Performance Implications of a Bounded Receive Buffer In Concurrent Multipath Transfer

Janardhan R. Iyengar
Dept. of Math and Computer Science
Connecticut College
iyengar@conncoll.edu

Paul D. Amer
Protocol Engineering Lab, CIS Dept.
University of Delaware
amer@cis.udel.edu

Randall Stewart
Internet Technologies Division
Cisco Systems
rrs@cisco.com

***Abstract*— We study the performance of Concurrent Multipath Transfer using SCTP multihoming (CMT) in the presence of a bounded receive buffer (rbuf). We demonstrate using simulation that if two paths are used for CMT, the lower quality (i.e., higher loss rate) path degrades overall throughput of an rbuf-constrained CMT association by blocking the rbuf. We argue that rbuf blocking is not specific to the transport layer, but applies to multipath transfers at other layers as well. We present and discuss CMT performance using several retransmission policies and various constrained rbuf values. We also study the impact of rbuf blocking with different combinations of end-to-end loss rate and delay on the two paths and show that when large differences exist in path delays and loss rates, using only the better path outperforms using two paths concurrently. While rbuf blocking cannot be eliminated, it can be reduced by choice of retransmission policy — a mechanism available to only the transport layer.**

I. Introduction

A host is multihomed if it can be addressed by multiple IP addresses, as is the case when the host has multiple network interfaces. Multihoming is increasingly economically feasible and can be expected to be the rule rather than the exception in the near future, particularly when fault tolerance is crucial. Multihomed nodes may be simultaneously connected through multiple access technologies, and even multiple end-to-end paths to increase resilience to path failure. For instance, a mobile user could have simultaneous Internet connectivity via a wireless local area network using 802.11b and a wireless wide area network using GPRS.

Previous work proposed using *Concurrent Multipath Transfer (CMT)* between multihomed source and destination hosts to increase an application's throughput [1]–[3]. CMT is the concurrent transfer of new data from a source to a destination host via two or more end-to-end paths. While concurrency can be arranged at other layers, the transport layer has the best knowledge to estimate end-to-end paths' characteristics. Implementing multipath transfer at the application layer increases redundancy and room for error by requiring a separate implementation by each application programmer.

The current transport protocol workhorses, TCP and UDP, do not support multihoming; TCP allows binding to only one network address at each end of a connection. When TCP was designed, network interfaces were expensive components, and multihoming was beyond the ken of research. Changing economics and increasing emphasis on end-to-end fault tolerance have brought multihoming within the purview of the transport layer. We research CMT at the transport layer using the Stream Control Transmission Protocol (SCTP) [4], [5]. SCTP is an IETF standards-track protocol that natively supports multihoming at the transport layer. SCTP multihoming allows binding of one transport layer *association* (SCTP's term for a connection) to multiple IP addresses at each association endpoint. This binding allows a sender to transmit data to a multihomed receiver through different destination addresses. *Concurrent* transfer of *new* data to multiple destination addresses is currently not allowed in SCTP due primarily to insufficient research. Our research attempts to fill that need.

Previously, we developed three algorithms for 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 [1]. Since CMT$_{scd}$[1] allows concurrent transmission of data to multiple destinations, a sender can send retransmissions to one of several destinations that are receiving new transmissions. Previously, we proposed five retransmission policies, and protocol modifications to SCTP to allow for correct functioning of these retransmission policies [2]. But in [2], we operated under the strong and limiting assumptions that (1) the receive buffer (rbuf) was infinite, and (2) the bottleneck queues on the end-to-end paths used in CMT were independent of each other.

In this paper, we drop assumption (1) and investigate how a bounded rbuf affects CMT performance. While assumption (2) remains limiting, several environments (eg., telephony signalling over IP, battlefield networks) exist with independent paths. Future work will relax this constraint. While we discuss performance considerations in the context of CMT, we note that these considerations apply to multipath transfer at other layers as well.

Section II describes the simulation topology for our investigation, and reviews relevant concepts and terminology. Section III describes, by example, the rbuf blocking problem.

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 University Research Program of Cisco Systems, Inc.

[1] Henceforth, we refer to "CMT$_{scd}$" as simply "CMT"

Section IV presents a detailed performance analysis of the different retransmission policies with different rbuf sizes. Section V then considers the impact of rbuf blocking with different loss rate and end-to-end delay combinations on the paths used for CMT. Section VII summarizes and concludes the paper.

II. Preliminaries

For our simulations, we use the University of Delaware's SCTP module, which is now part of the latest ns simulator distribution [6], modified to incorporate CMT and the different retransmission policies. The topology is shown in Figure 1. Edge links represent the last hop, and the core links represent end-to-end conditions on the Internet. The end-to-end delays are 45ms on both paths, representing typical US coast-to-coast delays experienced by a significant fraction of the flows on the Internet [7]. The loss rate on Path 1 is maintained at 1%, and on Path 2 is varied from 1 to 10%. A loss rate of x% means a forward path loss rate of x%, and a reverse path loss rate of x%.

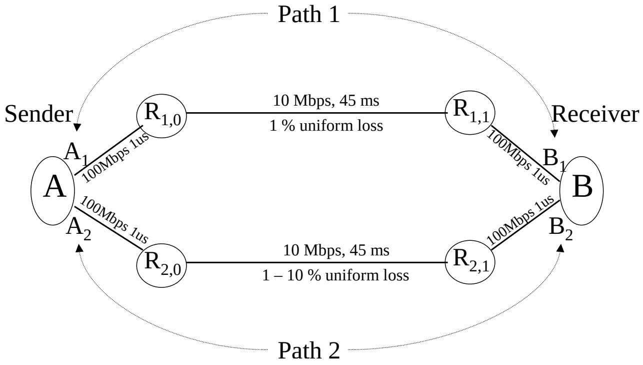


Fig. 1. Simulation topology

This simulation topology does not account for effects seen in the Internet such as network-induced reordering, delay spikes, etc.; these effects are beyond the scope of this study. Our simulation evaluation provides insight into the impact that a constrained rbuf poses to CMT, the fundamental differences between the retransmission policies, and their performance in a constrained rbuf environment. We chose a simple topology to avoid influence of other effects, and to focus on performance differences which we believe should hold true in a real environment as well[2].

We now review relevant concepts and terminology. A transport layer receiver maintains rbuf space for incoming data for two reasons: (i) to handle out-of-order data, and (ii) to receive data at a rate higher than that of the receiving application's consumption. In SCTP (and TCP), a receiver advertizes currently available rbuf space through window advertisements (normally accompanying acks) to a data sender. This value is the *advertized receive window (adv-rwnd)*. A sender computes a *peer-rwnd* to deduce how much *more* data can be buffered at the receiver. Beside the latest adv-rwnd received, the peer-rwnd takes into account data that has been sent but not yet acked by the receiver.

An SCTP receiver maintains a single rbuf per association. An SCTP sender, consequently, maintains a single peer-rwnd per association. Note that sender-side estimates such as congestion window (cwnd), slow start threshold (ssthresh) and roundtrip time (RTT) are maintained per destination - they represent the state of different network paths from a sender to each destination address. A sender has no reason to maintain separate rbufs or peer-rwnds per path since a receiver can consume data only in sequence, irrespective of the destination address they are sent to[3]. An SCTP sender's sending rate is bound by both the peer-rwnd and the pertinent destination's cwnd, i.e., *min(peer-rwnd, cwnd)*.

We define a *sub-association flow* as the set of transport protocol data units (PDUs) within an SCTP association that have the same destination address. In Figure 1, an SCTP association from the sender to the receiver spanning the two paths will have two sub-association flows - one consisting of PDUs with destination B_1, and the other with destination B_2.

A reference to “cwnd for destination X” means the cwnd maintained at the sender for destination X, and “timeout on destination X” refers to the expiration of a retransmission timer maintained for destination X at the sender. Since bottleneck queues on the end-to-end paths are assumed independent, each destination uniquely maps to a bottleneck-independent path. For instance in Figure 1, “cwnd for destination B_1” may be used interchangeably with “cwnd for path 1” (where path 1 ends at destination B_1).

CMT schedules new data to different destinations as bandwidth becomes available on corresponding paths, i.e., as corresponding cwnds allow. When cwnd space is available simultaneously for two or more destinations, data is sent to these different destinations in arbitrary order - a reasonable transmission policy when the CMT sender has no *apriori* knowledge of the paths' characteristics.

III. Receive Buffer Blocking: Problem Description

A CMT receiver maintains a single rbuf which is shared across the sub-association flows. Irrespective of the layer at which multipath transfer is performed, a similar shared buffer would exist at a receiver (likely at the transport or application layer). This buffer sharing has been shown to degrade throughput [8]. To elaborate, Figure 2 shows an excerpt from a simulation of a CMT association using the topology shown in Figure 1. In this example, the rbuf is 16KB, Path 1 (A_1 to B_1) has a loss rate of 1%, and Path 2 (A_2 to B_2) has a loss rate of 10%. Retransmissions are sent to the same destination as the original transmission, a policy called *RTX-SAME* (we revisit this retransmission policy in Section IV).

Figures 2(a) and (c) show Transmission Sequence Number (TSN) progression[4] over Path 1 and Path 2, respectively, and Figure 2(b) shows peer-rwnd evolution at the sender

[2] The simulation topology is clearly simplistic. We verified a subset of the results using a more complex topology with congestion losses based on cross-traffic, and the resulting trends were similar, supporting our conclusions in this paper.

[3] While SCTP supports *unordered* data delivery and *multistreaming* in an association [5], here we focus on ordered data delivery over a single stream.

[4] TSNs in SCTP are analogous to sequence numbers in TCP.

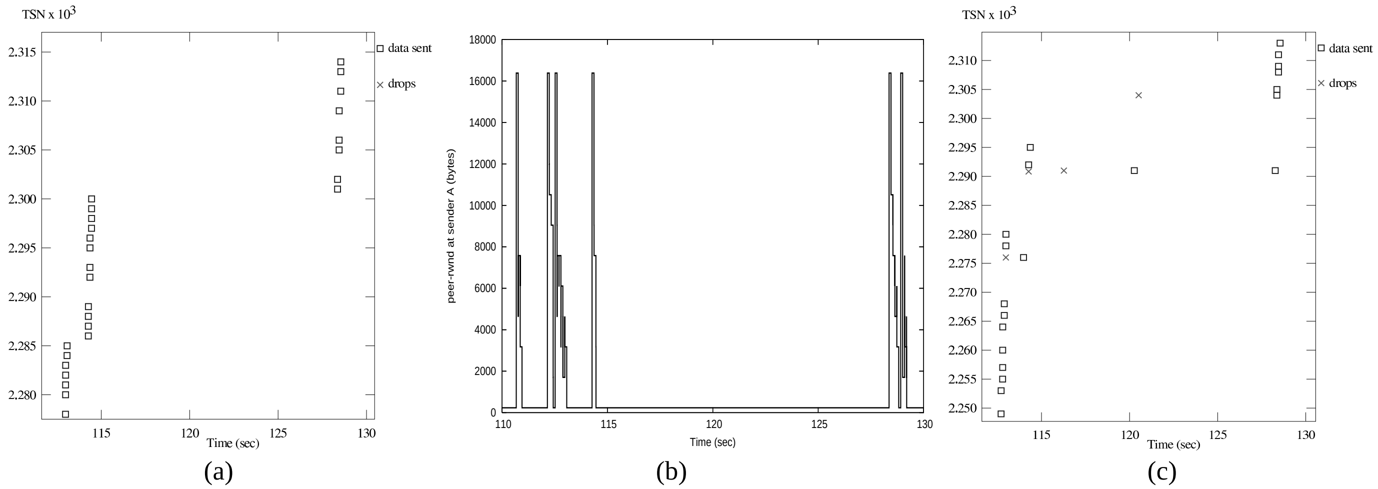


Fig. 2. Instantiation of rbuf blocking: (a) Progression of data sent to destination B_1 over Path 1 (loss rate 1%) over a select interval; (b) peer-rwnd value maintained at sender (endpoint A) over same interval; (c) Progression of data sent to destination B_2 over Path 2 (loss rate 10%) over same interval

(endpoint A) during the time interval from 110 to 130 seconds. Figure 2(a) shows that data transmission over the better (i.e., lower loss rate) path stops abruptly around 114.5 seconds and resumes around 128 seconds. This 13.5 second pause can be explained with the help of Figure 2(b). At 114.5 seconds, the peer-rwnd at the sender abruptly reduces from 16384 bytes to 236 bytes, constraining the sender from transmitting any new data. The cause for this abrupt rbuf reduction is explained as follows.

During the same time interval from 114.5 seconds to 128 seconds, Figure 2(c) shows that the lower quality (i.e., higher loss rate) path undergoes congestion, and recovers from losses through repeated retransmission timeouts - the longest recovery time being 8 seconds for TSN 2304. During this entire period of 13.5 seconds while loss recovery repeatedly occurs on Path 2, the receiver waits for retransmissions to come through, and is unable to deliver subsequent TSNs to the application (some of which were sent over Path 1). These subsequent TSNs are held in the transport layer rbuf until the retransmissions are received, thus blocking the rbuf and the peer-rwnd. Path 2 thus causes blocking of the rbuf, preventing data from being sent on either path and reducing overall throughput.

This example demonstrates how a shared rbuf results in a sub-association flow on a higher quality path getting lower throughput than expected. We note that the exact numbers used in this example do not hold special relevance. This example presents a phenomenon which occurs, in lesser or greater degree, throughout a CMT association.

Figure 3 shows the time taken to transfer an 8MB file using (i) CMT (with RTX-SAME retransmission policy) with a 16KB rbuf, and (ii) a single SCTP association which uses *only* the better path (Path 1 with loss rate 1%). Intuitively, using two paths should provide higher overall throughput than using one path. However, Figure 3 demonstrates that using two paths performs worse than using only the better path if a finite rbuf is shared across the paths[5]. This performance difference is due to rbuf blocking that occurs in CMT - an rbuf of 16KB does not constrain a single SCTP association (which uses one lower loss rate path) as much as it constrains CMT (which uses two paths with different loss rates).

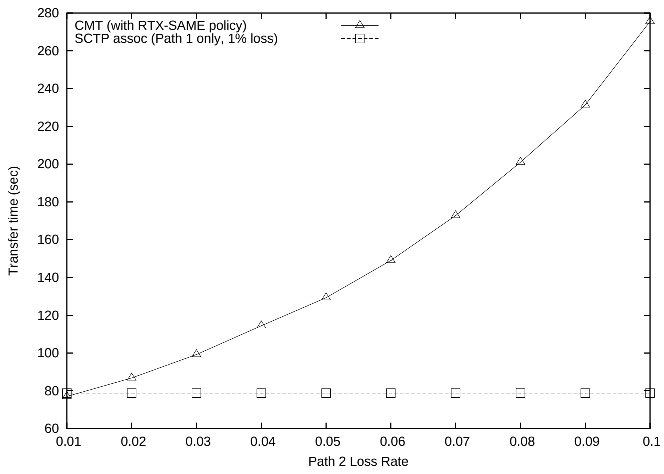


Fig. 3. rbuf blocking in CMT causes throughput degradation

We emphasize that rbuf blocking is not unique to the transport layer; it applies to multipath transfer at other layers as well. Rbuf blocking cannot be eliminated or avoided by moving CMT's parallelism to a different layer. Specifically, if the application layer distributes a single logical flow across multiple end-to-end paths, and the application layer receiver (the final destination) has finite buffer space, then rbuf blocking will occur.

After analyzing several traffic flows, we observe that chances of rbuf blocking are higher during periods of timeout recovery. Further, a larger timeout recovery period due to back-to-back timeouts with exponential backoff results in an even

[5] Presumably an SCTP sender does not have *apriori* knowledge about the better path and hence cannot always achieve best performance. We discuss *expected* SCTP performance later in Section V.

higher probability that a finite rbuf blocks a sender.

We therefore hypothesize that reducing (i) the number of timeouts, and/or (ii) the number of back-to-back timeout retransmissions will reduce the rbuf blocking problem. Consequently, we hypothesize that making an intelligent choice of destination for retransmissions, a choice available to only the transport layer, will help reduce rbuf blocking. As we mentioned earlier, retransmissions in the above example are sent to the same destination as the original transmission. We used this retransmission policy initially due to its simplicity. In the following section, we present several alternative "smarter" policies.

IV. Choosing a Retransmission Policy for CMT

Multiple paths present a sender with a choice where to send a retransmission of a lost transmission. Unlike SCTP where a sender "knows" a lot about one primary path and just a little (via probes) about all of the other paths, a CMT sender maintains accurate information about all paths, since new data is being sent to all destinations concurrently. This information allows a CMT sender to intelligently decide where to retransmit.

A. CMT Retransmission Policies

We review five retransmission policies for CMT [2]. In the latter four policies, a retransmission may be sent to any of the receiver's destinations, including the one used for the original transmission.

- **RTX-SAME** - Once a new data chunk is scheduled and sent to a destination, all retransmissions of the chunk are sent to the same destination (until the destination is deemed *inactive* due to failure [5]).
- **RTX-ASAP** - A retransmission of a data chunk is sent to any destination for which the sender has cwnd space available at the time of retransmission. If multiple destinations have available cwnd space, one is chosen randomly.
- **RTX-CWND** - A retransmission is sent to the destination for which the sender has the largest cwnd. A tie is broken by random selection.
- **RTX-SSTHRESH** - A retransmission is sent to the destination for which the sender has the largest ssthresh. A tie is broken by random selection.
- **RTX-LOSSRATE** - A retransmission is sent to the destination with the lowest loss rate path. If multiple destinations have the same loss rate, one is selected randomly.

Of the policies, RTX-SAME is simplest. Since an application that stripes data across multiple transport layer (SCTP or TCP) associations will not be able to move outstanding data from one association to another, such an application will effectively use the RTX-SAME policy at the transport layer. RTX-SAME thus represents the performance of an data striping application.

RTX-ASAP is a "hot-potato" policy - retransmit as soon as possible without regard to loss rate. RTX-CWND and RTX-SSTHRESH practically track, and attempt to move retransmissions onto the path with the estimated lowest loss rate. RTX-LOSSRATE uses information about loss rate provided by an "oracle" - information that RTX-CWND and RTX-SSTHRESH estimate. This policy represents a hypothetically ideal case; hypothetical since in practice, a sender typically does not know *apriori* path loss rates; ideal since the path with the lowest loss rate has highest chance of having a packet delivered. We hypothesized that policies that take loss rate into account would best reduce rbuf blocking by reducing the number of timeouts.

We do not propose different policies for new transmissions - we assume that a sender does not know path properties *apriori* and can therefore only react to network events such as congestion losses (see Section II for our transmission policy). This assumption holds true particularly in the Internet where the path properties are changing.

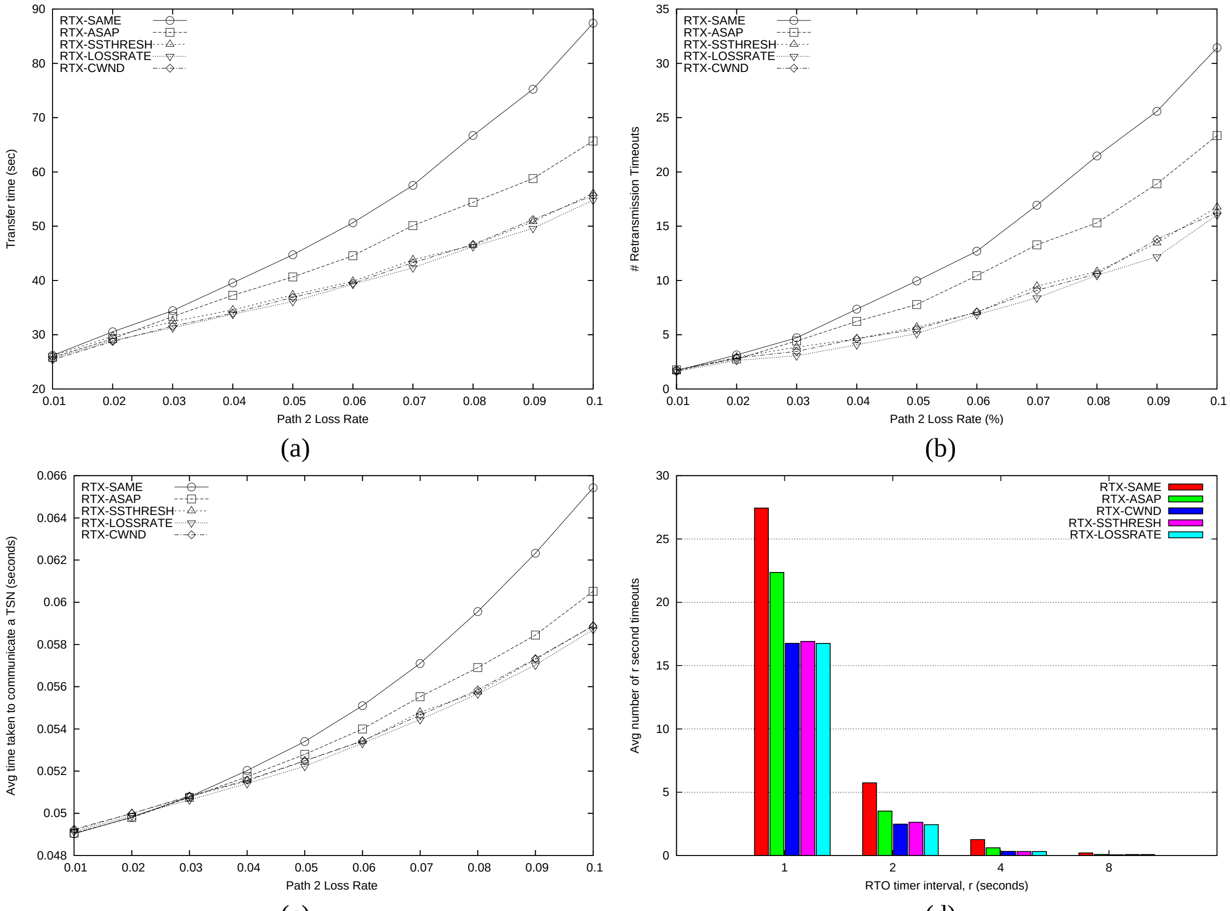
We now evaluate the five retransmission policies for CMT operating under a constrained rbuf. Default rbuf values in commonly used operating systems today vary from 16KB to 64KB and beyond. We believe that today, when a desktop computer can have gigabytes of memory, having an rbuf of at least 64KB is reasonable. We first study and analyze performance of the different policies with an rbuf of 64KB in Section IV-B. This section provides insight into the causes of the performance differences between the retransmission policies. We then summarize performance of the different policies under more and less constraining rbufs varying from 16KB to 256KB in Section IV-C. This analysis provides us with an understanding of how much impact a constraining rbuf (or the rbuf blocking problem) has on the different policies.

B. Evaluation with rbuf = 64KB

Figure 4(a) shows the time taken for a CMT sender to transfer an 8MB file when the rbuf is set to 64KB, using the five retransmission policies. Each plotted value is the mean of at least 100 simulation runs. RTX-SAME, the simplest to implement, performs worst. Its performance gap with the other policies increases as the loss rate on Path 2 increases. RTX-ASAP performs better than RTX-SAME, but still considerably worse than the three loss-rate-based policies. We present two causes for these differences.

Cause 1: Figure 4(b) shows the number of retransmission timeouts experienced when using the different policies. One may conclude that improvement in using the loss-rate-based policies is due partly to fewer timeouts (and hence, timeout recovery periods). RTX-SAME does not consider loss rate and experiences the largest number of timeouts. RTX-ASAP does not consider loss rate and does better than RTX-SAME, but still experiences more timeouts than the loss-rate-based policies. This analysis supports our intuitive hypothesis - taking path loss rate into consideration while deciding the retransmission destination improves the chances of a retransmission getting through, and improves overall performance due to reduction of rbuf blocking.

Cause 2: Figure 4(c) shows the average time taken to successfully communicate a TSN. This time is measured as the time taken from the first transmission of a TSN to the time when that TSN or one of its retransmissions finally



(a)

(b)

(c)

(d)

Fig. 4. With rbuf = 64K, and Path 1 loss rate = 1%,: (a) Time taken by CMT to transfer an 8MB file, (b) Number of retransmission timeouts for CMT with different retransmission policies, (c) Average time taken to successfully communicate a TSN with different retransmission policies, (d) Average number of consecutive timeouts with different retransmission policies (Path 1 loss rate = 1%; Path 2 loss rate = 10%)

reaches the receiver. RTX-SAME shows the highest average, suggesting that more retransmissions may be needed for a successfully communicating a TSN. Since recovery via fast retransmission can happen only once for a given TSN, the number of consecutive timeouts may be higher with RTX-SAME than with the other policies. Each consecutive timeout causes a sender's retransmission timeout value to double, thus doubling the timeout recovery period. Recall that the longer the timeout recovery period, the higher the probability and longer the duration for rbuf blocking to occur. Thus, more consecutive timeouts will degrade performance.

Figure 4(d) shows average number of timeouts that take r seconds (for r = 1, 2, 4 and 8) using the different retransmission policies with Path 1 loss rate = 1%, and Path 2 loss rate = 10%. Using SCTP's (and TCP's) default parameter values, these values of r are caused by consecutive timeouts — r = 2 seconds corresponds to 2 consecutive timeouts, and r = 4 seconds corresponds to 3 consecutive timeouts. Though the occurrences of 2 and 4 second timeouts are few, we note that their impact is significant due to rbuf blocking during these periods. Overall, loss-rate-based policies experience about half the consecutive timeouts that RTX-SAME does. Thus, *performance degradation due to consecutive timeouts can be significantly reduced by taking loss rate into account for making retransmission decisions.*

C. Evaluation with different rbufs

We also investigated the performance of the five retransmission policies using rbuf sizes of 16KB, 32KB, 128KB, and 256KB. The performance ranking of the different policies with these rbufs remains the same as with an rbuf of 64KB (Figure 4(a)); hence performance curves for these rbuf values are not shown. We discuss a few salient points.

- With a large (i.e., minimally constraining) rbuf of 256KB, RTX-SAME still performs poorly due to a high number of timeouts, and the consequent throughput degradation. Each timeout causes cwnd reduction at a sender, and entails idle time (i.e., the sender not transmitting data) — causing throughput reduction.
- As the rbuf size decreases and becomes more of a constraint, degradation in CMT throughput occurs due to increased rbuf blocking as explained in Section III. All retransmission policies suffer in the face of an increasingly constraining rbuf. Even with a reasonably large rbuf of 128KB, some performance degradation occurs; i.e., *even with large rbufs, rbuf blocking is not eliminated.*

- As the rbuf becomes more constraining, degradation in throughput of RTX-SAME policy is markedly more than with the other retransmission policies. The reasons for this degradation are the same as described in Section IV-B. Degradation is least in the loss-rate-based policies with an increasingly constraining rbuf.

In summary, retransmission policies that take loss rate into account perform better under the different rbuf values considered. Of the loss rate based policies, the practical ones (RTX-CWND and RTX-SSTHRESH) perform similarly under all conditions considered. We arbitrarily select RTX-SSTHRESH as CMT's retransmission policy in further evaluations.

V. CMT Performance Evaluation

In this section we explore the effect of different end-to-end delays (Section V-A), and different combinations of delays and loss rates (Sections V-B and V-C), on CMT throughput. This evaluation studies the impact of rbuf blocking on CMT under different network conditions. A discussion on the insights gained through this evaluation follows (Section V-D).

A. Evaluation under different end-to-end delays

Figure 5 shows relative throughput degradation of CMT under different end-to-end delays - 10ms, 25ms, 45ms, 90ms, 180ms, and 360ms, yielding RTTs of 20ms, 50ms, 90ms, 180ms, 360ms, and 720ms, respectively. The delays on both paths to the receiver are symmetric (We are currently studying rbuf blocking with asymmetric paths). These values cover a range of RTTs experienced by majority of flows on the Internet [7]. Relative throughput degradation is computed as the ratio

$$\frac{CMT\ throughput\ with\ infinite\ (INF)\ rbuf}{CMT\ throughput\ with\ rbuf = X}$$

as X varies from 16KB to 256KB along the X-axis (Note that along the Y-axis, smaller values are better). The degradation curves for all end-to-end delays in Figure 5 show a knee at 64KB. The largest degradation (i.e., worst throughput) occurs with the shortest delay of 10ms. At 10ms, throughput with infinite rbuf space is roughly 10 times what it would be with a 16KB rbuf. As the end-to-end delay increases, CMT's relative throughput degradation decreases. We now explain why associations with smaller delays are more sensitive to a constrained rbuf.

Overall SCTP throughput, similar to TCP throughput, varies inversely with delay. This relationship holds true for large rbuf conditions. Thus, in the relative throughput degradation measure, the numerator (CMT throughput with infinite rbuf) increases as delay decreases.

As the rbuf size increasingly becomes a bottleneck, a different dynamic dominates. According to the SCTP specification [5] and the specification for computing TCP's retransmission timer [9], retransmission timeouts (RTOs) should have a (conservative) minimum value of 1 second to avoid spurious timeouts. These timeout recovery periods are thus independent of the end-to-end delays considered, since the delays are far less than 1 second. As rbuf increasingly constrains, the number of timeouts increases. Consequently, total time spent in timeout recovery (which is roughly the same irrespective of the end-to-end delay) increasingly dominates association lifetime. Thus, the denominator in the relative throughput degradation measure (CMT throughput with rbuf=X, as X varies) does not increase as fast as the numerator with decreasing end-to-end delay, since the denominator is largely dictated by (constant) timeout recovery periods. Therefore, the influence of a constrained rbuf increases as end-to-end delay decreases. In summary, *CMT is more sensitive to rbuf constraints in environments (such as data centers [10]) with shorter end-to-end delay. Or, from a network engineering point of view, the shorter the end-to-end delay, the more important it is to have a larger rbuf to fully exploit CMT.*

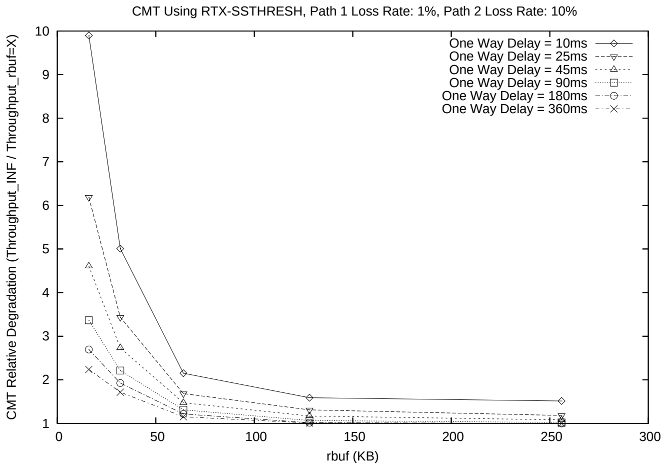


Fig. 5. Relative throughput degradation of CMT with different end-to-end delays

We can thus see that rbuf blocking has a larger impact on associations with shorter end-to-end delay due to a minimum RTO value which is recommended [5], [9] and largely in use. We note that shorter minimum RTOs together with better RTT estimation algorithms [11], [12] would lessen the bias against shorter delay associations (such a study is beyond the scope of this work.)

B. CMT vs UnawareApp

The potential parallelism gains of CMT decrease as the rbuf size decreases. In this section we attempt to quantify the gains, if any, in using CMT with a limited rbuf. We introduce a reference for comparison called *UnawareApp* and compare it to CMT. UnawareApp represents the expected throughput seen by an application using a single SCTP association to transfer data. UnawareApp sends data to one destination selected from the set of receiver destinations with equal probability. Without prior knowledge of path conditions, and without CMT, an application would arbitrarily pick one destination to send data. UnawareApp captures the expected throughput when such a decision is made.

It might seem unintuitive to compare CMT against UnawareApp, since UnawareApp uses just one path for the

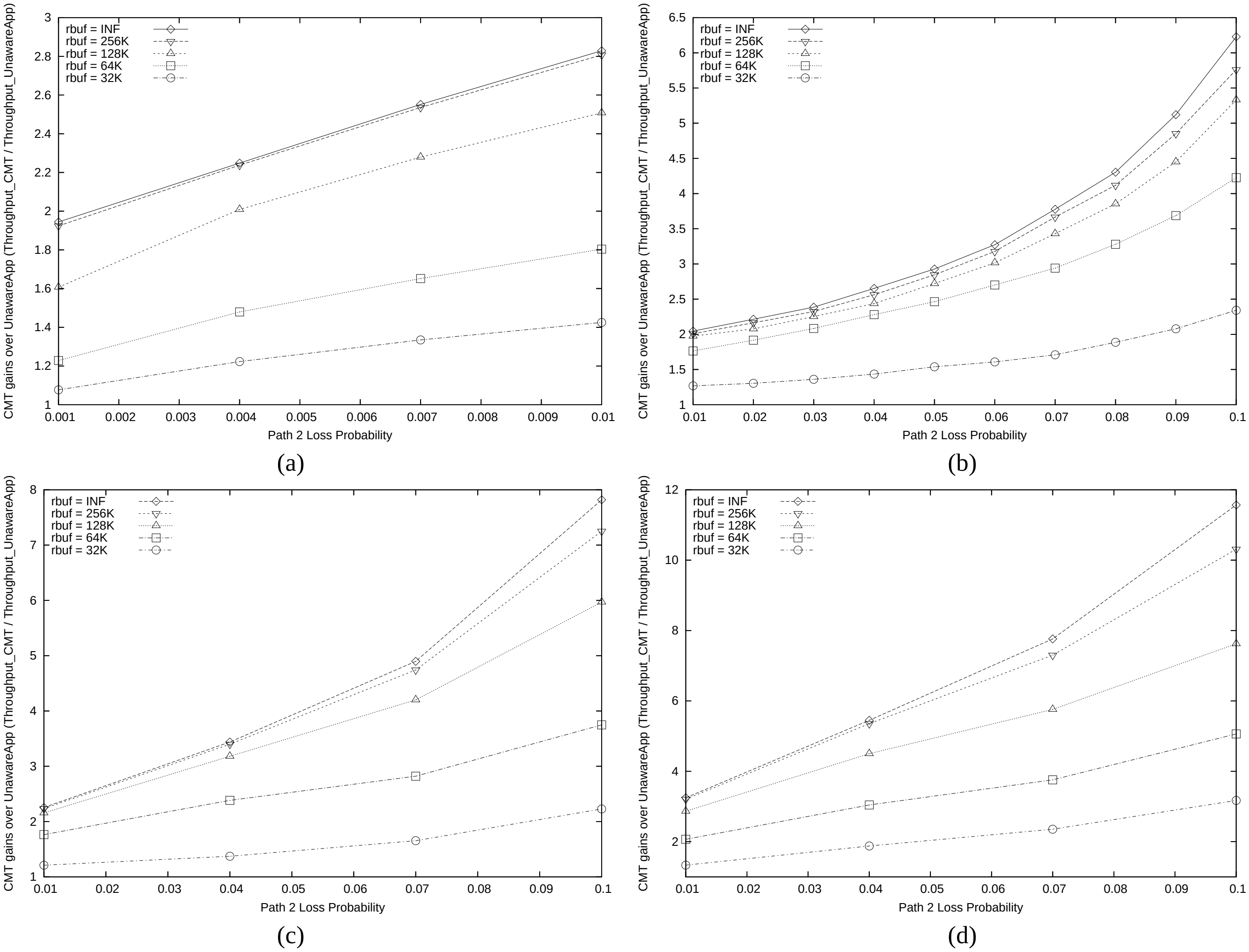


Fig. 6. CMT throughput gains over UnawareApp with a constraining rbuf:
(a) Path 1 loss probability = 0.001, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 45ms
(b) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 45ms
(c) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 90ms
(d) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 180ms

transfer (unlike CMT which sends data to all destinations). This evaluation explores the impact of the rbuf blocking problem on CMT. The rbuf blocking problem does not exist for data transfers that use only one path, and therefore, does not affect UnawareApp.

Further, rbuf blocking is caused by a shared finite receive buffer, and equally affects any data striping application such as AppStripe [2], which distributes load at the application layer across multiple transport paths. An application that stripes data across multiple paths requires a finite rbuf at the application layer for reassembly and ordering of incoming data. This rbuf will cause analogous degradation at the application layer as a bounded transport layer rbuf would for CMT. Therefore, we do not use a data striping application as reference in this evaluation since the rbuf blocking problem exists for such an application as well.

Figure 6 shows throughput gains in using CMT vs. UnawareApp, measured as a ratio of CMT throughput to UnawareApp throughput, for different rbuf values, different loss rates on the two paths, and different delay combinations on the two paths. We explored loss rate combinations in the range [0.1%, 10%], and end-to-end delay combinations in the range [25ms, 360ms]. These loss rate and delay combinations reflect network conditions experienced by flows on the Internet [7]. UnawareApp uses the same rbuf size as CMT in these evaluations.

The ratio plotted in Figure 6 is $\frac{Throughput\_CMT}{Throughput\_UnawareApp}$. Values greater than 1 imply that CMT performs better than UnawareApp; a value less than 1 means that UnawareApp performs better than CMT. All results show similar trends in throughput with CMT performing better; representative results are shown in Figure 6. The throughput gains with CMT are chiefly attributed to two reasons:

- UnawareApp chooses the worse path for transferring data half the time. Transfer times over the worse path increase significantly as loss rate on that path increases, thereby increasing the average transfer time for UnawareApp significantly. CMT uses both paths concurrently, and using RTX-SSTHRESH ensures that most of the data is sent over the better path, thus reducing overall transfer time for CMT.

- CMT is more resilient to reverse path loss than UnawareApp. CMT uses a single sequence space (TSN space, used for congestion control and loss detection and recovery) across the one association's multiple paths. Since CMT's acks are cumulative, sharing of sequence spaces across paths helps a CMT sender receive ack info on either of the return paths. Thus, CMT effectively uses *both* return paths for communicating ack info to the sender. For instance, if an ack is lost on one return path, traffic on the other path will cause the CMT receiver to respond with acks on the other return path. These cumulative acks carry the information that was lost in the previous ack(s). This resilience is a significant benefit of CMT, and occurs because CMT shares sequence space across the paths [2].

As compared to UnawareApp, using CMT is beneficial, even with the most constrained rbuf of 32KB; this benefit increases as the rbuf size increases.

C. CMT vs AwareApp

By using UnawareApp as a reference, we assumed that a sender has no prior knowledge of path conditions. This assumption causes UnawareApp to suffer throughput degradation, since UnawareApp picks and uses the higher loss rate path half the time. We now drop this assumption, and introduce *AwareApp*, an application which has *apriori* path information, and uses only the lower loss rate path for data transfer. AwareApp represents an application's throughput when using a single SCTP association over the best path to the destination. AwareApp avoids the rbuf blocking problem (as UnawareApp avoids), and also avoids throughput degradation due to using the higher loss rate path.

As in Section V-B, we explored different combinations of loss rate in the range from 0.1% to 10%, and end-to-end delays in the range from 25ms to 360ms with rbuf values ranging from 32KB to 256KB. Representative results from our exhaustive set of simulations are shown in Figure 7. The ratio plotted in Figure 7 is $\frac{Throughput\_CMT}{Throughput\_AwareApp}$. Values greater than 1 imply that CMT performs better than AwareApp; a value less than 1 means that AwareApp performs better than CMT. Salient points are as follows:

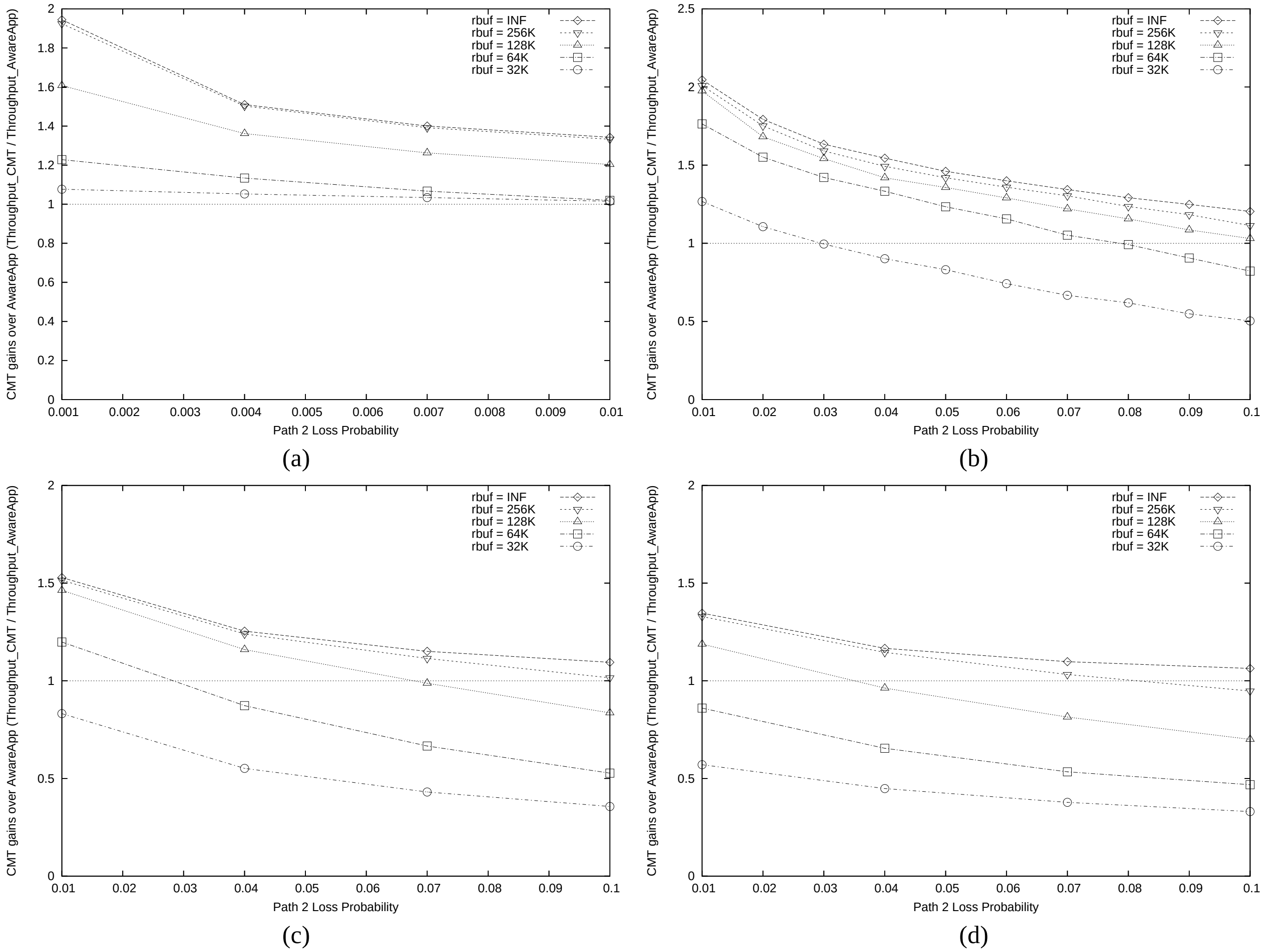
- *In some cases, AwareApp performs better than CMT.* These cases can be seen in Figure 7 whenever the curves drop below 1. This result is significant — rbuf blocking can degrade throughput to the point that *when large differences exist in path delays and loss rates, using only the better path outperforms using two paths concurrently*.
- CMT's throughput benefit over AwareApp decreases as the loss rate difference between the two paths increases for two — increased losses, and increased rbuf blocking. On the other hand, AwareApp, which uses only the lower loss rate path (i.e., Path 1), does not experience either of these throughput degradations.
- CMT's throughput benefit over AwareApp decreases as the delay difference between the two paths increases. See Figures 7(b), (c), and (d), where Path 2 delay is 45ms, 90ms, and 180ms, respectively. Path 1 delay is maintained at 45ms. This degradation is explained as follows: as the delay of Path 2 increases, more data can be sent on Path 1 within one roundtrip time on Path 2. An increase in Path 2's end-to-end delay therefore increases occurrences and periods of rbuf blocking due to increased loss recovery time on Path 2, especially for fast retransmit based recovery. This increased rbuf blocking degrades CMT's throughput.

From our simulations, we note that an rbuf of at least 128KB is required for CMT to perform better than AwareApp with a loss rate difference of upto 10 times *and* a delay difference of upto twice between the two paths used for CMT. This result holds within the range of loss rates and end-to-end delays that we explored in our simulations.

D. Discussion

As just discussed, a constrained rbuf can cause significant throughput degradation when multiple paths are used concurrently. One might expect that blocking can be avoided by multiplexing over paths at a different layer, but we note that *the rbuf blocking problem cannot be eliminated at any layer*; it can only be reduced. This problem equally affects an application layer (or network layer) data striping mechanism. Reservation of rbuf space per path will also not reduce blocking due to the need for in-order delivery to the application. As we have shown, use of an intelligent retransmission policy, which is possible in only the transport layer, and/or using a larger rbuf reduces rbuf blocking. Though the conditions studied represent Internet conditions [7], we acknowledge that we are using simplistic simulations to extract an exact value for use in real complex networks; we therefore *suggest with caution* that a minimum rbuf of 128KB is required to best exploit CMT's parallelism.

In addition to a receive buffer (rbuf) at the receiver, a transport sender maintains a send buffer (sbuf) for two reasons: (i) to buffer data received from a sending application that sends faster than can be transmitted into the network, and (ii) to maintain data that has been sent, but not cumulatively acked, for possible retransmission. While we discuss only rbuf throughout this paper, we note that these evaluations, analyses and recommendations apply to both sender and receiver socket buffers. If the sbuf is blocked by a higher loss rate path in a CMT association (as the rbuf can be), i.e., no data can be sent from the sending application to the sending transport, then the CMT association will suffer from the same degradation as described in this chapter. Our suggestion of using a minimum rbuf size of 128KB, therefore, holds for the sbuf as well. Using multistreaming or unordered data delivery will allow a transport receiver to deliver to the application data that may not be globally in-order within an association. Such "out-of-order" delivery will reduce rbuf blocking, but will not affect blocking of the sbuf, since loss recovery mechanisms use global ordering within an association. In using a single ordered



(a)

(b)

(c)

(d)

Fig. 7. CMT throughput gains over AwareApp with a constraining rbuf:
(a) Path 1 loss probability = 0.001, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 45ms
(b) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 45ms
(c) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 90ms
(d) Path 1 loss probability = 0.01, Path 1 end-to-end delay = 45ms, Path 2 end-to-end delay = 180ms

stream in our evaluations, we capture the effective blocking (rbuf or sbuf) that will degrade any CMT association.

With CMT, further gains can be had over UnawareApp and AwareApp (which use a single SCTP association) in fault tolerance as well. Fault tolerance is a major motivation for, and benefit of, the multihoming feature in SCTP. In case of a network or path failure, an SCTP sender can *failover* to a different destination for sending data to a multihomed receiver. An SCTP sender normally sends data to only one receiver destination (called *primary destination*), and gathers information about paths to all other receiver destinations (called *alternate destinations*) through infrequent probes called *heartbeats*. Since these probes are infrequent, an SCTP sender may have stale or inadequate information about the alternate paths to a receiver. Throughput degradation occurs when a sender uses such information about alternate paths [13].

Due to inadequate information, an SCTP sender is unable to make informed decisions about a new destination to use in case of a network failure. A CMT sender avoids this problem because data sent concurrently on all paths act as frequent probes, reflecting current conditions of all paths to a receiver. A CMT sender has more accurate information about *all* paths to a receiver, which better assists a sender in detecting and responding to network failure events.

VI. Related Work

A. Socket Buffer Sizing

Research in socket buffer sizing [14]–[17] has focussed on the relationship between socket buffer size and transport layer (typically, TCP) throughput. Reference [15] is an early piece of work, which proposes buffer size auto-tuning mechanisms to maximize the sender's throughput on all active connections. [14], [16], [17] propose automatic buffer size tuning at the application layer based on inline (using the data stream) or offline (using probes) measurements. In general, research efforts in socket buffer sizing focus on sizing the send and/or receive socket buffers depending on application and network congestion dynamics, and assume single path transmission.

Our work does not focus on the resizing of socket buffers. We explore rbuf blocking due to differing path characteristics in multipath transmission and the alleviation of such blocking, *given a buffer size*. We note that socket buffer sizing for multipath transmission is an open research area.

B. Load Balancing

Research in load balancing at the application layer [18]–[20], transport layer [21]–[24], and at the network layer [25] have all attempted to devise mechanisms to distribute traffic over multiple paths through the network (See [3], [26] for a detailed comparison of CMT with other techniques). Where rbuf constraints are discussed [21]–[23], little is done to understand or alleviate blocking due to multipath transfer. Our work attempts to fill this gap by exploring rbuf blocking, and providing means for a transport layer sender to alleviate the consequent degradation.

VII. Summary

We presented an rbuf blocking problem which causes throughput degradation during multipath transfer. We evaluated five retransmission policies for CMT under different rbuf constraints. Simulation results show that RTX-SAME, shown to work well in non-CMT environments [13], and whose performance represents that of a data striping application, performs poorest. Better performance will result from any retransmission policy that takes loss rate into account. We select RTX-CWND and RTX-SSTHRESH as the recommended retransmission policies for CMT.

Investigation under different end-to-end delays revealed that CMT is more sensitive to rbuf constraints in environments with shorter end-to-end delay. CMT performs better than UnawareApp under all conditions studied; performance comparisons with AwareApp revealed that rbuf blocking can be a significant cause for throughput degradation in CMT. We suggest that a minimum rbuf of 128KB be used in practice to fully exploit CMT's parallelism.

We reemphasize that rbuf blocking is not specific to the transport layer; it applies to multipath transfer at other layers as well. Rbuf blocking cannot be eliminated or reduced by moving the functionality of CMT to a different layer or by reserving rbuf space per path. A significant benefit of CMT is that retransmission decisions can be made at the transport layer, thus considerably reducing rbuf blocking—a benefit that multipath transfer at any other layer does not have.

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] J. Iyengar, K. Shah, P. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming," in *SPECTS 2004*, San Jose, California, July 2004.

[2] J. Iyengar, P. Amer, and R. Stewart, "Retransmission Policies For Concurrent Multipath Transfer Using SCTP Multihoming," in *ICON 2004*, Singapore, Nov. 2004.

[3] J. Iyengar, P. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, (to appear).

[4] R. Stewart, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues," draft-ietf-tsvwg-sctpimpguide-16.txt, Oct. 2005, (work in progress).

[5] 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, Oct. 2000.

[6] A. Caro and J. Iyengar, "ns-2 SCTP module," http://pel.cis.udel.edu.

[7] S. Shakkottai, R. Srikant, A. Broido, and k. claffy, "The RTT Distribution of TCP Flows in the Internet and its Impact on TCP-based Flow Control," Tech. Rep., Cooperative Association for Internet Data Analysis (CAIDA), Feb. 2004.

[8] J. Iyengar, P. Amer, and R. Stewart, "Receive Buffer Blocking In Concurrent Multipath Transport," in *IEEE GLOBECOM*, St. Louis, Missouri, Nov. 2005.

[9] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," RFC2988, IETF, Nov. 2000.

[10] N. Jani and Krishna Kant, "SCTP Performance in Data Center Environments," Tech. Rep., Intel Corporation, 2005.

[11] 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.

[12] H. Ekstrom and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport," in *IEEE INFOCOM 2004*, Hong Kong, Mar. 2004.

[13] A. Caro, P. Amer, and R. Stewart, "Retransmission Policies for Multihomed Transport Protocols," *Computer Communications*, (in press).

[14] R. Prasad, M. Jain, and C. Dovrolis, "Socket Buffer Auto-Sizing for High-Performance Data Transfers," *Journal of Grid Computing*, vol. 1, no. 4, pp. 361–376, 2004.

[15] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP Buffer Tuning," in *ACM SIGCOMM*, Aug. 1998.

[16] T. Dunigan, M. Mathis, and B. Tierney, "A TCP Tuning Deamon," in *Supercomputing*, Nov. 2002.

[17] M. K. Gardner, W-C. Feng, and M. Fisk, "Dynamic Right-Sizing in FTP (drsFTP): Enhancing Grid Performance in User-Space," in *IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[18] T. Hacker and B. Athey, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," in *IEEE IPDPS*, Ft. Lauderdale, FL, Apr. 2002.

[19] H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The Case For Application-Level Network Striping For Data Inttensive Applications Using High Speed Wide Area Networks," in *IEEE Supercomputing (SC)*, Dallas, TX, Nov. 2000.

[20] M. Day, B. Cain, G. Tomlinson, and P. Rzewski, "A Model For Content Internetworking (CDI)," RFC3466, IETF, Feb. 2003.

[21] H.Y. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multihomed Mobile Hosts," in *ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Atlanta, Georgia, Sept. 2002.

[22] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths," in *USENIX*, June 2004.

[23] A. Abd El Al, T. Saadawi, and M. Lee, "LS-SCTP: A Bandwidth Aggregation Technique For Stream Control Transmission Protocol," *Computer Communications, Special issue on Protocol Engineering for Wired and Wireless Networks*, vol. 27, no. 10, June 2004.

[24] A. Argyriou and V. Madisetti, "Bandwidth Aggregation With SCTP," in *IEEE Globecom 2003*, San Fransisco, CA, Dec. 2003.

[25] 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.

[26] J. Iyengar, *End-to-End Concurrent Multipath Transfer Using Transport Layer Multihoming*, Ph.D. thesis, CIS Dept, University of Delaware, Aug. 2006.