Volume 32, issue 15, 15 September 2009
ISSN 0140-3664
ELSEVIER
computer
communications
Available online at
ScienceDirect
www.sciencedirect.com

Computer Communications 32 (2009) 1577–1587

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

# Concurrent Multipath Transfer during path failure ☆

Preethi Natarajan [a,*], Nasif Ekiz [b], Paul D. Amer [b], Randall Stewart [c]

[a] *Cisco Systems, 425 East Tasman Drive, San Jose, CA 95134, USA*
[b] *CIS Dept., 101 Smith Hall , University of Delaware, Newark DE 19716, USA*
[c] *Researcher, Chapin, SC 29036, USA*

ARTICLE INFO

*Article history:*
Received 2 April 2009
Received in revised form 5 May 2009
Accepted 6 May 2009
Available online 13 May 2009

*Keywords:*
Stream Control Transmission Protocol (SCTP)
Concurrent Multipath Transfer (CMT)
Multihoming
Multipath transport
Path failure

ABSTRACT

We investigate how path failure influences Concurrent Multipath Transfer (CMT) using SCTP multihoming. We show that CMT suffers from significant "receive buffer blocking" which degrades performance during both permanent and short-term failure. To improve performance, we introduce a new "Potentially-Failed" (PF) destination state, and revise CMT's failure detection and (re)transmission policies to include the PF state. Using simulation, we demonstrate that the modification called CMT-PF outperforms CMT during failure – even with aggressive failure detection thresholds and varying receive buffer constraints. In non-failure scenarios, CMT-PF performs on par or better but never worse than CMT. Finally, we confirm these simulation results using FreeBSD implementations of CMT and CMT-PF. Based on our findings, we recommend CMT-PF be used in existing and future CMT implementations and RFCs.

## 1. Introduction

Multihoming among networked machines and devices is a technologically feasible and increasingly economical proposition. Unlike TCP and UDP, the Stream Control Transmission Protocol (SCTP) [16] 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 end of the association. This binding allows a sender to transmit data to a multihomed receiver through different destination addresses, providing network interface redundancy and improved end-to-end fault tolerance.

Multihomed nodes may also be connected via multiple end-to-end paths. For instance, users may be simultaneously connected through dial-up/broadband, or via multiple wireless technologies (e.g., 802.11b, GPRS). Concurrent Multipath Transfer (CMT) [7] is an experimental SCTP extension that assumes multiple *independent* paths, and exploits these paths for *simultaneous* transfer of new data between end hosts. This work investigates CMT performance during path failures. We also highlight the conditions under which our findings would be pertinent to other transports that support multipath transfer (e.g., multipath TCP [19]).

This paper is organized as follows. Section 2 motivates this research by discussing the commonness of link failures in the Internet. Section 2 also overviews CMT's failure detection process, and how CMT's throughput degrades during failure detection. Section 3 outlines our proposed improvement – CMT with a "Potentially-Failed" destination state (CMT-PF). Using ns-2 simulations, Sections 4 and 5 evaluate CMT vs. CMT-PF in failure and congestion scenarios, respectively. Section 6 validates the simulation results with empirical experiments using FreeBSD implementations of CMT and CMT-PF. Section 7 concludes this work.

## 2. Problem description

This section discusses the prevalence of Internet path failures, and outlines how CMT's performance degrades during path failures.

### *2.1. Prevalence of path failures*

Internet paths fail when a router or a link connecting two routers fails due to planned maintenance or unplanned accidents such as hardware malfunction or software error. Ideally, the routing system detects unplanned link failures, and reconfigures routing tables to avoid trying to route traffic via a failed link. Using data from an ISP's routing logs, Markopoulou et al. [9] observed that link failures are part of everyday operation. Around 80% of the failures are unplanned, and the time-to-repair for any particular failure can

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

* Corresponding author.
*E-mail addresses:* prenatar@cisco.com (P. Natarajan), ekiz@cis.udel.edu (N. Ekiz), amer@cis.udel.edu (P.D. Amer), randall@lakerest.net (R. Stewart).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.05.001

be on the order of hours. Using probes Feamster et al. [5] observed that Internet paths fail often. The failures appear everywhere and are not confined to a few bad links.

Existing research also highlights problems with Internet backbone routing that result in long route convergence times. Labovitz et al. [8] shown that Internet's interdomain routers may take as long as tens of minutes to reconstruct new paths after a failure. During these delayed convergences, end-to-end Internet paths experience intermittent loss of connectivity in addition to increased packet loss, latency, and reordering. Using probes, Paxson and Zhang et al. [14,20] found that "significant routing pathologies" prevent selected pairs of hosts from communicating about 1.5% to 3.3% of the time. Importantly, the authors also find that this trend has not improved with time. In [3], the authors use probes to confirm that failure durations are heavy-tailed, and report that 5% of detected failures last more than 2.75 h, and as long as 27.75 h. The pervasiveness of path failures in practice motivates us to study their impact on CMT.

### 2.2. Failure detection in CMT

Since CMT is an extension to SCTP, CMT retains SCTP's failure detection process. A CMT sender employs a tunable failure detection threshold called *Path.Max.Retrans (PMR)* [16]. As shown in the finite state machine of Fig. 1, a destination is in one of the two states – *Active* or *Failed* (inactive). A destination is active as long as acks come back for data or heartbeats (probes) sent to that destination. When a sender experiences more than PMR consecutive timeouts while trying to reach a specific active destination, that destination is marked as failed. Only heartbeats (i.e., no data) are sent to a failed destination. A failed destination returns to active when the sender receives a heartbeat ack. RFC4960 proposes a default PMR = 5, which translates to $\geqslant$63 s (6 consecutive timeouts) for failure detection.

### 2.3. Receive buffer blocking in CMT

Iyengar et al. [6] explored the "receive buffer blocking" problem in CMT, where Transport Protocol Data Unit (TPDU) losses throttle data transmission once the CMT receiver's buffer (rbuf) is filled with out-of-order data. Even though the sender's cwnd would allow new data to be transmitted, rbuf blocking (i.e., flow control) stalls the sender, causing throughput degradation.

Rbuf blocking problem cannot be eliminated in CMT [6]. To reduce rbuf blocking's negative impact during congestion, Iyengar et al. [6] proposed different retransmission policies that use heuristics for faster loss recovery. These policies consider different path properties such as loss rate and delay, and try to reduce rbuf blocking by sending retransmissions on a path with lower loss or delay. In practice, a path's loss rate can only be estimated, so Iyengar et al. [6] proposed the RTX_SSTHRESH policy, where retransmissions are sent on the path with the largest slow-start threshold. Since RTX_SSTHRESH outperformed other retransmission policies during congestion, Iyengar et al. [6] recommended the RTX_SSTHRESH policy for CMT. However, Iyengar et al. [6] did not consider CMT performance during path failures, which is the focus of this work.

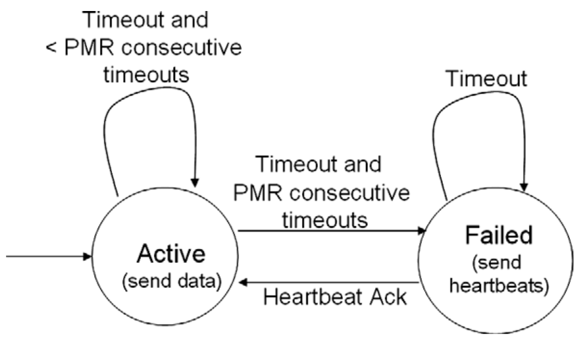

**Fig. 1.** Failure detection in CMT.

### 2.4. CMT performance during path failures

CMT's failure-induced rbuf blocking is explained via the timeline diagram shown in Fig. 2. The CMT sender (A) has two interfaces – $A_1$ and $A_2$, and transmits data to receiver (B) with two interfaces – $B_1$ and $B_2$. All four addresses are bound in the CMT association such that the sender employs the two independent paths – path 1 and path 2, for data transmission.

In Fig. 2, the initial cwnd for each path = 2 MTUs. Each TPDU consists of an MTU-sized data chunk, and is assigned a unique Transmission Sequence Number (TSN) [16]. C*i* and O*i* denote the cwnd in number of MTUs, and the number of outstanding TPDUs, respectively, on path *i*. A SACK labeled $\langle Sa, b - c; Rd \rangle$ cumulatively acknowledges all TSNs up to and including *a*, selectively acknowledges TSNs *b* through *c* (missing report for TSNs $a + 1$ through $b - 1$), and advertises a receiver window capable of buffering *d* more TSNs. In Fig. 2's example, the transport layer receive buffer can hold a maximum of 5 TSNs, and its contents are listed after the reception of every TSN.

Both forward and reverse paths between $A_1$ and $B_1$ fail just after TSN 2 enters the network. Hence, TSN 2 and the SACK for TSN 1 are presumed lost. TSNs 3 and 4 are received out of order and stored in the receive buffer. Each of these TSNs triggers a SACK to the sender. The CMT sender uses the *Cwnd Update for CMT* (CUC) algorithm [7] to decouple a path's cwnd evolution and data ordering. On receiving the SACK triggered by TSN 3, the sender uses CUC to increment C2 to 3, and decrement O1 and O2 to 1. The available receive buffer space for new data, calculated as advertised receive window (=4) – total outstanding (=2), allows the sender to transmit two TSNs, 5 and 6, on path 2. On path 1, even though 1 MTU worth of new data could be transmitted ($C1 > O1$), rbuf blocking, i.e., flow control, throttles data transmission. On receiving the SACK triggered by TSN 4, the sender increases C2 = 4, and decreases O2 = 2. However, lack of rbuf space inhibits transmission of new data on path 2.

Since $O2 < C2$, the SACKs triggered by TSNs 5 and 6 do not increment C2 [16] (discussed later). But these SACKs decrement O2. Even though $O2 < C2$, flow control continues to prevent data transmission on path 2.

When path 1's retransmission timer expires, the sender detects the loss of TSN 2. Note that this timeout is the first of the 6 (when PMR = 5) consecutive timeouts needed to detect path 1's failure. After this timeout, C1 = 1, O1 = 0, and path 1's RTO value is doubled [16]. The CMT sender employs the RTX_SSTHRESH policy and retransmits TSN 2 on path 2.

On receiving TSN 2, the CMT receiver delivers data from TSNs 2 to 6 to the application. The corresponding SACK advertises a receive window of 5 TSNs. Once receive buffer becomes available, the sender transmits TSN 7 on path 1, and TSNs 8–11 on path 2. Due to path 1 failure, TSN 7 is lost, and TSNs 8–11 are received out-of-order and fill the receiver's buffer. Once again, flow control inhibits data transmission until the sender experiences a timeout on path 1, detects loss of TSN 7, and successfully retransmits TSN 7 on path 2.

To generalize, a CMT sender continues to transmit data on a failed path until the corresponding destination is *marked* failed, i.e., until the sender experiences (PMR + 1) consecutive timeouts on the failed path. During failure detection, out-of-order data received via the non-failed path(s) fill the receive buffer and stall data transmission until the lost TPDUs are successfully retransmitted. Since losses on a failed path are detected only after a timeout,

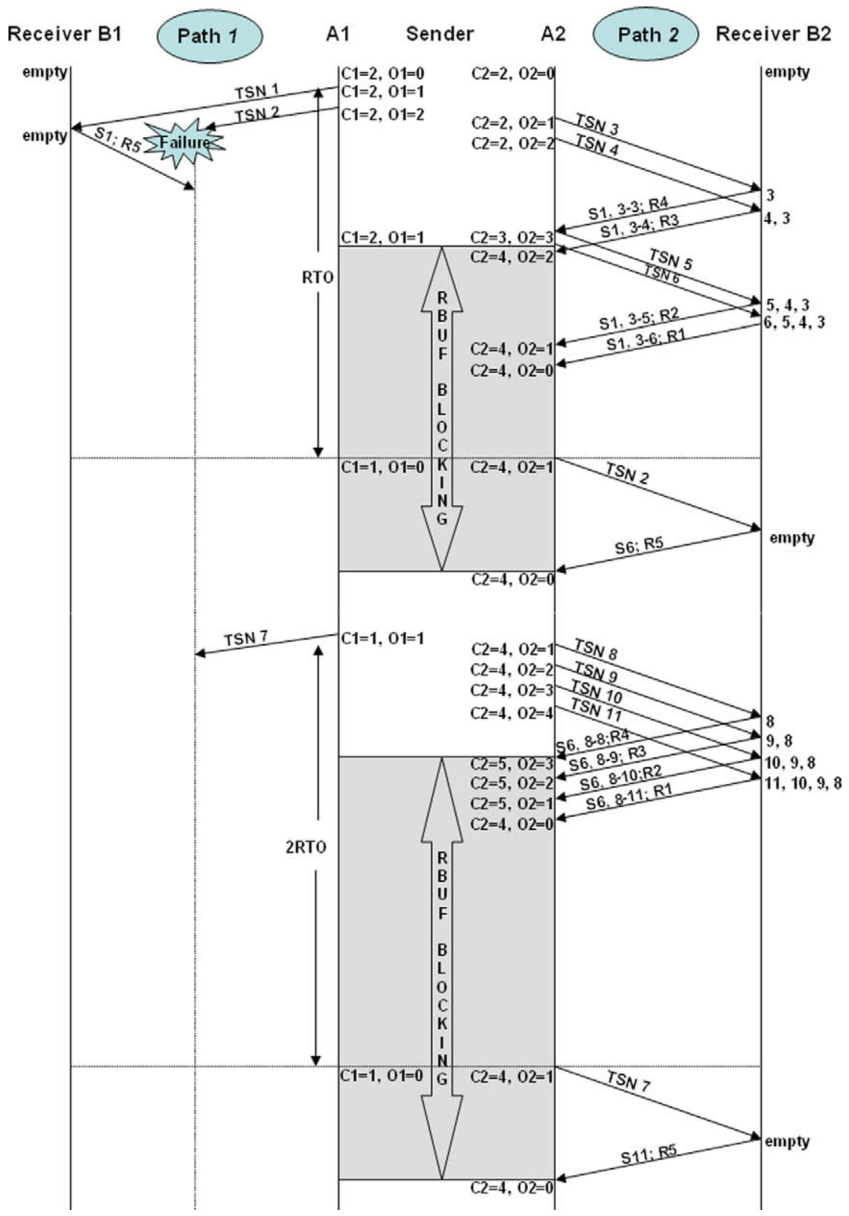

**Fig. 2.** CMT's rbuf blocking during path failure.

the length of an rbuf blocking instance is proportional to the failed path's RTO. Until failure detection, the sender keeps transmitting data on the failed path, resulting in successive instances of rbuf blocking. Each instance is exponentially longer than the previous instance due to the exponential backoff of RTO values.

Ref. [19] discusses the advantages of resource pooling and argues the need for multipath-capable TCP (a.k.a. multipath TCP) to facilitate resource pooling. Multipath TCP functions similar to CMT – multipath TCP extends TCP to support multihoming, and sets up multiple subflows to simultaneously transmit data over multiple paths. Ref. [19] acknowledges that a single receive window can interact badly with multipath TCP connections. Data received out-of-order on one subflow (path) may fill the receive window and flow control would stall data transmission on other subflows (paths). Note that this problem is conceptually similar to CMT's congestion-induced rbuf blocking. Consequently, when multipath TCP's subflows share a single receive window, the multipath TCP connection will endure failure-induced rbuf blocking and degraded throughput.

In CMT, rbuf blocking results in the following side-effects that further degrade throughput.

### 2.4.1. Preventing congestion window growth

Note that rbuf blocking prevents the sender from fully utilizing the cwnd. When the amount of outstanding data is less than the cwnd, RFC4960 prevents the sender from increasing the cwnd for future SACKs. For example, in Fig. 2, when the sender receives the SACKs for TSNs 5, 6, 9–11, the sender cannot increment C2.

### 2.4.2. Reducing congestion window

To reduce burstiness in data transmission, an SCTP sender employs a congestion window validation algorithm similar to [15]. During every transmission, the sender uses the *MaxBurst* parameter (recommended value of 4) as follows:

$$\text{If} \ \ ((outstanding + MaxBurst * MTU) < cwnd)$$
$$cwnd = outstanding + MaxBurst * MTU$$

This algorithm decays the cwnd after an idle period so that at the next sending opportunity, the sender will not transmit more than ($MaxBurst * MTU$) bytes of data. During rbuf blocking, the amount of outstanding data can become less than the cwnd. In such cases, the above rule is triggered and further reduces the cwnd. In Fig. 2, when the SACK triggered by TSN 11 arrives at the sender, O2 decrements to 0. The window validation algorithm causes C2 to be reduced to 4 (O2 (=0) + *MaxBurst* (=4)).

## 3. Solution

Caro [2] recommends lowering the PMR value for SCTP flows in Internet-like environments. Correspondingly, lowering the PMR for CMT flows reduces the number of rbuf blocking episodes during failure detection. However, lowering the PMR is an incomplete solution to the problem since a CMT flow may be rbuf blocked for any $PMR > 0$ (Section 4). Also, a tradeoff exists on deciding the value of PMR – a lower value reduces rbuf blocking but increases the chances of spurious failure detection, whereas a higher PMR increases rbuf blocking and reduces spurious failure detection in a wide range of environments.

### 3.1. Details of CMT-PF

To mitigate failure-induced rbuf blocking, our proposed solution introduces a new "Potentially-Failed" state to the FSM of Fig. 1. The expanded FSM, shown in Fig. 3, assumes that loss detected by a timeout implies either severe congestion or failure en route. After a single timeout on a path, a sender is unsure, and marks the corresponding destination as "Potentially-Failed" (PF). A PF destination is not used for data transmission or retransmission. CMT's retransmission policies are augmented to include the PF state. CMT with the new set of retransmission policies is called CMT-PF [10,12]. Details of CMT-PF are:

- If a TPDU loss is detected by RFC4960's threshold number of missing reports, one of CMT's current retransmission policies, such as RTX_SSTHRESH, is used to select an active destination for "fast" retransmission.
- If a TPDU loss is detected after a timeout, the corresponding destination transitions to the PF state (Fig. 3). The sender never transmits data to a PF destination. However, when all destinations are in the PF state, the sender transmits data to the destination with the fewest consecutive timeouts. In case of a tie, data is sent to the last active destination. This exception ensures that CMT-PF does not perform worse than CMT when all paths have potentially failed (discussed further in Section 5).
- Heartbeats are sent to PF destination(s) with an exponential backoff of RTO after every timeout until either (i) a heartbeat ack transitions the destination back to the active state, or (ii) an additional PMR consecutive timeouts confirm the path failure, upon which the destination transitions to the failed state, and heartbeats are sent with a lower frequency as described in RFC4960.
- If ever a heartbeat ack indicates a PF destination is alive, that destination's cwnd is set to either 1 MTU (CMT-PF1), or 2 MTUs (CMT-PF2), and the sender follows the slow start algorithm to transmit data to this destination. Detailed analysis on the cwnd evolution of CMT-PF1 vs. CMT-PF2 can be found in Section 5.
- Acks for retransmissions do not transition a PF destination back to the active state, since a sender cannot disambiguate whether the ack was for the original transmission or the retransmission(s).

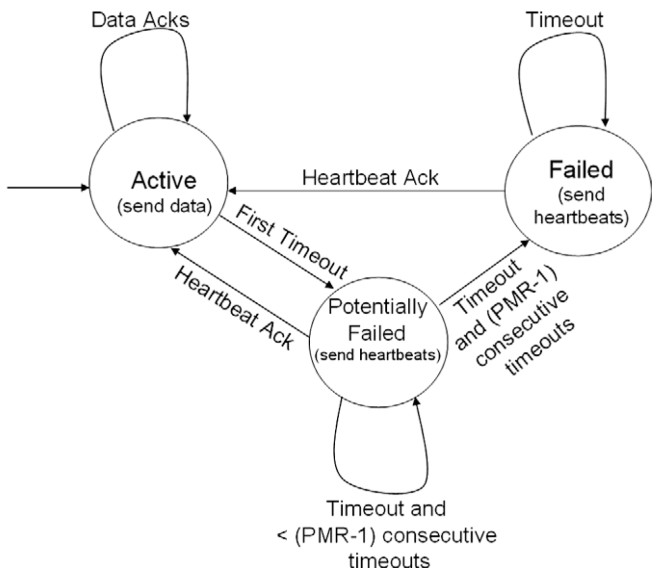

**Fig. 3.** Failure detection in CMT-PF (PMR >0).

### 3.2. CMT-PF data transfer during failure

Fig. 4 depicts an analogous CMT-PF timeline for the scenario described in Fig. 2. All events are identical between the two figures up to the first timeout on path 1. After this timeout, the CMT-PF sender transitions path 1 to the PF state, transmits a heartbeat on path 1, and retransmits TSN 2 on path 2. The heartbeat loss on the failed path (path 1) is detected on the next timeout. This timeout is the second of (PMR + 1) consecutive timeouts required to detect path 1 failure. Meanwhile, receiver buffer space is released once the retransmitted TSN 2 is received on path 2. From this point onwards, data is transmitted only on path 2, without further rbuf blocking.

Section 2D discussed how other multipath transports (e.g., multipath TCP [19]) may suffer from rbuf blocking during path failures. We note that the CMT-PF framework can be adapted to such multipath transports to mitigate the negative consequences of failure-induced rbuf blocking.

## 4. Evaluations during failure

CMT-PF is evaluated using the University of Delaware's ns-2 SCTP/CMT module [13,4]. The failure experiments discussed in this section use a simple topology and loss model (Fig. 5). In the next section, we employ a more realistic (and complex) simulation topology and loss model to evaluate CMT and CMT-PF during congestion.

In Fig. 5, the multihomed sender, A, has two independent paths to the multihomed receiver, B. The edge links between A (or B) to the routers represent last-hop link characteristics. The end-to-end one-way delay is 45 ms on both paths, representing typical coast-to-coast delays experienced by significant fraction of the flows in the Internet [18]. The sender A transfers an 8MB file to receiver B using both path 1 and path 2. This file transfer uses a single-streamed CMT or CMT-PF association such that all data is delivered in sequence to the receiving application. Path 2 fails during the file transfer; this failure is simulated by bringing down the bidirectional link between routers $R_{20}$ and $R_{21}$. Unless stated otherwise, the PMR = 5, rbuf = 64 KB, and both paths experience Bernoulli losses with low loss rate (<1%).

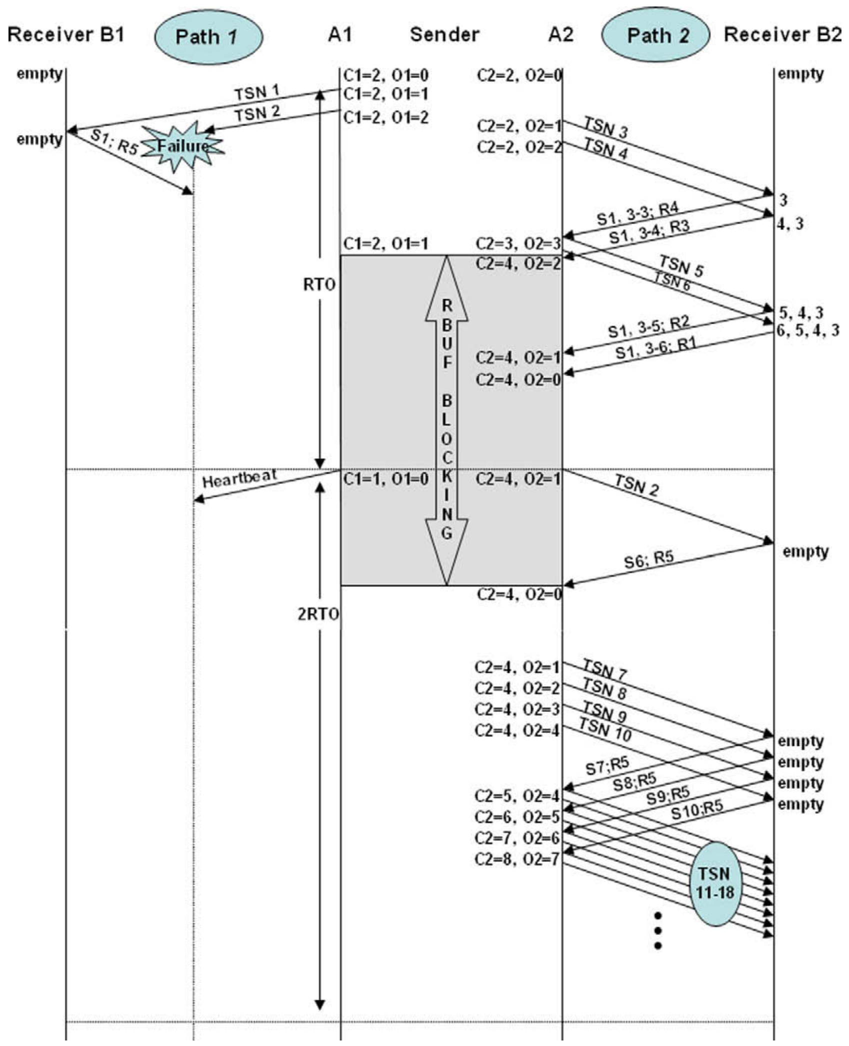

**Fig. 4.** CMT-PF reduces rbuf blocking during path failure.

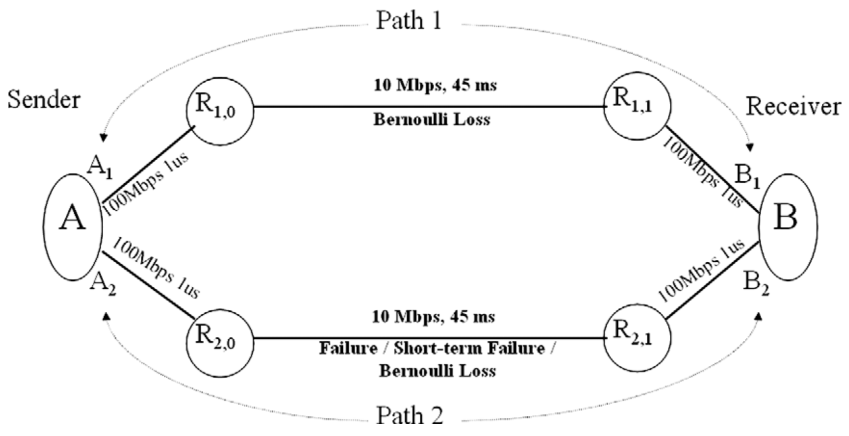

**Fig. 5.** Simulation topology for failure experiments.

### *4.1. Evaluations during permanent path failure*

In the following experiments, path 2 fails permanently 5 s after the file transfer begins.

#### *4.1.1. Single permanent path failure*

Neither path experiences congestion in this experiment, which helps to highlight how CMT and CMT-PF differ during failure detection.

Path 2's failure causes back-to-back timeouts at the sender. Both senders (CMT and CMT-PF) experience the first timeout on path 2 at ~6 s, and detect the failure after 6 back-to-back timeouts (PMR = 5), at ~69 s (Fig. 6). During the failure detection period, CMT continues to transmit data on path 2, experiencing consecutive timeouts and recurring rbuf blocking instances. CMT's throughput suffers until 69 s (i.e., until failure detection), after which CMT uses path 1 alone and completes the file transfer at around 80 s. On the other hand, CMT-PF transitions path 2 to PF

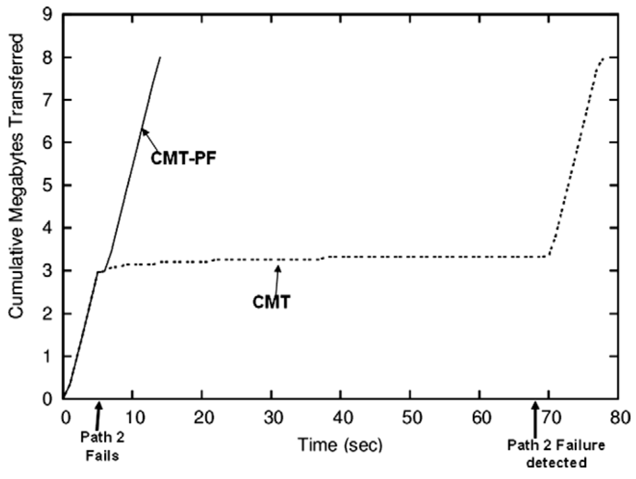

**Fig. 6.** CMT vs. CMT-PF during permanent path failure.

state after the first timeout, and then transmits only heartbeats on path 2 avoiding further rbuf blocking. Reduced rbuf blocking helps CMT-PF to complete the file transfer (~15 s) using path 1 alone, even before path 2's failure is detected.

*4.1.2. Varying failure detection thresholds (PMR values)*

To achieve faster yet robust failure detection, Caro [2] argued for varying the PMR based on a network's loss rate, and suggests PMR = 3 for the Internet. Since the sender detects a permanent path failure after (PMR + 1) consecutive timeouts, CMT's failure-induced rbuf blocking varies as the PMR varies. Let,

$T_f$ = time when path 2 fails.
$T_d$ = time when the sender detects path 2 failure (after PMR + 1 consecutive timeouts).
The goodput during failure detection ($G$) is defined as,
$G$ = (application data received during $T_f$ to $T_d$) $\div (T_d - -T_f)$.

Fig. 7 plots CMT vs. CMT-PF average goodput ($G$) with 5% error margin. The dashed line in Fig. 7 denotes the maximum attainable goodput of an SCTP file transfer (application data received ÷ transfer time) using path 1 alone.

When the failure detection threshold is most aggressive (PMR = 0), both CMT and CMT-PF detect path 2 failure after the first timeout. Effectively, CMT-PF's failure detection never engages the PF state transition (Fig. 3), and is equivalent to CMT's failure detection procedure. Both senders experience identical rbuf blocking, and perform similarly (Fig. 7). As PMR increases, the number of rbuf blocking instances during failure detection increases, resulting in increasing performance benefits with CMT-PF. In Fig. 7, as PMR and the failure detection period increase, CMT-PF's goodput increases, whereas CMT's goodput decreases. Starting from PMR = 3, CMT-PF's goodput is comparable or equal to the maximum attainable goodput. To conclude, *during path failures, CMT-PF performs as well as CMT for PMR = 0, and better than CMT for PMR >0.*

*4.2. Evaluations during short-term failure*

In the following experiments, path 2 in Fig. 5 fails temporarily during the file transfer between A and B. The link connecting routers $R_{20}$ and $R_{21}$ goes down after 5 s from the start of file transfer, and is restored 5 s later.

*4.2.1. During single short-term failure*

Neither path experiences congestion in this experiment, which helps to highlight the differences between CMT and CMT-PF during a short-term failure.

The short-term failure is long enough for the sender (CMT or CMT-PF) to experience three back-to-back timeouts on path 2. As in the failure case, CMT transmits data on path 2 after each of these timeouts, while CMT-PF does not. Therefore, CMT suffers from consecutive rbuf blocking and lower throughput than CMT-PF (Fig. 8). Once path 2 recovers at time = 10, CMT's data and CMT-PF's heartbeat transmissions on the path (after the 3rd timeout – ~12.5 s) are successful, and both CMT and CMT-PF complete the file transfer without further rbuf blocking.

*4.2.2. Varying receive buffer sizes*

This second short-term failure experiment analyzes CMT vs. CMT-PF for varying levels of receive buffer constraints (receive buffer sizes). Let

$T_f$ = time when path 2 fails.
$T_r$ = time when path 2 is restored.
The goodput during the short-term failure ($G$) is defined as,
$G$ = (application data received during $T_f$ to $T_r$) $\div (T_r - T_f)$.

Fig. 9 plots CMT vs. CMT-PF average goodput ($G$) with 5% error margin. As the receive buffer becomes more constrained, i.e., as

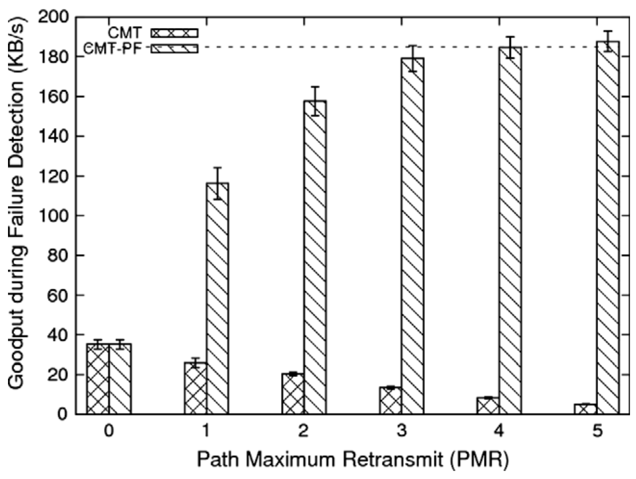

**Fig. 7.** CMT vs. CMT-PF under varying PMR values.

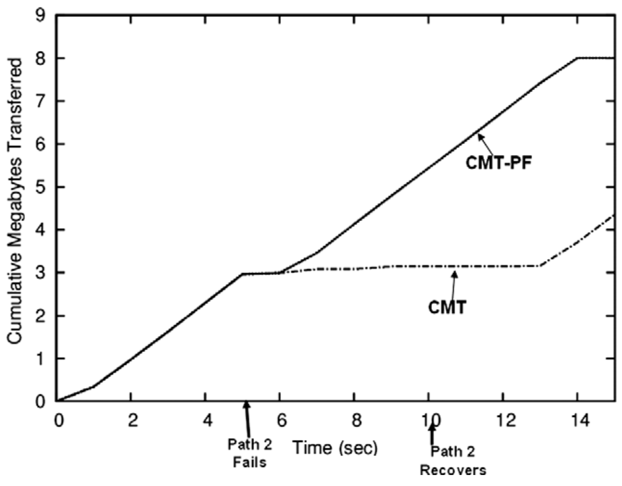

**Fig. 8.** CMT vs. CMT-PF during short-term path failure.

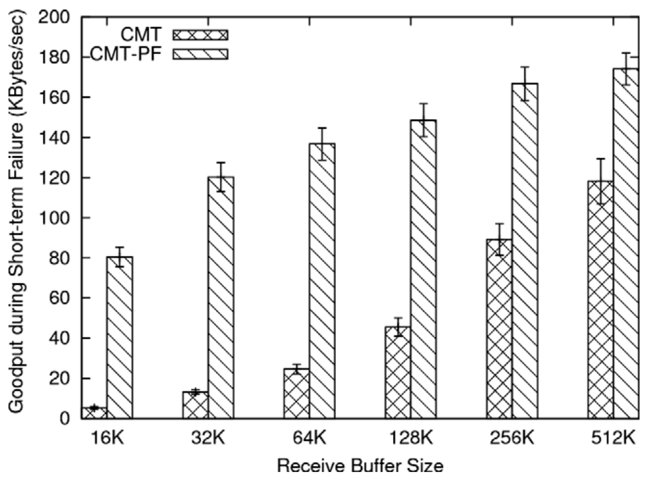

**Fig. 9.** CMT vs. CMT-PF under varying receive buffer sizes.

rbuf size decreases, the chances of rbuf blocking increases. *Consequently, CMT-PF's ability to alleviate rbuf blocking is more valuable for smaller rbuf sizes.*

## 5. Evaluations during congestion

In this section we investigate how CMT-PF performs when timeouts are caused by non-failure scenarios such as congestion [11,12]. Consider the case when a timeout on path $p$ is due to congestion rather than failure. Depending on the rbuf size and the different paths' characteristics, the transport sender may or may not be rbuf blocked, leading to the following two scenarios.

### 5.1. Sender is rbuf-limited

Both CMT and CMT-PF senders cannot transmit new data until the rbuf blocking is cleared, i.e., until after successful retransmission(s) of lost data. The only difference is that CMT considers $p$ for retransmissions, whereas CMT-PF transmits a heartbeat on $p$, and tries to retransmit lost data on other active paths. (If all destinations are in the PF state, the CMT-PF sender transitions the destination with the fewest consecutive timeouts to the active state (Section 3A), and retransmits lost data to this destination.)

### 5.2. Sender is not rbuf-limited

Assume that SCTP PDUs (data or heartbeats) transmitted after the first timeout on path $p$ are successfully received. In CMT, the cwnd allows 1 MTU of new data transmission on $p$ (Fig. 10a), and the corresponding SACK increments path $p$'s cwnd by 1 MTU. After1 RTT after the timeout (shown by point A in Fig. 10a), (i) $p$'s cwnd is 2 MTUs, and (ii) 1 MTU of new data has been successfully sent on $p$.

CMT-PF transmits a heartbeat on $p$ and new data on other active path(s). (*Note:* if all destinations are marked PF, the CMT-PF sender transitions one of the PF destinations to the active state.) Path $p$ is marked active when the heartbeat ack reaches the sender. Therefore, after 1 RTT from the timeout (point B in Fig. 10b), (i) $p$'s cwnd is 1 MTU (CMT-PF1), and (ii) no new data has been sent on $p$. Comparing points A and B in Fig. 10a and b, respectively, it can be seen that CMT has a 1 RTT "lead" in $p$'s cwnd growth. Assuming no further losses on $p$, after $n$ RTTs, the cwnd on $p$ will be $2^n$ with CMT, and $2^{n-1}$ with CMT-PF1.

To avoid the 1 RTT lag in CMT-PF1's cwnd evolution, we propose CMT-PF2 which initializes $p$'s cwnd to 2 MTUs after receiving a heartbeat ack (shown by point C in Fig. 10c). Assuming that today's

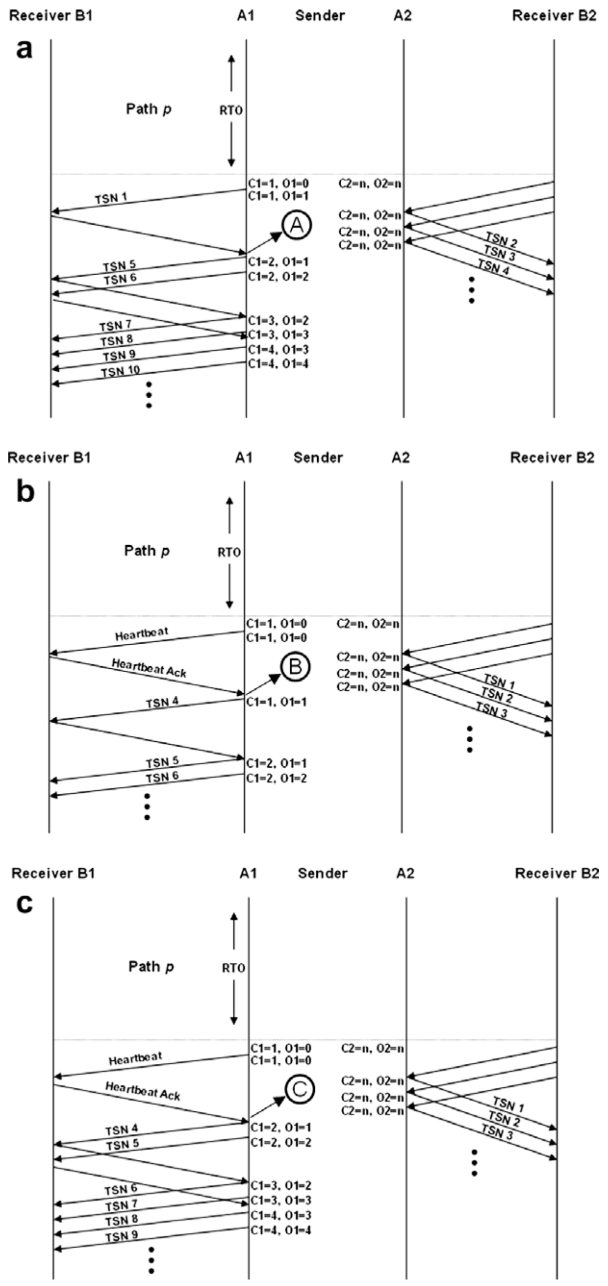

**Fig. 10.** CMT vs. CMT-PF during congestion. (a) CMT; (b) CMT-PF1; (c) CMT-PF2.

Internet router queues deal with packets rather than bytes, the successful routing of a heartbeat PDU is equivalent to the successful routing of a data PDU. Hence, a heartbeat ack can be used to clock the transport layer sender in the same way as a data ack. (Under similar reasoning, even an SCTP sender could initialize a destination's cwnd to 2 MTUs after transitioning the destination from failed to active state, but that recommendation is outside the scope of this paper.) Since CMT-PF2 is clearly superior, we no longer consider CMT-PF1, and in the following sections, any reference to CMT-PF implies CMT-PF2.

### 5.3. Simulation setup

In the dual-dumbbell topology shown in Fig. 11, each router, $R$, is attached to five edge nodes. Dual-homed edge nodes A and B are the transport sender and receiver, respectively. The other edge nodes are single-homed, and introduce background cross-traffic

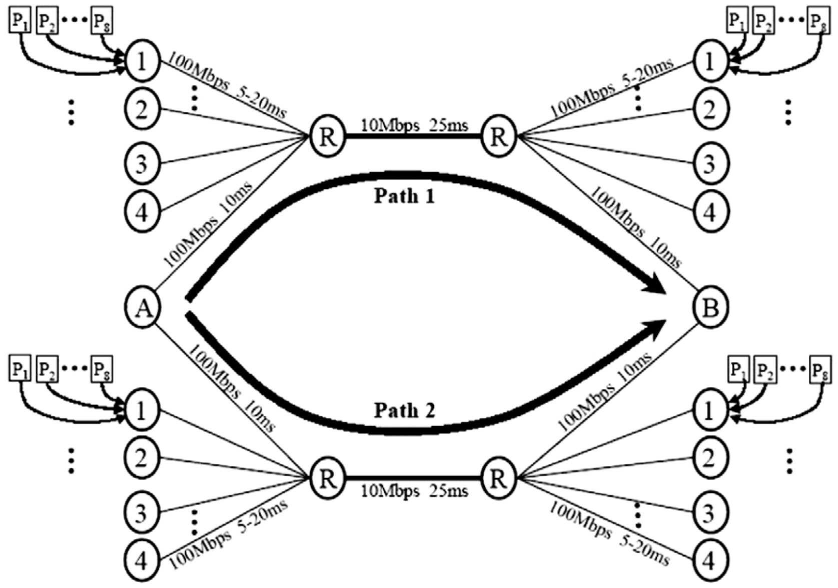

**Fig. 11.** Topology for non-failure experiments.

that instigates bursty periods of congestion, and hence bursty congestion losses at the routers. Their last-hop propagation delays are uniformly distributed between 5 and 20 ms, resulting in end-to-end one-way propagation delays ranging ~35–65 ms [18]. All links (both edge and core) have a buffer size twice the link's bandwidth-delay product, a reasonable setting in practice. Base on intuition, we believe that the final conclusions regarding CMT vs. CMT-PF are independent of the actual bandwidth and delay configurations used in the topology, as long as these configurations are similar on both paths.

Each single-homed edge node has eight traffic generators, introducing cross-traffic with a Pareto distribution [1]. The cross-traffic flows start at random times during the initial 5 s of the simulation. After an initial warm-up period, at time = 10 s, sender A begins transmitting a 32-MB file to receiver B over paths 1 and 2. This file transfer uses a single-streamed CMT or CMT-PF association such that all data is delivered in sequence to the receiving application. For both CMT and CMT-PF flows, rbuf = 64KB, PMR = 5, and loss rates are controlled by varying the cross-traffic load. The graphs in the subsequent discussions plot the average goodput (file size ÷ transfer time) of CMT vs. CMT-PF with 5% error margin.

## 5.4. Evaluations during symmetric loss conditions

In the symmetric loss case, the aggregate cross-traffic load on both paths are similar, and vary from 40% to 100% of the core link's bandwidth.

Both CMT and CMT-PF perform similarly (Fig. 12) during low loss rates (i.e., low cross-traffic), since, most of the TPDU losses are recovered via fast retransmits, not timeout recoveries. As the cross-traffic load increases causing the loss rate to increase, the number of timeouts on each path increases and *CMT-PF performs on par or insignificantly better than CMT (Fig. 12). This confirms that the PF state transition does not penalize CMT-PF performance during symmetric path loss conditions.*

## 5.5. Evaluations during asymmetric loss conditions

In practice, different paths are likely to experience asymmetric loss rates. For the next set of experiments, paths 1 and 2 experience different cross-traffic loads. The aggregate background cross-traffic on path 1 is set to 50% of the core link bandwidth, while the background on path 2 varies from 50% to 100% of the core link bandwidth.

Note that rbuf blocking depends on the frequency of loss events (loss rate), and the duration of loss recovery. As the loss rate increases, the probability that a sender experiences consecutive timeout events on the path increases. After the first timeout, CMT-PF transitions the path to PF, and avoids data transmission on the path (as long as another active path exists) until a heartbeat-ack confirms the path as active. But, a CMT sender suffers back-to-back timeouts on *data* sent on the path, with exponential backoff of RTO. As path 2's cross-traffic load increases, the probability that a sender experiences back-to-back timeouts on path 2 increases. CMT suffers more consecutive timeouts on data (Table 1) resulting in longer rbuf blocking periods when compared with CMT-PF. Therefore, as the asymmetry of the paths increases, CMT-PF performs better than CMT (Fig. 13).

The asymmetric loss experiment helps highlight an important difference in CMT vs. CMT-PF's transmission strategy. In CMT, RTX_SSTHRESH is a retransmission policy, and is not applied to new data transmissions. In CMT-PF, a path is marked PF after a

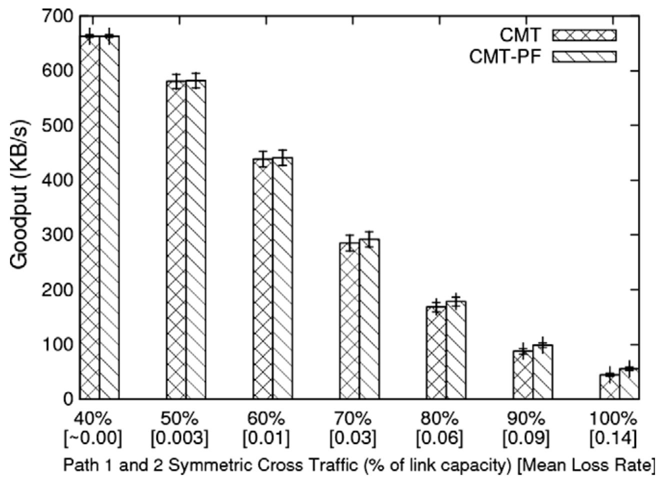

**Fig. 12.** CMT vs. CMT-PF during symmetric loss.

**Table 1**
CMT vs. CMT-PF mean consecutive data timeouts on path 2.

| Variant | Path 2 cross-traffic (%) | # of consecutive timeouts | | | |
|---|---|---|---|---|---|
| | | 2 | 3 | 4 | 5 |
| CMT | 70 | 0.49 | 0.02 | 0 | 0 |
| CMT-PF | | 0 | 0 | 0 | 0 |
| CMT | 80 | 1.13 | 0.07 | 0 | 0 |
| CMT-PF | | 0 | 0 | 0 | 0 |
| CMT | 90 | 3.73 | 0.60 | 0.09 | 0.02 |
| CMT-PF | | 0.02 | 0.02 | 0 | 0 |
| CMT | 100 | 9.42 | 1.62 | 0.18 | 0.04 |
| CMT-PF | | 0.04 | 0.04 | 0 | 0 |

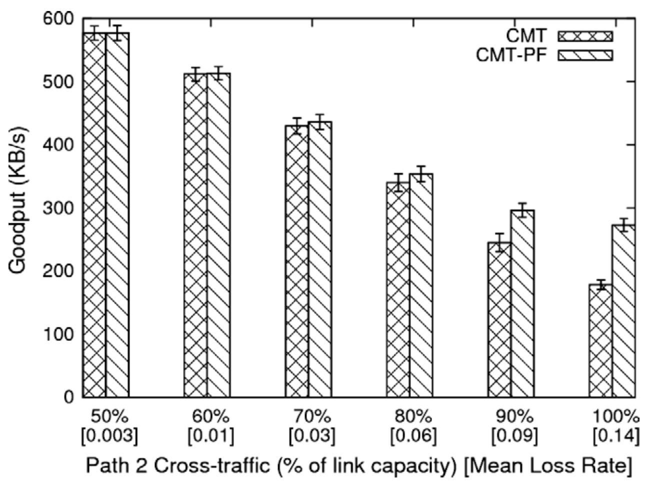

**Fig. 13.** CMT vs. CMT-PF during asymmetric loss conditions.

**Table 2**
CMT vs. CMT-PF mean number of transmissions.

| Variant | Path 2 cross-traffic % | Aggregate transmissions | | |
|---|---|---|---|---|
| | | Path 1 | Path 2 | Path 1/path 2 |
| CMT | 70 | 13,857 | 9486 | 1.5 |
| CMT-PF | | 14,002 | 9344 | 1.5 |
| CMT | 80 | 15,530 | 7902 | 2.0 |
| CMT-PF | | 16,029 | 7416 | 2.2 |
| CMT | 90 | 17,137 | 6401 | 2.7 |
| CMT-PF | | 18,153 | 5362 | 3.4 |
| CMT | 100 | 18,093 | 5508 | 3.3 |
| CMT-PF | | 20,318 | 3193 | 6.4 |

timeout, and as long as active path(s) exist, CMT-PF avoids retransmissions on the PF path. Once the retransmissions are all sent, CMT-PF's data transmission strategy is applied to new data, and CMT-PF avoids new data transmissions on the PF path. As shown in Table 2, when compared to CMT, CMT-PF reduces the number of (re)transmissions on the higher loss rate path 2 and (re)transmits more on the lower loss rate path 1. This transmission difference (ratio of transmissions on path 1 over path 2) between CMT-PF and CMT increases as the paths become more asymmetric in their loss conditions.

In summary, *CMT-PF does not perform worse than CMT during asymmetric path loss conditions. In fact, CMT-PF is a better transmission strategy than CMT, and performs better as the asymmetry in path loss increases.*

## 6. CMT-PF implementation

We extended the FreeBSD CMT implementation to incorporate CMT-PF. The following experiments were performed using this

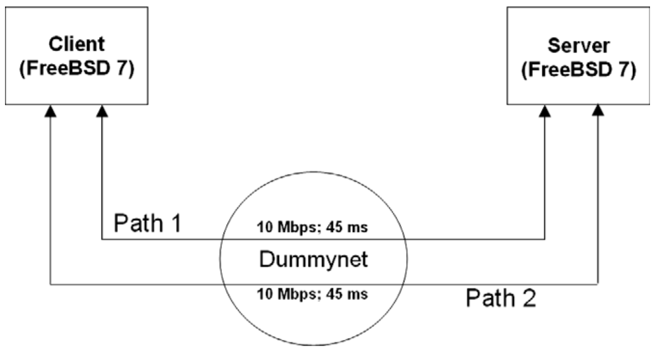

**Fig. 14.** Emulation topology for CMT vs. CMT-PF experiments.

implementation with the intention of validating the simulation-based conclusions in Sections 4 and 5.

The experimental topology, shown in Fig. 14, consists of a client and server node running FreeBSD 7 and an intermediate node running the Dummynet traffic shaper [17] which emulates Bernoulli losses. The server and client are connected by two independent paths, with symmetric bandwidth and propagation delay characteristics. The forward and reverse queue sizes for both paths are set to 1000 KB. The transport layer receive window = 64 KB, and PMR = 5. At time = 0, the server initiates a bulk file transfer to the client. This file transfer uses a single-streamed CMT or CMT-PF association such that all data is delivered in sequence to the receiving application.

### 6.1. Single permanent path failure

This single permanent path failure scenario is similar to the scenario described in Section 4A1. Neither path experiences loss. At time = 5 s, path 2 fails; this failure is emulated by having Dummynet block all packets traversing via path 2 to and from the client and server, respectively. Fig. 15 plots the cumulative bytes received at the client during this transfer.

As observed in the simulations (Fig. 6), path 2's failure causes consecutive timeouts and rbuf blocking instances in CMT, which prevents data transmission until failure detection time = 69 s). After failure detection, CMT transmits data using only path 1, completing the file transfer at time = 80 s. The CMT-PF sender transitions path 2 to PF after the first timeout (at time = 6.5 s), and transmits only heartbeats on path 2. Data transmission continues on path 1 and the file transfer finishes at time = 18 s. This

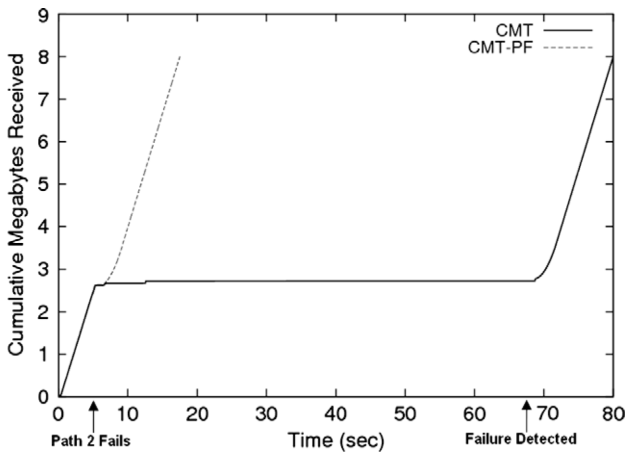

**Fig. 15.** CMT vs. CMT-PF during permanent path failure.

experiment confirms that the *FreeBSD CMT-PF implementation behaves as expected during a path failure*.

*6.2. Symmetric and asymmetric loss conditions*

Similar to the congestion scenarios in Section 5, the following two experiments compare CMT and CMT-PF when the paths experience symmetric and asymmetric congestion levels, respectively. Note that the emulation and simulation topologies assume different loss models. Therefore, to facilitate a reasonable comparison between the emulation and simulation results, the Bernoulli loss rates for paths in Fig. 14 were adjusted to be similar to the mean loss rates observed in the cross-traffic simulations.

In the first experiment, both paths experience symmetric loss rates, varying from 0.3% to 14%. Fig. 16 plots the average goodput (file size ÷ transfer time) of CMT vs. CMT-PF with 5% error margin. As observed in the simulations (Fig. 12), both CMT and CMT-PF perform similarly at low loss rates, and this trend continues at higher loss rates.

In the second experiment, paths 1 and 2 experience asymmetric loss rates; loss rate on path 1 is fixed at 0.3% while path 2's loss rate varies from 0.3% to 14%. As observed in the simulations (Fig. 13), (i) both CMT and CMT-PF perform similar under low loss rates (Fig. 17), and (ii) as the paths become more asymmetric in their loss conditions, CMT-PF performs similarly or slightly better than CMT. However, the absolute goodput difference between CMT and CMT-PF in Fig. 17 is not as significant as observed in Fig. 13,

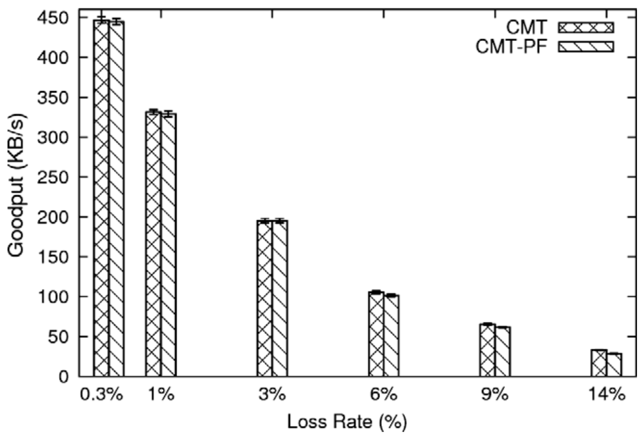

**Fig. 16.** CMT vs. CMT-PF during symmetric loss conditions.

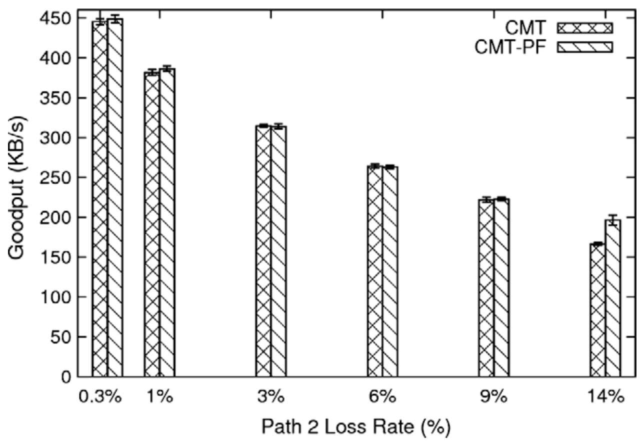

**Fig. 17.** CMT vs. CMT-PF during asymmetric loss conditions.

likely because the emulations and simulations employ different loss models. Nonetheless, our emulation experiments confirm that *CMT-PF performs on par or better but never worse than CMT during congestion.*

## 7. Summary

Using simulation, we demonstrated that retransmission policies using CMT with a "potentially-failed" destination state (CMT-PF) outperform CMT during permanent and short-term failures. During permanent failures, CMT-PF employs a better failure detection process even under aggressive failure detection thresholds and varying receive buffer constraints.

Investigations also revealed that the PF state transition does not penalize CMT-PF performance during congestion. CMT-PF performs as well as CMT when the paths experience symmetric congestion levels, and similar or better than CMT when the paths experience asymmetric congestion levels. These conclusions were confirmed using real implementations of CMT and CMT-PF in FreeBSD.

Since *our findings demonstrate CMT-PF performs better or similar but never worse than CMT*, we recommend CMT be replaced by CMT-PF in existing and future implementations and RFCs. Finally, we note that other multipath transports (e.g., multipath TCP [19]) may also suffer from rbuf blocking during path failures, and the CMT-PF framework can be adapted to such transports to alleviate the negative consequences of failure-induced rbuf blocking.

## Acknowledgments

The authors thank Joseph Szymanski for helping implement CMT-PF on FreeBSD. The authors also thank Janardhan Iyengar, Jon Leighton and Ertugrul Yilmaz for their helpful comments and suggestions.

## References

[1] CAIDA: Packet Sizes and Sequencing, March 1998. <www.caida.org>.
[2] A. Caro, End-to-End Fault Tolerance using Transport Layer Multihoming, Ph.D. Dissertation, Department of Computer and Info Sciences, University of Delaware, August 2005.
[3] M. Dahline, B. Chandra, L. Gao, A. Nayate, End-to-end WAN service availability, IEEE/ACM Transactions on Networking 11 (2) (2003) 300–313.
[4] N. Ekiz, P. Natarajan, J. Iyengar, A. Caro, ns-2 SCTP Module, Version 3.7, September 2007. pel.cis.udel.edu.
[5] N. Feamster, D. Andersen, H. Balakrishnan, M. Kaashoek, Measuring the Effects of Internet Path Faults on Reactive Routing, ACM SIGMETRICS, San Diego, 2003.
[6] J. Iyengar, P. Amer, R. Stewart, Receive Buffer Blocking in Concurrent Multipath Transfer, IEEE GLOBECOM, St. Louis, November 2005.
[7] J. Iyengar, P. Amer, R. Stewart, Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths, IEEE/ACM Transactions on Networking 14 (5) (2006) 951–964.
[8] C. Labovitz, A. Abuja, A. Bose, F. Jahanian, Delayed Internet Routing Convergence, ACM SIGCOMM, Stockholm, 2000.
[9] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, C. Diot, Characterization of Failures in an IP Backbone, IEEE INFOCOM, Hong Kong, China, 2004.
[10] P. Natarajan, J. Iyengar, P. Amer, R. Stewart, Concurrent Multipath Transfer using Transport Layer Multihoming: Performance under Network Failures, IEEE MILCOM, Washington, DC, USA, October 2006.
[11] P. Natarajan, N. Ekiz, P. Amer, J. Iyengar, R. Stewart, Concurrent Multipath Transfer Using Transport Layer Multihoming: Introducing the Potentially-Failed Destination State, IFIP-TC6 Networking, Singapore, 2008.
[12] P. Natarajan, Leveraging Innovative Transport Layer Services for Improved Application Performance, PhD Dissertation, Department of Computer & Info Sciences, University of Delaware, February 2009.
[13] ns-2 Documentation and Software, Version 2.33, March 2008. Available from: <www.isi.edu/nsnam/ns>.
[14] V. Paxson, End-to-end routing behavior in the internet, IEEE/ACM Transactions on Networking 5 (5) (1997) 601–615.
[15] M. Handley, J. Padhye, S. Floyd, TCP Congestion Window Validation, RFC 2861, June 2000.
[16] R. Stewart, Stream Control Transmission Protocol, RFC 4960, September 2007.

[17] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM Computer Communications Review 27 (1) (1997) 31–41. Jan.
[18] S. Shakkottai, R. Srikant, A. Broido, K. Claffy, The RTT Distribution of TCP Flows in the Internet and its Impact on TCP-based Flow Control, Tech. Report, Cooperative Association for Internet Data Analysis (CAIDA), February 2004.
[19] D. Wischik, M. Handley, M. Braun, The resource pooling principle, ACM Computer Communications Review 38 (5) (2008) 47–52.
[20] Y. Zhang, V. Paxson, S. Shenker, The Stationarity of Internet Path Properties: Routing, Loss, and Throughput, ACIRI Tech. Report, May 2000.