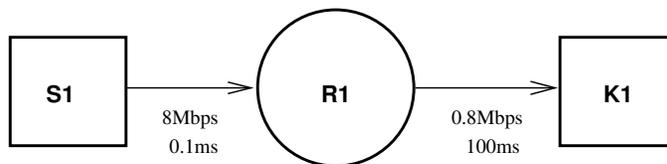

Figure 1: Simulation Topology

As shown in Figure 1, we use the same network topology used in [5]. The topology consists of a sending and receiving host (labelled S1 and K1, respectively) connected via a drop-tail router (labelled R1). The parameters of the links are indicated in the figure. Each simulation consists of only a single TCP connection or SCTP association between S1 and K1. The loss pattern in each

of the scenarios is specified explicitly using a "droplist" at the router.[5] For simplicity, and to be consistent with the results presented in [5], all simulations consist of one-way traffic and no delayed acks. Also, the link characteristics, buffer sizes, and loss patterns, as in [5], are not intended to be realistic. They provide simple scenarios which illustrate the congestion control algorithms of TCP and SCTP.

4   Original vs New-Reno SCTP

We present detailed simulations results for Original SCTP and New-Reno SCTP under two simulation scenarios: one and two drop scenarios. We choose the same loss scenarios for one and two drops used in [5]. Also, note that in practice the congestion control variables for SCTP are manipulated in bytes; our detailed analysis is done in packets (and sometimes fractions of packets) for simplicity.

We present the results by graphing the packets which enter and depart R1 over the course of the simulation. The x-axis depicts the simulation time in seconds, while the y-axis represents the packet number modulo 50 (packet numbering begins at 1). A data packet arrival is indicated on the graph as ■, whereas □ marks the departure of a data packet. Any horizontal space between these two marks represents the queuing delay a packet experiences at R1. Packets dropped upon arrival at R1 due to buffer overflow are marked with × on the graph. The acks in the reverse direction are marked with ◯ at the time they arrive at R1. Only the cumulative ack is represented on the graph; selective ack information is not conveyed explicitly and must be inferred from the graph. Also, we assume that each packet corresponds to a single TSN.

4.1   One Packet Loss

Figure 2 shows the behavior of Original SCTP and New-Reno SCTP with one dropped packet. Let us first consider Original SCTP. Packets 1-14 are sent successfully, and in the process, the cwnd increases exponentially from 1 to 15 according to the Slow Start algorithm. Since packet 15 is lost, packets 16-29 arrive at the receiver and trigger fourteen duplicate acks for packet 14. The first three duplicate acks for packet 14 decrease the number of outstanding packets by one each. Thus, packets 30-32 are clocked out.

Upon receiving the fourth duplicate ack for packet 14 (which serves as the fourth missing report for packet 15), the sender Fast Retransmits packet 15. The sender then halves the ssthresh to 7.5, and sets the cwnd to this new ssthresh.

The fifth duplicate ack for packet 14 reduces the number of outstanding packets from 15 to 14. Each additional duplicate ack for packet 14 received similarly decreases the number of outstanding packets by one. Since the number of outstanding bytes may exceed cwnd by at most $\text{MTU}-1$

---

[5]Ns-2 has a loss model which allows specific packets to be dropped at a node.

6

bytes [13], the eleventh duplicate ack reduces the outstanding packets to 7, and begins to clock out new packets once again. The last seven duplicate acks for packet 14 allow the sender to transmit packets 33-39.

Upon receiving the ack for packet 32, the cwnd and ssthresh are both 7.5, and hence, the sender is in Slow Start mode. In response to this ack for packet 32, the sender increases the cwnd to 8.5 and transmits packets 40-41. The sender then continues further transmissions in Congestion Avoidance mode.

As show in Figure 2, New-Reno SCTP shows no difference from Original SCTP with one packet drop (as expected).
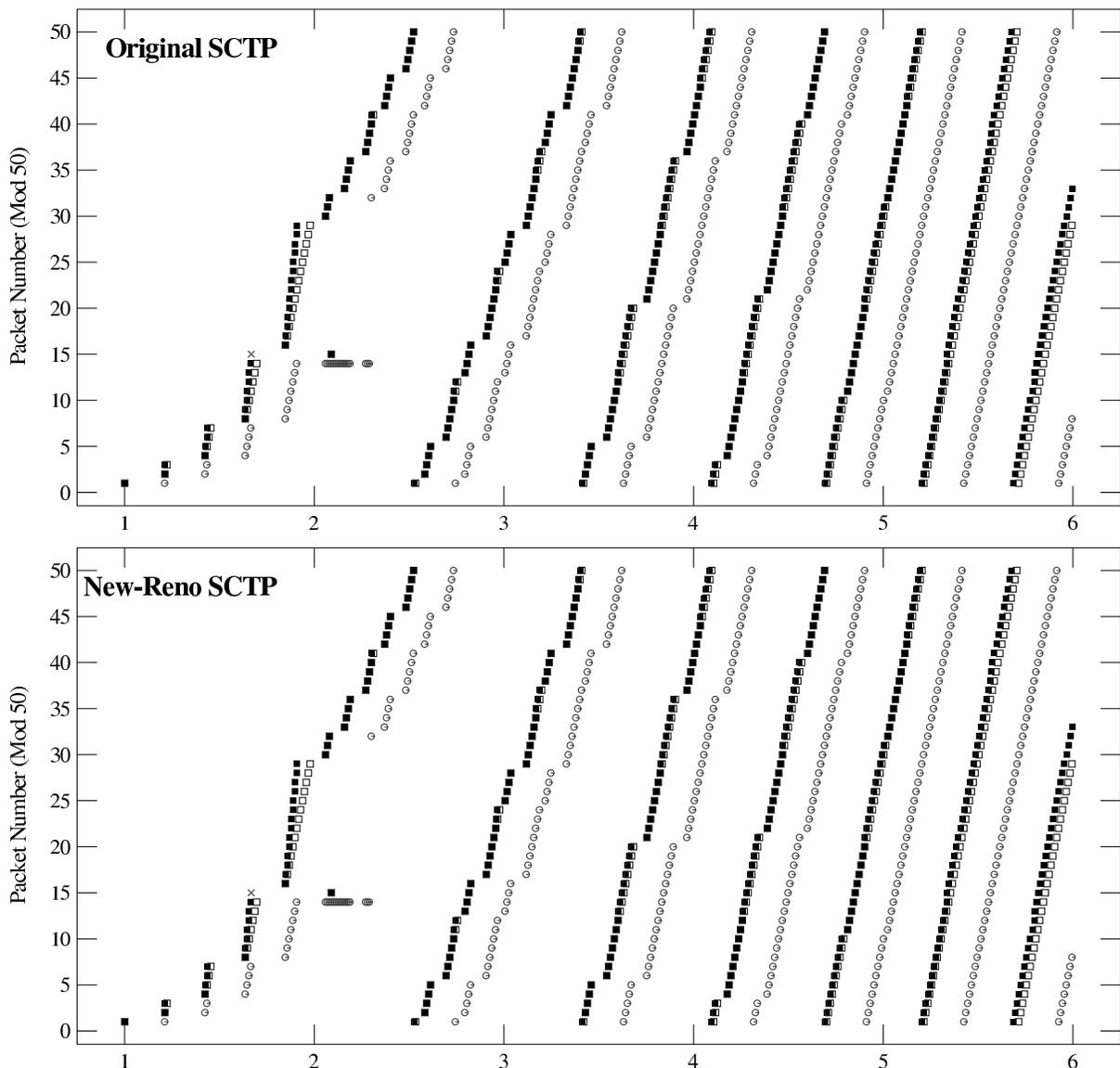

Figure 2: SCTP simulations with one dropped packet

4.2   Two Packet Losses

Figures 3 shows the behavior of Original SCTP and New-Reno SCTP with two dropped packets (packets 15 and 29). Original SCTP initially behaves the same as explained earlier with one drop, except that there is one less duplicate ack for packet 14 due to the loss of packet 29. Hence, the duplicate acks for packet 14 allow the sender to transmit only 30-38. The last three duplicate acks for packet 14 contain selective acks for packets 30-32, and hence count as missing reports for packet 29.

The next ack is for packet 28, and corresponds to the arrival of the retransmitted packet 15. Following the HTNA algorithm (see Section 2.1), this ack's highest newly acked packet is 15, which does not count as a missing report for packet 29. Here is a flaw with the current HTNA algorithm which is addressed in New-Reno SCTP. Since the sender is in Slow Start mode, the cwnd is increased to 8.5 in response to this ack. The sender now transmits packets 39-40, and moves into Congestion Avoidance mode with nine packets outstanding.

The first duplicate ack for packet 28 counts as the fourth missing report for packet 29, and triggers the Fast Retransmit of packet 29. As a result, the cwnd is reduced to 4.25.

Upon receiving the fifth duplicate ack for packet 28, the number of outstanding packets decreases to four. Thus, the sender transmits packet 41. Likewise, the next three duplicate acks for packet 28 allow the sender to transmit packets 42-44.

Finally, an ack for packet 40 arrives at the sender. Since the cwnd and ssthresh are both 4.25, the sender is in Slow Start mode and increases the cwnd to 5.25. The sender then transmits packets 45-46 and continues further transmissions in Congestion Avoidance mode.

New-Reno SCTP behaves as Original SCTP does until the first ack for packet 28 arrives at the sender. Due to the modified HTNA algorithm, this ack now triggers a Fast Retransmit of packet 29. With Original SCTP, this ack did not trigger the Fast Retransmit of packet 29 because it did not increment packet 29's missing report. Also, in contrast with Original SCTP, New-Reno SCTP does not reduce the cwnd for this Fast Retransmit since the sender is in Fast Recovery mode. Hence, the cwnd remains at 7.5. Since there are only seven outstanding packets at this point, the sender may transmit packet 39. The next six duplicate acks for packet 28 clock out packets 40-45.

The sender exits Fast Recovery mode when the ack for packet 38 arrives. The sender is in Slow Start mode at this time and increases the cwnd to 8.5. The sender then transmits packets 46-47 and continues further transmissions in Congestion Avoidance mode. Note that New-Reno SCTP recovers from the losses sooner, with a larger cwnd, and with more packets transmitted than Original SCTP. These differences are a result of correct congestion control in New-Reno SCTP, and lead to significant improvement in performance over Original SCTP (see Section 5).
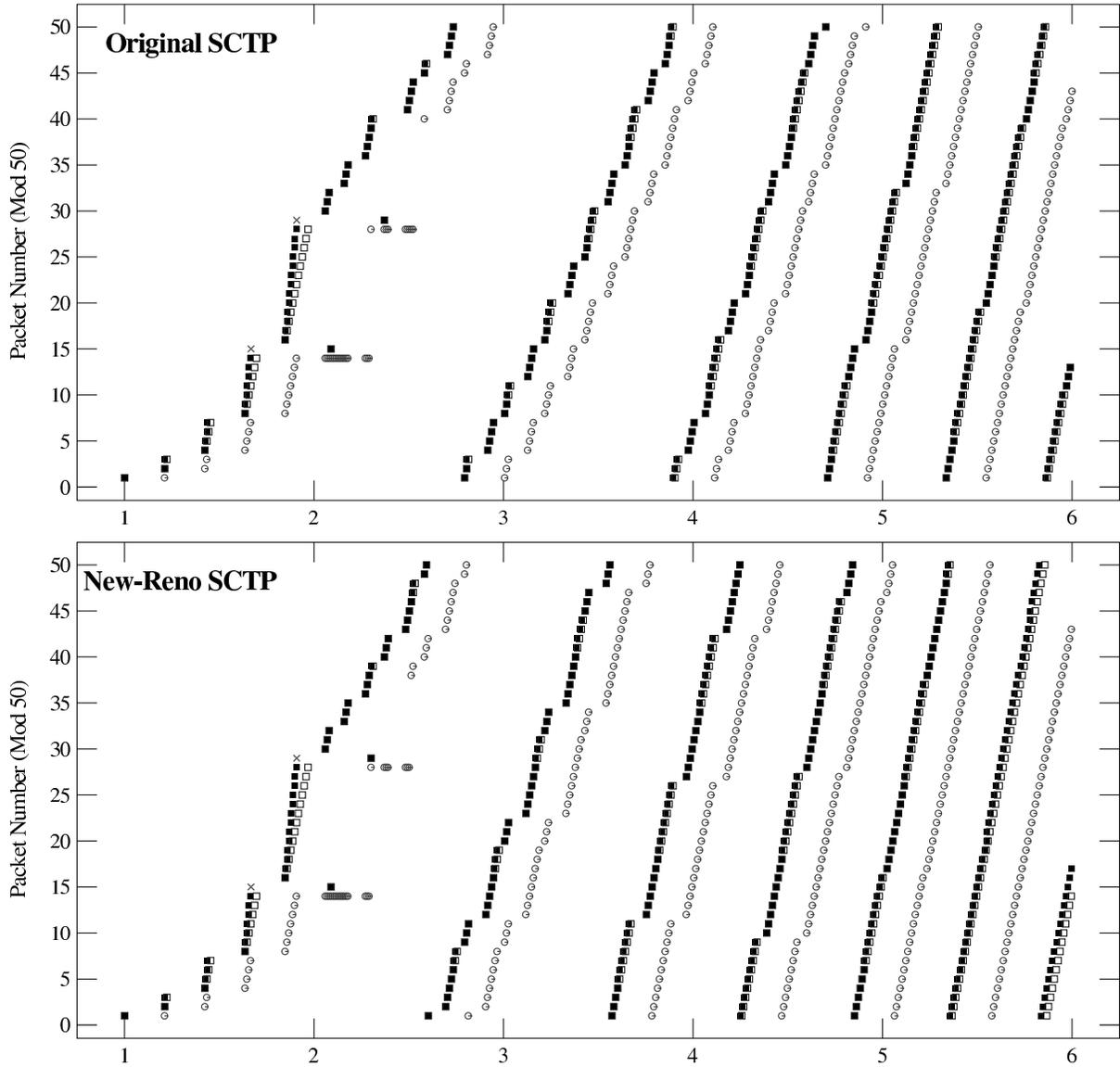
Figure 3: SCTP simulations with two dropped packets

5 Comparisons with TCP Variants

We present final simulation results for New-Reno TCP, SACK TCP, Original SCTP, and New-Reno SCTP under five different loss conditions. We compare these protocols in terms of performance and robustness to multiple losses in a single window. We include New-Reno TCP in our comparisons because it is currently the most widely deployed version of TCP [12], and it is the best performing non-SACK variant of TCP [5]. Since SCTP uses selective acks, SACK TCP [5, 11] is

also included.[6]

The ns-2 implementation of TCP performs cwnd and ssthresh reductions in whole packets, whereas the ns-2 implementation of SCTP reduces these variables in bytes. For example, if a cwnd of 15 is to be halved, TCP would set cwnd to 7, but SCTP would set it to 7.5. For the purpose of fair comparisons with the TCP variants, we modified the ns-2 implementation of SCTP to maintain the congestion control variables in whole packets as TCP does. Note that this change was not needed in Section 4's comparisons of Original SCTP and New-Reno SCTP.

We present results for five loss scenarios: zero to four dropped packets. The zero drop scenario is included to verify our ns-2 simulations accurately model that all the protocols succeed in transmitting the same number of packets when there are no losses. The other drop scenarios (shown in Figure 4) are included to compare the robustness of the protocols in handling multiple losses in a single window. These scenarios are the same as presented in [5].
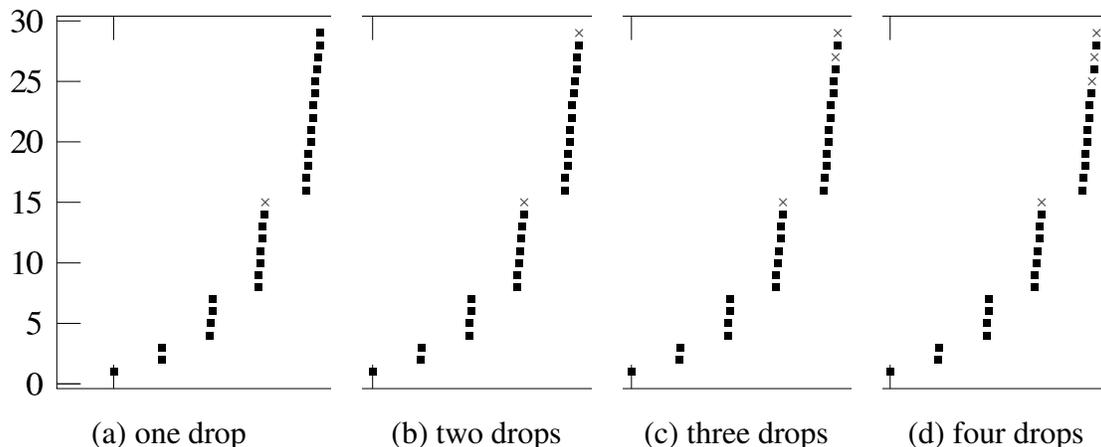

Figure 4: Drop scenarios

Figure 5 plots the number of data packets forwarded by the router (R1 in Figure 1) in six seconds of simulation for each of the five loss scenarios. Since the modifications introduced in New-Reno SCTP affect performance only when there are multiple losses in a window, New-Reno SCTP and Original SCTP perform identically for the zero and one drop scenarios. As expected, New-Reno SCTP significantly outperforms Original SCTP as the number of drops in a window increases. Also, since Original SCTP is the only protocol shown in the graph which suffers multiple cwnd reductions in a window, it performs worse than the others.

---

[6]The most recent advancements with SACK-based TCP are FACK TCP [9] and Rate-Halving TCP [10], but we exclude them in this paper for the following reasons. First, we learned through Matt Mathis (one of the Rate-Halving TCP authors) that there are a couple of issues which currently make Rate-Halving TCP unsuitable for deployment. Second, we observed that the ns-2 implementation of FACK TCP is inconsistent with [9], which states that "during [Fast Recovery], cwnd is held constant". The ns-2 implementation reduces the ssthresh to half of cwnd and then sets cwnd to 1 when entering Fast Recovery. Each subsequent ack (including duplicate acks) that arrives at the sender increments the cwnd according to Slow Start or Congestion Avoidance, depending on the values of cwnd and ssthresh. Due to this behavior, the cwnd becomes greater than ssthresh by approximately 1 packet by the time Fast Recovery is exited (based on our scenarios). Matt Mathis and Jamshid Mahdavi (the FACK TCP authors) have been unable to help us with this inconsistency.
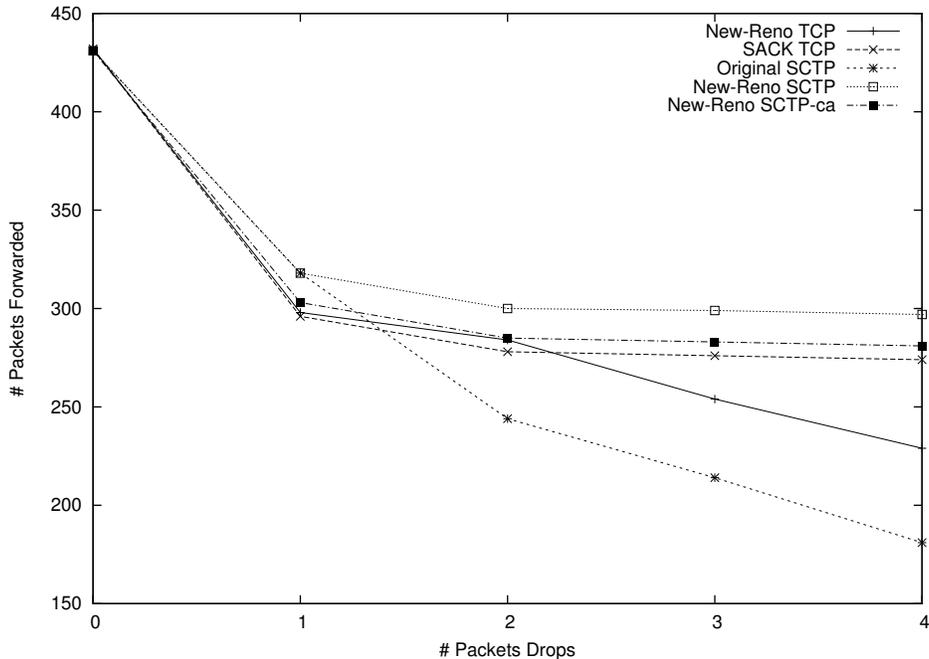
Figure 5: Simulation comparisons of protocols

Without SACK information, New-Reno TCP can retransmit at most one dropped packet per round-trip time. Hence, New-Reno TCP's performance steadily declines with additional packet losses, whereas SACK TCP and New-Reno SCTP can retransmit multiple dropped packets per round-trip, and are thus more robust to additional packet losses.

We expected New-Reno SCTP and SACK TCP to perform about the same. They exhibit nearly the same robustness to multiple drops. That is, the results in Figure 5 show that as the number of drops increase, the performance degradation of these two protocols is roughly the same. However, as Figure 5 shows, SACK does not perform as well as New-Reno SCTP.

As mentioned earlier, New-Reno SCTP performs Slow Start when cwnd = ssthresh. However, the ns-2 implementation of TCP performs Congestion Avoidance when cwnd = ssthresh.[7] To eliminate the effect of this difference while comparing with SACK TCP, we also present results for a protocol labelled *New-Reno SCTP-ca*. New-Reno SCTP-ca is equivalent to New-Reno SCTP, but differs in that Congestion Avoidance is performed when cwnd = ssthresh. New-Reno SCTP-ca is included only for the purpose of fair comparison with the TCP-variants; we are not proposing this change to SCTP.

Even with this difference eliminated, New-Reno SCTP-ca is as robust to multiple losses in a window, and slightly outperforms SACK TCP. SCTP's SACK mechanism precisely tracks the number of outstanding packets to allow the sender to continue clocking out new packets upon the arrival of duplicate acks (if cwnd allows). On the other hand, SACK TCP tracks the number of outstanding

---

[7]Note that SCTP is not being aggressive and is following acceptable congestion control as specified in RFC2581 [1], which allows TCP to perform either Slow Start or Congestion Avoidance when cwnd = ssthresh.

packets only during Fast Recovery, and not otherwise. Thus, the duplicate acks that lead up to a Fast Retransmit allow transmission of new packets in New-Reno SCTP-ca, but not in SACK TCP. This slight gain early on allows New-Reno SCTP-ca to perform slightly better than SACK TCP.

In summary, the results presented in Figure 5 show that the protocols which have both Fast Recovery and SACK mechanisms are more robust to multiple losses in a window. Note that the results presented may not represent how the protocols perform for all scenarios of a given number of packet drops. Instead, the drop scenarios are intended to illustrate the differences between congestion control algorithms of the SCTP and TCP variants.

6   Conclusion

In this paper we highlight the significant drawback of not having a Fast Recovery period in SCTP. We introduce an SCTP variant, called New-Reno SCTP, which is robust to multiple packet losses in a single window. We show that New-Reno SCTP performs signficantly better than the current specification of SCTP and New-Reno TCP. We also show that New-Reno SCTP maintains conservative behavior similar to SACK TCP, and is as robust as SACK TCP to multiple losses in a window. We recommend to SCTP the three modifications introduced with New-Reno SCTP in this paper.

7   Disclaimer

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U. S. Government.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC2581, Internet Engineering Task Force (IETF), April 1999.

[2] UC Berkeley, LBL, USC/ISI, and Xerox Parc. ns-2 documentation and software, Version 2.1b8, 2001. http://www.isi.edu/nsnam/ns.

[3] A. Caro and J. Iyengar. ns-2 SCTP module, Version 3.2, December 2002. http://pel.cis.udel.edu.

[4] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[5] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. In *ACM Computer Communications Review*, pages 5–21, July 1996.

[6] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. Experimental, RFC2582, Internet Engineering Task Force (IETF), April 1999.

[7] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. RFC2681, Internet Engineering Task Force (IETF), June 2000.

[8] V. Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM 1988*, Standford, CA, August 1988.

[9] M. Mathis and J. Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *ACM SIGCOMM 1996*, pages 281–291, Stanford, CA, August 1996.

[10] M. Mathis and J. Mahdavi. TCP Rate-Halving with Bounding Parameters. Technical report, December 1997. http://www.psc.edu/networking/papers/FACKnotes/current/.

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC2018, Internet Engineering Task Force (IETF), October 1996.

[12] J. Padhye and S. Floyd. On Inferring TCP Behavior. In *ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[13] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, and M. Tuexen. Stream Control Transmission Protocol (SCTP) Implementer's Guide. draft-ietf-tsvwg-sctpimpguide-07.txt, Internet Draft (work in progress), Internet Engineering Task Force (IETF), October 2002.

[14] R. Stewart and Q. Xie. *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison Wesley, New York, NY, 2001.

[15] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. Proposed standard, RFC2960, Internet Engineering Task Force (IETF), October 2000.