TRANSPORT LAYER LOAD BALANCING IN FCS NETWORKS[*]

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

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

ABSTRACT

The Stream Control Transmission Protocol (SCTP) allows for end-to-end load balancing in FCS networks to be performed at the transport layer, through repeated changeover. 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. We first illustrate the problem using an example scenario. We then briefly describe the proposed solutions for problem and our future direction.

1 INTRODUCTION

A node is *multihomed* if it can be addressed by multiple IP addresses [3], as would be the case when the host has multiple network interfaces. Network layer redundancy allows access to a host even if one of its IP addresses becomes unreachable; ideally packets can be rerouted to one of the host's alternate IP addresses. However, since IP is connectionless, end-to-end session persistence under failure conditions becomes the responsibility of the transport layer and above. To provide for such fault tolerance, the Stream Control Transmission Protocol (SCTP) supports multihoming at the transport layer. SCTP sessions, or *associations*, can dynamically span over multiple local and peer IP addresses so that an association can remain alive even if one of the endpoints' addresses becomes unreachable. Multihoming also allows for end-to-end load balancing to be performed at the transport layer. Bearing in mind that all resources available should be optimally used in FCS networks, our investigation focusses on utilizing all network resources visible at the transport layer.

SCTP [9] is a recent standards track transport layer protocol in the Internet Engineering Task Force (IETF). Of the salient features that distinguish SCTP from TCP, we concern ourselves with *multihoming*. SCTP multihoming allows binding of one transport layer association to multiple IP addresses. This binding allows an SCTP sender to send data to a multihomed receiver through different destination addresses. For instance, in figure 1, A could send data to B using destination address B_1 or B_2. SCTP's multihoming feature was motivated by fault tolerance; if one destination address becomes unreachable, the destination can still send and receive via other interfaces bound to the association.

In a multihomed 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 change the sender's primary destination address, thus moving the outgoing traffic to a potentially different path[1]. This feature can be used to perform end-to-end load balancing using SCTP, thus helping FCS network elements exploit all network resources visible at the transport layer. We uncovered a problem in the current SCTP (RFC2960) specification [9] that results in unnecessary retransmissions and "TCP-unfriendly" growth of the sender's congestion window under certain changeover conditions.

In section 2, we present a specific example which illustrates the problem of *cwnd* overgrowth with SCTP's currently

[*]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]SCTP was designed as a transport protocol for telephony signaling in SS7 networks. In an SS7 network the upper layers can dictate to which destination address packets will be sent, motivating the application-initiated changeover feature in SCTP.

specified handling of changeover. Section 3 presents two *changeover aware congestion control* algorithms: *Conservative CACC (C-CACC)* and *Split Fast Retransmit CACC (SFR-CACC)*, and the *Rhein* algorithm. In light of the bigger problem of end-to-end load balancing, we conclude with questions which describe our future direction in section 4.

2 CONGESTION WINDOW OVERGROWTH

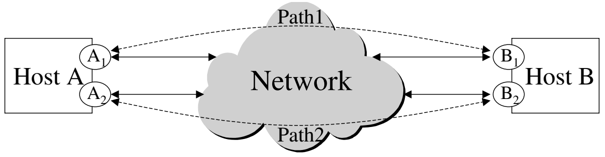


Figure 1: Architecture used in example

In this section, we present an example illustrating the occurence of *cwnd* mishandling and unnecessary retransmissions 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[2]. All four addresses are bound to the one SCTP association. For 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 locally routed through A_1, and from A to B_2 through A_2. Non overlapping paths are assumed with the bottleneck bandwidth of path 1 being 10Mbps, and that of path 2 being 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 our 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 of the form $ST_C(T_{GS} - T_{GE})$. 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.

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

The assignments such as initial TSN = 1 and initial time $t = 0$ are arbitrary assignments to signify the beginning of the snapshot. These assignments are not meant to imply the beginning of the association. Initially, $C_2 = 2$ because we assume that either there has been no transmission to B_2 before $t = 0$ during the lifetime of the association, or C_2 has decayed[3] to two $MTUs$ by $t = 0$.

2.1 Example Description

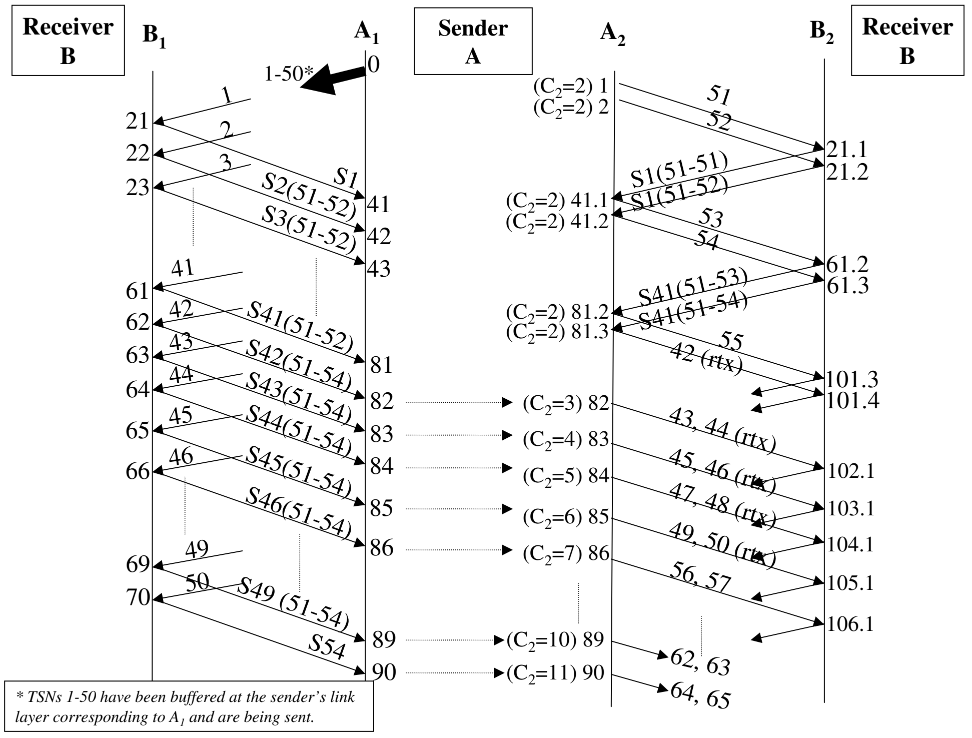
The sender (A) initially sends to the receiver (B) using primary destination address B_1. This setting causes packets to leave through A_1. Assume these packets leave the transport/network layers, and get buffered at A's link layer A_1, whereupon they get transmitted according to the channel's availability. 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, thus requiring any new data from A to be sent to B_2. In the example, we assume TSN 51 is transmitted to the new primary at $t = 1$. We refer to this moment 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 specific times depend on the delays of paths 1 and 2.

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 is based on TCP's Fast Retransmit algorithm [4], with the additional use of selective acks and a modification to handle some cases of reordering[4]. Accordingly, the 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

[3]The *cwnd* for a destination address decays exponentially if no data is transmitted to that destination address [9].

[4] [7] goes hand-in-hand with RFC2960. The Implementor's guide maintains all changes and additions to be included in RFC2960's next version. All implementations are expected to carry the specifications and modifications in this guide.



Receiver B
B1
A1
Sender A
A2
B2
Receiver B
1-50*
* TSNs 1-50 have been buffered at the sender's link layer corresponding to A1 and are being sent.

Figure 2: Timeline for the example

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 the third and the fourth missing reports for TSNs 42 - 50. Upon the fourth missing report, A retransmits only TSN 42, since C_2 permits only one more packet to be outstanding. Note that this falsely triggered retransmission leads to an unnecessary reduction of the C_1 by half, since the sender infers congestion from A_1 to B_1. 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.

2.2 Discussion

While 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 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. Further, the reduction in C_1 causes the sender to reduce its sending rate on path 1 unnecessarily. In our preliminary investigation of load balancing, we have observed multiple occurrences of such false retransmissions. Such false retransmissions cause the sending rate to reduce drastically on path 1, resulting in suboptimal utilization of the path.

3 PROPOSED SOLUTIONS

The TCP-unfriendly *cwnd* growth and incorrect retransmissions during changeover occur due to a current inadequacy of SCTP - either (i) the sender is unable to distinguish SACKs for transmissions from SACKs for retransmissions, or (ii) the sender's congestion control mechanism 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 problems of TCP-unfriendly *cwnd* growth and unnecessary retransmissions.

The *Rhein Algorithm* [6] solves the problems by addressing (i). The Rhein algorithm is based on the *Eifel* algorithm, which uses meta information in the TCP header. This meta information is used in disambiguating acks for transmissions from acks for retransmissions to improve the throughput of a TCP connection. The Rhein algorithm uses meta information in the SCTP header to curb the unnecessary *cwnd* growth and reduction due to spurious retransmissions. In our initial conception of the Rhein algorithm, each data packet has to carry an extra *Retransmission Identifier (RTID) Chunk*, and each SACK has to carry an extra *RTID Echo*. Additional complexity is also introduced at the sender and receiver for processing these new chunks.

The *Changeover aware congestion control (CACC)* algorithms solve the problem by addressing (ii). In other words, the CACC solutions introduce *changeover awareness* in the sender's congestion control mechanism. The *cwnd* overgrowth occurs due to the sender misinterpreting SACK feedback, and incorrectly sending fast retransmissions. CACC algorithms curb the *cwnd* miscalculations by eliminating these improper fast retransmissions. The key in a CACC algorithm is maintaining state at the sender for each destination when changeover happens. On receipt of a SACK, the sender selectively increases the missing report count for TSNs in the retransmission list, thus preventing incorrect fast retransmissions. [5] describes two CACC algorithms: *Conservative CACC (C-CACC)* and *Split Fast Retransmit CACC (SFR-CACC)*. The C-CACC algorithm has the disadvantage that in the face of loss, a significant number of TSNs could potentially wait for a retransmission timeout when they could have been fast retransmitted. The SFR-CACC algorithm alleviates this disadvantage. [5] provides the details of the CACC algorithms, including verification of the effectiveness of the SFR-CACC algorithm through ns-2 simulations.

4 ANALYSIS AND FUTURE WORK

Results from [5] suggest that the problem presented in Section 2 might not be a "corner case." By approaching the problem from different perspectives, the Rhein algorithm and the CACC algorithms all solve the problem of *cwnd* overgrowth. The Rhein algorithm recognizes that this growth occurs due to the sender's inability to distinguish between SACKs for original transmissions from SACKs for retransmissions. This algorithm does not solve the problem of unnecessary fast retransmissions on a changeover. This algorithm also adds the overhead of an extra chunk for every SCTP packet.

The CACC algorithms maintain state information during a changeover, and use this information to avoid incorrect fast retransmissions. These algorithms have the added advantage that no extra bits are added to any packets, and thus the load on the wire and network is not increased. One disadvantage of the CACC algorithms is that some of the TSNs on the old primary are ineligible for fast retransmit. Furthermore, complexity is added at the sender to maintain and use the added state variables.

We have implemented SFR-CACC in the NetBSD/FreeBSD release for the KAME stack [1, 2]. The implementation uses three flags and one TSN marker for each destination, as described in [5]. Approximately twenty lines of *C* code were needed to facilitate the SFR-CACC algorithm, most of which will be executed only when a changeover is performed in an association.

In light of the enveloping issue of end-to-end load balancing, we plan to research the following questions in the future:

- How well do the Rhein and CACC algorithms perform during *cycling* changeovers? A changeover where the sender repeatedly cycles through the destination address space while sending data is called a cycling changeover.
- In an FCS network wireless environment where paths have frequent disruptions, would load balancing improve or degrade overall performance?
- Given the path parameters for all paths between the sender and the receiver, is it possible for the sender to send data out of order such that the receiver receives all data in order? Is it possible for the sender to do the same while probing for path information at the same time? This line of thought leads to efficient load balancing for realtime and/or multimedia transfers.
- What is the effect of performing shared/separate congestion control at the sender among various paths to

the receiver when the bottlenecks along the paths are shared/separate? This question arises from an inquiry into the effects of shared/separate bottlenecks on SCTP congestion control. This study is important for SCTP to be TCP-friendly in sending data.

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

Thanks to Ivan Arias Rodriguez, Vern Paxson, Mark Allman, Phillip Conrad and Johan Garcia for their comments and inputs. Ivan helped in the formulation of the problem.

REFERENCES

[1] The SCTP Homepage. http://www.sctp.org.

[2] Webpage of the KAME project. http://www.kame.net.

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

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

[5] Janardhan R. Iyengar, Armando L. Caro Jr., Paul D. Amer, Gerard J. Heinz, and Randall Stewart. Making SCTP More Robust To Changeover. Technical Report TR 2002-03-01, CIS Department, University of Delaware, July 2002.

[6] Janardhan R. Iyengar, Armando L. Caro Jr., Paul D. Amer, Gerard J. Heinz, and Randall Stewart. SCTP Congestion Window Overgrowth During Changeover. Proc. SCI2002, Orlando, July 2002.

[7] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, and M. Tuexen. SCTP Implementers Guide. Internet Draft: draft-ietf-tsvwg-sctpimpguide-06.txt, Internet Engineering Task Force (IETF), May 2002. *(work in progress)*.

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

[9] 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.