CONCURRENT MULTIPATH TRANSFER USING TRANSPORT LAYER MULTIHOMING: PERFORMANCE UNDER VARYING BANDWIDTH PROPORTIONS*†

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

Randall Stewart
Cisco Systems
rrs@cisco.com

ABSTRACT

We have previously specified three algorithms for the Stream Control Transmission Protocol (SCTP) resulting in CMT$_{scd}$ - a protocol that uses SCTP's multihoming feature for correctly transferring data between multihomed end hosts using multiple separate end-to-end paths. In this paper, we evaluate CMT$_{scd}$ against Application Striping (AppStripe) under uniform losses varying from 0 to 10 percent. We also present results encouraging the use of multiple paths in lossy networks such as battlefield networks. 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$_{scd}$ are independent of each other.

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, 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. In battlefield networks in particular, where fault tolerance is crucial, multihoming will likely become an essential feature in most network elements. Figure 1 shows one such battlefield scenario where the endpoints are multihomed. These nodes may be simultaneously connected through multiple access technologies, and even multiple end-to-end paths to increase resilience to path failure. Multiple active interfaces suggest the simultaneous existence of multiple paths between the multihomed hosts. 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.

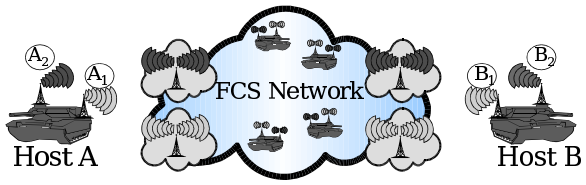

Figure 1: Example Multihoming Topology

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 con-

*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.

nection. 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. 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). CMT at the application layer will increase redundancy and room for error by requiring each application programmer to separately implement CMT in the application. Further, complexity at the transport-application interface will increase due to continuous information exchange between the transport and the application.

The Stream Control Transmission Protocol (SCTP) [4, 7, 8] is an IETF standards track transport layer protocol that natively supports multihoming at the transport layer for fault tolerance and mobility. 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.

We have previously specified three algorithms for SCTP resulting in CMT$_{scd}$ - a protocol that uses SCTP's multihoming feature for *correctly* transferring data between the multihomed end hosts using multiple separate end-to-end paths [5]. In this paper, we evaluate the performance of CMT$_{scd}$ under zero and higher loss conditions. In Section 2 we briefly describe the simulation topology and *Application Striping (AppStripe)*, a simulated application that performs ideal load balancing across multiple end-to-end paths using multiple SCTP associations. In Sections 3 and 4, we present our evaluation of CMT$_{scd}$ vs. AppStripe under zero and higher loss conditions. We conclude the paper in Section 5 with a summary of the results and some observations on the use of multiple paths in battlefield networks.

2 EVALUATION METHODOLOGY

Without CMT$_{scd}$, a multihoming-aware application can choose to perform CMT at the application layer by distributing data across multiple transport layer associations, one on each path to a receiver. The throughput obtained by such an application depends on the scheduling algorithm that the application uses to distribute data across the associations. We propose *Application-Striping (AppStripe)*, a hypothetical multihome-aware application that achieves the highest throughput achievable by an application that distributes data across multiple SCTP associations. An application can, on the other hand, use CMT$_{scd}$, allowing the transport layer to perform distribution of data across paths. To evaluate the performance and further the case for using CMT$_{scd}$, we compare AppStripe, against an application using CMT$_{scd}$. Figure 2 schematically describes AppStripe and CMT$_{scd}$. End-to-end load balancing is performed at the application layer by AppStripe, and at the transport layer by CMT$_{scd}$.

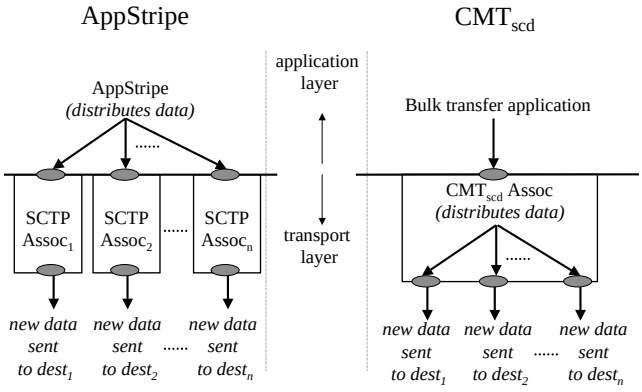

Figure 2: Schematic - AppStripe and CMT$_{scd}$

Since we do not have an ideal scheduling mechanism for AppStripe to distribute data across the two SCTP associations, we simulate an 8MB file transfer by AppStripe as follows: Step 1: We transfer 8MB files using two SCTP associations separately, one association one each path from the sender to the receiver; Step 2: We analyze the traces side by side to locate the point in time when the *total* data transferred by the two associations *in-order* together sums up to 8MB. This time is recorded as the optimal (i.e., minimum) time that would be needed by AppStripe to transfer an 8MB file. We use a filesize of 8MB since

it is large enough to enable the associations to spend much of their lifetime in the congestion avoidance phase, and small enough to maintain zero loss (for evaluation under zero loss) by avoiding buffer overflow at any intermediate buffers between the sender and the receiver.

We have implemented CMT$_{scd}$ using the University of Delaware's SCTP module for the ns-2 simulator [1, 3]. We use the simulation topology shown in Figure 3 for our evaluation. The simulation setup has two dualhomed hosts, sender A with local addresses A1, A2, and receiver B with local addresses B1, B2. The hosts are connected by two separate paths: Path1 (A1 - B1), and Path2 (A2 - B2). The roundtrip propagation delay on both paths is 70 milliseconds. We vary the end-to-end available bandwidth, but maintain a constant sum of the bandwidth-delay products on the two paths to obtain our results. We use a constant sum of bandwidth-delay products to observe the effect of varying bandwidth of the paths within a constant sum. Since we are interested in the sum of throughputs obtained on the paths by AppStripe or CMT$_{scd}$, varying the bandwidth within a constant sum of bandwidth-delay products provides insight into how bandwidth alone influences throughput in these simulations. Ideally, with a constant sum of bandwidth-delay products, the sum of throughputs obtained on the paths should remain constant as well. For a fixed delay of 70 milliseconds on both paths, we maintain the sum of bandwidths of the paths constant at 40Mbps. For a given set of network parameters, we compare:

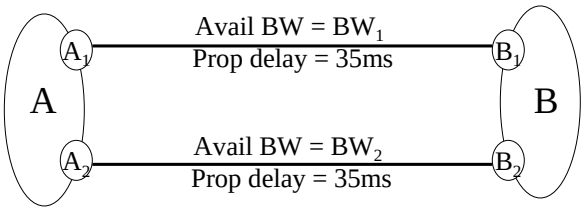

Figure 3: Simulation Topology

- the time taken to transfer an 8MB file using AppStripe. This measurement, as a base reference, represents the best performance that can be obtained by an application that uses multiple SCTP associations to distribute traffic on the two paths.
- the time taken to transfer a 8MB file using CMT$_{scd}$.

We now present evaluation of CMT$_{scd}$ vs. AppStripe under zero loss conditions (Section 3), and under lossy conditions (Section 4).

3 CMT VS. APPSTRIPE: EVALUATION UNDER ZERO LOSS

In Figure 4, the X-axis shows the difference in bandwidth between Path 1 and Path 2. The sum of the bandwidths is set to 40Mbps; at 0 on the X-axis, each path has 20Mbps bandwidth. The one way delay is set to 35ms on each path. Figure 4 compares the time take to transfer an 8MB file using CMT$_{scd}$ with the time taken to transfer an 8MB file using AppStripe.

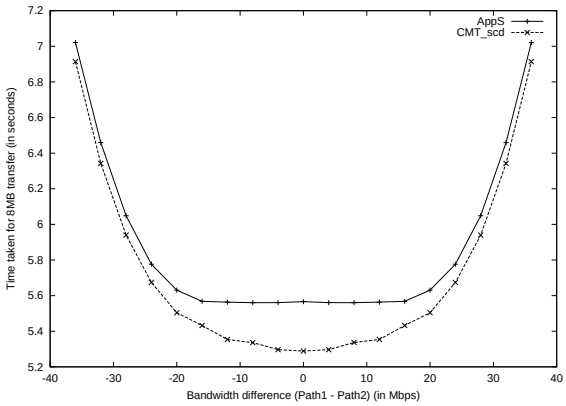

Figure 4: CMT$_{scd}$ vs. AppStripe under zero loss

At all bandwidth allocations between the two paths, CMT$_{scd}$ performs better than AppStripe. This improved performance is primarily due to a faster rate of cwnd growth during slow start with CMT due to acks for data on multiple paths. To understand this phenomenon better, note that in AppStripe, which uses two SCTP associations, each delayed ack can increase the cwnd by at most one MTU during slow start in each association, even if the ack acknowledges more than one MTU worth of data. With delayed acks, it is likely that an ack acknowledges more than one MTU worth of data. On the other

hand, a CMT$_{scd}$ receiver delays acks across the entire CMT$_{scd}$ association, i.e., without regard to which interface packets are received on. Thus, in CMT$_{scd}$, 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[1]. From analyzing the traces, we conclude that such delayed acks which simultaneously contribute to the cwnd growth of the two destinations cause CMT$_{scd}$ to perform better than AppStripe. In the absence of delayed acks, we have seen that both curves overlap exactly.

To understand the overall *'U'* shape of either of the AppStripe or CMT$_{scd}$ curves, consider the following two scenarios -
Scenario 1: two concurrent SCTP associations over paths each with end-to-end available bandwidth X bps, and
Scenario 2: a single SCTP association over one path with end-to-end available bandwidth 2X bps.

If we assume the same end-to-end propagation delay for all paths and same rwnd for both scenarios, it can be seen that in the slow start phase, Scenario 1 will have a cwnd growth rate faster than the Scenario 2. The following steps through the first few stages in the cwnd growth for the two scenarios may clarify this point. Assuming delayed acks result in one ack for every two data packets received:

1. At time 0: In Scenario 1, aggregate cwnd is 4 MTUs (2 MTUs per destination). In Scenario 2, aggregate cwnd is 2 MTUs.
2. After 1 RTT: In Scenario 1, aggregate cwnd is 6 MTUs. In Scenario 2, aggregate cwnd is 3 MTUs.
3. After 2 RTTs: In Scenario 1, aggregate cwnd is 8 MTUs. In Scenario 2, aggregate cwnd is 4 MTUs.

Such cwnd increase continues until the sender enters congestion avoidance. With our simulation parameters, the sender enters congestion avoidance after the cwnd increases beyond the rwnd. We can thus see that there is an faster rate of cwnd growth in Scenario 1 than in Scenario 2 during the slow start phase.

The overall *'U'* shape of either of the AppStripe or CMT$_{scd}$ curves in Figure 4 can be now partly explained on the basis of the two scenarios we have described. At the center of the X-axis, both paths have equal bandwidth (20Mbps each). This point on the X-axis is equivalent to Scenario 1 described above. Further from the center of the X-axis, the paths have increasingly disparate bandwidth characteristics (with the bandwidth of one path decreasing linearly to zero, while bandwidth of the other path increases linearly to 40Mbps), and the simulations tend towards Scenario 2 described above. Thus, due to differing cwnd growth rates, the time taken for Application-Striping to transfer an 8MB file increases as the bandwidth difference between the paths increases and the sum of the bandwidths is maintained a constant. A second phenomenon which further explains the shape of the curves is described in Section 4.1.

We thus demonstrate under zero loss conditions that even if the application uses an ideal scheduling algorithm to distribute data across multiple SCTP associations, CMT-SCTP$_{scd}$ will perform at least as well such an application.

4 CMT VS. APPSTRIPE: EVALUATION UNDER UNIFORM LOSS

In this section, we present an evaluation of CMT$_{scd}$ vs. AppStripe under uniform loss. For both AppStripe and CMT$_{scd}$, we use *rtx_to_same* retransmission policy - a retransmission is sent to the same destination as the original transmission. The default policy of SCTP, which specifies retransmission to an alternate destination performs markedly worse, and has been shown to have significant performance problems [2]. The topology for the simulations is described in Section 2. We first present performance results comparing CMT$_{scd}$ vs. AppS in Section 4.1. To demonstrate the the performance differences observed are attributable to delayed acks and the CMT

[1]This phenomenon is also described in Section 5 of [5]

delayed ack algorithm, we present performance results comparing CMT_{sc} (CMT_{scd} without the CMT delayed ack algorithm) vs. AppS, both without delayed acks, in Section 4.1. These experiments were performed under the hypothesis that CMT_{scd} would perform at least as well as AppS. We point out that the benefits of performing CMT at the transport layer vs. at the application layer have been presented earlier in Section 1, and provide justification for using CMT_{scd} instead of an application such as AppS. Further, we remind the reader that AppS is a simulated application that performs ideal scheduling of data, and is used only as a reference for our performance measurements.

4.1 EVALUATION WITH DELAYED ACKS

Figure 5 shows CMT_{scd} vs. AppS under 0.1, 2, 5 and 10 percent uniform loss on both paths. We describe three salient observations from the performance graphs:

Observation 1: When only one path is used, i.e, at the edges in the graphs where the available bandwidth combination of the paths is 40Mbps and 0Mbps, CMT_{scd} performs worse than AppS. CMT_{scd} performance degrades further as the loss rate increases. This performance degradation is attributed to the CMT delayed ack algorithm [5]. Since the receiver observes significant reordering with CMT (without the CMT delayed ack algorithm), acks are almost never delayed, thus increasing the number of acks being sent back to the data sender. To curb the increase in ack traffic, the CMT delayed ack algorithm delays acks even when the receiver observes reordering. In the presence of loss, the CMT_{scd} data sender compensates for fewer acks received to allow fast retransmits to happen "on time"[2]. The data receiver sends an ack on the expiration of a delayed ack timer, which is set to the recommended value of 200ms [8].

Under high loss rates, the data sender often has a small cwnd value, and thus there is no continuous flow of data packets to ensure continuity in delayed acks being sent by the CMT_{scd} data receiver. The CMT_{scd} receiver thus often waits for the delayed ack timer to expire before sending an ack, even if a loss has occured, thus often adding 200ms to the roundtrip time. The performance degradation of CMT_{scd} is thus due to the increase in number of delayed ack timeouts at the data receiver. As loss rates increase, the number of losses increase, and hence the number of loss recoveries that are delayed due to the delayed ack timer increase, thus increasing total transfer time.

This phenomenon of increased delayed ack timeouts is not observed when multiple paths are used in our simulation settings, mostly because there is almost always flow of data on either of the two paths used, thus helping maintain continuity in flow of delayed acks. We are currently investigating if the increase in the number of delayed ack timeouts can occur under other path parameter combinations, and if so, will propose extensions to the CMT delayed ack algorithm. If increased ack traffic is not an concern in the deployment domain, then we suggest that the CMT delayed ack algorithm be turned OFF in general to avoid such performance degradation. In general, we suggest that CMT not be used when only one path is used since there is no benefit from CMT with only a single path.

Observation 2: CMT_{scd} performs the same as, or better than AppS under all loss rates and all other bandwidth combinations. In these cases, two phenomena occur simultaneously: (i) increased number of delayed ack timeouts increase the transfer time of CMT_{scd} (as described in Observation 1), and (ii) delayed acks simultaneously contributing to the cwnd growth of both destinations during slow start (as described in Section 3 to explain better performance of CMT_{scd} over AppS under zero loss). Since, under higher loss rates, the sender experiences more data timeout retransmissions, the sender is often operating in the slow start phase. From the performance graphs, it is evident that the influence of the benefit to CMT_{scd} during slow start is greater than that of the increased number of delayed ack timeouts, thus causing performance improvement of CMT_{scd} over AppS.

[2]The interested reader is referred to [5] for more details on the CMT delayed ack algorithm.

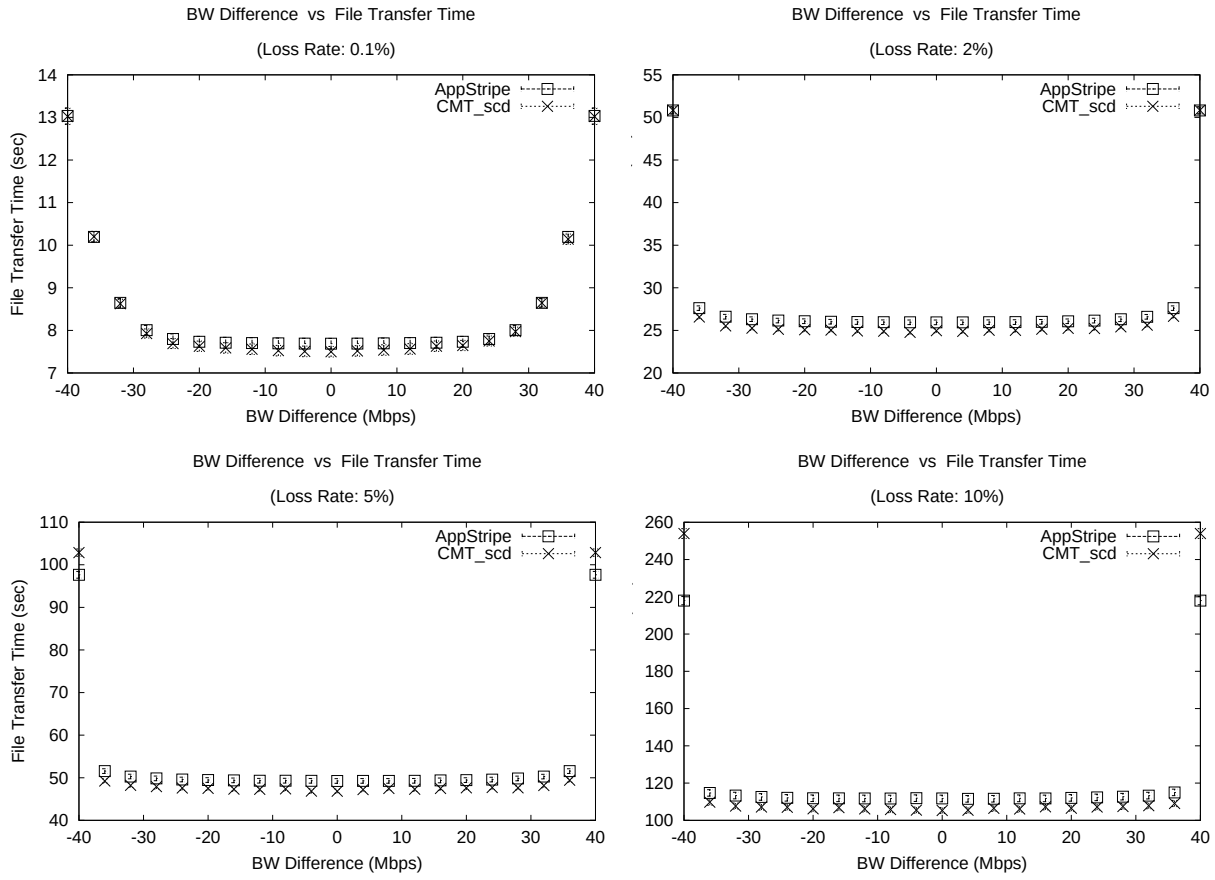

Figure 5: 8MB Transfer Time: CMT$_{scd}$ Vs. AppStripe

Observation 3: For both CMT$_{scd}$ and AppS curves, as the loss rate increases, the overall *'U'* shape can still be seen. Since the influence of the initial slow start phase decreases as the loss rate increases, the phenomenon described during the initial slow start phase in Section 3 for explaining the *'U'* shape of the curves has lesser impact on the shape of the curves under high loss rates. A second phenomenon, which we now present, becomes dominant and describes the shape of the curves. Consider the approximate TCP (also approximately SCTP) throughput equation [6] which can be presented as:

$$Throughput \propto 1/(RTT \times \sqrt{p})$$

where p is the loss rate, and RTT is the roundtrip time. We make use of the same two scenarios used in Section 3 for our explanation here. In Scenario 1, the total throughput obtained by either AppStripe or CMT$_{scd}$ is effectively the sum of throughputs obtained by two SCTP associations, one on each path to the destination host. Since in our simulation settings, the roundtrip times of the two paths are dominated by the same roundtrip propagation delay (RTT_{prop}) of 70ms, and the loss rates of both paths are the same, the total throughput obtained in Scenario 1 by either CMT$_{scd}$ or AppStripe can be approximated as:

$$Throughput_{(Scenario\ 1)} \propto 2/(RTT_{prop} \times \sqrt{p})$$

In Scenario 2, the total throughput obtained by either AppStripe or CMT$_{scd}$ is effectively the throughput obtained by a single SCTP association on the one path. Again, since in our simulation settings, roundtrip time of the path is dominated by a roundtrip propagation delay (RTT_{prop}) of 70ms, the throughput obtained in Scenario 2 by either CMT$_{scd}$ or AppStripe can be approximated as:

$$Throughput_{(Scenario\ 2)} \propto 1/(RTT_{prop} \times \sqrt{p})$$

which is only half the throughput obtained in Case 1. This difference in throughput between Scenario 1

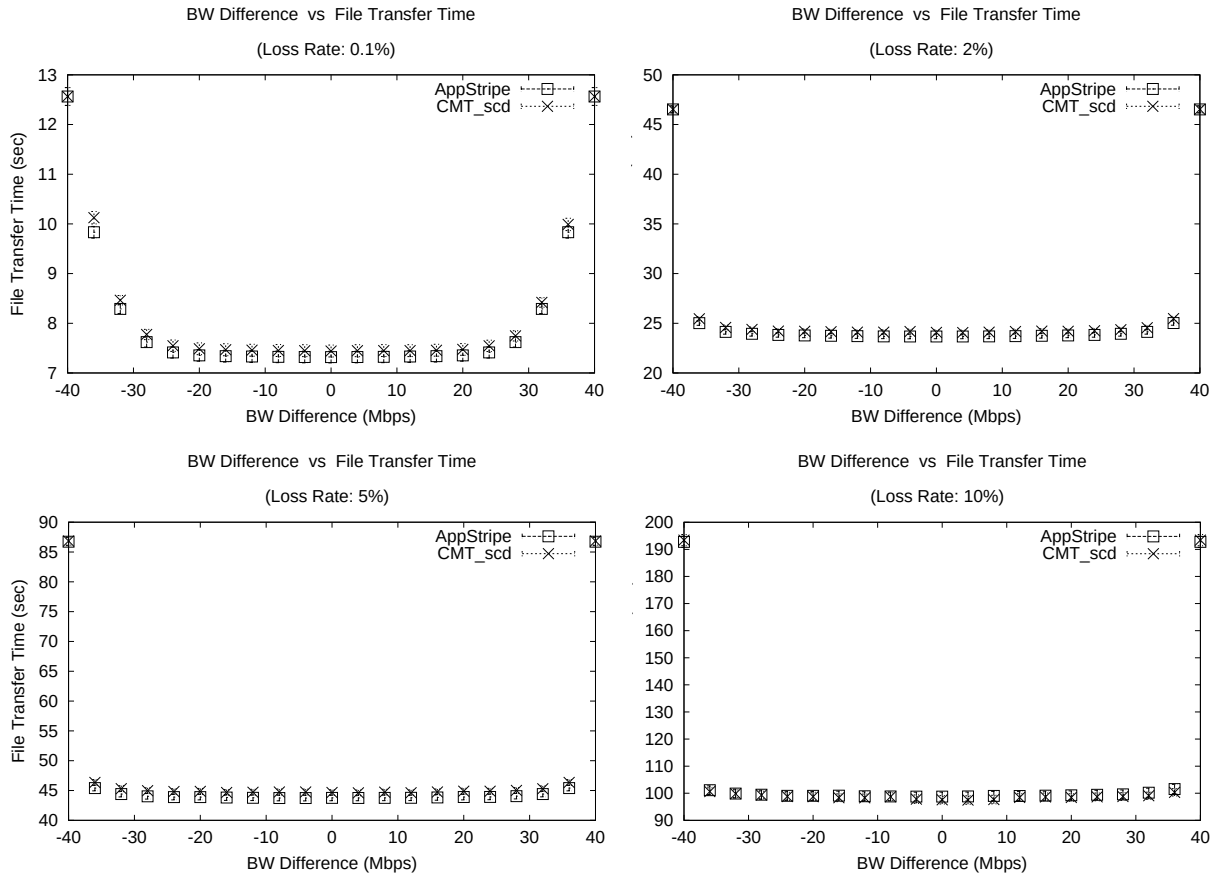

Figure 6: 8MB Transfer Time: CMT$_{scd}$ Vs. AppStripe (no delayed acks)

and Scenario 2 explains the large increase in transfer time from the center to the edges (roughly twice) in Figure 5.

Thus, since the propagation delay dominated roundtrip time is almost the same for all simulation settings, a large difference is observed when the topology changes from a single path (40Mbps and 0Mbps combinations) to multiple paths (38Mbps and 2Mbps combinations), even though the bandwidth delay product sum is maintained constant. Thus, if the loss rates on all paths are the same, transmission delays are negligible compared to the propagation delays, and propagation delays on all paths are the same, then the sender gains significantly by adding another path instead of adding an equal amount of bandwidth to the existing path.

4.2 EVALUATION WITHOUT DELAYED ACKS

We now present performance comparison of CMT$_{sc}$ - CMT$_{scd}$ without the CMT delayed ack algorithm [5] - vs. AppStripe; without delayed acks. In this set of simulations, each data packet resulted in an ack being sent by the data receiver to the data sender. Figure 6 shows CMT$_{sc}$ vs. AppStripe under 0.1, 2 and 5 percent uniform loss on both paths. CMT$_{sc}$ performs as well as AppStripe under all loss rates and bandwidth combinations[3].

[3]Some variability, though negligible, can be observed in the performance difference between CMT$_{sc}$ and AppStripe in the graphs. This variability is due to association startup and teardown effects, not due to protocol mechanisms, and can be ignored.

5 CONCLUSION

In this paper, we evaluated CMT$_{scd}$ against AppStripe under uniform losses varying from 0 to 5 percent. We demonstrate that CMT$_{scd}$ performs as well as, if not better than AppStripe under most conditions, except when a single path is used. We attribute this performance degradation to increase in the number of delayed ack timeouts. We are currently investigating if an increase in the number of delayed ack timeouts can occur under other path parameter combinations, and if so, will propose extensions to the CMT delayed ack algorithm. If increased ack traffic is not an concern in the deployment domain, then we suggest that the CMT delayed ack algorithm be turned OFF in general to avoid such performance degradation. In general, we suggest that CMT not be used when only one path is used since there is no benefit from CMT with only a single path.

We presented results encouraging the use of multiple paths in lossy networks such as battlefield networks in Section 4.1. If the loss rates on all paths are the same, transmission delays are negligible compared to the propagation delays, and propagation delays on all paths are the same, then the sender gains significantly by adding another path instead of adding an equal amount of bandwidth to the existing path.

We are currently evaluating CMT$_{scd}$ against AppStripe using a more general topology under common network parameter settings in wireline and wireless networks. We plan to also perform evaluation in the presence of cross-traffic and congestion losses. We then plan to propose and investigate the performance of different retransmission policies for CMT$_{scd}$.

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.

REFERENCES

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

[2] A. Caro, P. Amer, and R. Stewart. Transport Layer Multihoming for Fault Tolerance in FCS Networks. In *MILCOM 2003*, Boston, MA, October 2003.

[3] A. Caro and J. Iyengar. ns-2 SCTP module, Version 3.0, April 2002. http://pel.cis.udel.edu.

[4] A. Caro, J. Iyengar, P. Amer, S. Ladha, G. Heinz, and K. Shah. SCTP: A Proposed Standard for Robust Internet Data Transport. *IEEE Computer*, 36(11):56–63, November 2003.

[5] J. Iyengar, K. Shah, P. Amer, and R. Stewart. Concurrent Multipath Transport Using SCTP Multihoming. Tech Report TR2004-02, CIS Dept, University of Delaware, September 2003.

[6] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM 1998*, pages 303–314, Vancouver, CA, 1998.

[7] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tuexen. Stream Control Transmission Protocol (SCTP) Implementer's Guide. draft-ietf-tsvwg-sctpimpguide-10.txt, Internet Draft (work in progress), Internet Engineering Task Force (IETF), November 2003.

[8] 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, Internet Engineering Task Force (IETF), October 2000.