Concurrent Multipath Transfer Using SCTP Multihoming*†

Janardhan R. Iyengar, Keyur C. Shah, Paul D. Amer
Protocol Engineering Lab, Computer and Information Sciences, University of Delaware
{iyengar, shah, amer}@cis.udel.edu
Randall Stewart
Cisco Systems, rrs@cisco.com

ABSTRACT

We propose CMT - Concurrent Multipath Transfer using the Stream Control Transmission Protocol (SCTP). CMT uses SCTP's multihoming feature to simultaneously transfer new data across multiple end-to-end paths to the receiver. Through ns-2 simulations, we observe significant reordering at the receiver due to CMT. We identify three negative side-effects of reordering introduced by CMT that must be managed before the full performance gains of parallel transfer can be achieved: (i) unnecessary fast retransmissions at the sender, (ii) reduced cwnd growth due to fewer cwnd updates at the sender, and (iii) more ack traffic due to fewer delayed acks. We propose three algorithms which augment and/or modify current SCTP to counter these side-effects and present initial simulations indicating correctness of the proposed solutions. In this work, we operate under the strong assumptions that the receiver's advertised window does not constrain the sender, and that the bottleneck queues on the end-to-end paths used in CMT are independent of each other.

Keywords: SCTP, CMT, Multihoming, Load Sharing, Transport Protocols

1 INTRODUCTION

Multihoming among networked machines and devices is a technologically feasible and increasingly economical proposition. A host is *multihomed* if it can be addressed by multiple IP addresses [7], as is the case when the host has multiple network interfaces. Though feasibility alone does not determine adoption of an idea, multihoming can be expected to be the rule rather than the exception in the near future. For instance, cheaper access to the Internet may motivate a home user to have simultaneous connectivity through multiple ISPs. Wireless devices may be simultaneously connected through multiple access technologies. More and more machines will have wired and wireless connections. The use of multihoming increases a host's fault tolerance at an economically feasible cost. Multiple active interfaces also suggest the *simultaneous* existence of multiple paths between the multihomed hosts. In this paper, we propose using these multiple paths between multihomed source and destination hosts through *Concurrent Multipath Transfer (CMT)* to increase throughput for a networked application. *CMT* is the simultaneous transfer of new data from a source host to a destination host via two or more end-to-end paths. In our initial efforts, we assume that the bottleneck queues on the end-to-end paths are independent of each other.

The current transport protocol workhorses, TCP and UDP, are ignorant of multihoming; TCP allows binding to only one network address at each end of a connection. At the time TCP was designed, network interfaces were expensive components, and hence multihoming was beyond the ken of research. Increasing economical feasibility and a desire for networked applications to be fault tolerant at an end-to-end level, have brought multihoming within the purview of the transport layer. In this paper, we investigate CMT at transport layer, using transport layer multihoming. As opposed to the application layer, CMT at the transport layer is desirable since the transport layer, being the first end-to-end layer, has finer information about the end-to-end path(s). Further, CMT at the application layer would increase complexity at the transport-application interface, due to continuous information exchange between the transport and the application.

Two recent transport layer protocols, the Stream Control Transmission Protocol (SCTP) [22], and the Datagram Congestion Control Protocol (DCCP) [16] support multihoming

*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.

†Supported in part by the University Research Program of Cisco Systems, Inc.

at the transport layer. The motivation for multihoming in DCCP is mobility, while SCTP is driven by a broader and more generic application base, which includes fault tolerance and mobility. Of the two, we use SCTP primarily because it is a reliable protocol (and due to our expertise with it). The issues presented in this paper and the corresponding algorithms should be applicable to CMT using other reliable, SACK-based transport layer protocols; some issues are applicable to unreliable protocols as well.

SCTP is an IETF standards track transport layer protocol. SCTP multihoming allows binding of one transport layer *association* (SCTP's term for a connection) to multiple IP addresses at each end of the association. This binding allows an SCTP sender to send data to a multihomed receiver through different destination addresses. Due primarily to insufficient research in the area, simultaneous transfer of new data to multiple destination addresses is currently not allowed in SCTP. In this paper, we investigate CMT at the transport layer using SCTP as a reliable, multihome-aware, SACK-based transport layer protocol.

In Section 2 we briefly describe SCTP mechanisms relevant to CMT, our simulation setup and assumptions, and the graphs presented here. In Sections 3, 4, and 5, we present three negative side-effects of reordering with CMT, and propose algorithms to avoid these side-effects. Though the simulations in these sections represent specific cases, they should be viewed as illustrations of the larger issues described. Section 6 concludes our work presented in this paper with the current focus of our research. Section 7 describes related work in the area of concurrent multipath transfer (or load sharing).

2 PRELIMINARIES

We first present an overview of select ideas and mechanisms used by SCTP, also in comparison with TCP to highlight relevant similarities and differences.

SCTP is defined in RFC2960 [22] with changes and additions included in the SCTP Implementer's Guide [21]. An SCTP packet consists of one or more concatenated building blocks called *chunks*: either control or data. For the purposes of reliability and congestion control, each data chunk in an association is assigned a unique Transmission Sequence Number (TSN), similar in function to sequence numbers in TCP. Since SCTP is message-oriented and chunks are atomic, TSNs are associated only with chunks of data, as opposed to a TCP bytestream which associates a sequence number with each byte of data. In our simulations, we assume one data chunk per packet for ease of illustration; each packet thus carries, and is associated with a single TSN.

SCTP uses a selective ack scheme similar to SACK TCP. SCTP's congestion control algorithms are based on RFC2581 [3], and include SACK-based mechanisms for better performance. Similar to TCP, SCTP uses three control variables: receiver's advertised window (rwnd), sender's congestion window (cwnd), and sender's slow start threshold (ssthresh). However, unlike TCP, SCTP's cwnd reflects how much data can be sent, not which data to send. In SCTP, rwnd is shared across an association. Unlike in TCP, SCTP uses a separate set of congestion control parameters (cwnd and ssthresh, among others) *per destination* since each destination address may result in a different path to the destination. Currently, due to lack of research in CMT, RFC2960 does not allow a sender to simultaneously send *new* data on multiple paths; an SCTP sender maintains a *primary destination* to which all transmissions of new data are sent (Note: retransmissions are sent to alternate destinations).

Thus far, in our investigation of CMT, we assume independent paths. Though we use disjoint paths from sender to receiver in the simulation results presented in this paper, our premise of independent paths means that the paths have separate bottlenecks. Overlap in the paths is acceptable, but bottlenecks are assumed independent. We also assume that the rwnd is large enough to not constrain the sender. This assumption enables us to study cwnd dynamics with CMT, without introducing the dynamics of rwnd sharing across different paths. Our initial simulations also do not have any loss. Even without loss, conventional mechanisms such as cwnd growth and roundtrip time (RTT) estimation mechanisms, are significantly affected by CMT. We will relax these unrealistic constraints in our continued efforts with CMT.

The simulations presented in this paper use the University of Delaware's SCTP module for ns-2 [4, 8]. The simulation setup has two dualhomed hosts, sender A with local addresses A_1, A_2, and receiver B with local addresses B_1, B_2. The hosts are connected by two separate paths: Path 1 ($A_1 - B_1$), and Path 2 ($A_2 - B_2$) whose end-to-end available bandwidths are 0.2 Mbps and 1 Mbps, respectively. The roundtrip propagation delay on both paths is 70 milliseconds, which roughly reflects the U. S. coast-to-coast delay. The CMT sender (host A) uses a scheduling algorithm that sends new data to a destination as soon as its corresponding cwnd allows new data to be sent.

The simulation results described in the paper (Figures 1, 3, 5, and 7) all show cwnd evolution with time. The figures have four curves, which show the CMT sender's (1) cwnd evolution for destination B_1 ($+$), (2) cwnd evolution for destination B_2 ($\times$), (3) aggregate cwnd evolution (sum of (1) and (2)) ($\triangle$), and (4) expected aggregate cwnd evolution (–). The expected aggregate cwnd evolution curve is obtained as the sum of the cwnd evolution curves of two independent SCTP runs, using B_1 and B_2 as the primary destination, respectively.

We now introduce some notation which is used in this paper; the meaning and usage of this notation will be clear as the reader progresses through the paper. CMT refers to a host involved in concurrent multipath transfer using current SCTP. CMT$_s$, CMT$_c$, and CMT$_d$ refer to a host involved in CMT using SCTP with the SFR-CACC algorithm (Section 3), the Cwnd Update for CMT (CUC) Algorithm (Section 4) and the Delayed Ack for CMT (DAC) algorithm (Section 5), respectively. Using more than one subscript suggests inclusion of more than one algorithm. For instance, CMT$_{sc}$ refers to a host using CMT with the SFR-CACC and CUC algorithms.

3 FAST RETRANSMISSIONS WITH CMT

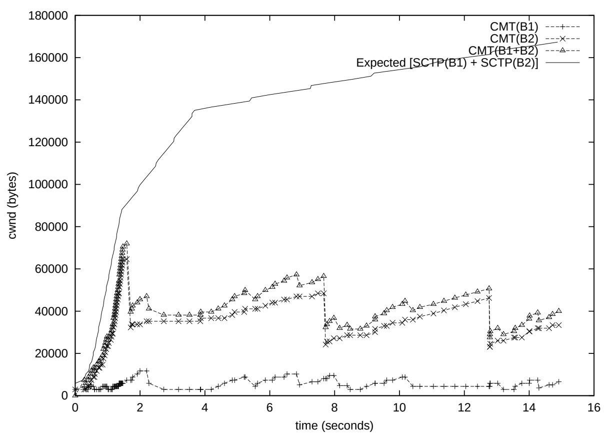

Figure 1: CMT with SCTP: Evolution of the different cwnds

When multiple paths being used for CMT have disparate delay and bandwidth characteristics, additional packet reordering is observed at the receiver. When reordering is observed, a receiver sends gap reports through SACKs to the sender, and the sender uses the gap reports to detect loss through the fast retransmission procedure [3, 22]. With CMT, the observed reordering not due to loss can be significant enough to trigger unnecessary fast retransmissions [14], which has two negative consequences: (1) Since each retransmission is assumed to occur due to a congestion loss, the sender reduces its cwnd for the destination on which the retransmitted data was outstanding, and (2) the cwnd overgrowth problem [13] causes a sender's cwnd to grow aggressively for the destination on which the retransmissions are sent, due to acks received for original transmissions.

Figure 1 shows how unnecessary fast retransmissions can significantly hinder cwnd growth. The aggregate cwnd growth obtained by CMT is much slower than by even a single SCTP association on any of the two paths (not shown). Note that all cwnd reductions seen are due to unnecessary fast retransmissions; no packet loss was simulated.

For our proposed solution, we suggest a different interpretation of SACK information. Conventional understanding of a SACK chunk in SCTP (or ack with SACK option in TCP) is that gap reports imply loss. The probability of a gap report indicating loss increases with the number of gap reports received for the same TSN (or sequence numbers in TCP). With CMT, we suggest that the SACK information be treated as a concise description of the TSNs received thus far by the receiver. Hence, a loss may not be immediately obvious from just SACK information. In other words, gap reports do not necessarily imply a lost TSN; the sender infers lost TSNs using information in SACKs, *and* history information in the retransmission queue.

The proposed solution to address the side-effect of incorrect cwnd evolution due to unnecessary fast retransmissions is the Split Fast Retransmit Changeover Aware Congestion Control (SFR-CACC) algorithm, shown in Figure 2. This algorithm is based on a previous incarnation which could not handle *cycling changeover* [14], and hence could not be directly applied to CMT. This revised SFR-CACC is simpler, and is applicable to CMT as well as to single changeover. SFR-CACC introduces a *virtual queue* per destination within the sender's retransmission queue. The sender then uses SACK information in conjunction with history information in the retransmission queue to correctly deduce missing reports for a TSN by inferring cumulative ack and gap report information per destination.

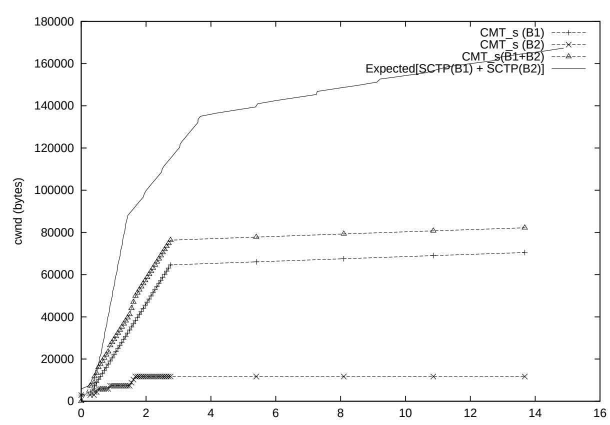

Figure 3: Including the SFR-CACC algorithm (CMT$_s$): Evolution of the different cwnds

On receipt of a SACK containing gap reports [Sender side behavior]:

1) initialize *cacc_saw_newack* = FALSE for all destination addresses;
2) **for** each TSN t_a being acked that has not been acked in any SACK thus far **do**
   - (i) let d_a be the destination to which t_a was sent;
   - (ii) set $d_a.cacc\_saw\_newack$ = TRUE;
3) $\forall$ destinations d_n, set $d_n.highest\_in\_sack\_for\_dest$ to highest TSN being newly acked on d_n;
4) to determine whether missing report count for a TSN t_m should be incremented:
   - (i) let d_m be the destination to which t_m was sent;
   - (ii) **if** ($d_m.cacc\_saw\_newack$ = TRUE) **and** ($d_m.highest\_in\_sack\_for\_dest > t_m$) **then**
     increment missing report count for t_m;
     else do not increment missing report count for t_m;

NOTE 1: The HTNA algorithm [21] does not need to be applied separately, since step (4) covers the function of the HTNA algorithm.
NOTE 2: This SFR-CACC algorithm requires that after retransmission due to a timeout, the retransmitted TSN must be made ineligible for a further fast retransmission.

Figure 2: SFR-CACC Algorithm – Eliminating unnecessary fast retransmissions

In SFR-CACC, two variables are introduced per destination:

1. *highest_in_sack_for_dest* - stores the highest TSN acked per destination by the SACK being processed.
2. *cacc_saw_newack* - a flag used during the processing of a SACK to infer the causative TSN(s)'s destination(s). Causative TSNs for a SACK are those TSNs which caused the SACK to be sent.

Figure 3 shows cwnd evolution for CMT including the SFR-CACC algorithm, i.e., CMT$_s$. We see that SFR-CACC eliminates the unnecessary fast retransmissions, reflected by the absence of unnecessary cwnd reductions in the graph. The aggregate cwnd growth obtained by CMT$_s$ while better, is still slower than expected; this slower growth with CMT is addressed in the next section.

4 CWND UPDATES WITH CMT

The cwnd evolution algorithm for SCTP [22] (and also TCP [3]) dictates growth in cwnd only when a new cum ack is received by the sender. In other words, when SACKs with unchanged cum acks are received (say due to reordering), a sender does not modify its cwnd. This mechanism again reflects the conventional view that a SACK which does not advance the cum ack indicates possibility of loss.

Figure 3 illustrates that there still remains reduced cwnd growth with CMT$_s$. The aggregate cwnd growth for CMT$_s$ (CMT$_s$[B1+B2] in Figure 3) is slower than expected. We will now discuss reasons for this behavior. Since a CMT$_s$ receiver observes reordering, many SACKs are sent containing new gap reports but not new cum acks. When reported gaps are later filled by a new cum ack, cwnd growth occurs, but only for the newly acked data. The data previously acked through gap reports will not contribute to cwnd growth. Even though data may have reached the receiver "in-order per destination", without changing the current SCTP cwnd management process, the updated cwnd will not reflect this fact.

This inefficient behavior can be attributed to SCTP's current design principle that the cum ack in the SACK, which tracks the latest TSN received in-order at the receiver, applies to an entire association, not per destination. TCP and current SCTP use only one destination address at any given time to transmit new data to, and hence, this design principle works fine. Since CMT uses multiple destinations simultaneously, cwnd growth in CMT demands tracking the latest TSN received in-order *per destination*. This information is not coded directly in a SACK. A sender must infer cum ack per destination, possibly through SACKs and history information in the retransmission queue.

We also note from Figure 3 that of the constituent paths, cwnd growth for destination B_2 (which is the higher bandwidth path, Path 2) is stunted; in fact, the cwnd ceases to increase after some initial growth. This behavior, which may be specific to this illustration, is attributed to the fact that though data gets through at a faster rate to destination B_2, the sender receives most of the new cum acks (and after a while all of the new cum acks) from destination B_1. This causes the cwnd for

Initialize *find_exp_pseudo_cumack* = TRUE at beginning of the association;
On receipt of a SACK [Sender side behavior]:

1) $\forall$ destinations d, reset *d.new_pseudo_cumack* = FALSE;
2) **if** the SACK carries a new cum ack **then**
   for each TSN t_c being cum acked for the first time, that was not acked through prior gap reports **do**
      (i) let d_c be the destination to which t_c was sent;
      (ii) set $d_c.find\_exp\_pseudo\_cumack$ = TRUE;
      (iii) set $d_c.new\_pseudo\_cumack$ = TRUE;
3) **if** gap reports are present in the SACK **then**
   for each TSN t_p being processed from the retransmission queue **do**
      (i) let d_p be the destination to which t_p was sent;
      (ii) **if** ($d_p.find\_exp\_pseudo\_cumack$ = TRUE) **and** t_p was not acked in the past **then**
         $d_p.exp\_pseudo\_cumack = t_p$;
         $d_p.find\_exp\_pseudo\_cumack$ = FALSE;
      (iii) **if** t_p is acked via gap reports for first time **and** ($d_p.exp\_pseudo\_cumack = t_p$) **then**
         $d_p.new\_pseudo\_cumack$ = TRUE;
         $d_p.find\_exp\_pseudo\_cumack$ = TRUE;
4) **for** each destination d **do**
   if (*d.new_pseudo_cumack* = TRUE) **then** update cwnd according to [21, 22];

Figure 4: Cwnd Update gor CMT (CUC) Algorithm – Handling side-effect of reduced cwnd growth due to fewer cwnd updates

destination B_2 to grow considerably slower than expected.

We propose a cwnd growth algorithm to track the earliest outstanding TSN *per destination* and update the cwnd, even in the absence of new cum acks. The algorithm uses SACKs and history information to deduce in-order delivery per destination. In understanding our proposed solution, again bear in mind that gap reports do not (necessarily) imply a missing TSN; SACK information is treated only as a concise description of the TSNs received thus far by the receiver.

Figure 4 shows the proposed Cwnd Update for CMT (CUC) algorithm. We propose the idea of a *pseudo-cumack* that tracks the earliest outstanding TSN per destination at the sender. The sender tracks changes in the pseudo-cumack of each destination using SACKs and history information in the retransmission queue. An advance in a pseudo-cumack is used by a sender to trigger a cwnd update for the corresponding destination. Thus, if a SACK causes the pseudo-cumack for a destination to be advanced, then the cwnd for that destination is updated, even when the actual cum ack is not advanced. The pseudo-cumack should be used only for cwnd updates; only the actual cum ack can be used for dequeuing data in the sender's retransmission queue since a receiver can reneg on data that has been acked through gap reports, but not cumulatively acked. In the CUC algorithm (Figure 4), three variables are introduced per destination:

1. *exp_pseudo_cumack* - maintains next expected pseudo-cumack at a sender.
2. *new_pseudo_cumack* - flag used to indicate if a new pseudo-cumack has been received.
3. *find_exp_pseudo_cumack* - flag used to find a new expected pseudo-cumack. This flag is set after a new pseudo-cumack has been received.

Figure 5 shows cwnd growth for CMT$_{sc}$. The figure shows that the CUC algorithm resolves the side-effect of reduced cwnd growth due to fewer cwnd updates. Of significant interest is the observation that the aggregate cwnd obtained by CMT$_{sc}$ *exceeds* the expected aggregate cwnd. This unexpected behavior is discussed at the end of Section 5.

5 DELAYED ACKS WITH CMT

SCTP specifies that a receiver should use the delayed ack algorithm as given in RFC2581 while acknowledging data. Specifically, RFC2581 states that "Out-of-order data segments SHOULD be acknowledged immediately..." With CMT's frequent reordering, this rule causes an SCTP receiver to frequently not delay acks. Hence a negative side-effect

On receipt of a data packet [Receiver side behavior]:

1) delay sending an ack as given in [22], with the additional change that acks should be delayed even if reordering is observed.
2) in each ack, report number of data packets received since sending of previous ack.

When incrementing missing report count through SFR-CACC:Step 4(ii) (Figure 2) [Sender side behavior]:

1) let t_m be the TSN for which missing reports should be incremented;
2) let d_m be the destination to which t_m was sent;
3) **if** ($d_m.cacc\_saw\_newack$ = TRUE) **then**
    if (∀ destinations d_o such that $d_o \neq d_m$, $d_o.cacc\_saw\_newack$ = FALSE) **then**
    /** all newly acked TSNs were sent to the same destination as t_m **/
        if (∃ newly acked TSNs t_b, t_a such that $t_b < t_m < t_a$) **then**
            (conservatively) increment missing report count for t_m by 1;
        else if (∀ newly acked TSNs t_a, $t_a > t_m$) **then**
            increment missing report count for t_m by number of packets reported by receiver;
    else /** Mixed SACK - newly acked TSNs were sent on multiple destinations **/
        (conservatively) increment missing report count for t_m by 1;

Figure 6: Delayed Ack for CMT (DAC) Algorithm – Handling side-effect of increased ack traffic

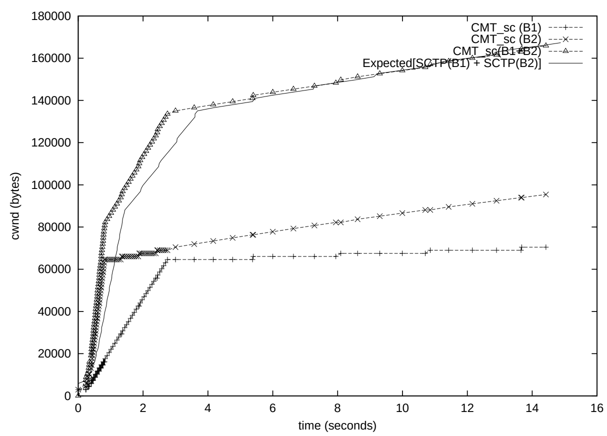

Figure 5: Including the CUC algorithm (CMT$_{sc}$): Evolution of the different cwnds

of reordering with CMT is increased ack traffic on the return path. To prevent this increase in ack traffic, we suggest that a CMT receiver ignore the rule mentioned above. That is, a CMT receiver does not immediately ack an out-of-order packet, but delays the ack. Though this modification at the receiver eliminates the observed increase in ack traffic, the rule from RFC2581 mentioned above has another purpose which gets hampered.

According to RFC2851, "Out-of-order data segments SHOULD be acknowledged immediately, in order to accelerate loss recovery. To trigger the fast retransmit algorithm, the receiver SHOULD send an immediate ... ACK when it receives a data segment above a gap in the sequence space." In SCTP, four acks with gap reports for a missing TSN (i.e., four missing reports for a TSN) suggest that the receiver received at least four data packets sent after the missing TSN. Receipt of four missing reports for a TSN triggers the fast retransmit algorithm at the sender. In other words, the sender has a *reordering threshold* (or *dupack threshold* in TCP terminology) of four packets. Since a CMT receiver cannot distinguish between loss and reordering introduced by a CMT sender, the modification suggested above by itself would cause the receiver to delay acks even in the face of loss. Consequently, when a loss does occur, fast retransmit would be triggered at the CMT sender only if the receiver receives at least seven data packets sent after a lost TSN. Thus, the effective reordering threshold at the sender increases to at least seven packets.

This effective increase in reordering threshold at the sender can be countered by reducing the actual number of acks required to trigger a fast retransmit at the sender. In other words, if a sender can increment the number of missing reports more accurately per ack received, fewer acks will be required to trigger a fast retransmit. The receiver can provide more information in each ack to assist the sender in accurately inferring the number of missing reports per ack for a lost TSN.

We suggest that in each ack, a receiver report the count of data packets received since the previous ack was sent. The final algorithm to enable delayed acks with CMT is given in Figure 6. This algorithm specifies a receiver's behavior on receipt of data, and also a sender's behavior when the missing report count for a TSN needs to be incremented.

Since SCTP (and TCP) acks are cumulative, loss of an ack will result in loss of the data packet count reported by the receiver, but the TSNs acked will be acknowledged by the following ack. Receipt of this following ack can cause ambiguity in inferring missing report count per destination. As shown in Figure 6, our algorithm conservatively assumes a single missing report count per destination in such ambiguous cases.

Figure 7 shows cwnd evolution for CMT$_{sc}$ after including the Delayed Ack for CMT (DAC) Algorithm, i.e., CMT$_{scd}$. We observe that cwnd growth remains almost the same as in Figure 5, but the amount of ack traffic (not shown) is reduced with CMT$_{scd}$.

We still observe that aggregate cwnd growth of CMT$_{scd}$ exceeds the expected aggregate cwnd growth, a surprising positive side-effect. To recap, the expected aggregate cwnd is the sum of the cwnd growth of two independent SCTP runs, each using one of the two destination addresses as its primary destination. The number of acks received in the CMT$_{scd}$ simulation run is the same as the total number of acks received in both the SCTP simulation runs. In the SCTP runs, each delayed ack can increase the cwnd by at most one MTU during slow start, even if the ack acknowledges more than one MTU worth of data.

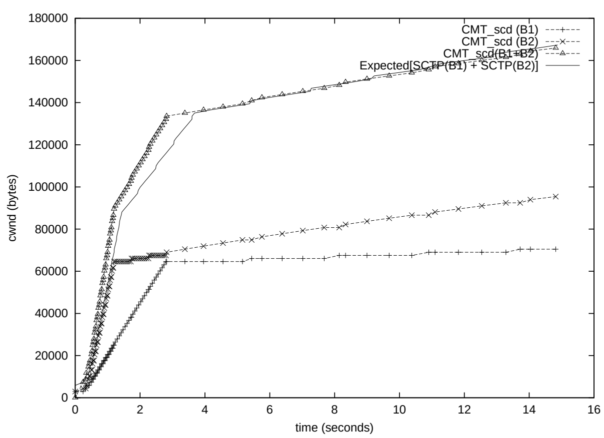

Figure 7: Including the DAC algorithm (CMT$_{scd}$): Evolution of the different cwnds

On the other hand, in the CMT$_{scd}$ run, if a delayed ack simultaneously acknowledges an MTU of data on each of the two destinations, the sender can simultaneously increase the two cwnds by one MTU each. Thus, a single delayed ack that acknowledges the data flows on the two paths can cause an aggregate cwnd growth of two MTUs. From analyzing the traces, we conclude that such delayed acks which simultaneously contribute to the cwnd growth of the two destinations cause the aggregate cwnd growth of CMT$_{scd}$ to exceed the expected aggregate cwnd growth.

Though the aggregate cwnd growth exceeds expected aggregate cwnd growth, we argue that the sender is not aggressive. The sender does not create bursts of data during slow start, and tries to build up the ack clock as expected. The sender is able to clock out more data due to delayed acks that acknowledge data flows on multiple paths.

6 CONCLUSION AND FUTURE WORK

We have identified three negative side-effects of introducing CMT with SCTP, and propose algorithms to avoid these side-effects. We show that initial simulation results indicate correctness of the proposed algorithms. The side-effects presented in this paper and the corresponding algorithms should be applicable to concurrent multipath transfer using other reliable, SACK-based transport layer protocols; some issues may be applicable to unreliable protocols as well. With simulations, we are currently evaluating CMT$_{scd}$ using different combinations of bandwidth, delay and lossrate on the paths.

There are some other effects which we feel may demand attention in our continued efforts. For instance, a positive synergy exists between the paths used for CMT; acks sent later on the faster path may reach the sender prior to acks sent earlier on the slower path. The acks received on the faster path also carry information about data received on the slower path due to cumulative information contained in the acks. Thus, the slower path will experience a faster cwnd growth due to a faster return path, and consequently a smaller effective RTT. We suspect that the impact of this phenomenon would be higher when paths with largely different end-to-end delays are used. This phenomenon also suggests a negative side-effect – spurious timeouts may occur due to an inaccurate RTT estimate for the slower path, requiring reevaluation of the RTT estimation algorithm (see Figure 8). We plan to investigate the effects of this phenomenon on RTT estimation at the sender.

We observe that inefficient sharing of rwnd at the sender may reduce the performance benefits of CMT$_{scd}$. We believe that a shared and limited receiver's advertised window (rwnd) will become a performance bottleneck, specifically when the

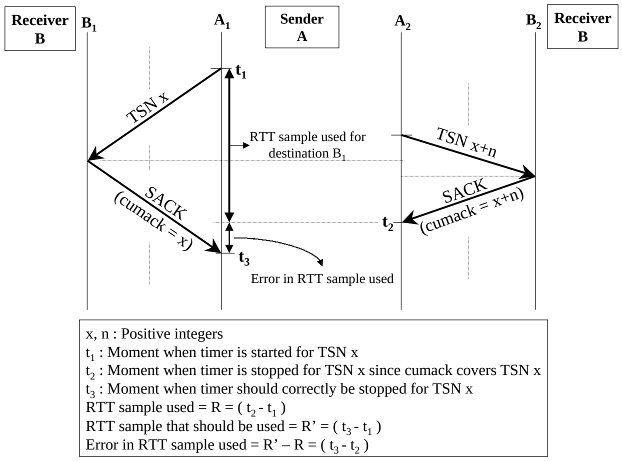

Figure 8: Erroneous RTT Calculation at CMT Sender

paths have different loss rates. We plan to investigate different rwnd sharing mechanisms. We are continuing work on CMT. Currently, we are looking into different retransmission policies for a CMT sender. We then plan on investigating rwnd sharing mechanisms, and then investigating end-to-end techniques for shared bottleneck detection [12, 19] to enable the sender to dynamically decide from either shared or distinct congestion control across paths.

7 RELATED WORK

Mao et al. [17] extend RTP (Realtime Transport Protocol) to support use of multiple paths in *Multi-path Realtime Transport Protocol (MRTP)*, an application layer protocol which could use one of TCP, SCTP or UDP as transport. MRTP specifies session establishment and maintenance mechanisms and scheduling mechanisms over multiple paths, possibly using SCTP multihoming or UDP. The authors propose, as one option, to use SCTP multihoming for simultaneously using multiple paths. This work is complementary to CMT-SCTP work, since the authors provide motivation and an application that would benefit from using CMT-SCTP in a multipath environment.

Al et al. [2] suggest ideas for *load sharing with SCTP*, but requires that more metadata be added to the packets. We believe that the SCTP (and TCP-SACK) packets already contain sufficient information for the data sender to infer the information that [2] explicitly codes as metadata into the packets. The authors introduce new sequence numbers to maintain per-path ordering information, but fail to suggest modified procedures for mechanisms which are immediately affected, such as initialization of the per-path sequence numbers, association initialization and shutdown procedures with multiple sequence numbering schemes, and response to reneging by a receiver.

Phatak [18] proposes distributing data at the network (IP) layer transparent to the higher layers using IP-in-IP encapsulation. Under the questionable assumption that that end-to-end delays are dominated by transmission delay, Phatak identifies conditions under which this mechanism would work without triggering incorrect retransmission timeouts. Phatak fails to adequately address key issues such as reordering and relevance in propagation delay dominated paths or paths with dynamically changing bandwidths and/or delays.

Blanton et al. [5], Zhang et al. [24] and Bohacek et al. [6] describe algorithms to eliminate the effects of reordering due to the network. With CMT, we discuss reordering introduced at the sender, not in the network. The sender has more information about sender introduced reordering, and can hence address this reordering more effectively. [5, 24] can be applied to CMT independently, since they address reordering introduced by the network.

Gerla et al. [11] show that *TCP Westwood* [9], which uses a different mechanism from TCP Reno [3] for bandwidth estimation, is robust to packet reordering introduced by the network. [11] also demonstrates that TCP Westwood is capable of obtaining aggregated throughput when the network layer uses multiple paths, but does not discuss performance in the presence of a shared bottleneck. Gerla et al.[11] assume that multiple paths at the network layer will be optimally utilized by the routing infrastructure. There exist scenarios where the end user has knowledge of and control over only the multihomed endpoints but not the intermediate routers, such as in the Internet. In such cases the endpoint cannot dictate or govern use of multiple paths in the network, but can certainly distribute traffic over the multiple end-to-end paths that may be available at the endpoint, thus motivating transport layer CMT.

Research in link layer load balancing, also known as inverse multiplexing or link aggregation [1, 10, 15, 20, 23] has generally not been end-to-end, and the operating conditions do not represent the conditions that end-to-end CMT over the Internet has to operate in.

8 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] H. Adiseshu, G. Parulkar, and G. Varghese. A Reliable and Scalable Striping Protocol. In *ACM SIGCOMM 1996*, Stanford, California, August 1996.

[2] A. Abd El Al, T. Saadawi, and M. Lee. Load Sharing in Stream Control Transmission Protocol. draft-ahmed-lssctp-00.txt, Internet Draft (expired), IETF, May 2003.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC2581, IETF, April 1999.

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

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

[6] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for Persistent Packet Reordering. In *IEEE ICDCS 2003*, Rhode Island, May 2003.

[7] R. Braden. Requirements for Internet hosts–communication layers. RFC1122, IETF, October 1989.

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

[9] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sansadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wireless Networks. *Wireless Networks Journal*, (8):467–479, March 2002.

[10] The ATM Forum Technical Committee. Inverse Multiplexing for ATM (IMA) Specification, Version 1.0, July 1997. AF-PHY-0086.000.

[11] M. Gerla, S. S. Lee, and G. Pau. TCP Westwood Simulation Studies in Multiple-Path Cases. In *SPECTS 2002*, San Diego, California, July 2002.

[12] K. Harfoush, A. Bestavros, and J. Byers. Robust Identification of Shared Losses Using End-to-End Unicast Probes. In *ICNP 2000*, Osaka, Japan, October 2000.

[13] J. Iyengar, A. Caro, P. Amer, G. Heinz, and R. Stewart. SCTP Congestion Window Overgrowth During Changeover. In *SCI 2002*, Orlando, FL, July 2002.

[14] J. Iyengar, A. Caro, P. Amer, G. Heinz, and R. Stewart. Making SCTP More Robust to Changeover. In *SPECTS 2003*, Montreal, Canada, July 2003.

[15] D. A. Khotimsky. A Packet Resequencing Protocol for Fault-Tolerant Multipath Transmission with Non-Uniform Traffic Splitting. In *IEEE GLOBECOM 1999*, Rio de Janeiro, Brazil, December 1999.

[16] E. Kohler, S. Floyd, M. Handley, and J. Padhye. Datagram Congestion Control Protocol (DCCP). draft-ietf-dccp-spec-04.txt, Internet Draft (work in progress), IETF, June 2003.

[17] S. Mao, D. Bushmitch, S. Narayanan, and S. S. Panwar. MRTP: A Multi-Flow Realtime Transport Protocol for Ad Hoc Networks. In *IEEE Vehicular Technology Conference*, Orlando, Florida, October 2003.

[18] D. S. Phatak and T. Goff. A Novel Mechanism for Data Streaming Across Multiple IP Links for Improving Throughput and Reliability in Mobile Environments. In *IEEE INFOCOM 2002*, New York, NY, June 2002.

[19] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows Via End-to-end Measurement. *IEEE/ACM Transactions on Networking*, 10(3), June 2002.

[20] A. C. Snoeren. Adaptive Inverse Multiplexing for Wide-Area Wireless Networks. In *IEEE GLOBECOM 1999*, Rio de Janeiro, Brazil, December 1999.

[21] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, and M. Tuexen. Stream Control Transmission Protocol (SCTP) Implementer's Guide. draft-ietf-tsvwg-sctpimpguide-08.txt, Internet Draft (work in progress), IETF, March 2003.

[22] 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. RFC2960, IETF, October 2000.

[23] C.B.S. Traw and J. M. Smith. Striping Within the Network Subsystem. *IEEE Network*, 9(4):22–32, July 1995.

[24] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. Technical Report TR-02-006, ICSI, July 2002.