SCTP Congestion Window Overgrowth During Changeover*

Janardhan R. Iyengar, Armando L. Caro, Jr., Paul D. Amer, Gerard J. Heinz
Computer and Information Sciences
University of Delaware
{iyengar, acaro, amer, heinz}@cis.udel.edu

Randall R. Stewart
Cisco Systems Inc.
rrs@cisco.com

Abstract

Unlike TCP, the Stream Control Transmission Protocol (SCTP) supports IP multihoming at the transport layer. SCTP allows an association to span multiple local and peer IP addresses, and allows the application to dynamically change the primary destination during an active association. We present a problem in the current SCTP (RFC2960) specification that results in unnecessary retransmissions and "TCP-unfriendly" growth of the sender's congestion window during certain changeover conditions. The problem is presented in a specific case, and an algorithm we name the *Rhein algorithm* is proposed and analyzed as a possible solution.

Keywords: SCTP, Changeover, Congestion Control

*Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

1 Introduction

To provide for fault tolerance, the Stream Control Transmission Protocol (SCTP) supports multihoming at the transport layer. A host is *multihomed* if it can be addressed by multiple IP addresses [1], as would be the case when the host has multiple network interfaces. Network layer redundancy allows access to a host at a time when its primary IP address becomes unreachable; packets are rerouted dynamically to one of the host's alternate IP addresses. But since IP is connectionless, end-to-end session persistence under failure conditions becomes the responsibility of the transport layer and above. SCTP sessions, or *associations*, can dynamically span over multiple local and peer IP addresses so that an association remains alive as long as one of each endpoint's addresses remains reachable.

In an SCTP association, the sender transmits data to its peer's *primary destination address*. SCTP provides for application-initiated changeovers so that the sending application can move the outgoing traffic to another path by changing the the sender's primary destination address. In this paper, we uncover a problem in the current SCTP (RFC2960) specification [3] that results in unnecessary retransmissions and "TCP-unfriendly" growth of the sender's congestion window under certain changeover conditions. We present the problem here in a specific case. We isolate inadequacies of SCTP which cause the congestion window overgrowth problem. We then present a solution, which we name the *Rhein algorithm*, and analyze its advantages and disadvantages. In the future, we plan to investigate other possible solutions, and

plan to make recommendations for modifications to SCTP.

2 Congestion Window Overgrowth: An Example

2.1 Preliminaries

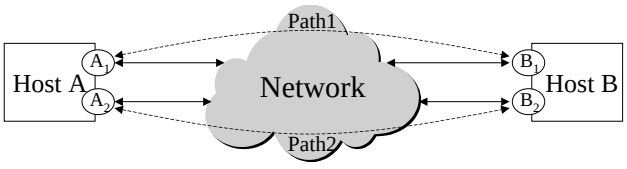

Figure 1: Architecture used in example

We present a specific example which illustrates a problem with SCTP's currently specified handling of changeover. The example uses the architecture shown in figure 1. Endpoints A and B have an SCTP association between them. Both endpoints are multihomed, A with network interfaces A_1 and A_2, and B with interfaces B_1 and B_2[1]. All four addresses are bound to the SCTP association.

For one of several possible reasons (e.g., path diversity, policy based routing, load balancing), we assume in this example that the data traffic from A to B_1 is routed through A_1, and from A to B_2 is routed through A_2. The bottleneck bandwidth of path 1 is 10Mbps, and that of path 2 is 100Mbps. The propagation delay of both paths is 200ms, and the path MTU for both paths is 1250 bytes.

Figure 2 shows a timeline of events for the described example. The vertical lines represent interfaces B_1, A_1, A_2 and B_2. The numbers along the lines represent times in milliseconds. Each arrow depicts the departure of a packet from one interface and its arrival at the destination. The labels on the arrows are either SCTP Transmission Sequence Numbers (TSN) or labels $ST_C(T_{GS} - T_{GE})$. Assuming one chunk per packet, every packet in the example corresponds to one TSN. A number represents the TSN of the chunk in the packet being transmitted. A label $ST_C(T_{GS} - T_{GE})$ represents a packet carrying a SACK chunk with cumulative ack T_C, and gap ack for TSNs T_{GS} through T_{GE}. C_1 is the *cwnd* at A for destination B_1, and C_2 is the *cwnd* at A for destination B_2. C_1 and C_2 are denoted in terms of MTUs, not bytes.

[1]More precisely, A_1, A_2, B_1 and B_2 are IP addresses associated with link layer interfaces. Here we assume exactly one address per interface, so address and interface are used interchangeably.

2.2 Example Timeline Description

The sender (host A) initially sends to the receiver (host B) using primary destination address B_1. This setting causes packets to leave through A_1. Assume these packets leave the transport/network layers, and get buffered at A's link layer A_1, whereupon they get transmitted according to the channel's availability. This initial condition is depicted in figure 2 at time $t = 0$, when in this example A has 50 packets buffered on interface A_1.

At $t = 1$, as TSNs 1 - 50 are being transmitted through A_1, the sender's application changes the primary destination to B_2, that is, any new data from the sender is sent to B_2. In the example, we assume C_2 = 2 at the moment of changeover and TSN 51 is transmitted to the new primary at $t = 1$. We refer to this event as the *changeover time*. This new primary destination causes new TSNs to leave the sender through A_2. Concurrently, the packets buffered earlier at A_1 are still being transmitted. Previous packets sent through A_1, and the packets sent through A_2, can arrive at the receiver B in an interleaved fashion on interfaces B_1 and B_2 respectively. In figure 2, TSNs 1, 51, 52 and 2 arrive at times 21, 21.1, 21.2, 22, respectively. This reordering is introduced as a result of changeover.

The receiver starts reporting gaps as soon as it notices reordering. If the receiver communicates four missing reports to the sender before all of the original transmissions (TSNs 1 - 50) have been acked, the sender will start retransmitting the unacked TSNs. SCTP's Fast Retransmit algorithm [3] is based on TCP's Fast Retransmit algorithm [2], with the additional use of selective acks and a modification to handle some cases of reordering[2]. Accordingly, the

[2] [4] is an Internet Draft which goes hand-in-hand with RFC2960. The Implementor's guide maintains all changes and additions to be included in RFC2960's next version. All imple-

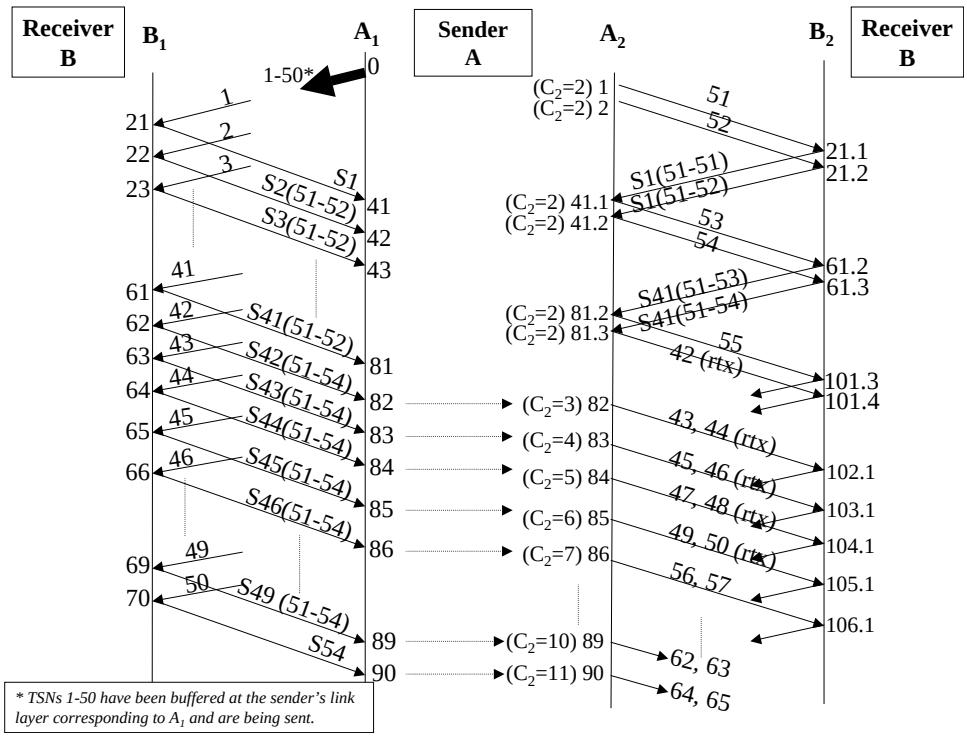

Figure 2: Timeline for the example

SACKs resulting from the receipt of TSNs 51-54 will be the only ones generating missing reports. The SACKs received by A on A_2 at $t = 41.1$ and $t = 41.2$ will be considered as the first and second missing reports for TSNs 2 - 50. Since these SACKs do not carry new cumulative acks, they do not cause growth of C_2. Between $t = 42$ and $t = 81$, the cumulative ack in the SACKs received by A on A_1 increases as a consequence of the original transmissions to destination B_1 reaching B. In this period, A receives 40 SACKs which incrementally carry cumulative acks of 2 - 41.

The SACKs received by A on A_2 at $t = 81.2$ and $t = 81.3$ carry a cumulative ack of 41, and are considered as the third and the fourth missing reports for TSNs 42 - 50. On the fourth missing report, A retransmits only TSN 42, since C_2 permits only one more packet to be outstanding. At $t = 82$, the SACK for the original transmission of TSN 42 reaches A on A_1. Since the sender cannot distinguish between SACKs generated by transmissions from SACKs generated by retransmissions, this SACK incorrectly acks the retransmission of TSN 42, thereby increasing C_2 by one, reducing the amount of data outstanding on destination B_2, and triggering the retransmission of TSNs 43 and 44. At $t = 83$, the SACK for the original transmission of TSN 43 arrives at A on A_1. As before, this SACK acks the retransmission (of TSN 43), further incorrectly increasing C_2, and triggering retransmission of TSNs 45 and 46. This behaviour of SACKs for original transmissions incorrectly acking retransmissions continues until the SACKs of all the original transmissions to B_1 (up to TSN 50) are received by A. Thus, the SACKs from the original transmissions cause C_2 to grow (possibly drastically) from wrong interpretation of the feedback.

2.3 Discussion

The values chosen in our example illustrate but a single case of the congestion window overgrowth problem. Our preliminary investigation shows that the problem occurs for a range of {propagation delay, bandwidth, MTU} settings. For example, with both paths having RTTs of 200ms (bandwidth = 100Kbps, propagation delay = 40ms) and MTU = 1500 bytes, the incorrect retransmission starts much earlier (at

mentations are expected to carry the specifications and modifications in this guide.

TSN 3), and the *cwnd* overgrowth is even more dramatic.

The congestion window overgrowth problem exists even if the buffering occurs not at the sender's link layer, but in a router along the path (in figure 1, path 1). In essence, the transport layers at the endpoints can be thought of as the sending and receiving entities, and the buffering could potentially be distributed anywhere along the end-to-end path.

3 Congestion Window Overgrowth: A Proposed Solution

The TCP-unfriendly *cwnd* growth and unnecessary retransmissions during changeover can be seen to occur due to inadequacies of SCTP - either (i) the sender is unable to distinguish between SACKs for transmissions and SACKs for retransmissions, or (ii) the congestion control algorithm at the sender is unaware of the occurrence of a changeover, and hence is unable to identify reordering introduced due to changeover. Addressing either of these inadequacies will solve the more important problem of TCP-unfriendly *cwnd* growth. We propose the Rhein algorithm that will prevent the incorrect *cwnd* growth by enabling the sender to distinguish between SACKs for transmissions and SACKs for retransmissions.

3.1 The Rhein Algorithm

The *Eifel algorithm* [5] uses meta information in the TCP header to disambiguate acks for transmissions from acks for retransmissions, thereby improving the throughput of a TCP connection. Based on the extra header information, in [5], the TCP sender can make a more accurate RTT estimate and curb unnecessary *cwnd* reduction due to spurious retransmits. We propose the *Rhein algorithm* which, like the Eifel algorithm, adds meta information to the SCTP packet to disambiguate the acks. The Rhein algorithm uses the meta information in the SCTP packet to curb unnecessary *cwnd* growth due to spurious retransmits.

As in the Eifel algorithm, we initially considered adding one bit per SCTP packet to distinguish between an original transmission and later retransmissions. This bit would be echoed by the receiver in the corresponding SACK, allowing the sender to distinguish between a SACK for an original transmission vs. a SACK for a retransmission. But since SCTP concatenates TSNs in packets, and rebundling on retransmissions may result in a different combination of TSNs in packets, a single bit per packet is insufficient.

Therefore we propose the addition of two new chunks called the *Retransmission Identifier (RTID) Chunk* and the *Retransmission Identifier (RTID) Echo*. The RTID chunk is added to every outgoing data packet at the sender, and carries one bit per TSN in the packet. The bit is 0 if its respective TSN is a first transmission, and is 1 if the TSN is a retransmission. The receiver echoes back these bits in the RTID echo chunk in the SACK. In the case of delayed SACKs, the receiver stores this chunk from the packet for which the SACK is delayed. When the delayed SACK is sent, the RTID chunks from the packets being SACKed are merged and echoed as one RTID echo.

When sending a packet, the sender (i) locally maintains the latest RTID information for each TSN in the packet, and (ii) maintains the number of retransmissions of each TSN. On the receipt of a SACK, the sender compares the corresponding bit for each TSN in the RTID echo with the corresponding local RTID bit. If the bits match, the corresponding destination's *cwnd* is increased accordingly. Otherwise, none of the *cwnd*s are increased.

With one bit per TSN, the protocol can distinguish between the transmission and the first retransmission of a TSN. If the retransmission count increases beyond one, to avoid TCP-unfriendly *cwnd* growth due to ambiguity in the SACKs again, we apply *Karn's algorithm* to *cwnd* growth. In other words, a TSN which has be retransmitted more than once will not cause any *cwnd* growth.

3.2 Discussion

Though it does not solve the problem of unnecessary retransmissions, the Rhein algorithm solves the problem of TCP-unfriendly *cwnd* growth. The RTID chunk and the RTID echo can also be used to implement the Eifel algorithm to improve SCTP performance in the face of spurious retransmits. Using the

Eifel algorithm would help SCTP improve throughput in situations where timeouts are likely to occur.

The Rhein algorithm is backwards compatible in that a receiver not recognizing the RTID chunk will not hinder RFC2960 behaviour. If the receiver does not recognize the RTID chunk, it will respond with an OPERATIONAL ERROR chunk (*Unrecognized Chunk Type* error) [3]. The sender can attempt to infer information about the SACK when such an OPERATIONAL ERROR is received. If the receiver responds with the OPERATIONAL ERROR chunk when it receives an RTID chunk, the sender can conservatively fall back on applying Karn's algorithm to *cwnd* growth from the first retransmission itself.

There are disadvantages to using the Rhein algorithm. Every packet has to carry an additional RTID chunk, and every SACK has to carry an RTID echo. Additional complexity is also introduced at the sender and receiver for processing these new chunks.

4 Discussion and Conclusion

The general consensus at the IETF has been to dissuade the usage of SCTP's multihoming feature for simultaneous data transfer to the multiple destination addresses, largely due to insufficient research in the area. Though there is some amount of simultaneous data transfer in the described scenarios, this phenomenon is an effect of changing the primary destination. Among other reasons, this changeover could be initiated by an application searching for a better path to the peer host for a long session, or attempting to perform a more efficient failover.

We have proposed a solution to curb the TCP unfriendly *cwnd* growth observed during changeover. The Rhein algorithm recognizes that this growth occurs due to the sender's inadequacy in distinguishing between the SACKs for original transmissions and the SACKs for retransmissions. This algorithm has the drawbacks though, that it does not solve the problem of unnecessary fast retransmissions on a changeover, and adds the overhead of an extra chunk for every SCTP packet.

We are currently investigating the possibility of introducing changeover awareness at the SCTP sender, to prevent the unnecessary fast retransmissions. After exhaustively analyzing the possible solutions, we plan to make recommendations for modifications to SCTP. We also plan to investigate the performance of the solutions through simulations using the SCTP ns-2 module developed at the University of Delaware [6].

5 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.

6 Acknowledgments

Many thanks to Ivan Arias-Rodriguez and Phillip Conrad for their comments and inputs. Ivan helped in the development of the described example.

7 References

[1] R. Braden. Requirements for internet hosts–communication layers. RFC1122, Internet Engineering Task Force (IETF), October 1989.

[2] M. Allman et al. TCP Congestion Control. RFC2581, Internet Engineering Task Force (IETF), April 1999.

[3] R. Stewart et al. Stream Control Transmission Protocol. Proposed standard, RFC2960, Internet Engineering Task Force (IETF), October 2000.

[4] R. Stewart et al. SCTP Implementors Guide. Internet Draft: draft-ietf-tsvwg-sctpimpguide-04.txt, Internet Engineering Task Force (IETF), March 2002. *(work in progress)*.

[5] R. Ludwig and R. H. Katz. The Eifel Algorithm: Making TCP Robust against Spurious Retransmissions. *ACM Computer Communication Review*, Vol. 30(1), January 2000.

[6] Protocol Engineering Lab, University of Delaware. SCTP Module for ns-2. http://pel.cis.udel.edu.