Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP

Preethi Natarajan[1], Nasif Ekiz[1], Ertugrul Yilmaz[1], Paul D. Amer[1], Janardhan Iyengar[2], Randall Stewart[3]

[1]CIS Dept., University of Delaware, {nataraja,ekiz,yilmaz,amer}@cis.udel.edu,

[2]Math & Computer Science, Franklin & Marshall College, jiyengar@fandm.edu

[3]Cisco Systems, rrs@lakerest.net

***Abstract*— In both TCP and SCTP, selectively acked (SACKed) out-of-order data is implicitly renegable; that is, the receiver can later discard SACKed data. The possibility of reneging forces the transport sender to maintain copies of SACKed data in the send buffer until they are cumulatively acked. In this paper, we investigate the situation where all out-of-order data is non-renegable, such as when the data has been delivered to the application, or when the receiver simply never reneges. Using simulations, we show that SACKs result in inevitable send buffer wastage, which increases as frequency of loss events and loss recovery durations increase. We introduce a fundamentally new ack mechanism, Non-Renegable Selective Acknowledgments (NR-SACKs), for SCTP. Using NR-SACKs, an SCTP receiver can explicitly identify some or all out-of-order data as being non-renegable, allowing the sender to free up send buffer sooner than if the data were only SACKed. Simulation comparisons show that NR-SACKs enable efficient utilization of a transport sender's memory. Further investigations show that NR-SACKs also improve throughput in Concurrent Multipath Transfer (CMT) [4].**

***Index Terms*—Network Protocols, Protocol Design and Analysis**

I. INTRODUCTION

Reliable transport protocols such as TCP and SCTP (Stream Control Transmission Protocol) [RFC4960] employ two kinds of data acknowledgment mechanisms: (i) cumulative acks indicate data that has been received in-sequence, and (ii) selective acknowledgments (SACKs) indicate data that has been received out-of-order. In both TCP and SCTP, while cumulatively acked data is the receiver's responsibility, SACKed data is not. SACK information is only *advisory* [RFC3517, RFC4960]. While SACKs notify a sender about the reception of specific out-of-order TPDUs, the receiver is permitted to later discard the TDPUs. Discarding data that has been previously SACKed is known as *reneging*. Though reneging is a possibility, the conditions under which current transport layer and/or operating system implementations renege, and the frequency of these conditions occurring in practice (if any) are unknown and needs further investigation.

Data that has been delivered to the application, by definition, is non-renegable by the transport receiver. Unlike TCP which never delivers out-of-order data to the application, SCTP's *multistreaming* and *unordered data delivery* services result in out-of-order data being delivered to the application and thus becoming non-renegable. Interestingly, TCP and SCTP implementations can be configured such that the receiver is not allowed to and therefore never reneges on out-of-order data (details in Section II). In these configurations, even non-deliverable out-of-order data becomes non-renegable.

The current SACK mechanism in both TCP and SCTP does not differentiate between out-of-order data that "has been delivered to the application and/or is non-renegable" vs. data that "has not yet been delivered to the application and is renegable". In this work, we introduce a fundamentally new acknowledgment mechanism called Non-Renegable Selective Acknowledgments (NR-SACKs). NR-SACKs enable a transport receiver to explicitly convey non-renegable information on out-of-order data to the sender. While this work introduces NR-SACKs for SCTP, the NR-SACKs idea can be applied to any reliable transport protocol that uses selective acks and/or permits delivery of out-of-order data.

In this work, we investigate the effect of SCTP's SACK mechanism in situations where out-of-order data is non-renegable. We also compare SACKs vs. NR-SACKs in Concurrent Multipath Transfer (CMT) [4]. CMT is an experimental extension to SCTP that exploits SCTP's multihoming feature for simultaneous transfer of new data over multiple paths (details in Section II). Section III introduces the NR-SACK chunk for SCTP, and briefly explains sender and receiver side NR-SACK processing details. Section IV elaborates evaluation preliminaries such as

- Prepared through collaborative participation in the Communication and Networks Consortium sponsored by the US Army Research Lab under Collaborative Tech Alliance Program, Coop Agreement DAAD19-01-2-0011. The US Gov't is authorized to reproduce and distribute reprints for Gov't purposes notwithstanding any copyright notation thereon.
- Supported by the University Research Program, Cisco Systems, Inc.

ns-2 simulation topologies, parameters for cross-traffic generation, and performance metrics. Section V analyzes SACK vs. NR-SACK results for both SCTP and CMT. Section VI concludes our work

II. PROBLEM DESCRIPTION

A. Background

The SCTP (or TCP) layer send buffer, or the sender-side socket buffer (Figure 1), consists of two kinds of data: (i) new application data waiting to be transmitted for the first time, and (ii) copies of data that have been transmitted at least once and are waiting to be cum-acked, a.k.a. the *retransmission queue* (rtxq). Data in the rtxq is the transport sender's responsibility until the receiver has guaranteed their delivery to the receiving application, and/or the receiver guarantees not to renege on the data.

In traditional in-order data delivery service (e.g. TCP), a receiver cumulatively acknowledges (cum-acks) the latest in-order data. Cum-acked data has either been delivered to the application or is ready for delivery. In either case, cum-acks are an explicit assurance that the receiver will not renege on the corresponding data. Upon receiving a cum-ack, the sender is no longer responsible, and removes the corresponding data from the rtxq. In the current SACK mechanism, cum-acks are the only means to convey non-renegable information; all selectively acknowledged (out-of-order) data are renegable.

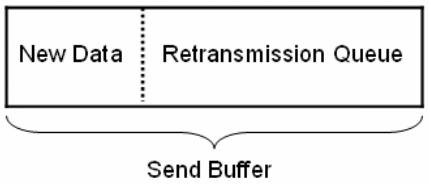

Figure 1: Transport Layer Send Buffer

SCTP's *multistreaming* service divides an end-to-end *association* (SCTP's term for a transport connection) into independent logical data streams. Data arriving in-sequence within a stream can be delivered to the receiving application even if the data is out-of-order relative to the association's overall flow of data. Also, data marked for *unordered delivery* can be delivered immediately upon reception, regardless of the data's position within the overall flow of data. Thus, SCTP's data delivery services result in situations where out-of-order data is delivered to the application, and is thus non-renegable.

Operating systems allow configuration of transport layer implementations such that out-of-order data is never reneged. For example, in FreeBSD, the *net.inet.tcp.do_tcpdrain* or *net.inet.sctp.do_sctp_drain* sysctl parameters can be configured to never revoke kernel memory allocated to TCP or SCTP out-of-order data [9]. Thus, out-of-order data can also be rendered non-renegable through simple user configuration.

In the following discussions, "non-renegable out-of-order data" refers to data for which the transport receiver takes full responsibility, and guarantees not to renege either because (i) the data has been delivered (or is deliverable) to the application, or (ii) the receiving system (OS and/or transport layer implementation) guarantees not to revoke the allocated memory until after the data is delivered to the application. With the current SACK mechanism, non-renegable out-of-order data is selectively acked, and is (wrongly) deemed renegable by the transport sender. Maintaining copies of non-renegable data in the sender's retransmission queue is unnecessary.

B. SCTP Unordered Data Transfer with SACKs

This section discusses the effects of SACKs in transfers where all out-of-order is non-renegable. The discussion is applicable to any type of reliable data delivery service (in-order, partial-order, unordered) where all out-of-order data is non-renegable, but uses the simple unordered SCTP data transfer example shown in Figure 2.

In this example, the SCTP send buffer denoted by the rectangular box can hold a maximum of eight TPDUs. Each SCTP PDU is assigned a unique Transmission Sequence Number (TSN). The timeline slice shown in Figure 2 picks up the data transfer at a point when the sender's cwnd C=8, allowing transmission of 8 TPDUs (arbitrarily numbered TSNs 11-18). Note that when TSN 18 is transmitted, the rtxq grows to fill the entire send buffer.

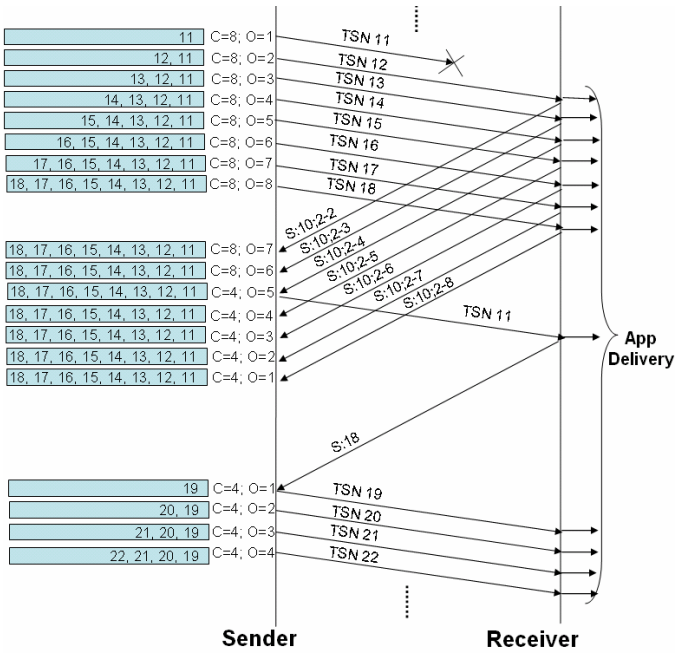

Figure 2: Timeline of an Unordered SCTP Data Transfer using SACKs

In Figure 2's example, TSN 11 is presumed lost in the network. The other TSNs are received out-of-order and immediately SACKed by the SCTP receiver. The SACKs shown have the following format: (S)ACK: CumAckTSN; GapAckStart-GapAckEnd. Each Gap-ack start and Gap-ack end value is relative to the cum-ack value, and together specifies a block of received out-of-order TSNs.

At the sender, the first SACK (S:10;2-2) is also a dupack

and gap-acks TSN 12. Though data corresponding to TSN 12 has been delivered to the receiving application, the SACK does not convey the non-renegable nature of TSN 12, requiring the sender to continue being responsible for the TSN. Starting from the time that this SACK arrives at the sender, the copy of TSN 12 in the sender's rtxq is *unnecessary*. The gap-ack for TSN 12 reduces the amount of outstanding data (O) to 7 TPDUs. Since O<C, the sender could in theory transmit new data, but in practice cannot do so since the completely filled send buffer *blocks* the sending application from writing new data into the transport layer. We call this situation *send buffer blocking*. Note that send buffer blocking prevents the sender from fully utilizing the cwnd.

The second and third dupacks (S:10;2-3, S:10;2-4) further increase the number of unnecessary TSNs in the rtxq. Send buffer blocking continues to prevent new data transmission. On receipt of the third dupack, the sender halves the cwnd (C=4), fast retransmits TSN 11, and enters fast recovery. Dupacks received during fast recovery further increase the amount of unnecessary data in the rtxq, prolonging inefficient rtxq usage. Note that though these dupacks reduce outstanding data (O<C), send buffer blocking prevents transmission of new data.

The sender eventually exits fast recovery when the SACK for TSN 11's retransmission (S:18) arrives. The sender removes the unnecessary copies of TSNs 12-18 from the rtxq, and concludes the current instance of send buffer blocking. Since send buffer blocking prevented the sender from fully utilizing the cwnd before, the new cum ack (S:18) does not increase the cwnd [RFC4960]. The application writes new data into the newly available send buffer space and the sender now transmits TSNs 19-22.

Figure 2 demonstrates the following on transfers involving non-renegable out-of-order data:

- The unnecessary copies of non-renegable out-of-order data waste kernel memory (rtxq). The amount of wasted memory is a function of *flightsize* (amount of data "in flight") during a loss event; a larger flightsize wastes more memory.
- When the rtxq grows to fill the entire send buffer, send buffer blocking ensues, which can degrade throughput.

C. Implications to Concurrent Multipath Transfer (CMT)

A host is multihomed if it can be reached via multiple IP addresses, as is the case when the host has multiple network interfaces. SCTP supports *transport layer multihoming* for fault-tolerance purposes [RFC4960]. SCTP multihoming allows binding of an association to multiple IP addresses at each endpoint. An endpoint chooses a single destination address as the primary destination, which is used for all data traffic during normal transmission. SCTP also monitors the reachability of each destination address. Failure in reaching the primary destination results in failover, where an SCTP endpoint dynamically chooses an alternate destination to transmit the data.

Multiple active interfaces suggest the possibility of multiple independent end-to-end paths between the multihomed hosts. Concurrent Multipath Transfer (CMT) [4] is an experimental extension to SCTP that assumes multiple independent paths, and exploits these paths for *simultaneous* transfer of new data between end hosts. Similar to an SCTP sender, a CMT sender uses a single send buffer and rtxq for data transfer. However, the CMT sender's total flightsize is the sum of flightsizes on each path. Since the amount of wasted kernel memory and the probability of send buffer blocking increase as the transport sender's flightsize increases, we hypothesize that a CMT association is even more likely to suffer from the inefficiencies of the SACK mechanism than an SCTP association.

III. NON-RENEGABLE SELECTIVE ACKNOWLEDGMENTS (NR-SACKS) FOR SCTP

NR-SACKs enable an SCTP receiver to explicitly convey non-renegable information on out-of-order data [5]. NR-SACKs provide the same information as SACKs for SCTP's congestion and flow control, and the sender is expected to process this information identically as in SACK processing. In addition, NR-SACKs report any non-renegable out-of-order TSNs. The sender can use this non-renegable information to free up kernel memory sooner than with SACKs.

A. NR-SACK Chunk Details

The proposed NR-SACK chunk for SCTP is shown in Figure 3. Before sending/receiving NR-SACKs, the endpoints first negotiate NR-SACK usage during association establishment. An endpoint supporting the NR-SACK extension lists the NR-SACK chunk in the Supported Extensions Parameter carried in the INIT or INIT-ACK chunk [RFC5061]. During association establishment, if both endpoints support the NR-SACK extension, then each endpoint acknowledges received data with NR-SACK chunks instead of SACK chunks.

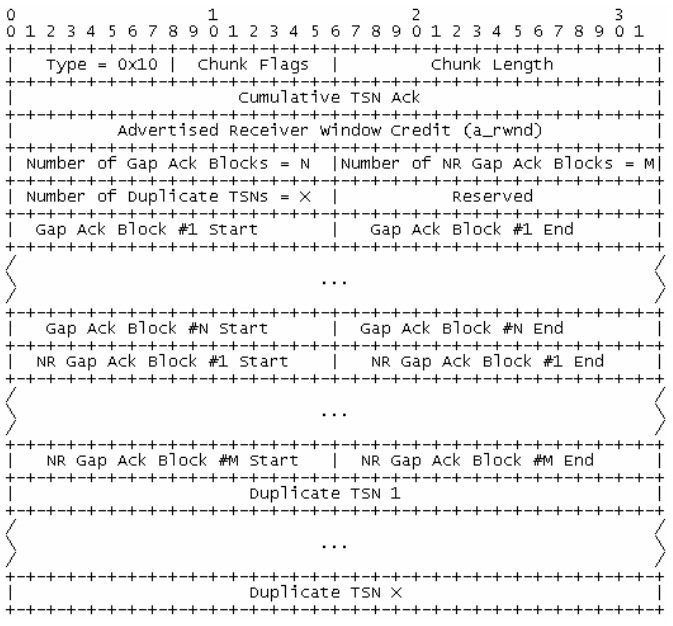

Figure 3: NR-SACK Chunk for SCTP

Since NR-SACKs extend SACK functionality, an NR-SACK chunk has several fields identical to the SACK chunk: the *Cumulative TSN Ack,* the *Advertised Receiver Window Credit (a_rwnd), Gap Ack Blocks,* and *Duplicate TSNs.* These fields have identical semantics to the corresponding fields in the SACK chunk [RFC4960]. NR-SACKs also report non-renegable out-of-order data chunks in the *NR Gap Ack Blocks*, a.k.a. "nr-gap-acks". All data chunks with TSNs ≥ (Cumulative TSN Ack + NR Gap Ack Block Start) and ≤ (Cumulative TSN Ack + NR Gap Ack Block End) of each NR Gap Ack Block are reported as non-renegable. The *Number of NR Gap Ack Blocks (M)* field indicates the number of NR-Gap Ack Blocks included in the NR-SACK chunk.

Note that each sequence of TSNs in an NR Gap Ack Block will be a subsequence of one of the Gap Ack Blocks, and there can be more than one NR Gap Ack Block per Gap Ack Block. Also, non-renegable information cannot be revoked. If a TSN is nr-gap-acked in any NR-SACK chunk, then all subsequent NR-SACKs gap-acking that TSN should also nr-gap-ack that TSN. Complete details of NR-SACK chunk can be found in [5].

B. SCTP Unordered Data Transfer with NR-SACKs

When a receiver guarantees not to renege an out-of-order data chunk and nr-gap-acks the chunk, the sender no longer needs to keep that particular data chunk in its rtxq, thus allowing the data sender to free up kernel memory sooner than if the data chunk were only gap-acked.

Figure 4 is analogous to Figure 2's example, this time using NR-SACKs. The sender and receiver negotiate the use of NR-SACKs during association establishment. As in Figure 2, TSNs 11-18 are initially transmitted, and TSN 11 is presumed lost. For each TSN arriving out-of-order, the SCTP receiver transmits an NR-SACK chunk instead of a SACK chunk. The NR-SACKs in Figure 4 have the following format: (N)R-SACK: CumAckTSN; NRGapAckStart-NRGapAckEnd.

Note that in an unordered data transfer all out-of-order data are non-renegable. The first NR-SACK (N:10;2-2) is also a dupack. This NR-SACK cum-acks TSN 10, and (nr-)gap-acks TSN 12. Once the data sender is informed that TSN 12 is non-renegable, the sender frees up the kernel memory allocated to TSN 12, allowing the application to write more data into the newly available send buffer space. Since TSN 12 is also gap-acked, the amount of outstanding data (O) is reduced to 7, allowing the sender to transmit new data – TSN 19.

On receipt of the 2nd and 3rd dupacks that newly (nr-)gap-ack TSNs 13 and 14, the sender removes these TSNs from the rtxq. On receiving the second dupack, the sender transmits new data – TSN 20. On receipt of the third dupack, the sender halves its cwnd (C=4), fast retransmits TSN 11, and enters fast recovery. Dupacks received during fast recovery (nr-)gap-ack TSNs 15-20. The sender frees rtxq accordingly, and transmits new TSNs 21, 22 and 23. The sender exits fast recovery when the NR-SACK with new cum-ack (N:20) arrives. This new cum-ack increments C=5, and decrements O=3. The sender now transmits new TSNs 24 and 25.

The explicit non-renegable information in NR-SACKs ensures that the rtxq contains only necessary data − TPDUs that are actually in flight or "received and renegable". Comparing Figures 2 and 4, we observe that NR-SACKs use the rtxq more efficiently.

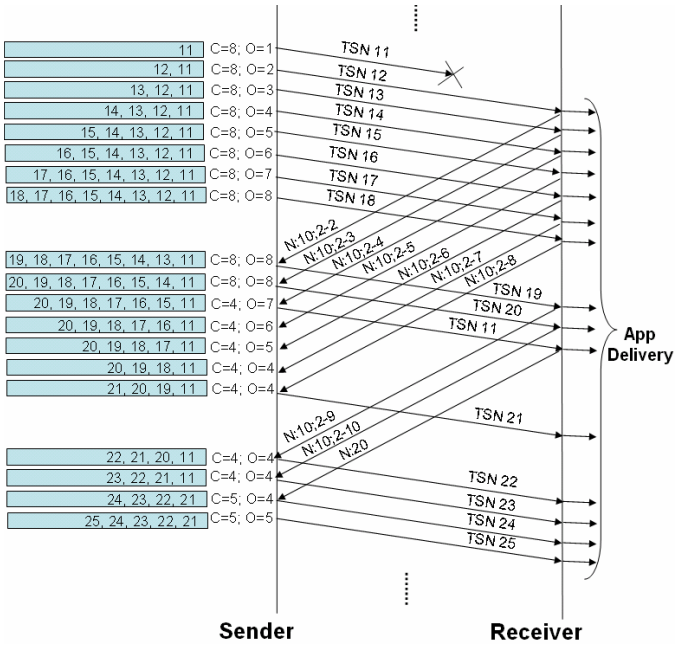

Figure 4: Timeline of an Unordered SCTP Data Transfer using NR-SACKs

IV. Evaluation Preliminaries

Ns-2's SCTP and CMT modules [11, 3] were both extended to support and process NR-SACK chunks. The simulation-based evaluations compare long-lived SCTP or CMT flows using SACKs vs. NR-SACKs under varying cross-traffic loads. This section discusses the experiment setup and other evaluation preliminaries in detail.

A. Simulation Setup

Reference [1] recommends specific simulation scenarios and parameters for performing realistic evaluations of TCP extensions and congestion control algorithms. These recommendations include network topologies, details of cross-traffic generation, and delay distributions mimicking patterns observed in the Internet. We adhere to these recommendations for a realistic evaluation of SACKs vs. NR-SACKs.

The SCTP evaluations use the dumb-bell topology shown in Figure 5, which models the access link scenario specified in [1]. The central bottleneck link connects routers R_1 (left) and R_2 (right), has a 100Mbps capacity, and 2ms one-way propagation delay. Both routers employ drop tail queuing and the queue size is set to the bandwidth-delay product of a 100ms flow. Each router is connected to three cross-traffic generating edge nodes via 100Mbps edge links with the following propagation delays: 0ms, 12ms, 25ms (left) and 2ms, 37ms, 75ms (right). Each left edge node generates cross-traffic destined to every right edge node and vice-versa. Thus, without considering queuing delays, the RTTs for cross-traffic

flows sharing the bottleneck link range from 8ms—204ms.

Reference [1] recommends application level cross-traffic generation over packet level generation, since, in the latter scenario, cross-traffic flows do not respond to the user/application/transport behavior of competing flows. Also, [1] proposes the use of Tmix [8] traffic generator. However, the recommended Tmix *connection vectors* were unavailable at the time of performing our evaluations. Therefore, we decided to employ existing ns-2 application level traffic generation tools, recommended by [7]. Since our simulation setup uses application level cross-traffic, we believe that the general conclusions from our evaluations will hold for evaluations using the Tmix traffic generator.

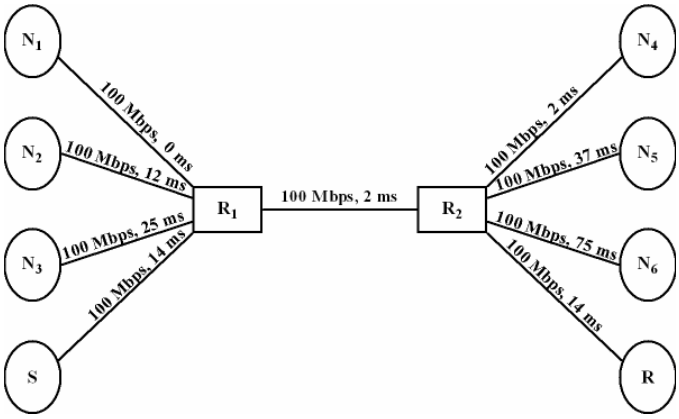

Figure 5: Topology for SCTP Experiments (Topology 1)

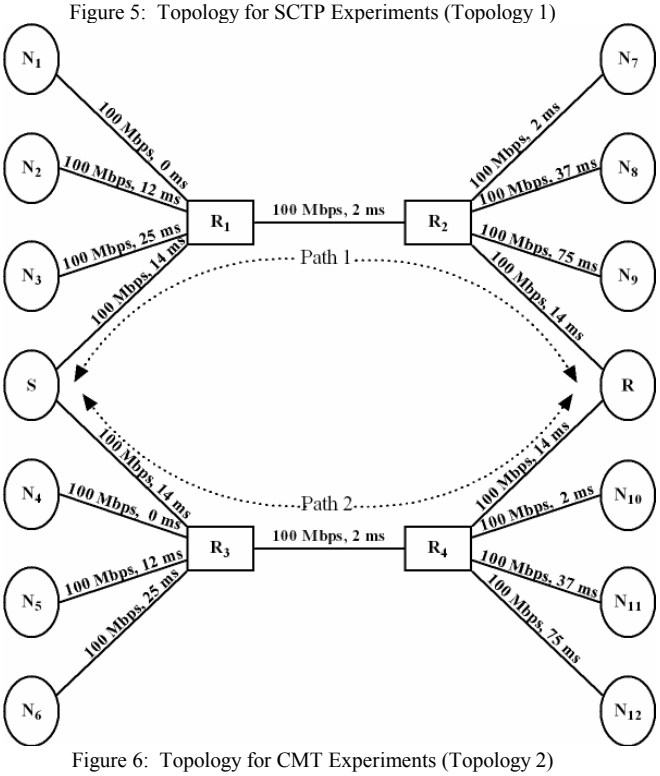

Figure 6: Topology for CMT Experiments (Topology 2)

Cross-traffic generated by three kinds of applications are considered: (i) non-greedy, responsive HTTP sessions − generated by PackMime implementation [2], (ii) rate controlled, unresponsive video sessions over UDP, and (iii) greedy, responsive bulk file transfer sessions over TCP. We are unaware of existing measurement studies on the proportion of each kind of traffic observed in the Internet. Therefore the simulations assume a simple, yet reasonable rule for traffic mix proportion − more HTTP traffic than video or FTP traffic.

Each edge node runs a PackMime session to every edge node on the other side, and the amount of HTTP traffic generated is controlled via the PackMime *rate* parameter. Similarly, each edge node establishes video and FTP sessions to every edge node on the other side, and the number of video/FTP sources at each node impacts the amount of video/FTP traffic. To avoid synchronization issues, the PackMime, video, and FTP sessions start at randomly chosen times during the initial 5 seconds of the simulation. The default segment size for all TCP traffic results in 1500 byte IP PDUs; the segment size for 10% of the FTP flows is modified to result in 576 byte IP PDUs. Also, the PackMime *request* and *response* size distributions are seeded in every simulation run, resulting in a range of packet sizes at the bottleneck [1].

The bottleneck router load is measured as (L) = (mean queue length ÷ total queue size). Four packet-level load/congestion variations are considered: (i) Low (~15% load, < 0.1% loss), (ii) Mild (~45% load, 1-2% loss), (iii) Medium (~60% load, 3-4% loss), (iv) Heavy (~85% load, 8-9% loss).

Topology 1 (Figure 5) is used to evaluate SCTP flows. CMT evaluations are over the dual-dumbbell topology shown in Figure 6 (topology 2), with two independent bottleneck links between routers R_1-R_2 and R_3-R_4. Similar to topology 1, each router in topology 2 is attached to 3 cross-traffic generating edge nodes, with similar bottleneck and edge link bandwidth/delay characteristics. In both topologies, nodes S and R are the SCTP or CMT sender and receiver, respectively. In topology 2, both S and R are multihomed, and the CMT sender uses the two independent paths (paths 1 and 2) for simultaneous data transfer. In both topologies, S and R are connected to the bottleneck routers via 100Mbps duplex edge links, with 14ms one-way delay. Thus the one-way propagation delay experienced by the SCTP or the CMT flow corresponds to 30ms, approximately the US coast-to-coast propagation delay [6].

In both topologies, the bottleneck links experience bi-directional cross-traffic; the cross-traffic load is similar on both forward and reverse directions. In topology 1, the cross-traffic load varies from low to heavy. For CMT evaluations using topology 2, the bottlenecks experience asymmetric path loads; path 1 cross-traffic load varies from low to heavy, while path 2 experiences low load.

The SCTP or CMT flow initiates an unordered data transfer ~18-20 seconds after the simulation begins such that, all data received out-of-order at R is deliverable, and thus, non-renegable. Trace collection begins after a 20 second warm-up period from the start of SCTP or CMT traffic, and ends when the simulation completes after 70 seconds. The CMT sender uses the recommended RTX-SSTHRESH retransmission

policy, i.e., retransmissions are sent on the path with highest ssthresh [4].

B. Metric: Retransmission Queue Utilization

In transfers using SACKs, the rtxq consists of two kinds of data (Figure 2): (i) *necessary* data – data that is either "in flight" and has not yet reached receiver's transport layer, or data that has been received but is renegable by the transport receiver, and (ii) *unnecessary* data – data that has arrived out of order at the receiver and is non-renegable. *The rtxq is most efficiently utilized when all data in the rtxq are necessary.* As the fraction of unnecessary data increases, the rtxq is less efficiently utilized.

The transport sender modifies the rtxq as and when SACKs/NR-SACKs arrive. The rtxq size varies during the course of a file transfer, but can never exceed the send buffer size. For time duration t_i in the transfer, let

r_i = size of rtxq, and

k_i = amount of *necessary* data in rtxq.

During t_i, only $k_i \div r_i$ of the rtxq is efficiently utilized, and the efficiency changes whenever k_i or r_i changes.

Let $\frac{k_0}{r_0}, \frac{k_1}{r_1}, \ldots \frac{k_n}{r_n}$ be the efficient rtxq utilization values during time durations $t_0, t_1, \ldots t_n \left(\sum t_i = T\right)$, respectively. The time weighted efficient rtxq utilization averaged over T is calculated as $RtxQ_Util = \left(\sum t_i \times \frac{k_i}{r_i}\right) \div T$. To measure rtxq utilization, the ns-2 SCTP or CMT sender tracks k_i, r_i and t_i until association shutdown.

Let,

W = time when trace collection begins after the initial warm-up time, and

E = simulation end time.

The time weighted efficient rtxq utilization averaged over the *entire trace collection time, i.e., T = (E – W),* is referred to as *RtxQ_Util*. In an unordered transfer using NR-SACKs, all out-of-order data will be nr-gap-acked and the rtxq should contain only necessary data. Therefore, we expect an SCTP or CMT flow using NR-SACKs to most efficiently utilize the rtxq (*RtxQ_Util* =1) under all circumstances.

C. Retransmission Queue Utilization during Loss Recovery

Typically, in SCTP transfers, data is always received in-order during no loss, unless the intermediate routers reorder packets. Consequently, during no loss, SCTP flows employing either SACKs or NR-SACKs utilize the rtxq most efficiently, and both their *RtxQ_Util*=1. The two ack mechanisms differ in rtxq usage only when data is received out-of-order, which ensues when an SCTP flow suffers packet losses. Specifically, in SCTP, *the duration of NR-SACKs' impact on the rtxq is limited to loss recovery periods.*

To evaluate the impact of the two ack schemes during loss recovery periods, the ns-2 SCTP sender timestamps every entry/exit to/from loss recovery. Since none of the routers reorder packets in our simulations, the SCTP sender uses the following naive rule − sender enters loss recovery on the receipt of SACKs/NR-SACKs with at least one gap-ack block, and exits loss recovery on the receipt of SACKs/NR-SACKs with a new cum-ack and zero gap-acks. We find that this simple rule results in a good approximation of the actual loss recovery periods. In addition to *RtxQ_Util*, an SCTP sender also tracks *RtxQ_Util_L*, which corresponds to rtxq utilization averaged over only the loss recovery durations of trace collection.

Depending on the paths' bandwidth/delay characteristics, a CMT association experiences data reordering even under no loss conditions [4]. Data transmitted on the shorter RTT path will be received out-of-order w.r.t. data transmitted on other path(s). Therefore, the naïve rule mentioned above cannot be employed to estimate entry/exit of CMT sender's loss recovery. Therefore, the CMT sender tracks only *RtxQ_Util*.

V. Results

For each type of sender (SCTP or CMT), different send buffer sizes imposing varying levels of memory constraints are considered: 32K, 64K and INF (infinite or unconstrained space) for SCTP, and 128K, 256K and INF for CMT. The results are averaged over 30 runs, and plotted with 95% confidence intervals. In the following discussions, SCTP transfers using SACKs or NR-SACKs are referred to as SCTP-SACKs and SCTP-NR-SACKs, respectively. Similarly, CMT using SACKs or NR-SACKs are referred to as CMT-SACKs and CMT-NR-SACKs.

A. Retransmission Queue Utilization

As the end-to-end path gets more congested, SCTP-SACKs' *RtxQ_Util_L* remains fairly consistent ~0.5 (Figure 7), while the *RtxQ_Util* decreases (Figure 8). The *RtxQ_Util_L* values indicate that irrespective of path loss rate, SCTP-SACKs efficiently utilize only ~50% of rtxq during loss recovery; ~50% of rtxq is wasted buffering *unnecessary* data.

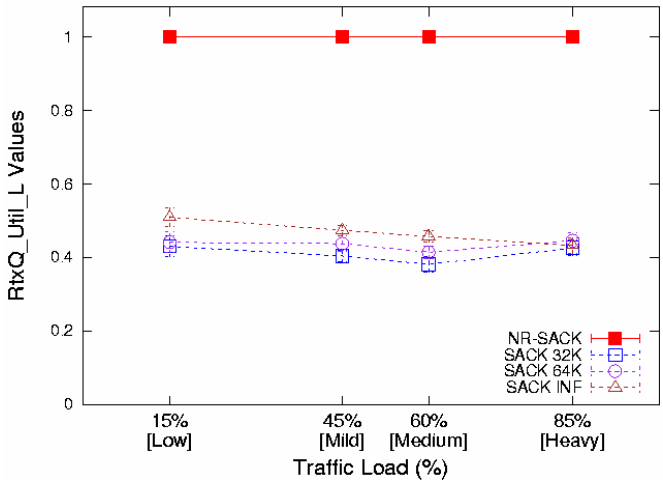

Figure 7: Rtxq Utilization during Loss Recovery in SCTP

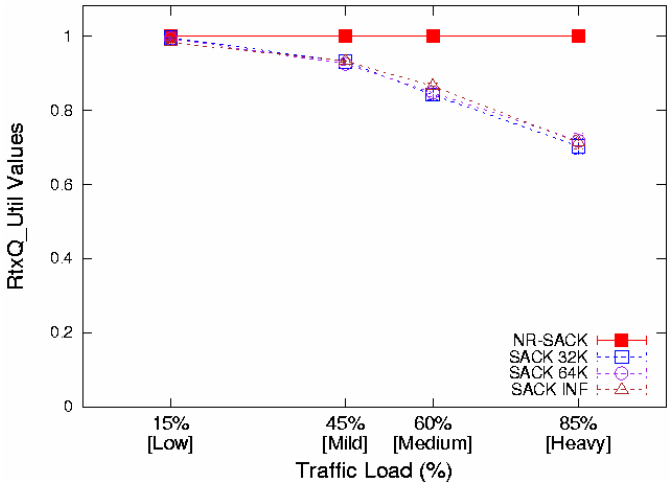

Figure 8: Rtxq Utilization in SCTP

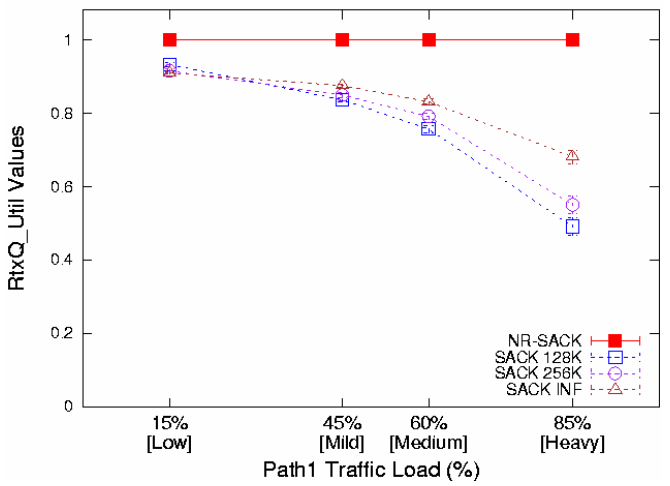

Figure 9: Rtxq Utilization in CMT

At lower congestion levels (lower cross-traffic), the frequency of loss events and the fraction of transfer time spent in loss recovery are smaller, resulting in negligible rtxq wastage during the entire trace collection period (*RtxQ_Util*). As loss recoveries become more frequent, SCTP-SACKs' inefficient rtxq utilization during loss recovery lowers the corresponding *RtxQ_Util* values (Figure 7). Results show that SCTP-SACKs waste on average ~20% of the rtxq during moderate congestion and ~30% during heavy congestion conditions. The amount of wasted kernel memory increases as the number of transport connections increase, and can be significant at a server handling large numbers of concurrent connections, such as a web server.

By definition of the *RtxQ_Util* metric, NR-SACKs are expected to utilize the rtxq most efficiently, even during loss recovery periods (Section IVB). The simulation results confirm this hypothesis. *RtxQ_Util* values for both SCTP-NR-SACKs and CMT-NR-SACKs are unity.

In CMT evaluations, path 2 experiences low traffic load, while path 1's traffic load varies from low to heavy (Figure 6). Recall that a CMT sender transmits data concurrently on both paths. Asymmetric path congestion levels aggravate data reordering in CMT. As path 1 congestion level increases, TPDU losses on the higher congested path 1 cause data transmitted on the lower congested path 2 to arrive out-of-order at the receiver. CMT congestion control is designed such that losses on path 1 do not affect the cwnd/flightsize on path 2 [4]. While losses on path 1 are being recovered, a sender continues data transmission on path 2, increasing the amount of non-renegable out-of-order data in the rtxq. As the paths become increasingly asymmetric in their congestion levels, the amount of non-renegable out-of-order data in the rtxq increases, and brings down CMT-SACKs' *RtxQ_Util* values (Figure 9).

Increasing the send buffer/rtxq space improves SCTP-SACKs' or CMT-SACKs' kernel memory (rtxq) utilization only to a certain degree. In Figures 8 and 9, *RtxQ_Util* for INF send buffer is essentially the upper bound on how efficient SCTP or CMT employing SACKs utilizes rtxq. Therefore, we conclude that *TPDU reordering results in inevitable rtxq wastage in transfers using SACKs. The amount of wasted memory increases as TPDU reordering and loss recovery durations increase.*

Smaller send buffers further degrade *RtxQ_Util_L* and *RtxQ_Util* values. This degradation is more pronounced in CMT (Figure 9) than in SCTP. Further investigation reveals this effect to be due to send buffer blocking, discussed next.

B. *Send Buffer Blocking in CMT*

When the rtxq grows to fill the entire send buffer, send buffer blocking ensues, preventing an application from writing new data into the transport layer (Section IIA). In both SCTP and CMT, send buffer blocking increases as the send buffer is more constrained, i.e., decreases in size. In addition, CMT employs multiple paths for data transfer, increasing a sender's total flightsize in comparison to SCTP. Therefore, we hypothesized that CMT would suffer more send buffer blocking than SCTP (Section IIC). Indeed, in the simulations, CMT suffered significant send buffer blocking even for 128K and 256K send buffer sizes. In this section, we focus on the effects of send buffer blocking in CMT.

CMT using either ack scheme suffers from send buffer blocking for 128K and 256K buffer sizes. In CMT-SACKs, send buffer blocking continues until the cum-ack point moves forward, i.e., until loss recovery ends. As path 1 congestion level increases, more frequent timeout recoveries cause longer causing longer loss recovery durations. Therefore, as congestion increases, the CMT-SACKs sender is blocked for larger fractions of transfer time. On the other hand, send buffer blocking in CMT-NR-SACKs is unaffected by the congestion level on path 1. As and when NR-SACKs arrive (on path 2), a CMT-NR-SACK sender removes nr-gap-acked data from the rtxq, allowing transmission of more new data. CMT-SACKs' longer send buffer blocking durations adversely impact performance as discussed next.

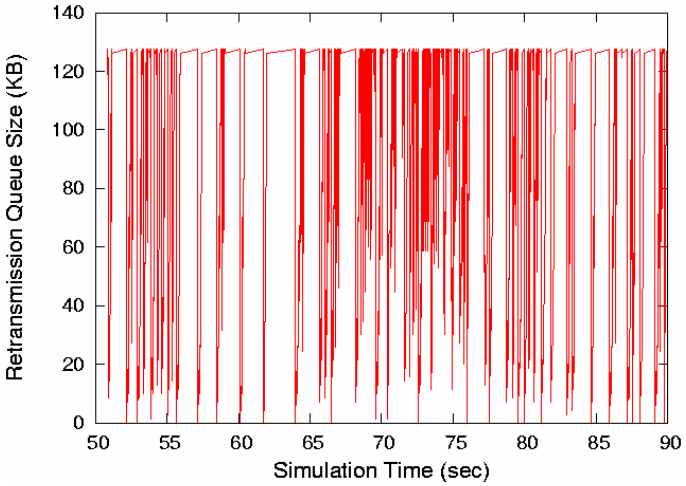

Figure 10: Retransmission Queue Evolution in CMT-SACKs

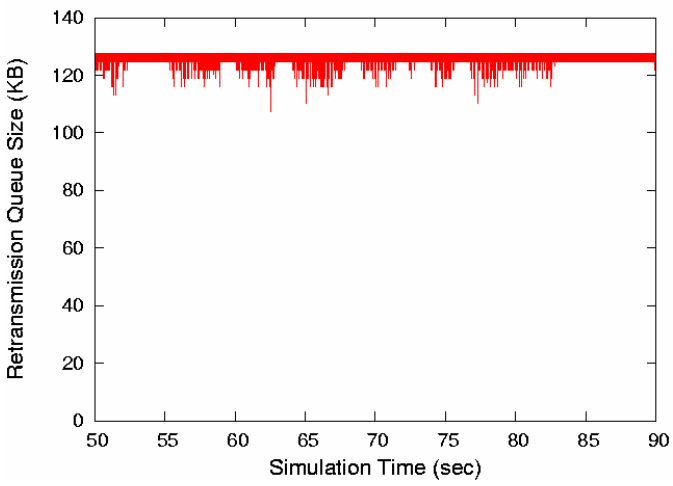

Figure 11: Retransmission Queue Evolution in CMT-NR-SACKs

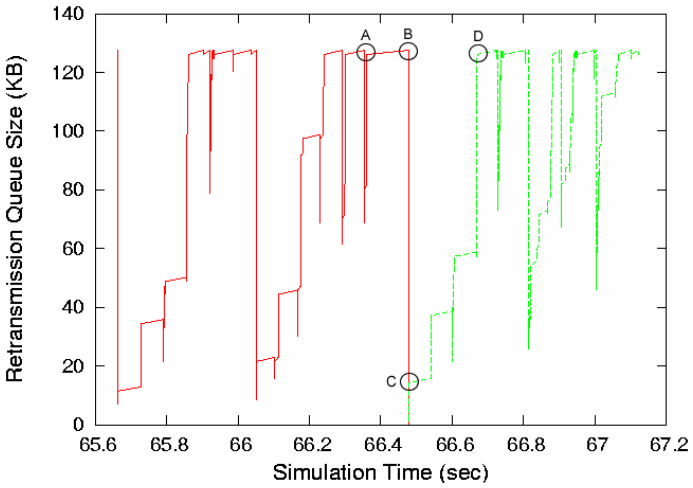

Figure 12: Retransmission Queue Evolution in CMT-SACKs (1.5 sec)

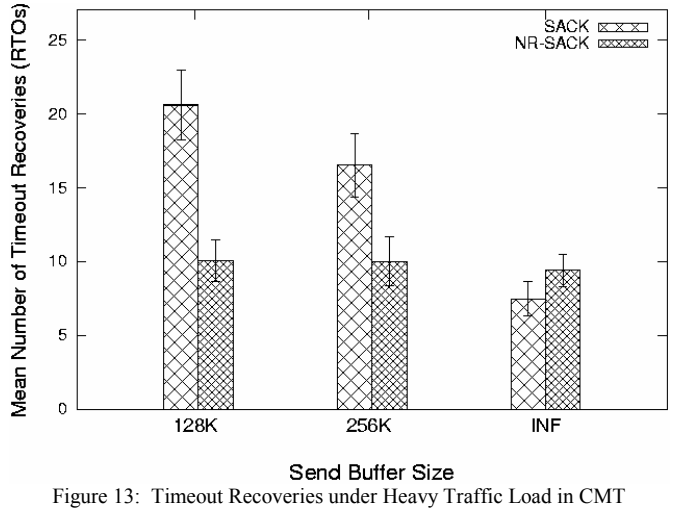

Figure 13: Timeout Recoveries under Heavy Traffic Load in CMT

1) Ineffective Use of Send Buffer Space

Send buffer blocking limits rtxq growth and reduces throughput. The negative impact on throughput is minimized when the available send buffer space is utilized *as much as possible*.

Figures 10 and 11 illustrate CMT sender's rtxq evolution over 40 seconds of a transfer using SACKs and NR-SACKs, respectively. The figures show that both CMT-SACKs and CMT-NR-SACKs suffer from send buffer blocking − the maximum rtxq size in the figures corresponds to 100% of send buffer (128K). However, the rtxq evolution in CMT-SACKs (Figure 10) exhibits more variance – reaches the maximum and drops to 0 multiple times, while CMT-NR-SACKs' rtxq size stays closer to 128K most of the time (Figure 11).

Figure 12 is a zoom of CMT-SACKs' rtxq evolution over an arbitrary 1.5 second period. At point A (time 66.36sec), rtxq size hits the maximum, and the sender is blocked from transmitting any more data. Subsequent SACKs reduce the amount of outstanding data, but send buffer blocking prevents the sender from clocking out new data. At time 66.42sec, path 1's retransmission timer expires; the sender detects loss, and retransmits TSN 20369 on path 2. At time 66.48sec (point B), sender receives a SACK with a new cum-ack (TSN=20457) and completely clears rtxq contents, ending the current instance of send buffer blocking. The sender immediately transmits new data on both paths, and the rtxq evolution after the new cum-ack (TSN=20457) is shown by the dashed line. The cwnd on path 1 allows transmission of 2 MTU sized TPDUs (TSNs 20458 and 20459). The cwnd on path 2 is 127162 bytes, but the *Maxburst* parameter [RFC4960] limits the sender to transmit only 4 MTU sized TPDUs − TSNs 20460-20463. Once the sender transmits data on both paths, rtxq size increases to ~8.6K, shown by point C. Subsequent SACKs allow more data transmission and at point D the sender's rtxq reaches the maximum causing the next instance of send buffer blocking.

Though CMT-NR-SACKs (Figure 11) also incurs send buffer blocking, nr-gap-acks free up rtxq space allowing the sender to steadily clock out more new data. *A constrained send buffer is better utilized, and the transmission is less bursty with NR-SACKs than SACKs.* The improved send buffer use contributes to throughput improvements (discussed below).

2) Retransmission Queue Utilization

In Figure 9, CMT-SACKs' *RtxQ_Util* worsens as send buffer blocking increases (send buffer size decreases). As discussed earlier, in CMT-SACKs, send buffer blocking

prevents transmission of new data until loss recovery ends. Lack of new data transmission resulted in fewer and sometimes insufficient acks to trigger fast retransmits. Consequently, blocked CMT-SACKs experienced more timeout recoveries (RTOs) at heavy traffic loads than non-blocked CMT-SACKs (Figure 13). As the send buffer is more constrained, the average number of RTOs increase, and the fraction of transfer time spent in loss recovery increases. Longer loss recovery durations increase the duration of inefficient rtxq utilization, and further lower blocked CMT-SACKs' *RtxQ_Util* values than non-blocked (INF) CMT-SACKs' *RtxQ_Util*.

On the other hand, CMT-NR-SACKs steadily clock out new data, and do not incur excessive RTOs during send buffer blocking. CMT-NR-SACKs' mean number of RTOs for 128K and 256K buffer sizes are similar to the INF case (Figure 13). To conclude, *send buffer blocking worsens CMT-SACKs' rtxq utilization. Blocked CMT-SACKs' inefficient send buffer usage increases the number of timeout recoveries, and degrades throughput when compared to CMT-NR-SACKs.*

3) Throughput

When the send buffer never limits rtxq growth (INF send buffer size), both CMT-SACKs and CMT-NR-SACKs experience no send buffer blocking, and perform similarly (Figure 14). However, CMT-SACKs achieve the same throughput as CMT-NR-SACKs at the cost of larger rtxq sizes.

Using terminology defined in Section IVB, the average rtxq size, *RtxQ* over the entire trace collection period (*T*) is calculated as, $RtxQ = \left(\sum t_i \times r_i\right) \div T$. Figure 15 plots CMT-SACKs vs. CMT-NR-SACKs *RtxQ* for the unconstrained send buffer (INF) case. As path 1 cross-traffic load increases, the bandwidth available for the CMT flow decreases, and CMT-NR-SACKs' *RtxQ* decreases (Figure 15). Similarly, CMT-SACKs' *RtxQ* decreases as traffic load increases from low to mild. However, a different factor dominates and increases CMT-SACKs' *RtxQ* during medium and heavy traffic conditions. Since the rtxq growth is never constrained in the INF case, the CMT sender transmits as much data as possible on path 2 while recovering from losses on path 1. At medium and heavy cross-traffic loads, loss recovery durations increase due to increased timeout recoveries, and the CMT-SACKs sender transmits more data on path 2 compared to mild traffic conditions. This factor increases CMT-SACKs' *RtxQ* during medium and heavy traffic conditions.

Returning to Figure 14, when the send buffer size limits rtxq growth, CMT-NR-SACKs' efficient rtxq utilization alleviates send buffer blocking, and CMT-NR-SACKs perform better than CMT-SACKs. *The throughput improvements in CMT-NR-SACKs increase as conditions that aggravate send buffer blocking increases.* That is, NR-SACKs improve throughput more as send buffer becomes more constrained and/or when the paths become more asymmetric in the congestion levels. Alternately, *CMT-NR-SACKs achieve similar throughput as CMT-SACKs using smaller send buffer sizes.* For example, during mild, medium and heavy path 1 cross-traffic load, CMT-NR-SACKs with 128K send buffer performs similar or better than CMT-SACKs with 256K send buffer. Also, CMT-NR-SACKs with 256K send buffer performs similar to CMT-SACKs with larger (unconstrained) send buffer.

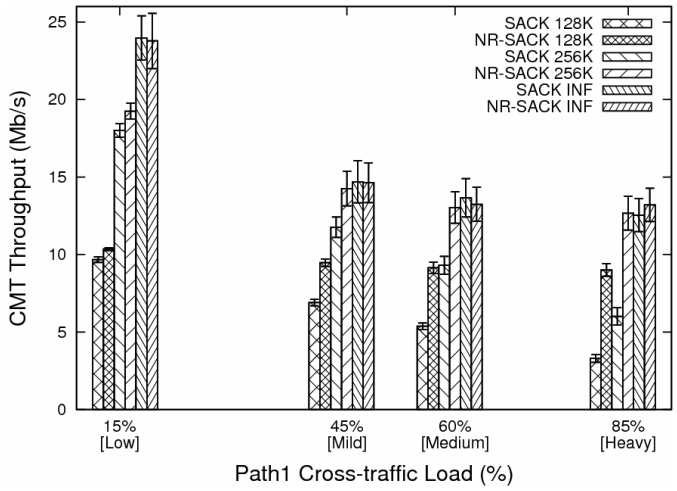

Figure 14: CMT- SACKs vs. CMT-NR-SACKs Throughput

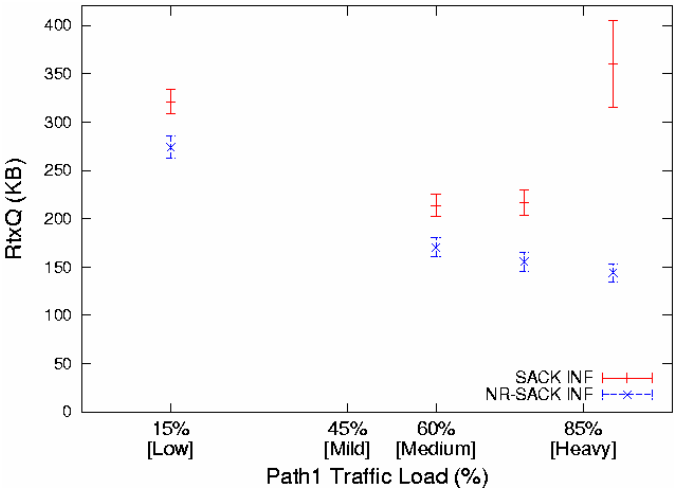

Figure 15: CMT-SACKs vs. CMT-NR-SACKs Average Rtxq Size

VI. Conclusion & Future Work

This work investigated the effect of existing SACK mechanism when data received out-of-order is non-renegable. We conclude that SACKs cause inevitable sender memory wastage, which worsens as data reordering and loss recovery durations increase. We proposed NR-SACKs which enable the transport receiver to convey non-renegable information about out-of-order data.

Note that *a transfer employing NR-SACKs never performs worse than a transfer using SACKs*. When out-of-order data is non-renegable, NR-SACKs perform better than SACKs. Simulations confirmed that in both SCTP and CMT, NR-SACKs utilize send buffer and rtxq space most efficiently. Send buffer blocking in CMT with SACKs adversely impact

end-to-end performance, while efficient send buffer use in CMT with NR-SACKs alleviates send buffer blocking. Therefore, NR-SACKs not only reduce sender's memory requirements, but also improve throughput in CMT. NR-SACKs are being pursued as an experimental extension to SCTP in the IETF [5]. We are in the process of implementing NR-SACKs in FreeBSD.

We plan to investigate the impact of asymmetric path delays in CMT with SACKs. Asymmetric path delays can aggravate send buffer blocking in CMT with SACKs, and we expect NR-SACKs to alleviate this blocking.

ACKNOWLEDGMENTS

The authors acknowledge the valuable comments and suggestions from Dr. Philip Conrad at the University of California Santa Barbara; and Jonathan Leighton and Joseph Szymanski at the University of Delaware Protocol Engineering Lab.

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] L. Andrew, C Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, I. Rhee. Towards a Common TCP Evaluation Suite. 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), March 2008.

[2] J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith, M.C, Weigle, "Stochastic Models for Generating Synthetic HTTP Source Traffic", INFOCOM, March 2004.

[3] N. Ekiz, P. Natarajan, J. Iyengar, A. Caro, "ns-2 SCTP Module," Version 3.7, September 2007.

[4] J. Iyengar, P. Amer, R. Stewart, "Concurrent Multipath Transfer using SCTP Multihoming over Independent End-to-end Paths," IEEE/ACM Transactions on Networking, 14(5), pp. 951-964, October 2006.

[5] P. Natarajan, P. Amer, E. Yilmaz, R. Stewart, J. Iyengar, "Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP," Internet Draft, draft-natarajan-tsvwg-sctp-nrsack (work in progress).

[6] S. Shakkottai, R. Srikant, A. Broido, K. Claffy, "The RTT distribution of TCP Flows in the Internet and its Impact on TCP-based flow control," TR, CAIDA, February 2004.

[7] G. Wang, Y. Xia, D. Harrison, "An NS2 TCP Evaluation Tool: Installation Guide and Tutorial," April 2007. URL: http://labs.nec.com.cn/tcpeval.htm.

[8] M. C. Weigle, P. Adurthi, F. Hern´andez-Campos, K. Jeffay, and F. D. Smith, "Tmix: a Tool for Generating Realistic TCP Application Workloads in ns-2," SIGCOMM Computer Communication Review (CCR), 36(3), pp. 65–76, July 2006.

[9] FreeBSD TCP and SCTP Implementation, URL: http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/netinet/#dirlist.

[10] J. Iyengar, P. Amer, R. Stewart, "Performance implications of a bounded receive buffer in concurrent multipath transfer," Computer Communications, 30(4), pp. 818-829, February 2007.

[11] Ns-2 documentation and software, Version 2.33, March 2008. URL: www.isi.edu/nsnam/ns.