Making SCTP More Robust to Changeover*

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

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

ABSTRACT

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. To gain insight into the ambient conditions under which *cwnd* overgrowth can be observed, we present an analytical model of this problem. As solutions, we then propose two *changeover aware congestion control (CACC)* algorithms which incorporate *changeover awareness* in SCTP's congestion control mechanism: *Conservative CACC (C-CACC)*, and *Split Fast Retransmit CACC (SFR-CACC)*. Using ns-2 simulations, we validate the model and evaluate the recommended solution. Based on the analysis, we make recommendations for modifications to SCTP.

Keywords: SCTP, Changeover, Multihoming, Reordering, Congestion Control, Transport Protocols

1 INTRODUCTION

A node is *multihomed* if it can be addressed by multiple IP addresses [5], 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.

SCTP [13] 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]. We uncovered a problem in the current SCTP (RFC2960) specification [13] that results in unnecessary retransmissions and "TCP-unfriendly" growth of the sender's congestion window under certain changeover conditions.

* 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.

We wish to point out that the problem of unnecessary fast retransmits observed is applicable to TCP as well, under reordering of traffic by the network. Such reordering has been known to occur, and there's been work done in the area [4, 7, 14]. But, under a single association, SCTP has the unique feature of multihoming which allows multiple congestion windows to co-exist. Such a feature has not been known to TCP, and hence the problem of congestion window overgrowth is unique to SCTP. Nevertheless, any transport layer protocol equipped with multihoming awareness would probably observe the described problems. Though the solutions as described in Section 4 are specific to SCTP, they also indicate that such multihoming aware transport protocols should incorporate *changeover awareness* in their congestion control algorithms.

In [8], we present a specific example which illustrates the problem of *cwnd* overgrowth with SCTP's currently specified handling of changeover. In this paper, we generalize the problem and develop an analytical model in Section 2. The model abstractly quantifies the *cwnd* overgrowth under various network and changeover conditions. This model provides insight into the ambient conditions under which *cwnd* overgrowth can be observed. Based on the model, we present some results estimating conditions for *cwnd* overgrowth in Section 3. Due to the fact that transport layer multihoming is not a current practice, it is extremely difficult to use any empirical data to reinforce the importance of the observed congestion window overgrowth. Hence, we use analytical results in Section 3 to suggest that the problem might not be a "corner case". Section 4 presents two *changeover aware congestion control* algorithms as solutions: *Conservative CACC (C-CACC)* and *Split Fast Retransmit CACC (SFR-CACC)*. By approaching the problem from different perspectives, the Rhein algorithm (described in a previous work [8]) and the CACC algorithms (Section 4) all solve the problem of TCP-unfriendly *cwnd* growth. After analyzing their advantages and disadvantages in Section 5, we recommend the addition of the SFR-CACC algorithm to SCTP.

2 CONGESTION WINDOW OVERGROWTH: A GENERAL MODEL

In this section, we generalize the scenario described in [8]. This model abstractly quantifies the *cwnd* overgrowth and the number of unnecessary retransmissions caused by the changeover under various network conditions. In Section 2.1, we present a generalized timeline of SCTP behaviour during changeover. We then derive analytic results from this general model in Section 2.2, followed by a discussion of these results in section 2.3.

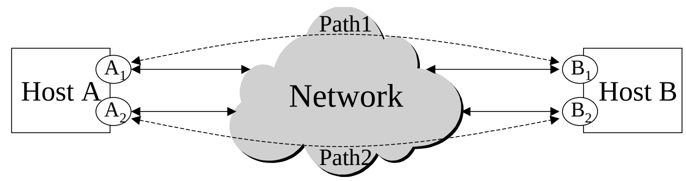

Figure 1: Architecture used in example

The general model 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 model that the data traffic from A to B_1 is locally routed through A_1, and from A to B_2 through A_2.

2.1 Model Description

We now present a generalized timeline of SCTP behaviour during changeover in figure 2. This timeline is an excerpt from an association and is based on the example scenario described in [8] . The vertical lines in the timeline represent interfaces B_1, A_1, A_2 and B_2. The numbers along the lines represent time periods or moments. 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 $ST_C(T_{GS} - T_{GE})$. SCTP transmits data and control information in transport layer entities called *chunks*. Each DATA chunk carries a unique TSN, as against the sequence numbering scheme in TCP, which assigns a sequence number per byte. 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. SCTP uses cumulative acks and selective acks in acknowledgments, where the selective acks indicate the TSNs received out of order. Such selective acks in SCTP, which are sent in SACK chunks, are called *gap acks*. 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.

Some parameters used in the model are described below. The rest of the notation is described in Section 2.2.

- C_1, C_2 : Congestion windows at A for B_1 and B_2, respectively
- t_c : Changeover time - Moment after a changeover when sender A starts sending packets to new primary destination B_2
- t_2 : Time when fast retransmission (incorrectly) starts.
- G_1 : Number of Transmission Sequence Numbers (TSNs) sent in initial group transmitted to destination B_1 in the time interval $\{0, \ t_c\}$.
- $K+1$: First TSN to be fast retransmitted (incorrectly) by A.

At $t = 0$, host A starts to transmit G_1 TSNs (TSN 1 through G_1) to destination address B_1. By time t_c the transport layer at host A has G_1 TSNs outstanding. This group of TSNs (1 through G_1) is referred to as the *initial group*. Note that these TSNs are outstanding at the transport entity at host A and could be buffered anywhere along the end-to-end path, even at interface A_1. By time t_c, A has changed its primary destination to B_2. At the instant $t = t_c$, A starts transmitting new data to B_2 through interface A_2. t_c can also be thought of as the time elapsed from the transmission of the first outstanding TSN on destination B_1 to the time of transmission of the first TSN on destination B_2 after changeover. Note that the SCTP receiver normally responds with delayed SACKs, but immediately returns a SACK whenever reordering is observed.

The critical instant in the scenario, denoted t_2, occurs when A receives the fourth missing report [11, 13]. At this instant, TSNs $K+1$ through G_1 get marked for retransmission. Due to the receipt of a SACK acking TSN $G_1 + 4$, (at t_2) C_2 allows one MTU sized chunk to be transmitted, hence TSN $K+1$ gets retransmitted to destination B_2. According to RFC2960, "... when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk to an active destination transport address that is different from the last destination address to which the DATA chunk was sent." Since the original transmission of TSN $K+1$ went to B_1, the retransmission of TSN $K+1$ is sent to B_2. The value of K is estimated and its relevance to the *cwnd* overgrowth is explained in Section 2.2.

The retransmission of TSN $K+1$ at $t = t_2$ is a consequence of the fourth missing report (SACK received on interface A_2 at $t = t_2$) carrying cumulative ack K. Since TSNs $G_1 + 1$ through $G_1 + 4$ reached host B by time t_1, the SACK also carries a gap ack for TSNs $G_1 + 1$ through $G_1 + 4$, resulting in the marking of TSNs $K+1$ through G_1 for retransmission. The cumulative ack K is an indication that the receiver B has received K TSNs *in-sequence* by time t_1. This in-sequence data is clearly the data received by B on the interface B_1 by time t_1.

Following the retransmission of TSN $K+1$, the SACK for the original transmission of TSN $K+1$ arrives at A. Since host A now considers TSN $K+1$ to be outstanding on destination B_2, the receipt of this SACK incorrectly increases C_2, and allows TSNs $K+2$ and $K+3$ to be retransmitted. The receipt of a SACK for TSN $K+1$ immediately after TSN $K+1$ is retransmitted is not a coincidence. At time t_1 when host B sends a SACK with a cumulative ack of K acking the receipt of TSN $G_1 + 4$, TSN $K+1$ is concurrently being received on interface B_1. Immediately after the receipt of TSN $K+1$ on interface B_1, host B sends a SACK with cumulative ack $K+1$. Consequently, the sequence of events at host A is the receipt of a SACK with cumulative ack K (which is also the fourth missing report for TSNs $K+1$ through G_1) followed by a SACK with cumulative ack $K+1$. As shown, this behaviour continues until the SACKs for all the original transmissions to B_1 (up to TSN G_1) have been received at host A.

2.2 Analytic Results

We will now estimate the *cwnd* overgrowth of C_2, and the number of unnecessary retransmissions. The parameters used in the following analysis are:

L_{1F}, L_{2F} : Maximum Transmission Unit (MTU) sizes on forward paths A_1 to B_1 and A_2 to B_2, respectively
B_{1F}, B_{2F} : End-to-End available bandwidths [9] on forward paths A_1 to B_1 and A_2 to B_2, respectively
e : Delay experienced by a packet along a path, given by:

$$e = \sum_{i \,=\, each\ hop} (prop)_i + (proc)_i + (queue)_i + (trans)_i \tag{1}$$

where $prop$ = propagation delay, $proc$ = processing delay, $queue$ = queueing delay, and $trans$ = transmission delay.

e_F : Delay experienced by a data packet, along the forward path. *Assumption:* Each data packet is MTU sized, therefore, e_F is estimated by:

$$e_F = \sum_{i \,=\, each\ hop\ in\ forward\ path} \left\{ \begin{array}{l} (prop)_i + (proc)_i \\ +(queue)_i + \frac{L}{B^i} \end{array} \right. \tag{2}$$

where, L is the MTU of the path, and B^i is available bandwidth at hop i.
e_{1F}, e_{2F} : Delays experienced by a data packet on forward paths A_1 to B_1 and A_2 to B_2, respectively,

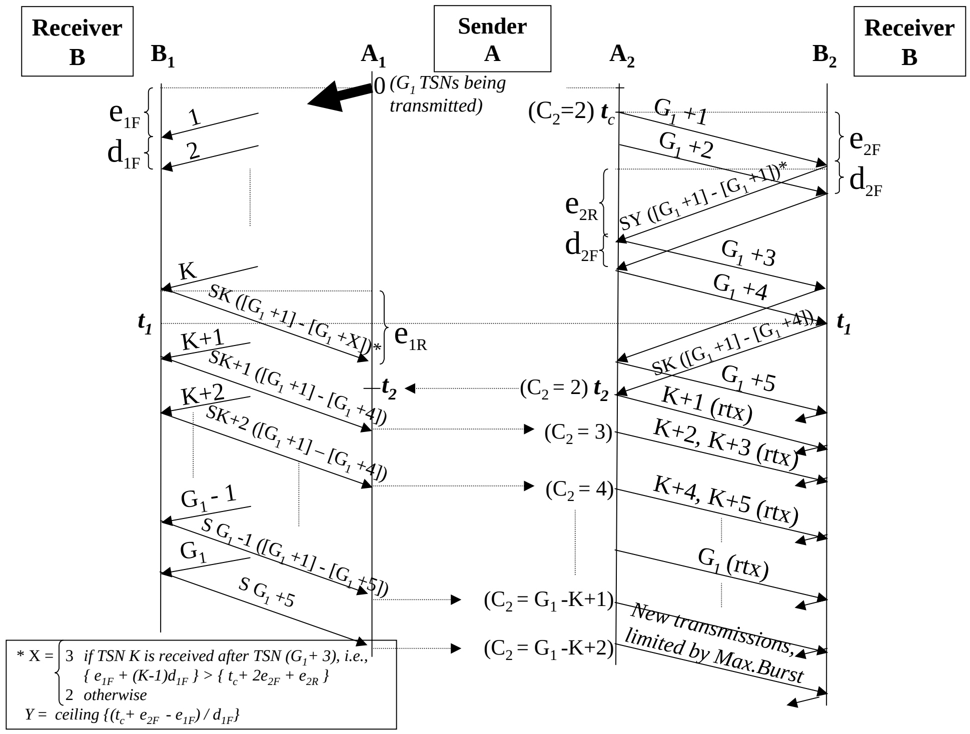

Figure 2: General timeline for the problem

e_R : Delay experienced by a pure SACK packet, along the reverse path. *Assumption:* that transmission delays for pure SACK packets are negligible, therefore, e_R is estimated by:

$$e_R = \sum_{i\,=\,each\ hop\ in\ reverse\ path} (prop)_i + (proc)_i + (queue)_i \tag{3}$$

e_{1R}, e_{2R} : Delays experienced by a pure SACK packet on reverse paths B_1 to A_1 and B_2 to A_2, respectively.

d : Minimum delay observed between consecutive packets transmitted along a same path by the receiver of the packets. This delay is dictated by end-to-end available bandwidth of the path, which is determined by the hop with the minimum available bandwidth on the path (in other words, the path bottleneck). d is given by:

$$d = \frac{L}{\min_{i\,=\,each\ hop}\{B^i\}} \tag{4}$$

where, L is the MTU of the path, and B^i is available bandwidth at hop i.

d_{1F}, d_{2F} : Minimum delays between consecutive data packets from A_1 to B_1 observed at B_1, and from A_2 to B_2 observed at B_2, respectively.

d_{1R}, d_{2R} : Minimum delays between consecutive SACK packets from B_1 to A_1 observed at A_1, and from B_2 to A_2 observed at A_2, respectively.

Assumption: The reverse path does not change the delay between SACKs. In other words, the forward path's bottleneck dictates the rate at which SACKs are transmitted and then received, not the reverse path's bottleneck. Therefore, the delay observed *between* SACKs is the same as the delay observed between the data packets. In other words,

$$d_{1R} \ = \ d_{1F}, \ \ and \ d_{2R} \ = \ d_{2F} \tag{5}$$

Packet transmission on path 2 starts at time t_c; it takes some time for the fourth legitimate missing report to reach the sender A. This time instant is shown in figure 2 as t_2, which is given by:

$$t_2 \ = \ t_c \ + \ 2e_{2F} \ + \ 2e_{2R} \ + \ d_{2F} \tag{6}$$

t_1 is the instant when this fourth legitimate missing report *leaves* the receiver B through B_2, and is given by:

$$t_1 \ = \ t_2 \ - \ e_{2R} \ = \ t_c \ + \ 2e_{2F} \ + \ e_{2R} \ + \ d_{2F} \tag{7}$$

As shown in figure 2, we assume that the SACK received at t_2 on A_2 contains the highest cumulative ack received by A so far[3].

Let K be defined as the TSN that was most recently cumulatively acked at A prior to time t_2. In other words, K is the last TSN that reached the receiver B on B_1 at t_1, where,

$$\begin{aligned} K &= \lceil \frac{t_1 - e_{1F}}{d_{1F}} \rceil \\ &= \lceil \frac{t_c + 2e_{2F} + e_{2R} + d_{2F} - e_{1F}}{d_{1F}} \rceil \end{aligned} \quad (8)$$

The result is that TSNs $(K+1)$ through G_1 will be retransmitted on Path 2 and the total number of unnecessary retransmissions = $\max\{0, G_1 - (K+1) + 1\} = \max\{0, G_1 - K\}$. The *cwnd* overgrowth for C_2 will be $\max\{0, G_1 - K\}$.

2.3 Discussion

For an AIMD (Additive Increase Multiplicative Decrease) congestion control algorithm, a *round* refers to the period from the transmission of *cwnd* amount of data to the receipt of acks for that data. After receipt of these acks, the next round starts as the sender transmits *cwnd*+1 amount of data. In our general model, the period between $t = 0$ and $t = t_c$ represents the beginning of such a round when the sender transmits G_1 amount of data. This transmission of data may or may not be in a burst, but the receiver receives the data with packet interarrival times of at least d_{1F} on interface B_1 since d_{1F} is the delay due to the available bandwidth of the bottleneck link on path 1. Thus, in our model, we assume that TSNs are received on interfaces B_1 and B_2 uniformly with interarrival times d_{1F} and d_{2F}, respectively.

If $K \geq G_1$, then all of the original transmissions to B_1 are received by host B by time t_1. Hence, the SACK received by host A at time t_2 would carry a cumulative ack of $G_1 + 4$ and no gap acks. In this case, no unnecessary retransmission and no TCP-unfriendly *cwnd* growth occurs.

On the other hand, if $K < G_1$, then K is the last TSN cumulatively acked prior to time t_2. Consequently, $K+1$ is the first TSN to be retransmitted incorrectly, and C_2 overgrows by $G_1 - K$. A higher value of K results in a higher cumulative ack at A at t_2, hence fewer retransmissions and consequently less error in C_2. Similarly, as K decreases, more unnecessary retransmissions occur, and the error in C_2 also increases.

From equation (8), K decreases with an increase in d_{1F}, or a decrease in d_{2F}. Further, K decreases with an increase in e_{1F}, or a decrease in e_{2F}. These relationships between K and the characteristics of the two paths imply that *when a changeover is made to a higher quality path, there is a likelihood of TCP-unfriendly cwnd growth and unnecessary retransmissions, and the bigger the improvement in quality that the new path provides, the larger the TCP-unfriendly growth and number of incorrect retransmissions will be.*

[3]This assumption is made for simplicity of analysis. If this assumption does not hold, the *cwnd* overgrowth will be lesser by $\lceil \frac{e_{2R} - e_{1R}}{d_{1F}} \rceil$.

3 ANALYTIC RESULTS: VALIDATION AND VISUALIZATION

It is clear from the analytic results derived in Section 2.2 that *cwnd* overgrowth occurs if the sender has more than K packets outstanding at the time of changeover. The value of K, given by equation (8), is thus pivotal in quantifying *cwnd* overgrowth. We first validate this analytical value of K using ns-2 simulations in Section 3.1. We then estimate the value of K using the model under various network and changeover conditions in Section 3.2.

3.1 Analytic Results: Validation

We now validate the analytical value of K derived in Section 2.2 through simulations using the SCTP module for ns-2 which was developed in the Protocol Engineering Lab at the University of Delaware [1, 10]. The topology is the same as in figure 1. The simulations do not have any cross traffic, hence the end-to-end available bandwidths on each of paths 1 and 2 is equal to the minimum of link capacities on the corresponding path. Each of paths 1 and 2 has three links - two edge links and one core link. The edge links have a capacity of 10Mbps and propagation delay of 1ms. The available bandwidths of the paths, i.e., the capacities of the core links are chosen randomly between 10Kbps and 1Mbps. The propagation delays of the core links are chosen randomly between 25ms and 50ms. The sender's sending window is fixed at 20KB by setting the receiver's advertised window to 20KB. We fix the sending window to make it easier to extract parameters from the traces. Changeover occurs at time 5 seconds.

Of 1000 simulation runs, 511 runs showed the occurrence of incorrect fast retransmissions due to changeover. Only the runs which showed these retransmissions could be used for validation because to infer the value of K from a simulation run (denoted K_{sim}), at least one such retransmission had to occur. The first incorrect retransmission would correspond to TSN $K_{sim} + 1$.

We extracted the values of the parameters e_{1F}, e_{1R}, e_{2F}, e_{2R}, d_{1F}, d_{2F} and t_c from the traces for each of the 511 runs.

Feeding these parameters into equation (8) gave us the analytic value of K (denoted K_{anal}).

Simulation results show that of the 511 comparisons of K_{sim} and K_{anal}, 431 results agreed exactly. In the remaining 80 results that did not agree, K_{anal} was equal to $K_{sim} - 1$. This underestimation of K by the analytic model could be attributed to the assumption made in the derivation of analytic expression for K in Section 2.2, or to approximations made in extracting the parameters from the traces.

The simulations thus agree with our analytic results.

3.2 Analytic Results: Visualization

In graphing the analytically derived value of K, we reduce the number of independent variables by making the following assumptions so as to visualize the graphs better:

- Forward paths 1 and 2 have the same MTU. Hence, $L_{1F} = L_{2F} = L$
- The forward and reverse paths have the same propagation, processing and queueing delays. Using equations (2) and (3),

$$e_F = e_R + \sum_{i = each\ hop\ in\ forward\ path} \frac{L}{B^i} \quad (9)$$

- The transmission delays at the other links along a path are assumed negligible in comparison to the transmission delay at the bottleneck link. Using equation (4),

$$\sum_{for\ i = each\ hop} \frac{L}{B^i} \approx \frac{L}{\min_{i = each\ hop}\{B^i\}} = d \quad (10)$$

- Combining the above two assumptions, we get

$$e_F = e_R + d_F = e_R + \frac{L}{B_F} \quad (11)$$

  For the forward paths 1 and 2, the equation 11 can be rewritten as

$$e_{1F} = e_{1R} + \frac{L}{B_{1F}}\ and\ e_{2F} = e_{2R} + \frac{L}{B_{2F}} \quad (12)$$

Figures 3.2 and 3.2 (left) graph K as a function of B_{2F}, for fixed values of B_{1F}, e_{1R} and e_{2R}. In these 2-D graphs, the changeover time, t_c, is fixed at 10ms. Each 3-D graph in figures 3.2 and 3.2 (right) picks one representative curve from the corresponding 2-D graph (left), and shows the influence of t_c on K. These 3-D graphs thus show K as a function of B_{2F} and t_c, for fixed values of B_{1F}, e_{1R} and e_{2R}.

The graphs are organized as follows:

- The results in figure 3.2 use the range 10kbps - 100kbps for the available bottleneck bandwidths B_{1F} and B_{2F}. t_c is set to 10ms in the 2-D graphs. The curve corresponding to B_{1F} = 50kbps is used as a representative curve to show the influence of t_c on K. t_c varies over 10ms - 100ms in the 3-D graphs. Three combinations of (e_{1R}, e_{2R}) are used: (50ms, 50ms), (50ms, 25ms), and (25ms, 50ms).
- The results in figure 3.2 use the range 100kbps - 1Mbps for the available bottleneck bandwidths B_{1F} and B_{2F}. t_c is set to 10ms in the 2-D graphs. The curve corresponding to B_{1F} = 500kbps is used as a representative curve to show the influence of t_c on K. t_c varies over 10ms - 100ms in the 3-D graphs. Three combinations of (e_{1R}, e_{2R}) are used: (50ms, 50ms), (50ms, 25ms), and (25ms, 50ms).

We split the range (10kbps - 1Mbps) into two subranges (10kbps - 100kbps and 100kbps - 1Mbps), because the variation observed in K with both B_{1F} and B_{2F} ranging from 10kbps to 1Mbps is large. We are thus able to visualize the behaviour of K over a large range of available bandwidths, with the assumption that the available bandwidths of the two paths are comparable.

In figure 3.2, K varies between 0 and 30, and mostly has a value below 10. Remember that the smaller K is, the more unnecessary retransmissions will occur, and the more *cwnd* grows when it should not. Changes in e_{1R}, e_{2R} and t_c seem to have little influence on K, as compared to the variation due to B_{1F}, B_{2F}. That is because in this set, since the available bandwidths are low, the total delay is dominated by transmission delay.

In figure 3.2, K varies between 0 and 40. The median value of K in this set has increased from the first set. This increase can be attributed to the greater range of the bottleneck bandwidths. Another important factor can be understood by considering equation (8). With an increase in the bottleneck bandwidth, the value of d_{1F} decreases, consequently increasing K. We also observe the increased influence of e_{1R}, e_{2R} and t_c in this set of results, since the transmission delay is lesser dominant in this set.

In both sets, we note that K decreases with a decrease in B_{1F} or an increase in B_{2F}, as is expected.

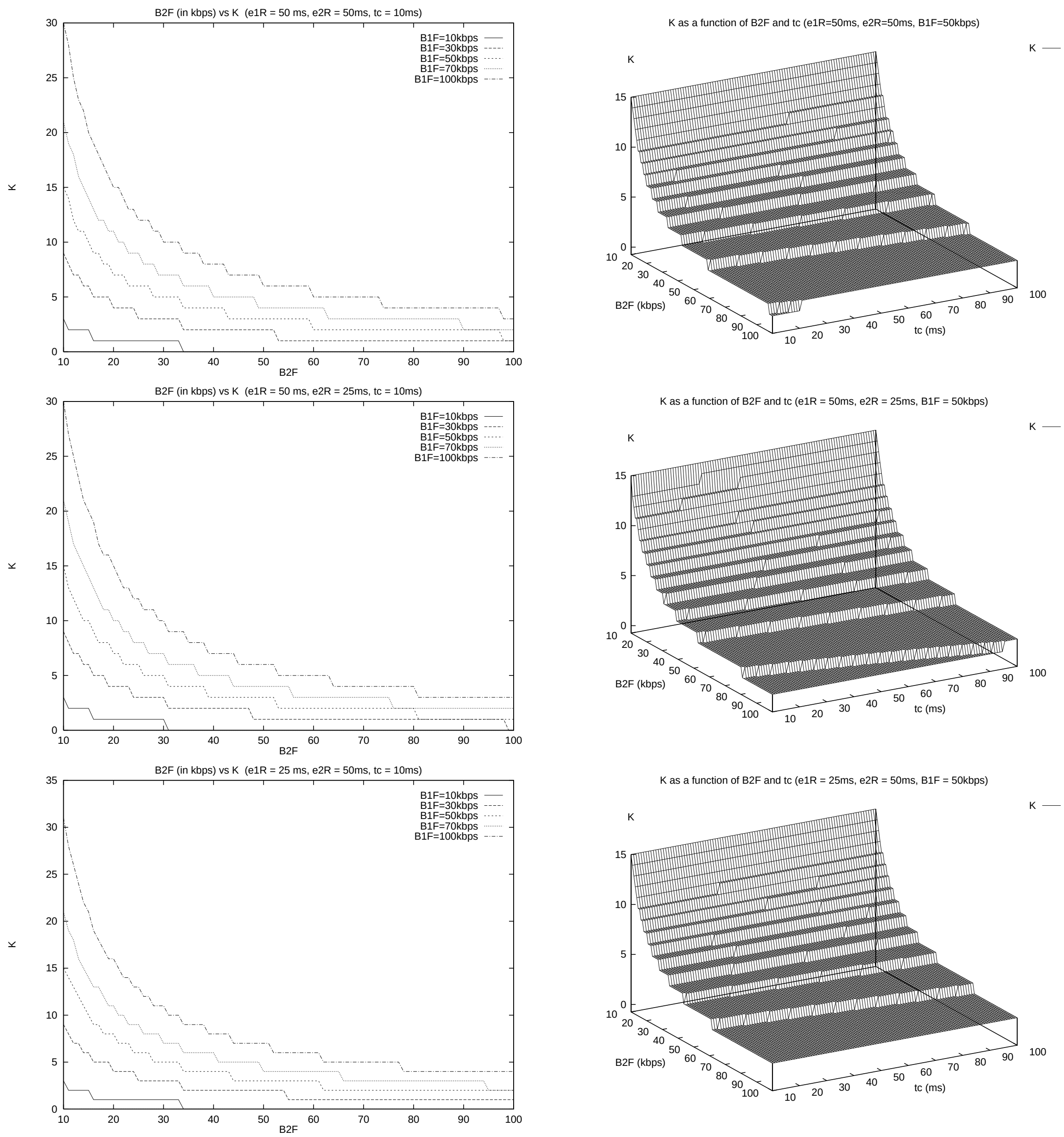

Figure 3: Graphing K analytically: 10kbps $\leq B_{1F}, B_{2F} \leq$ 100kbps

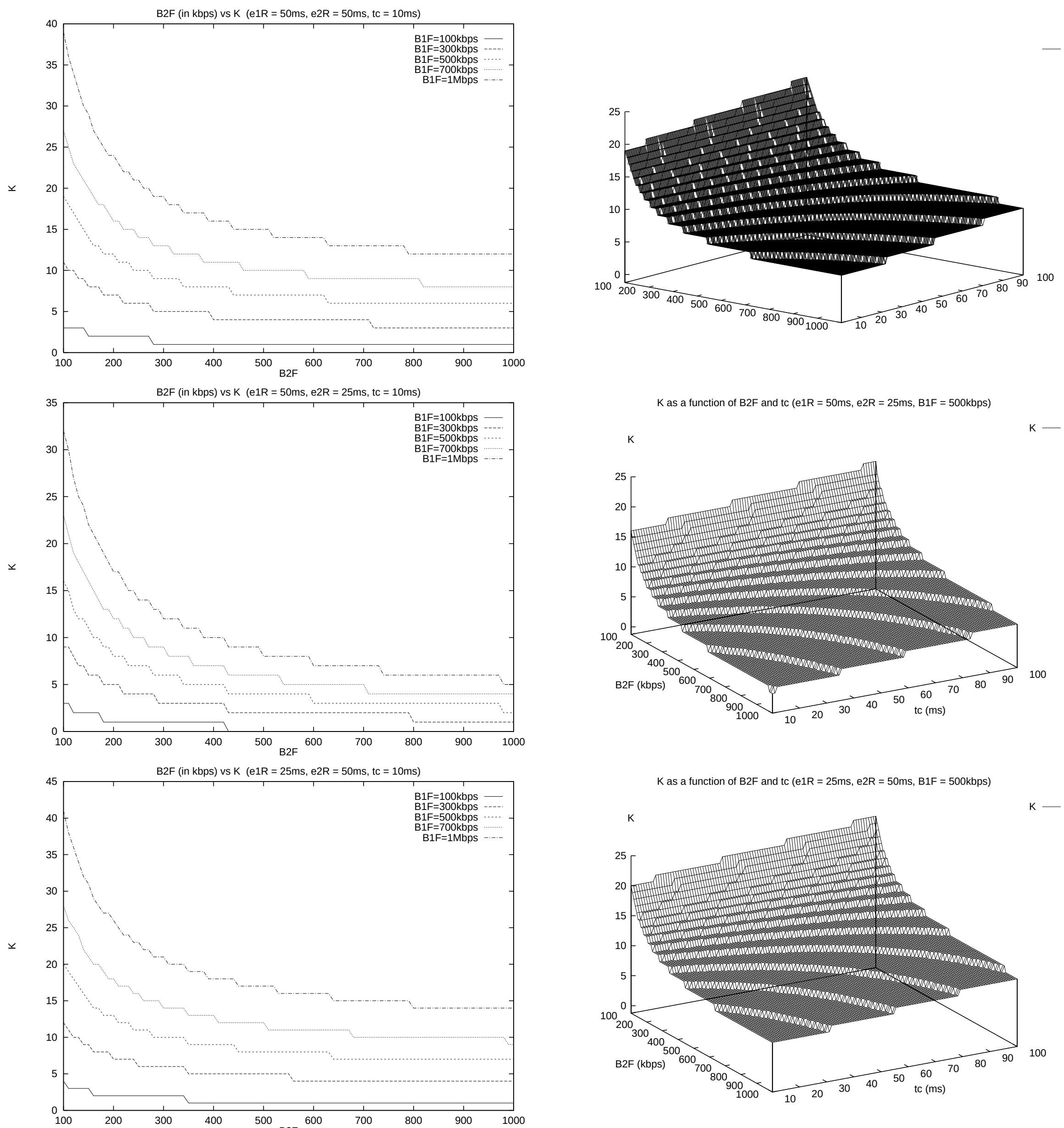

Figure 4: Graphing K analytically: 100kbps $\leq B_{1F}, B_{2F} \leq$ 1Mbps

4 PROPOSED SOLUTION: CHANGEOVER AWARE CONGESTION CONTROL

As mentioned earlier, the TCP-unfriendly *cwnd* growth and incorrect retransmissions during changeover occur due to current inadequacies of SCTP - (i) the sender is unable to distinguish SACKs for transmissions from SACKs for retransmissions, and (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 more important problem of TCP-unfriendly *cwnd* growth. The *Rhein Algorithm* [8] solves the problem by addressing (i). In this section, we propose solutions which solve the problem by addressing (ii). In other words, the following 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. *Changeover aware congestion control (CACC)* algorithms curb the TCP-unfriendly *cwnd* growth 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.

Section 4.1 describes the *Conservative CACC (C-CACC)* algorithm which 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. In Section 4.2, we describe the *Split Fast Retransmit CACC (SFR-CACC)* algorithm which alleviates this disadvantage. We verify the effectiveness of the SFR-CACC algorithm through simulation in Section 4.3. In Section 5, we discuss the advantages of the CACC algorithms over the Rhein algorithm in solving the *cwnd* overgrowth problem.

4.1 Conservative CACC

As mentioned previously, C-CACC maintains state at the sender when changeover happens, on a per-destination basis. This state is used to conservatively increment missing report counts for TSNs. This conservative approach prevents incorrect triggering of fast retransmissions, thus eliminating the *cwnd* overgrowth problem.

As was discussed in Section 2.1, the receiver could observe reordering of TSNs due to changeover. According to C-CACC, the sender uses state maintained for the current primary destination to identify SACKs that are sent by the receiver after the receiver observes this reordering. The state is constituted by two variables per-destination:

1. *CHANGEOVER_ACTIVE* - a flag which indicates the occurence of a changeover.
2. *next_tsn_at_change* - the next TSN to be used by the sender, at the moment of changeover.

The algorithm is described in figure 5. On changeover, the sender sets the state as described[4]. The sender is considered to be in *active changeover* state until the *CHANGEOVER_ACTIVE* flag is cleared. The flag is cleared when a SACK which cumulatively acks TSNs up to and including *next_tsn_at_change* is received. At that time, all TSNs which were sent to the receiver before changeover occurred at the sender have been received, andreordering due to changeover no longer happens. This period during which the sender is in active changeover state is referred to as the *active changeover period*, and the outstanding TSNs which have not yet been acked at the sender at the moment of changeover constitute the *changeover range*.

During the changeover period, receipt of a SACK that reports a TSN greater than or equal to *next_tsn_at_change* indicates to the sender that reordering has been observed at the receiver. Since this reordering is likely due to changeover, the sender does not increment missing report counts for TSNs in the changeover range, thus preventing the incorrect fast retransmissions.

C-CACC is conservative because when reordering due to changeover is observed at the receiver and consequently reported to the sender, the sender conservatively chooses to not increment missing reports for *any TSN in the changeover range*. In the face of loss, the sender will not perform fast retransmission on any TSN in the changeover range. The TSNs in the changeover range would thus have to wait for retransmission timeouts to be retransmitted. Furthermore, C-CACC does not take into account the possibility of multiple changeovers at the sender.

4.2 Split Fast Retransmit CACC (SFR-CACC)

To alleviate the limitations of C-CACC, note that the reordering observed during changeover happens because TSNs which are supposed to reach the receiver *in-sequence* end up reaching the receiver in *concurrent groups, in-sequence within each group*. With this observation, we reason that the

[4] Unless explicitly stated, the variables used in the CACC algorithms refer to the state for the current primary destination, from the sender's viewpoint.

On changeover, the sender maintains the following state for the new primary destination:

1) Set *CHANGEOVER_ACTIVE* to 1, indicating that a changeover has occured.
2) Store the next TSN to be sent in *next_tsn_at_change*.

On receipt of a SACK,

1) If the cumulative ack in the SACK is $\geq$ the *next_tsn_at_change*, the *CHANGEOVER_ACTIVE* flag is cleared.
2) The following algorithm dictates when the missing report count for a TSN t should be incremented in accordance with [13, 11], and when the count should not be incremented:

```
if    (CHANGEOVER_ACTIVE == 1) and
      (the SACK reports at least one TSN ≥ next_tsn_at_change)
then
      if (t ≥ next_tsn_at_change)
      then
            Increment missing report count for t according to [13, 11];
      else
            Do not increment missing report count for t;
else
      Increment missing report count for t according to [13, 11];
```

Figure 5: Conservative CACC Algorithm

fast retransmit algorithm can be applied independently within each group. That is, on the receipt of a SACK, if the sender can estimate the TSN(s) that causes this SACK to be sent from the receiver, the sender can use the SACK to increment missing report counts *within the causative TSN(s)'s group*.

In SFR-CACC, four variables for each destination are introduced:

1. *CHANGEOVER_ACTIVE* - a flag which indicates the occurrence of a changeover.
2. *CYCLING_CHANGEOVER* - a flag which indicates whether the change of primary is the first changeover to this destination address during an active changeover. This flag helps determine changeovers cycling through destination address space.
3. *next_tsn_at_change* - the next TSN to be used by the sender, at the moment of changeover.
4. *cacc_saw_newack* - a temporary flag, used during SACK processing to estimate the causative TSN(s)'s group.

SFR-CACC is broken up into three logical parts. SFR-CACC(1) is very similar to the initial part of C-CACC algorithm, except for the *CYCLING_CHANGEOVER* flag which we will discuss shortly. SFR-CACC(2) and SFR-CACC(3) specify sender actions on receipt of a SACK.

On receipt of a SACK that cumulatively acks up to and including *next_tsn_at_change*, the sender leaves the active changeover state. In SFR-CACC(2) the sender estimates the causative TSN(s)'s destination. The sender estimates the causative TSN(s) as TSN(s) getting acked for the first time in a SACK. TSNs sent to the same destination as the causative TSN(s) form the causative TSN(s)'s group.

In SFR-CACC(3), the sender does not increment missing report counts for TSNs *outside* the causative TSN(s)'s group. In other words, the sender applies the SACK selectively to fast retransmit *within* the causative TSN(s)'s group. If more than one group are being acked, then fast retransmit is conservatively applied only to TSNs in the current primary destination's group.

SFR-CACC does the in-group marking of TSNs only as long as the sender does not changeover to a previously used destination address which was already used during the current active changeover period. If the sender starts to cycle through destination address space, then the sender switches to a more conservative behaviour of marking only TSNs in the latest outstanding group. The protection from such cycling changeovers is necessary because SFR-CACC assumes that the latest outstanding TSNs were transmitted to the current

On changeover, for the new primary destination:

1) If *CHANGEOVER_ACTIVE* is 1, then there was a changeover to this destination address earlier. The sender sets *CYCLING_CHANGEOVER* to 1, indicating that this changeover is a cycling switch to the same destination address during an active changeover.
2) The sender sets *CHANGEOVER_ACTIVE* to 1, indicating that a changeover has occured.
3) The sender stores the next TSN to be sent in *next_tsn_at_change*.

Figure 6: Split Fast Retransmit CACC Algorithm (Part 1)

primary. One could envision a scenario where the sender has TSNs outstanding on two destination addresses, B_1 and B_2, having performed changeover in that order. The sender then performs a changeover back to B_1, and a SACK acking both TSNs from both groups is received. The sender could now end up incorrectly fast retransmitting TSNs sent to destination B_1, causing *cwnd* overgrowth on destination B_2 - precisely what we are trying to avoid. There may be other scenarios where the original problem of *cwnd* overgrowth may occur due to cycling changeovers. For the moment, we have not looked into cycling changeover in greater depth, and design SFR-CACC to be conservative when a cycling changeover occurs.

4.3 Simulations

Verification of the effectiveness of SFR-CACC was done through ns-2 simulations. Using SFR-CACC under the same conditions as in section 3.1 for which *cwnd* overgrowth was observed, the simulations showed no unnecessary retransmissions, or *cwnd* overgrowth due to changeover.

5 CONCLUSION AND FUTURE WORK

Results from Section 3 suggest that the problem might not be a “corner case”, since for a large range of network settings, the value of K, which governs the minimum packets required to be outstanding at the time of changeover so as to observe *cwnd* overgrowth, is low. By approaching the problem from different perspectives, the Rhein algorithm [8] and the CACC algorithms all solve the problem of TCP-unfriendly *cwnd* growth. 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. Consequently, these algorithms prevent the TCP-unfriendly *cwnd* growth. These algorithms have the added advantage that no extra bits are added to any packets, and thus the load on the wire and the 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.

The fast retransmit algorithm is active on the changeover range for a longer time in SFR-CACC than with C-CACC. To quantify the number of TSNs which will be ineligible for fast retransmit in the face of loss, let us assume that only one changeover is performed and that SACKs are not lost. Under these assumptions, potentially only the last four packets sent to the old primary destination will be forced to be retransmitted with an RTO instead of a fast retransmit. In other words, under these assumptions, if a TSN is lost, and at least four packets are successfully transmitted to the same destination after the loss, then the TSN will be retransmitted via fast retransmit. With C-CACC however, any TSN in the changeover range will require an RTO to recover from loss. C-CACC is also incapable of handling multiple changeovers, whereas SFR-CACC is equipped to do so.

We have implemented SFR-CACC in the NetBSD/FreeBSD release for the KAME stack [2, 3]. The implementation uses three flags and one TSN marker for each destination, as described in Section 4.2. 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. Since the writing of this paper, we have made modifications to the SFR-CACC algorithm presented

SFR-CACC (Part 2): On receipt of a SACK,

1) If the cumulative ack in the SACK is $\geq$ *next_tsn_at_change*,
   the *CHANGEOVER_ACTIVE* and *CYCLING_CHANGEOVER*
   flags are cleared for all destinations.
2) If (*CHANGEOVER_ACTIVE* == 1) and (the SACK contains Gap Acks)
   then

```
        for each destination d
        do
              initialize d.cacc_saw_newack = 0;
        done;

        for each TSN t being acked, that has not been acked in any SACK so far
        do
              let d be the destination to which t was sent;
              set d.cacc_saw_newack = 1;
        done
```

SFR-CACC (Part 3): On receipt of a SACK (contd.),

3) The following algorithm dictates when the missing report count for a TSN
   t should be incremented in accordance with [13, 11], and when the count
   should not be incremented:

```
        if    (CHANGEOVER_ACTIVE == 1) and (CYCLING_CHANGEOVER == 0)
        then
              let count_of_newacks be number of destinations for which cacc_saw_newack is set;
              if (count_of_newacks == 1)
              then /* SACK acks only one dest */
                    let d be the destination to which t was sent;
                    if (d.cacc_saw_newack == 1)
                    then
                          Increment missing report count for t according to [13, 11];
                    else
                          Do not increment missing report count for t;
              else /* Mixed SACK - SACK acks more than one dest */
                    if (t was sent to the current primary)
                    then
                          Increment missing report count for t according to [13, 11];
                    else
                          Do not increment missing report count for t;
        else if    (CHANGEOVER_ACTIVE == 1) and (CYCLING_CHANGEOVER == 1)
        then /* Cycling observed, hence mark only in latest group */
              if (t ≥ next_tsn_at_change)
              then
                    Increment missing report count for t according to [13, 11];
              else
                    Do not increment missing report count for t;
        else /* Sender is not in changeover active state */
              Increment missing report count for t according to [13, 11];
```

Figure 7: Split Fast Retransmit CACC Algorithm (Parts 2 and 3)

in Section 4.2. The modifications simplify the algorithm, and handle cycling changeovers. We are currently proposing addition of the modified SFR-CACC algorithm to SCTP.

6 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.

7 ACKNOWLEDGMENTS

Thanks to Ivan Arias Rodriguez, Vern Paxson, Mark Allman, Phillip Conrad and Johan Garcia for their comments and inputs.

References

[1] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

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

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

[4] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM Computer Communication Review*, Vol. 32(1), January 2002.

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

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

[7] M. Gerla, S. S. Lee, and G. Pau. TCP Westwood Simulation Studies in Multiple-Path Cases. Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS) 2002, San Diego, CA, July 2002.

[8] 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.

[9] M. Jain and C. Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth. Proc. 3rd Passive and Active Measurements Workshop, Fort Collins, March 2002.

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

[11] 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)*.

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

[13] 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.

[14] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. Technical Report TR-02-006, International Computer Science Institute (ICSI), July 2002.