Rethinking End-to-End Failover with Transport Layer Multihoming

Armando L. Caro, Jr.*
Internetwork Research Department
BBN Technologies
acaro@bbn.com

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

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

***Abstract*— Using the application of bulk data transfer, we investigate end-to-end failover mechanisms and thresholds for transport protocols that support multihoming (e.g., SCTP). First, we evaluate temporary failovers, and measure the tradeoff between aggressive (i.e., lower) thresholds and spurious failovers. We surprisingly find that spurious failovers do not degrade performance, and often actually improve goodput regardless of the paths' characteristics (bandwidth, delay, and loss rate). A permanent failover mechanism tries to avoid throttling the sending rate by not returning to a primary path when it recovers. We demonstrate that such a mechanism can be beneficial if the sender can estimate each path's RTT and loss rate. We advocate a new approach to end-to-end failover that temporarily redirects traffic to an alternate path on the first sign of a potential failure (i.e., a timeout) on the primary path, but conservatively proceeds with failure detection of the primary path in the background.**

***Résumé*—En appliquant les transferts de données par rafale, nous étudions des mécanismes de bout en bout et les seuils de basculement dans les protocoles de transport (par exemple SCTP) qui supportent le multihoming. Nous évaluons tout d'abord des basculements temporaires et mesurons le compromis entre les seuils agressifs (les plus bas) et les faux basculements. Nous avons eu la surprise de dcouvrir que loin de dégrader les performances, les faux basculements améliorent souvent le débit utile et ce indépendamment des caractéristiques des chemins (bande passant, délai, et taux de perte). Un mécanisme de basculement permanent tente d'éviter de ralentir le taux d' émission en ne retournant pas à un chemin primaire lors d'un recouvrement d' erreur. Nous démontrons le bénéfice qu'apporte un tel mécanisme à un utilisateur capable d'estimer le RTT et le taux de perte de chaque chemin. Nous préconisons une nouvelle approche de basculement de bout en bout qui redirige temporairement le trafic vers un chemin alternatif dès le premier signe d'une panne potentielle (expiration de temporisateur) sur le chemin primaire, mais traite de manière conservative et en arrière plan la détection de panne du chemin principal.**

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 can be expected to be the rule rather than the exception in the near future as cheaper network interfaces and Internet access motivate content providers to have simultaneous connectivity through multiple ISPs, and more home users install wired and wireless connections for added flexibility and fault tolerance. Furthermore, wireless devices may be simultaneously connected through multiple access technologies, such as wireless LANs (e.g., 802.11) and cellular networks (e.g., GPRS, CDMA).

Our research focuses on transport layer techniques that exploit host multihoming at the transport layer to provide end-to-end fault tolerance. While fault tolerance can be addressed at other layers, the transport layer is in the best position to detect failure (i.e., loss of connectivity) and make failover decisions. After all, the transport layer is the lowest layer responsible for end-to-end quality of service, and has knowledge about path characteristics.

TCP does not support multihoming; it binds to only one network address at each end of a connection. When TCP was designed, network interfaces were expensive components, and hence multihoming was beyond the ken of research. However, newer transport protocols are emerging that support multihoming. The Stream Control Transmission Protocol (SCTP) [1], [2] and the Datagram Congestion Control Protocol (DCCP) [3] support multihoming at the transport layer. The motivation for multihoming in DCCP is mobility [4], while SCTP is driven by a broader and more generic application base – fault tolerance. We use SCTP in our experiments primarily because of its relative maturity and our focus on fault tolerance, but we believe the results and conclusions presented in this paper apply in general to reliable SACK-based transport protocols that support multihoming.

SCTP 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 gives an SCTP sender more than one destination address for transmitting data to a multihomed receiver. However, SCTP currently uses multihoming for fault tolerance purposes only, and not for concurrent multipath transfer [5]. Each endpoint chooses a single peer IP address as the primary destination address to transmit new data during normal transmission. If the primary destination address becomes unreachable, the SCTP sender detects the failure, and temporarily *fails over* to using an alternate destination address without requiring action by the

*This research results from the first author's PhD dissertation while with the Protocol Engineering Lab, CIS Department, University of Delaware.

Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

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

user or application layer.

SCTP has a tunable failover threshold that RFC2960 recommends should be set to a conservative value of six consecutive timeouts, which translates to a failure detection time of at least 63 seconds – unacceptable for many applications. In this paper, we evaluate non-failure lossy conditions to measure the tradeoff between more aggressive failover (i.e., lower thresholds) and spurious failovers. We focus on the process of detecting failure, which may be a correct detection or a spurious detection. Regardless of the failover threshold value used, the behavior after correctly detecting a loss of connectivity is the same. Thus, an aggresive failover threshold is clearly better in failure scenarios, but may cause spurious failovers in non-failure lossy conditions. Hence, our study shows how different failover values affect performance in the case of a spurious failure detection. We surprisingly find that spurious failovers do not degrade performance, and often actually improve goodput regardless of the paths' characteristics (bandwidth, delay, and loss rate).

Since failovers are temporary, traffic migrates back to the primary path when the primary path recovers. This migration throttles the sending rate, because upon returning to using the primary path, the sender must enter slow start with a cwnd of one MTU. To avoid this slowdown, the concept of permanent failovers dictates that a sender make the failover permanent if the primary path does not respond within some threshold amount of time. We find that permanent failovers can improve performance if a sender can accurately estimate each path's RTT and loss rate to make an informed decision.

We advocate a new approach to end-to-end failover that temporarily redirects traffic to an alternate path on the first sign of a potential failure (i.e., a timeout) on the primary path, but conservatively proceeds with failure detection of the primary path in the background.

Section II describes SCTP's current failover mechanism. Section III presents the tradeoffs between more aggressive failover and spurious failovers. Section IV introduces and evaluates a modified failover mechanism that allows failovers to become permanent. We conclude the paper in Section V.

II. SCTP's Failover Mechanism

Each endpoint uses both implicit and explicit probes to dynamically maintain knowledge about the reachability of its peer's IP addresses. Transmitted data serve as implicit probes to a destination (generally, the primary destination), while explicit probes, called *heartbeats*, periodically test reachability and measure the RTT of idle destinations. Each timeout (for data or a heartbeat) on a particular destination increments an error count for that destination. A destination's error count is cleared whenever data or a heartbeat sent to that destination is acked. A destination "fails" should its error count *exceed* the failover threshold, called Path.Max.Retrans (PMR).

Figure 1 specifies SCTP's current failover mechanism for n destinations. The association begins in Phase I, where destination D_i is the primary destination, D_i is in the active state, and all new data are sent to D_i. When D_i fails, "failover" occurs and the association moves into Phase II.

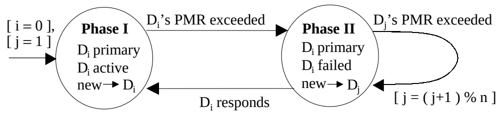

Fig. 1. FSM for current failover mechanism

In Phase II, D_i remains the primary destination, but in a failed state; all new data are redirected to an alternate destination, D_j. If more than one alternate destination address exists, RFC2960 leaves the alternate destination selection method unspecified. In this work, we assume a round-robin selection method. If D_j's error count should exceed PMR, a failover occurs to yet another alternate destination and the association stays in Phase II.

While in Phase II, the sender explicitly probes the primary destination, D_i, with periodic heartbeats. If D_i ever responds (i.e., recovers), failover is cancelled and the association returns to Phase I.

Failure detection time depends on three tunable parameters, which RFC2960 recommends to be set as: (1) minimum RTO $= 1$s, (2) maximum RTO $= 60$s, and (2) PMR $= 5$. Using these defaults, the first timeout towards failure detection takes 1s *in the best case*. Then, the exponential back-off procedure doubles the RTO on each subsequent timeout towards failure detection. With RFC2960's current recommended PMR $= 5$, six consecutive timeouts are needed to detect failure, taking at least $1 + 2 + 4 + 8 + 16 + 32 = 63$s. In the worst case, the first timeout takes the maximum of 60s, and failure detection requires $6 * 60 = 360$s!

III. Reducing PMR

Reducing PMR decreases failure detection time, but increases the possibility of *spurious failover*, where a sender mistakenly concludes a failure has occurred. In this section, we measure the tradeoff between lower PMR settings and spurious failovers. The goal is to determine how much failure detection time can be improved without having detrimental effects on goodput in non-failure scenarios.

III.1. Methodology

We evaluate different PMR settings using the University of Delaware's SCTP module [6] for the ns-2 network simulator [7]. Figure 2 illustrates the network topology. The multihomed sender, A, has two paths (labeled *Primary* and *Alternate*) to the multihomed receiver, B. The primary path's core link has a 10Mbps bandwidth and a 25ms one-way delay. The alternate path's core link has a 10Mbps bandwidth and one-way delays of 25ms, 85ms, and 500ms. Each router, R, uses drop-tail queuing and is attached to a dual-homed node (A or B) via an edge link with 100Mbps bandwidth and 10ms one-way delay.

The end-to-end RTTs are 90ms, 210ms, and 1040ms, which sample reasonable delays on the Internet today. Although

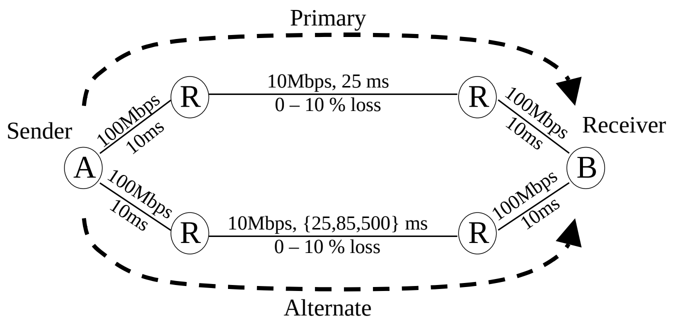

Fig. 2. Simulation network topology

1040ms may seem large, flows passing through cellular networks often experience RTTs as high as 1 or more seconds [8]–[10]. In any case, the delays are selected to demonstrate relative performance, and we believe our results and conclusions are independent of the actual bandwidth and delay configurations.

Note that we do not simulate different link bandwidths. Reducing the alternate path's bandwidth simply increases the RTT, which we already independently control.

We introduce uniform loss on these paths (0-10% each way) at the core links. We realize that using cross-traffic to cause congestion would more realistically simulate loss, but we found the simulation time for such a technique became impractical. On the other hand, uniform loss is a simple, yet sufficient model to provide insight about the effectiveness of different PMR settings accurately detecting failure. To evaluate if Figure 2's loss model was reasonable, we compared representative simulations using a cross-traffic model, shown in [11], to produce self-similar, bursty traffic. Although the absolute results differed for those examples compared, relative relationships remained consistent – leading to the same conclusions. We therefore proceed with the simpler uniform loss model, and refer the interested reader to [11] for an explanation of the cross-traffic model.

In our simulations, the sender uses a different retransmission policy than specified in RFC2960. The sender transmits (a) fast retransmissions to the same peer IP address as new data transmissions, and (b) timeout retransmissions to a non-failed alternate peer IP address (if one exists). This policy has been shown to perform better [11], and has been proposed to the IETF as a change to SCTP [12]. In our simulations, our Multiple Fast Retransmit algorithm [11] is also used to reduce the number of timeouts.

To observe long term averages, we simulate 80MB file transfers with PMR $= \{0, 1, 2, 3, 4, 5\}$. In this study, no link or interface failures are introduced; hence, all failovers that do occur are spurious. Each simulation has four parameters:

1) primary path's loss rate
2) alternate path's loss rate
3) alternate path's core link delay
4) PMR setting

III.2. Spurious Failovers

Figure 3 plots, for each PMR setting, the fraction of transfers that experience at least one spurious failover at primary path loss rates 0-10%. The graph aggregates all alternate path loss rates for each particular primary path loss rate.

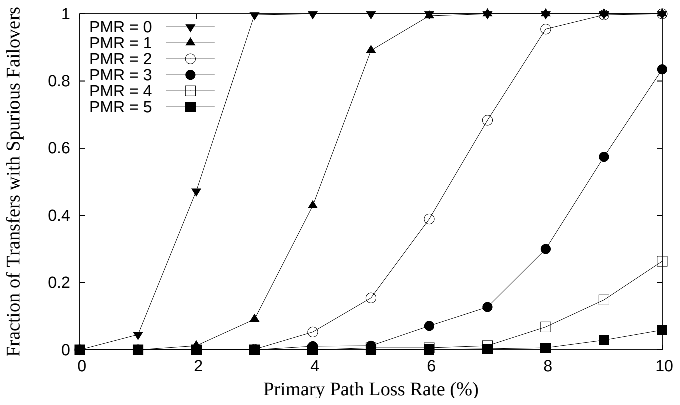

Fig. 3. Fraction of transfers with spurious failovers

Since PMR $= 0$ triggers a failover on a single timeout, this setting provides little robustness against spurious failovers at loss rates greater than 1%. At the other extreme, PMR $=$ 5 experiences nearly no spurious failovers at loss rates less than 8%. As the PMR increases from 0-5, their corresponding curves shift to the right by a loss rate of about 2%. This trend implies a simple linear relationship between the PMR setting and the robustness against spurious failovers. However, the slopes of the curves slowly flatten as the PMR increases, which argues that the robustness increases by more than a constant for each PMR setting.

The frequency of spurious failovers is also important when considering the robustness of various PMR settings. Figure 4 plots the cumulative distribution function (CDF) of the number of spurious failovers for primary path loss rates 2-10%. The CDFs for 1% primary path loss rate are omitted, because PMR $= \{1, 2, 3, 4, 5\}$ experience no spurious failovers, and PMR $= 0$ experiences spurious failovers in only 5% of the transfers. Again, each graph in Figure 4 aggregates all alternate path loss rates for each primary path loss rate.

At a 2% primary path loss rate, 53% of transfers with PMR $= 0$ are completely robust against spurious failovers, and 84% of transfers spuriously failover at most once. When the loss rate increases to 3%, less than 1% of transfers with PMR $= 0$ experience no spurious failovers. Then with 4% loss, only 1% of transfers experience less than ten spurious failovers.

As expected, PMR $= 1$ is more robust against spurious failovers than PMR $= 0$. At 3% loss, 91% of the transfers do not spuriously failover. Furthermore, at 4% loss, 57% of the transfers are free of spurious failovers, and no transfers experience more than four failovers. When the loss rate is 8%, less than 1% of transfers observe less than ten spurious failovers.

This trend continues for PMR $= \{2, 3, 4, 5\}$. More than 25% of the transfers observe spurious failovers at $\{6, 8, 10\}\%$ loss for PMR $= \{2, 3, 4\}$. With PMR $= 5$, only 3% and 6% of transfers have spurious failovers at 9% and 10% loss, respectively.

To conclude, determining which failover threshold is "robust enough" largely depends on the networking environment. For

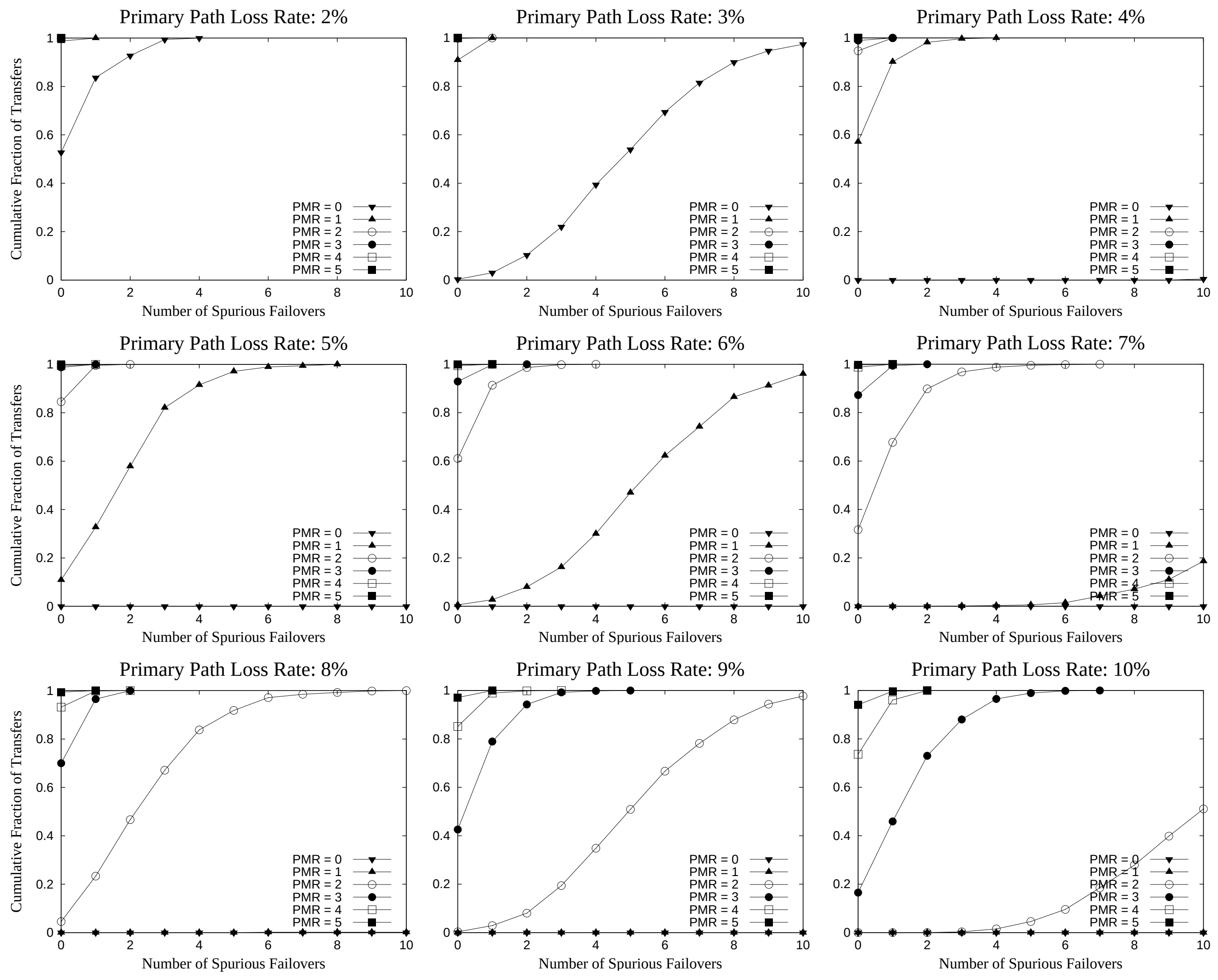

Fig. 4. CDF of the number of spurious failovers for primary path loss rates 2-10%

example, Zhang et al. [13] use end-to-end Internet measurements to report that 84% of their traces experienced less than a 1% loss rate (i.e., essentially “lossless”), and 15% of their traces had loss rates of 1-10% (with an average of 4%). Thus, to be completely robust against spurious failovers on 99% of paths, PMR should be set to 6 (even PMR $= 5$ spuriously fails over 6% of the time at 10% loss), but that translates to a failover time of 123 seconds! *Therefore, we would conclude that PMR $= 3$ is robust enough for the Internet. This setting translates to a 15 second failover time, and is robust for all “lossless” paths and the average “lossy” path.*

III.3. Symmetric Path Delays

While the frequency of spurious failovers is important in providing intuition about overall behavior, of greater importance is how these spurious failovers affect performance. We collected results for 0-10% loss on the primary and alternate paths, but due to space constraints in this paper, we do not include all results. The optimal transfer time (i.e., the primary path loss rate is 0%) of an 80MB file is 122.3 seconds. Figure 5 plots the average 80MB file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 90ms alternate path RTT. Each graph has a fixed primary path loss rate, and varies the alternate path loss rate on the x-axis from 0-10%. Note that the scale of the y-axis is different for each primary path loss rate to allow the reader to observe a performance difference between the different threshold settings at each primary path loss rate.

Counter to our intuition, we observe that the PMR setting has little effect on the goodput for primary path loss rates less than 8%. Above 8%, the results show that lower (!) PMR settings begin to improve performance, with PMR $= 0$ providing the most improvement. That is, surprisingly, being more aggressive with failover often provides improved performance, even when the alternate path loss rate is higher than that of the primary path. For example, reducing the PMR from 5 to 0 improves the performance by 4% when the primary and alternate path loss rates are 8% and 10%, respectively. These counter-intuitive results are explained later in Section III.7.

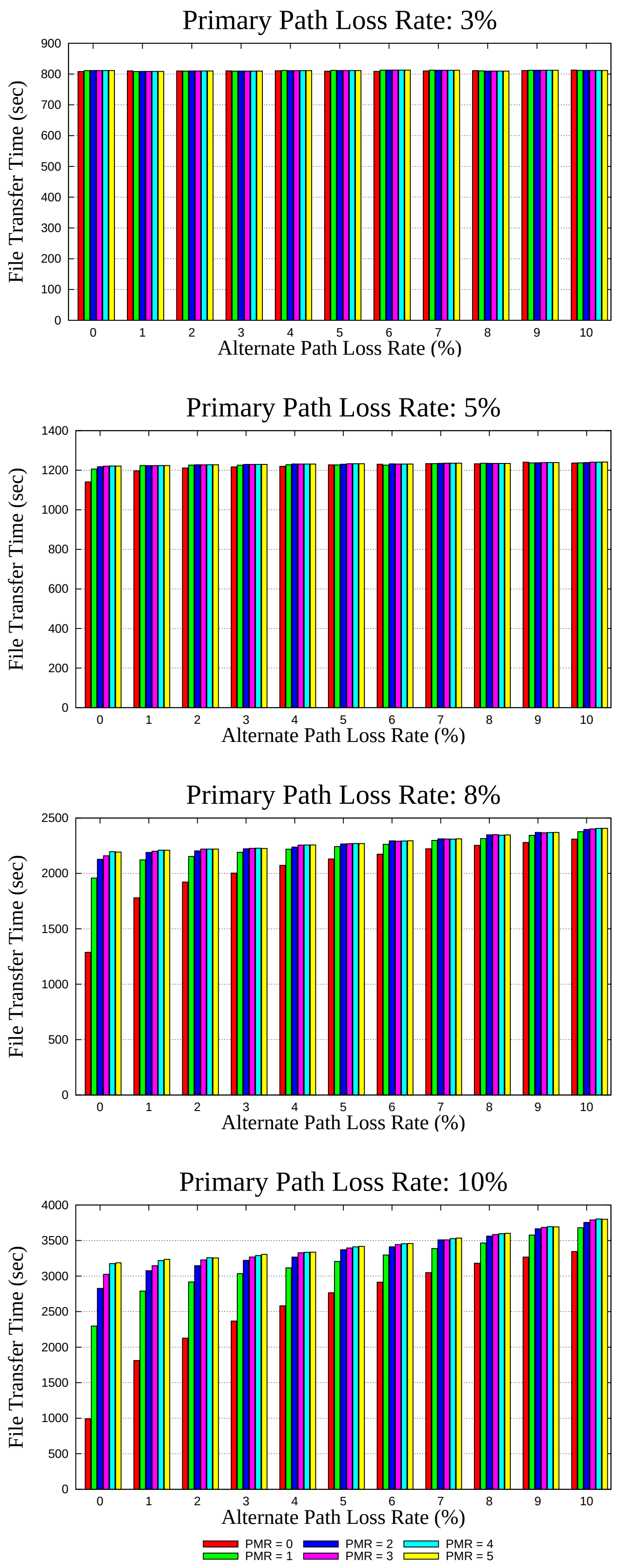

Fig. 5. 90ms primary path RTT and 90ms alternate path RTT

III.4. Asymmetric Path Delays

We are also surprised to find that being aggressive with failover does not change with asymmetric path delays. We expected larger alternate path RTTs to degrade performance of lower PMR settings. However, we find that the results remain nearly constant regardless of the alternate path delay. Figure 6 plots the results for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 1040ms alternate path RTT. Comparing these results with those in Figure 5 shows that the alternate path's longer RTT does not affect the performance. Even though the alternate path's RTT is more than ten times longer than the primary path's, PMR $= 0$ outperforms other PMR settings. Again, these unexpected results will be explained in Section III.7.

III.5. Three Paths

To determine if our conclusions hold when the number of paths between the endpoints increases, we add an additional alternate path to the topology in Figure 2. We configure both alternate paths to have the same properties (bandwidths, delays, and loss rates). Otherwise, the number of simulation parameters would be unmanageable. The results (not shown) are consistent with those for two paths. That is, the relationships between the different PMR settings remain the same. We expect that the trends will remain the same for configurations with more than three paths between endpoints.

III.6. Dormant State Behavior

As the finite state machine in Figure 1 shows, if a sender fails over to an alternate destination that in turn fails, the sender will failover to yet another alternate destination. If needed, the sender continues to failover to other alternate destinations until all alternate destinations are exhausted. When all destinations have failed, the association enters the *dormant state* [14], not represented in Figure 1.

RFC2960 does not specify dormant state behavior. Implementations are provided the freedom of choosing what action a sender takes when all destinations fail. The association leaves the dormant state when one of the destinations (primary or alternate) responds. Otherwise, the association is aborted when the association exceeds the Association.Max.Retrans threshold, which is an SCTP parameter to limit the number of consecutive timeouts across all destinations.

Dormant state behavior is generally considered unimportant, because high PMR settings make it unlikely to reach. However, if PMR is lowered to 0, as our results thus far argue should be done, entering the dormant state becomes more likely. Thus, we consider three different dormant state behaviors to evaluate how they impact behavior: (1) Dormant LastDest, (2) Dormant Primary, and (3) Dormant Hop.

The Dormant LastDest behavior dictates that when the dormant state is entered, the sender continues sending new data to whichever destination was last used in Phase II. The other destinations still are periodically probed in the background with heartbeats. If the primary destination replies, the dormant state is exited, and the association returns to Phase I. If an

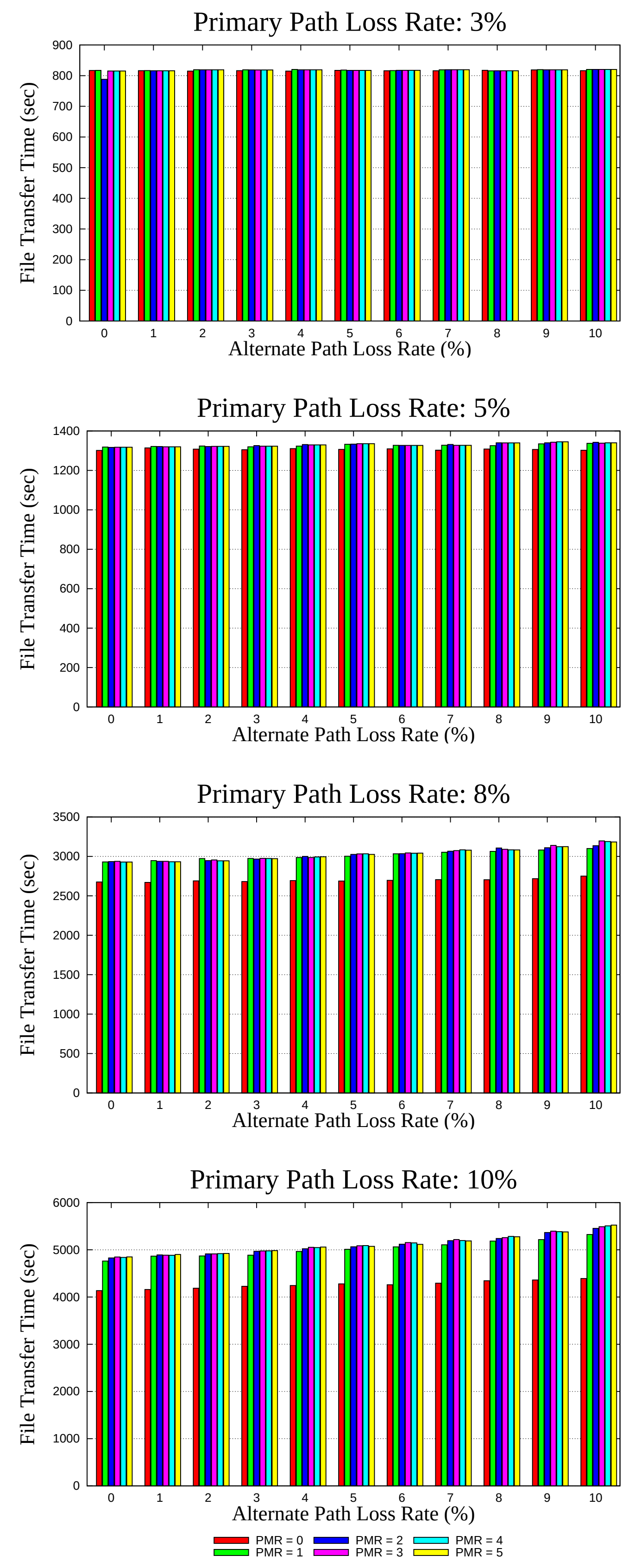

Fig. 6. 90ms primary path RTT and 1040ms alternate path RTT

alternate destination replies, the association returns to Phase II with the destination that replied as D_j.

The Dormant Primary behavior differs only slightly from the Dormant LastDest behavior. Instead of continually sending new data to whichever destination was last used in Phase II, the sender continually sends new data to the primary destination.

The Dormant Hop behavior, shown in Figure 7, attempts to be more aggressive in finding an active destination. While in the dormant state, the sender transmits new data to a different destination after each timeout. The sender cycles through all the destinations in a round-robin fashion until either a destination responds, or the association aborts.

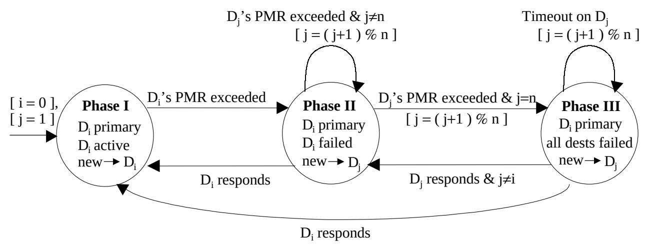

Fig. 7. FSM with Dormant Hop behavior

The results in Sections III.2 through III.5 use the Dormant Hop behavior, but we also evaluate the performance of the other two dormant state behaviors. We find that dormant state behavior does not affect goodput, and the trend reported in those sections remains consistent for all dormant state behaviors (results not shown).

III.7. Explaining the Results

Our results document that aggressive failover settings (in particular, PMR $= 0$) improve performance regardless of the path loss rates, path delays, and/or dormant state behavior – a result counter to our intuition. We spent considerable time investigating this surprising conclusion, which we now explain.

The underlying advantage of aggressive failover is that an association spends less time blocked during failure detection. With PMR $= 0$ for example, a single timeout moves new data transmission to the alternate path while the primary destination is probed with heartbeats. The primary destination may respond on the first probe, or it may not respond for a long time. In either case, data transmission continues on the alternate path, and migrates back to the primary path if and when the primary destination responds. Less aggressive failover settings (e.g., PMR $= 5$) cause a sender to wait longer before sending new data to the primary destination; in the meantime, essentially no useful communication takes place. Therefore, even if the alternate path has a higher loss rate and/or longer RTT, the sender always has the potential to gain (without risking doing worse) by failing over sooner.

The remainder of this section presents four detailed timeout scenarios (shown in Figure 8) for PMR $= \{0, 1\}$ to demonstrate the merits of more aggressive failover. They all begin with TSN 1 (i.e., packet 1) being lost in transit to the primary destination and subsequently timing out. For PMR

$= 0$, the sender immediately fails over, retransmits TSN 1 to the alternate destination, and sends a heartbeat to the primary destination. For PMR $= 1$, the sender retransmits TSN 1 to the alternate destination as the retransmission policy dictates (see Section III.1), and sends TSN 2 to the primary destination. We compare the behavior of these two PMR settings by following the details of four (of many) possible scenarios beyond this point.

III.7.1. Scenario 1: The first packet sent to the primary destination and the first packet sent to the alternate destination following TSN 1's timeout are both delivered successfully.

- **PMR = 0** The failover is cancelled when the heartbeat is acked. Although the figure shows both TSN 1 and the heartbeat are acked at the same time, it is a race condition. If the heartbeat gets acked first (as shown in Figure 8's Scenario 1), then TSN 2 is sent on the primary and normal data transfer continues from this point. If TSN 1 gets acked first (not shown), then TSNs 2-3 are sent to the alternate destination, TSN 4 is sent to the primary destination when the heartbeat is acked, and normal data transfer continues to the primary destination.
- **PMR = 1** As both TSN 1 and 2 are sent at about the same time, again a race condition occurs. If TSN 1 arrives at the receiver first, the receiver's delayed ack algorithm causes a single cumulative ack (denoted SACK 2) to be generated for both TSN 1 and 2 (as shown in Figure 8). When this ack arrives, TSNs 3-4 are sent to the primary destination and normal data transfer continues to the primary destination. If TSN 2 arrives at the receiver first, the receiver generates two acks (not shown). The first selectively acks TSN 2 with a missing report for TSN 1, and the second cumulatively acks TSN 2. Upon receiving the first, the sender sends TSN 3 to the primary destination and normal data transfer continues to the primary destination.

This scenario presents a case where both PMR settings perform *roughly* similar in our experiments. Let RTT_1 and RTT_2 be the primary path's RTT and the alternate path's RTT, respectively. If $RTT_1 \leq RTT_2$ (as is the case in our experiments), then PMR $= 1$ has a marginal advantage in that it sends one more packet than PMR $= 0$.

On the other hand, if $RTT_1 > RTT_2$ (not shown in Figure 8 and not the case in our experiments), then PMR $= 0$ gets ahead of PMR $= 1$ in the overall transfer. The amount by which PMR $= 0$ gets ahead depends on the ratio of the two paths' RTTs. However, since $RTT_1 \leq RTT_2$ in our experiments, we omit detailed analysis of PMR $= 0$'s performance gain when $RTT_1 > RTT_2$.

III.7.2. Scenario 2: The first packet sent to the primary destination following TSN 1's timeout is successfully delivered, and the first packet sent to the alternate destination is lost.

- **PMR = 0** The failover is cancelled when the heartbeat is acked. TSN 2 is sent to the primary destination. When TSN 2 is selectively acked, TSN 3 is then sent to the primary destination. The sender continues sending one packet at a time to the primary destination until TSN 1's retransmission times out. TSN 1 is then re-retransmitted to the primary destination and normal data transfer continues to the primary destination.
- **PMR = 1** When TSN 2 is selectively acked, TSN 3 is sent to the primary destination, and when it is selectively acked, TSN 4 is sent to the primary destination. The sender continues sending one packet at a time to the primary destination until TSN 1's retransmission times out. TSN 1 is then re-retransmitted to the primary destination and normal data transfer continues to the primary destination.

Again, both PMR settings perform *roughly* similar. PMR $= 1$ has only a marginal advantage in that it sends one more packet than PMR $= 0$. This scenario shows that loss on the alternate path alone has little effect on the performance gap between PMR settings.

III.7.3. Scenario 3: The first packet sent to the primary destination following TSN 1's timeout is lost, and the first packet sent to the alternate destination is delivered successfully.

- **PMR = 0** When TSN 1 is acked, TSNs 2-3 are sent to the alternate destination, and normal data transfer continues temporarily to the alternate destination. Eventually, the heartbeat times out, and another heartbeat is then sent to the primary destination. Since this timeout is the second consecutive timeout on the primary destination, it will take at least 2 seconds to expire (assuming RTO.Min is 1 second). Once the second heartbeat is successfully acked, the sender cancels the failover, and resumes normal data transmission to the primary destination.
- **PMR = 1** When TSN 1 is acked, the sender is temporarily blocked and does not send any new data. When TSN 2 times out (again, at least 2 seconds later), the sender fails over to the alternate destination, retransmits TSN 2 to the alternate destination, and sends a heartbeat to the primary destination. From this point, normal data transfer continues to the alternate destination until the heartbeat is acked and the failover is cancelled. Then the sender resumes normal data transfer to the primary destination.

In Scenario 3, PMR $= 0$ may potentially perform significantly better than PMR $= 1$. With PMR $= 0$, the sender transmits new data on the alternate path until the sender receives a heartbeat ack from the primary destination. We estimate the number of packets, d, sent to the alternate destination during this period as follows. From the time TSN 1 is retransmitted, the time it takes to receive a heartbeat ack from the primary destination is $(RTO_1^2 + RTT_1)$, where RTO_1^2 is the primary path's RTO for the second consecutive timeout, and RTT_1 is the primary path's RTT. The number of alternate path round trips, r, that will take place during this period is

$$r = min\left[1, \frac{RTO_1^2 + RTT_1}{RTT_2}\right] \tag{1}$$

where RTT_2 is the alternate path's RTT. Note that since at least one packet (TSN 1) is successfully sent on the alternate path, r must be at least 1.

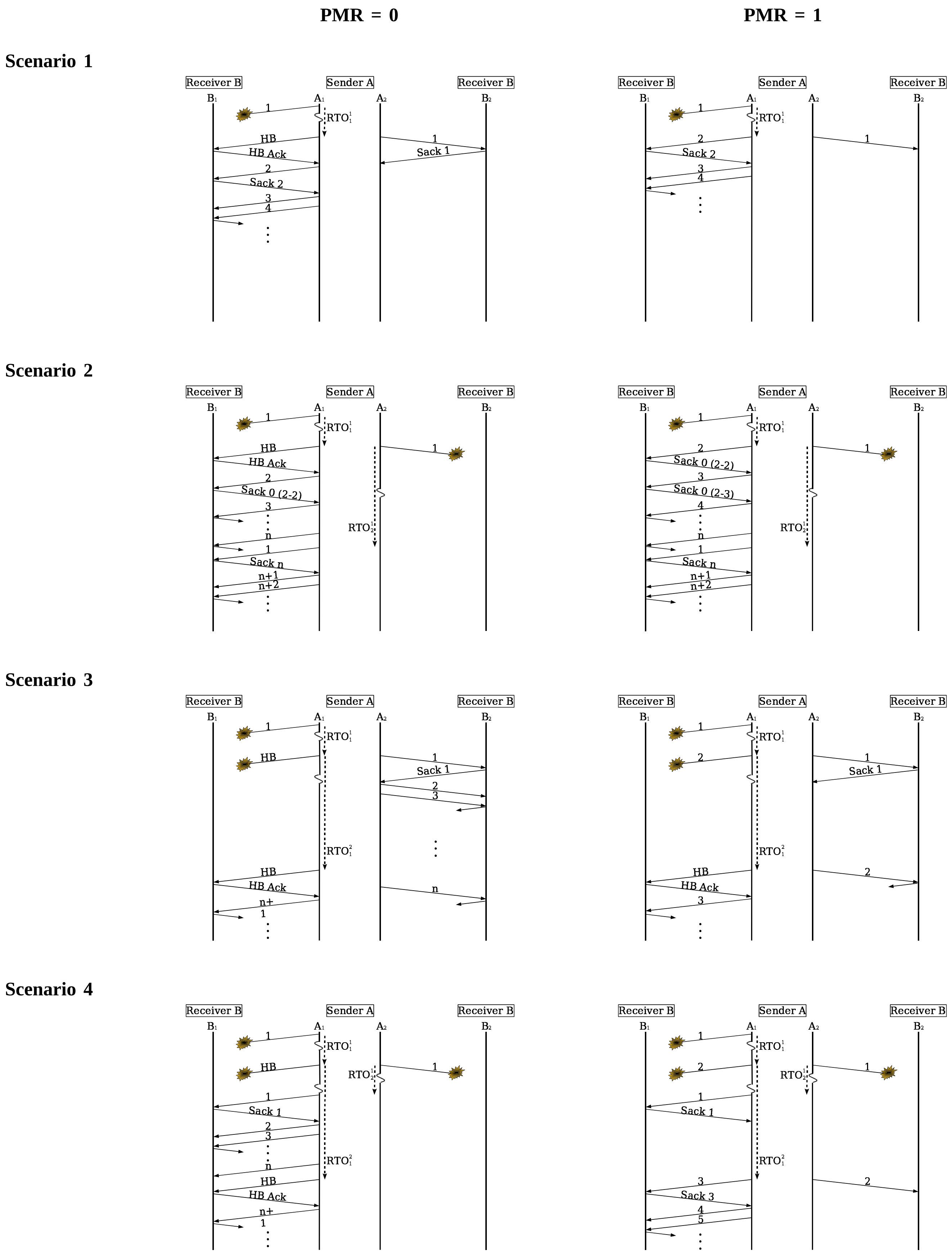

Fig. 8. Timeout scenarios

To estimate the number of packets, d, sent to the alternate destination during r alternate path round trips, we first assume that no loss occurs on the alternate path during this period. Hence, the transfer on the alternate path exits slow start when cwnd exceeds ssthresh. Using the slow start cwnd growth model from [15], the last alternate path cwnd before exiting slow start is

$$cwnd = ssthresh = init\_cwnd \cdot \left(1 + \frac{1}{b}\right)^{r_{ss}-1} \quad (2)$$

where $init\_cwnd$ is the initial cwnd, b is the number packets per ack the receiver's delayed ack algorithm uses, and r_{ss} is the number of alternate path round trips spent in slow start. Since $init\_cwnd = 1$, $b = 2$, and $r_{ss} \leq r$, we can solve for r_{ss} to arrive at

$$r_{ss} = max\left[r, 1 + log_{\frac{3}{2}}\left(ssthresh\right)\right] \quad (3)$$

Using a component of the slow start data transfer model from [15], the number of packets sent during the first r_{ss} round trips on the alternate path is

$$\begin{aligned} d_{ss} &= 1 \cdot \frac{\left(\frac{3}{2}\right)^{r_{ss}} - 1}{\frac{3}{2} - 1} \\ &= 2\left[\left(\frac{3}{2}\right)^{r_{ss}} - 1\right] \end{aligned} \quad (4)$$

The remaining round trips, r_{ca}, are the number of round trips the transfer on the alternate path spends in congestion avoidance:

$$r_{ca} = r - r_{ss} \quad (5)$$

During congestion avoidance, cwnd grows by 1 MTU each round trip. Thus, we use $cwnd_i$ to denote the sender's cwnd during the i-th round trip in congestion avoidance:

$$cwnd_{i+1} = cwnd_i + 1 \quad (6)$$

Then since a sender begins in congestion avoidance with $cwnd = ssthresh + 1$, we have:

$$cwnd_{i+1} = ssthresh + i \quad (7)$$

Thus, the number of data packets sent during congestion avoidance is

$$\begin{aligned} d_{ca} &= \sum_{i=1}^{r_{ca}} (ssthresh + i) \\ &= (r_{ca} \cdot ssthresh) + \sum_{i=1}^{r_{ca}} i \\ &= (r_{ca} \cdot ssthresh) + \frac{r_{ca} + (r_{ca})^2}{2} \end{aligned} \quad (8)$$

Combining (4) and (8), we estimate the number of successful data packets that PMR $= 0$ sends to the alternate destination in r alternate path round trips as

$$d = d_{ss} + d_{ca} \quad (9)$$

Since PMR $= 1$ only sends only one packet to the alternate destination in r alternate path round trips, the difference in the number of packets that PMR $= 0$ and PMR $= 1$ send in this scenario is $(d - 1)$. Therefore, the relative performance difference between PMR $= 0$ and PMR $= 1$ in this scenario depends on r. When $r = 1$, it follows that $d = 1$, and thus PMR $= 0$ performs no better than PMR $= 1$. However, when $r > 1$, PMR $= 0$ outperforms PMR $= 1$ since $d > 1$.

This analysis assumes that the alternate path does not experience loss, but we now relax this constraint by considering alternate path losses after TSN 1 (the case where TSN 1 is lost is presented next in Scenario 4). Without getting into the details of such scenarios (there are an infinite number), it suffices to say that our estimate of d in (9) is an overestimate when loss is introduced. However, the fact that $d \geq 1$ remains true. Therefore, it remains that, in this scenario, PMR $= 0$ performs no worse than PMR $= 1$, and may outperform PMR $= 1$ by as much as $(d - 1)$ packets, depending on r and the loss conditions on the alternate path.

III.7.4. Scenario 4: The first packet sent to the primary destination and the first packet sent to the alternate destination following TSN 1's timeout are both lost.

- **PMR = 0** TSN 1's retransmission times out first, and TSN 1 is re-retransmitted to the primary destination. When TSN 1 is acked, the failover is cancelled and normal data transfer continues to the primary destination from this point. Note that the heartbeat times out later, but does not affect the data transfer.
- **PMR = 1** TSN 1's retransmission times out first, and TSN 1 is re-retransmitted to the primary destination. When TSN 1 is acked, the failover is cancelled, but the sender cannot send any new data until TSN 2 times out. Once TSN 2 times out, the sender retransmits it to the alternate destination, and sends TSN 3 to the primary destination. From this point, normal data transfer continues to the primary destination.

Similar to Scenario 3, this scenario shows that PMR $= 0$ outperforms PMR $= 1$ when the primary path experiences consecutive timeouts. Again, the improvement is based on r, but in this scenario, r is the number of primary path round trips defined as

$$r = min\left[1, \frac{RTO_1^2 - RTO_2^1}{RTT_1}\right] \quad (10)$$

where RTO_1^2 is the primary path's RTO for the second consecutive timeout, RTO_2^1 is the alternate path's RTO, and RTT_1 is the primary path's RTT. Using this value of r in (3) and (5), we can use (9) to estimate the number of successful data packets, d, that PMR $= 0$ sends to the primary destination by the time RTO_2^1 expires. Therefore, this scenario also shows PMR $= 0$ performing no worse than PMR $= 1$, and possibly outperforming PMR $= 1$ by as much as $(d - 1)$ packets.

The chances of encountering each of these four scenarios depends on the loss conditions of the two paths. Regardless of which scenario is encountered when a timeout occurs on the primary path, lower PMR settings (PMR $= 0$ in particular) provide a transfer with more to gain (potentially several more packets successfully transferred) and less to lose (at most, one less packet successfully transferred). Therefore, lower PMR settings do not degrade performance and may actually improve performance.

IV. Permanent Failovers

When failovers are temporary, traffic migrates back to the primary path when it recovers. This migration throttles the sending rate, because the sender returns to slow start's cwnd of one MTU. To avoid this slowdown, we introduce a major potential change to SCTP – the concept of "permanent failover" using a *Change Primary Threshold (CPT)*. Permanent failover is based on a two-level threshold failover mechanism proposed in [16]. Once failover occurs, the sender can make the failover permanent (i.e., change the primary destination) if more than CPT heartbeat probes sent to the primary destination time out.

The specification for permanent failovers, shown in Figure 9, adds two new transitions to the finite state machine in Figure 7. While the association is in Phase II or III, if the primary destination's CPT threshold is exceeded, the primary destination is changed to the alternate destination currently in use. In Phase II, the association returns to Phase I with the new primary destination. In Phase III, however, the association remains in Phase III when a new primary destination is set; that is, changing the primary destination does not change the status of any destination, and thus the association remains in the dormant state.

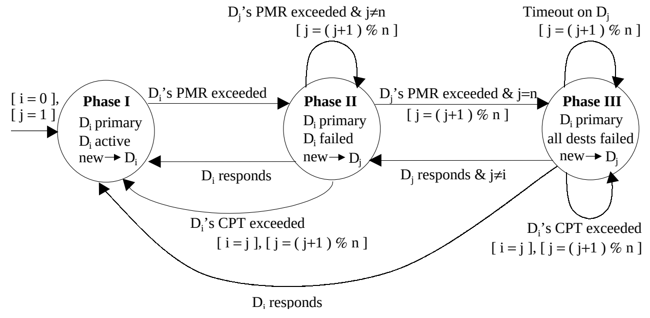

Fig. 9. FSM for permanent failovers

We evaluate different CPT settings using the same methodology explained in Section III, except here we only focus on PMR $= 0$ and the Dormant Hop behavior.

IV.1. Symmetric Path Delays

Figure 10 plots the average 80MB file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 90ms alternate path RTT. When the alternate path loss rate is lower than the primary path loss rate, more aggressive permanent failover (i.e., lower CPT settings) dramatically improve performance. On the flip side, the performance is degraded relatively little when the alternate path loss rate is higher than that of the primary path. For example, when the primary path loss rate is 5%, reducing CPT from 5 to 0 improves performance by as much as 88% and degrades performance by at most 9%.

Since paths with lower loss rates are less likely to exceed CPT, associations with lower CPT settings tend to spend less

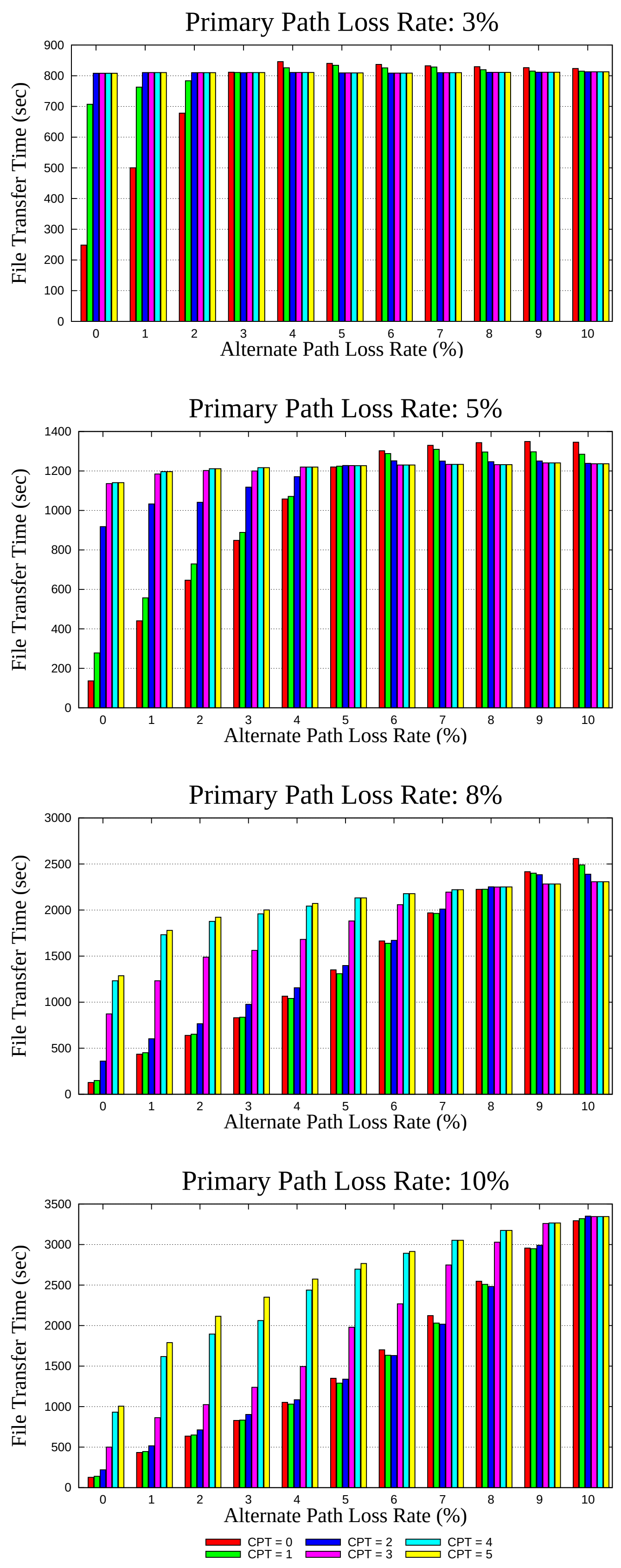

Fig. 10. PMR $= 0$, 90ms primary path RTT, and 90ms alternate path RTT

time on the higher loss rate path. The intuition is as follows. If a sender permanently fails over to a path with a higher loss rate, the performance may degrade, but only temporarily. Eventually, CPT will be exceeded again and the sender will switch back to the lower loss rate path. *Therefore, when the path delays are symmetric, the most aggressive permanent failover (i.e., CPT $= 0$) provides the best performance.*

IV.2. Asymmetric Path Delays

Figure 10 shows that lowering CPT improves performance when path delays are symmetric, but what happens when path delays are asymmetric? Figure 11 plots the average 80MB file transfer for $\{3, 5, 8, 10\}$ primary path loss, a 90ms primary path RTT, and a 210ms alternate path RTT. These results show that lower CPT settings *may* improve performance, but only when the alternate path's loss rate is *much* lower than the primary path loss rate. Otherwise, aggressive permanent failover degrades performance significantly. For example, when the primary path loss rate is 5%, reducing CPT from 5 to 0 improves performance 76% and 23% for 0% and 1% alternate path loss rates, respectively. On the other hand, the performance suffers (by as much as 54%) for all other alternate path loss rates. Thus, to benefit from a change primary, the difference in path delays requires an alternate path loss rate low enough to offset the alternate path's relatively large delay.

Note that worst performance for aggressive permanent failover occurs when both paths' loss rates are similar. As expected, aggressive permanent failover's performance improves as the alternate path's loss rate decreases relative to the primary's. Surprisingly, however, aggressive permanent failover's performance also improves as the alternate path's loss rate increases relative to the primary's. As explained in Section IV.1, lower CPT settings allows an association to reduce the time spent on the higher loss rate path. Therefore, as the alternate path's loss rate increases, the association will spend less time on the alternate path, thereby reducing the negative effects of its longer RTT.

Recognize that the results in Figure 11 present only one perspective with respect to asymmetric path delays. We show in Figure 12 that if the association begins with the longer delay path as the primary, lower CPT settings are advantageous regardless of the paths' loss rates. When starting on the longer delay path, the sender has only to gain with more aggressive permanent failovers. If the alternate path's loss rate is lower, the association will spend more time on the shorter delay path. Otherwise, the association will spend more time on the longer delay path, which it would have anyway with higher CPT settings.

These results seem to demonstrate that failovers should be permanent only when the alternate path has a shorter RTT, but both RTT and loss conditions need to be considered in the decision process. A path with shorter RTT and higher loss rate may provide lower throughput than a path with longer RTT and lower loss rate. With an estimated RTT and loss rate (p) for each path, a sender can apply Padhye's simplified throughput model, $\frac{1}{RTT}\sqrt{\frac{3}{4p}}$, from [17] to compare paths and determine if a permanent failover would be advantageous. Future work

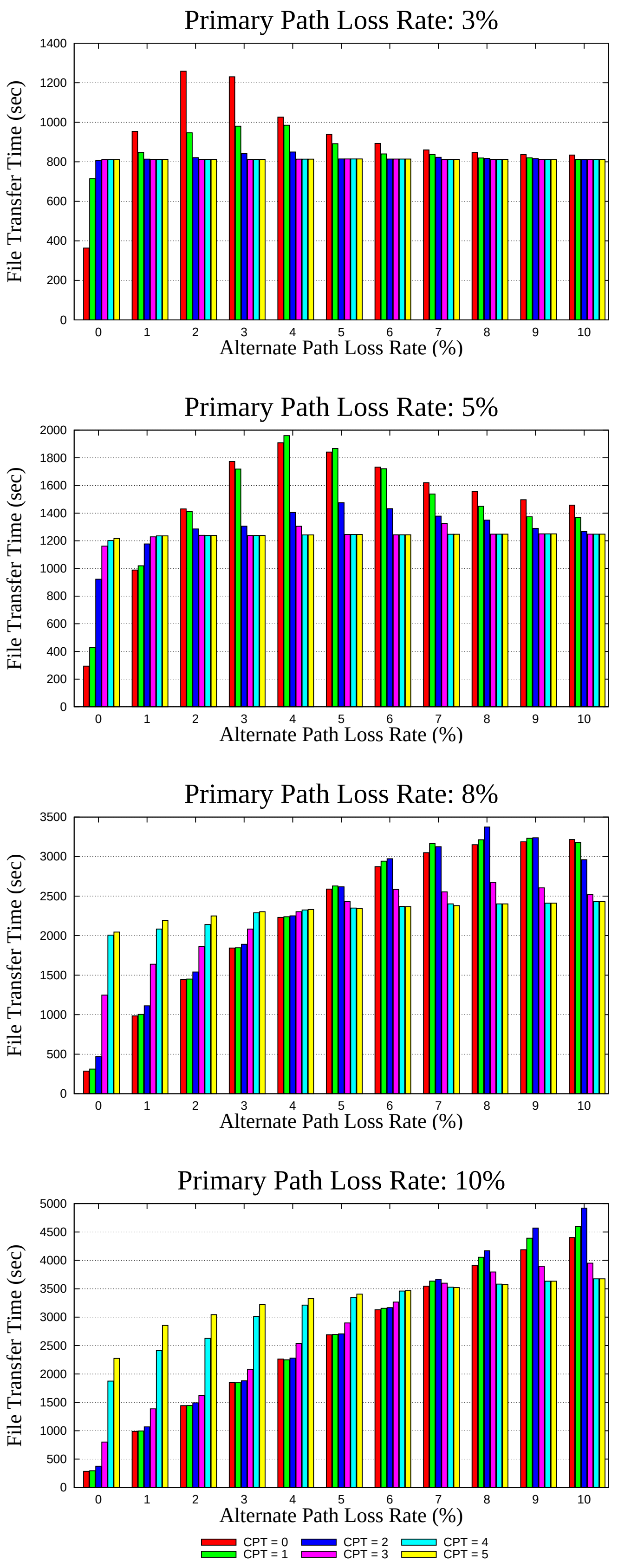

Fig. 11. PMR $= 0$, 90ms primary path RTT, and 210ms alternate path RTT

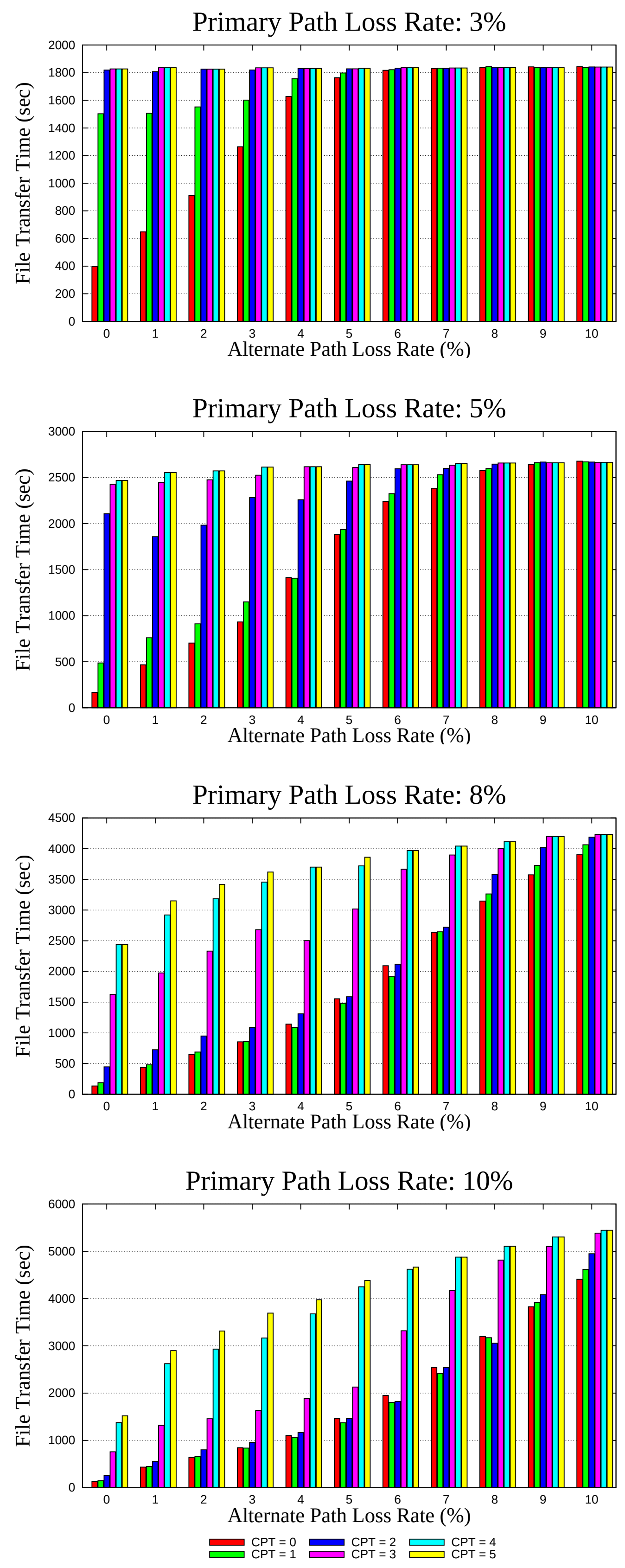

Fig. 12. PMR = 0, 210ms primary path RTT, and 90ms alternate path RTT

is to develop a mechanism to measure the loss rate of an idle alternate path without introducing unnecessary overhead.

V. Conclusion

We investigated the affects of reducing SCTP's failure detection threshold, Path.Max.Retrans (PMR), to less than the currently specified six consecutive timeouts. As expected, the number of spurious failovers increased as PMR was lowered, but we found that spurious failovers do not degrade performance. In fact, we found that aggressive failover settings have little effect on long term goodput averages for primary path loss rates less than 8%. At higher primary path loss rates, lower PMR settings improve goodput (even when the loss rate and/or delay is higher on the alternate path). Furthermore, since lower PMR settings provide less blocking during timeout events, short transfers may benefit even at low primary path loss rates.

We also explored the concept of permanent failovers to further improve performance by avoiding a slowdown of the sending rate after a failed path recovers. We found that that permanent failovers can improve performance if a sender has an estimate of each path's RTT and loss rate to make an informed decision.

Our results lead us to advocate a new approach to end-to-end failover. We have shown that aggressively failing over can significantly improve performance even if failure has not occurred. However, accurate failure detection is important for performance. When aggressively migrating traffic, a sender would benefit from avoiding destinations which have actually failed.

Figure 13 specifies our proposed failover mechanism for n destinations. The association begins in Phase I, with D_i as the primary destination, D_i in the active state, and all new data sent to D_i. On a single timeout on D_i, the association transitions to Phase II, where D_i remains the primary destination, D_i is probed with heartbeats, and new data are sent to an alternate destination (D_j). If D_i's probes cause PMR to be exceeded, the association transitions to Phase III, where D_i is marked failed. While in Phase II or III, each timeout redirects new data to a different destination (skipping failed destinations). Any time D_i responds, the association returns to Phase I.

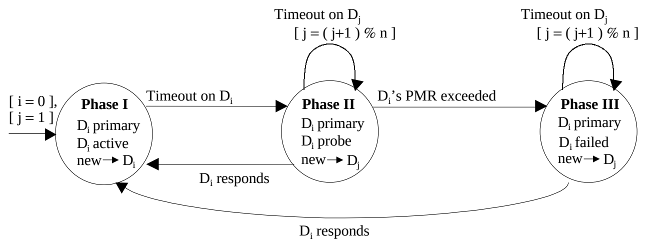

Fig. 13. FSM for proposed failover mechanism

This proposed mechanism provides improved performance without sacrificing failure detection accuracy. Its aggressive traffic migration design, however, may draw concern. First, traditional thinking is that frequent traffic redirection is counter-productive, but that intuition comes from research in congestion-based routing algorithms. Migrating traffic back-and-forth on an end-to-end basis does not suffer the side-effects (e.g., reordering, inaccurate RTT estimates, etc.) that are introduced, for example, when an intermediate router "flip-flops" traffic between routes. These side-effects are avoided because each time a flow moves to a new path, it begins from slow start as if it were a new flow. Furthermore, SCTP maintains path information (e.g., RTT, cwnd, ssthresh, etc.) per destination.

Second, "global failover synchronization" becomes possible with an aggressive traffic migration design. A cycle is formed when a bottleneck router drops a burst of packets, causing multiple flows to timeout and move their traffic to an alternate path. These flows then simultaneously probe their primary destination, and if successful, simultaneously migrate back to their primary path and increase their cwnds up to the point where a burst of drops occurs again.

However, we argue that global failover synchronization is no worse than the existing well-known phenomenon of global TCP congestion control synchronization [18]. In both cases, synchronized timeouts cause synchronized slow starts and cwnd evolution, but in the case of failover, the cwnd evolution may occur on alternate paths that do not share bottlenecks. If so, a single flow's traffic migration appears no different than a new end-to-end flow, because each time a flow migrates to a new path, the flow begins from slow start with a cwnd of one MTU. In fact, since new flows may begin with a cwnd as large as four MTUs [19], a single flow's traffic migration is more conservative than a new flow.

On the other hand, if multiple flows do migrate to alternate paths that share a bottleneck, these flows will not disturb the network any more than a synchronized TCP timeout would. In both cases, multiple flows begin from slow start with cwnd = one MTU, and simultaneously grow their cwnd. The only difference being that in the case of failover, the cwnd evolution happens to be on a different path than where the synchronized timeout occurred. In any case, AQM techniques eliminate global synchronization [18], which also includes global failover synchronization.

One limitation of our study is that the evaluation is indifferent to the causes of packet loss. However, packet losses may occur due to a variety of reasons, some of which include network congestion, transmission errors, and transient loss of connectivity. Each cause has inherent characteristics that may skew the results. In future work, it would be interesting to isolate and investigate how these particular scenarios influence the results.

Acknowledgements

The authors acknowledge Ryan Bickhart, Mark Hufe, Janardhan Iyengar, Sourabh Ladha, and the anonymous reviewers for their valuable comments and suggestions.

Disclaimer

References

[1] Stewart, (R.), Xie, (Q.), Morneault, (K.), Sharp, (C.), Schwarzbauer, (H.), Taylor, (T.), Rytina, (I.), Kalla, (M.), Zhang, (L.), and Paxson, (V.), "Stream Control Transmission Protocol," RFC2960, Oct. 2000.

[2] Caro, (A.), Iyengar, (J.), Amer, (P.), Ladha, (S.), Heinz, (G.), and Shah, (K.), "SCTP: A Proposed Standard for Robust Internet Data Transport," *IEEE Computer*, vol. 36, no. 11, pp. 56–63, Nov. 2003.

[3] Kohler, (E.), Handley, (M.), and Floyd, (S.), "Datagram Congestion Control Protocol (DCCP)," draft-ietf-dccp-spec-11.txt, Mar. 2005, (work in progress).

[4] Kohler, (E.), "Datagram Congestion Control Protocol Mobility and Multihoming," draft-kohler-dccp-mobility-00.txt, July 2004, (work in progress).

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

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

[7] Berkeley, (UC), LBL, USC/ISI, and Parc, (Xerox), "ns-2 documentation and software," Version 2.26, 2003, www.isi.edu/nsnam/ns.

[8] Gurtov, (A.), Passoja, (M.), Aalto, (O.), and Raitola, (M.), "Multi-Layer Protocol Tracing in a GPRS Network," in *International Conference on Ubiquitous Computing*, Sept. 2002.

[9] Inamura, (H.), Montenegro, (G.), Ludwig, (R.), Gurtov, (A.), and Khafizov, (F.), "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks," RFC3481, Feb. 2003.

[10] Jayaram, (R.) and Rhee, (I.), "A Case for Delay-based Congestion Control for CDMA 2.5G Networks," in *International Conference on Ubiquitous Computing*, Oct. 2003.

[11] Caro, (A.), Amer, (P.), and Stewart, (R.), "Retransmission Policies for Multihomed Transport Protocols," *Computer Communications*, (to appear).

[12] Stewart, (R.), Arias-Rodriguez, (I.), Poon, (K.), Caro, (A.), and Tuexen, (M.), "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues," draft-ietf-tsvwg-sctpimpguide-15.txt, Feb. 2005, (work in progress).

[13] Zhang, (Y.), Paxson, (V.), and Shenker, (S.), " On the Constancy of Internet Path Properties," in *CM SIGCOMM Internet Measurement Workshop (IMW 2001)*, San Francisco, CA, Nov. 2001.

[14] Stewart, (R.) and Xie, (Q.), *Stream Control Transmission Protocol (SCTP): A Reference Guide*, Addison Wesley, New York, NY, 2001.

[15] Cardwell, (N.), Savage, (S.), and Anderson, (T.), "Modeling TCP Latency," in *IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000.

[16] Caro, (A.), Iyengar, (J.), Amer, (P.), Heinz, (G.), and Stewart, (R.), "Using SCTP Multihoming for Fault Tolerance and Load Balancing," *ACM Computer Communication Review*, vol. 32, no. 3, pp. 23, July 2002, ACM SIGCOMM 2002 Poster.

[17] Padhye, (J.), Firoiu, (V.), Towsley, (D.), and Kurose, (J.), "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM 1998*, Vancouver, CA, 1998, pp. 303–314.

[18] Braden, (B.), Clark, (D.), Crowcroft, (J.), Davie, (B.), Deering, (S.), Estrin, (D.), Floyd, (S.), Jacobson, (V.), Minshall, (G.), Partridge, (C.), Peterson, (L.), Ramakrishnan, (K.), Shenker, (S.), Wroclawski, (J.), and Zhang, (L.), "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC2309, IETF, Apr. 1998.

[19] Allman, (M.), Floyd, (S.), and Partridge, (C.), "Increasing TCP's Initial Window," RFC3390, IETF, Oct. 2002.

[20] Caro, (A.), "End-to-End Fault Tolerance Using Transport Layer Multihoming," Phd dissertation, CIS Dept, University of Delaware, Aug. 2005.