Retransmission Policies for Multihomed Transport Protocols

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*— We evaluate three retransmission policies for transport protocols that support multihoming (e.g., SCTP). The policies dictate whether retransmissions are sent to the same peer IP address as the original transmission, or sent to an alternate peer IP address. Each policy presents tradeoffs based on the paths' bandwidth, delay, loss rate, and IP destination reachability. We find that sending all retransmissions to an alternate peer IP address is useful when the primary IP address becomes unreachable, but often degrades performance in non-failure scenarios. On the other hand, sending all retransmissions to the same peer IP address as the original transmission reverses the tradeoffs. We balance the tradeoffs by proposing a hybrid policy that sends fast retransmissions to the same peer IP address as the original transmission, and sends timeout retransmissions to an alternate peer IP address. We show that even with extensions which we proposed to improve the policies' performance, the hybrid policy is the best performing policy in failure and non-failure scenarios.**

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).

The current transport protocol workhorses, TCP and UDP, do not support multihoming; TCP allows binding to only one network address at each end of a connection. When TCP was designed, network interfaces were expensive components, and hence multihoming was beyond the ken of research.

Two recent transport layer protocols, the Stream Control Transmission Protocol (SCTP) [9], [23] and the Datagram Congestion Control Protocol (DCCP) [18] support multihoming at the transport layer. The motivation for multihoming in DCCP is mobility [17], 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. SCTP's n to m binding allows a multihomed sender with n interfaces to send to any of a multihomed receiver's m destination addresses. For example, an SCTP multihomed association between hosts A and B in Figure 1 could be bound to both IP addresses at each host: $(\{A_1, A_2\}, \{B_1, B_2\})$. Such an association allows data transmission from host A to host B to be sent to either B_1 or B_2.

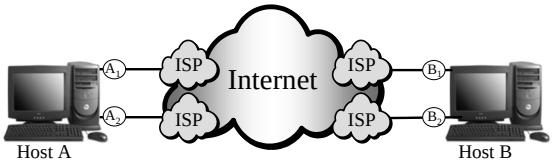

Fig. 1. Example multihoming topology

Currently, SCTP uses multihoming for fault tolerance purposes only, and not for concurrent multipath transfer [14]. 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 *fails over* to using an alternate destination address without requiring action by the user or application layer.

When data is lost, a sender uses an alternate destination address for retransmissions. SCTP's current retransmission policy [23] states that "when its peer is multihomed, an endpoint SHOULD try to retransmit [data] to an active destination transport address that is different from the last destination address to which the [data] was sent." This policy, which we refer to as *AllRtxAlt (All Retransmissions to Alternate)*, attempts to improve the chance of success by sending all retransmissions to an alternate destination address [22]. 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.

underlying assumption is that loss indicates either that the network path to the primary destination is congested, or the primary destination is unreachable. Thus, retransmitting to an alternate destination might avoid yet another loss of the same data.

We show that this policy actually degrades performance in many circumstances. We explore two alternative retransmission policies and find that the best policy, for both failure and non-failure scenarios, is to send (a) fast retransmissions to the primary destination, and (b) timeout retransmissions to an alternate destination. We show that this hybrid policy performs best when combined with two enhancements: our Multiple Fast Retransmit algorithm, and either timestamps or our Heartbeat After RTO mechanism. The Multiple Fast Retransmit algorithm reduces the number of timeouts. Timestamps and the Heartbeat After RTO mechanism both improve performance when timeouts are common by providing extra RTT measurements and maintaining low RTO values.

This paper combines and extends results published by the authors in three incremental conference publications [5]–[7], thereby documenting the complete development of this research. Section II demonstrates the problem with SCTP's current retransmission policy (AllRtxAlt) by comparing it to an alternative policy, *AllRtxSame (All Retransmissions to Same)*. Section III introduces and evaluates a third hybrid policy, *FrSameRtoAlt (Fast Retransmissions to Same, Timeouts to Alternate)*, which attempts to balance the tradeoffs between AllRtxAlt and AllRtxSame. Section IV introduces and evaluates three extensions to further improve the performance of the three policies. Section V compares the policies' performance with their best extensions in non-failure scenarios, and Section VI compares them in failure scenarios. Section VII concludes the paper.

II. AllRtxAlt's Problem

AllRtxAlt is the retransmission policy currently specified for SCTP in RFC2960. This policy attempts to bypass transient network congestion and path failures by sending all retransmissions to an alternate destination. Intuitively, we would expect that sending retransmissions to an alternate path would be beneficial, particularly when the alternate path's quality is better (i.e., higher bandwidth, lower delay, and/or lower loss). Similarly, when the alternate path's quality is worse, we expect sending retransmissions to the same destination as their original transmission should provide better performance. To test these hypotheses, we evaluate the performance of AllRtxAlt and the AllRtxSame policy – send all retransmissions to the same destination as their original transmission [6].

A. Analysis Methodology

We evaluate the retransmission policies using University of Delaware's SCTP module [8] for the ns-2 network simulator [3]. Figure 2 illustrates the network topology simulated: a dual-dumbbell topology whose core links have a bandwidth of 10Mbps and a one-way propagation delay of 25ms. Each router, R, is attached to five edge nodes. One of these five nodes is a dual-homed node for an SCTP endpoint, while the other four are single-homed and introduce cross-traffic that creates loss for the SCTP traffic.

The links to the dual-homed nodes have a bandwidth of 100Mbps and a one-way propagation delay of 10ms. The single-homed nodes also have 100Mbps links, but their propagation delays are randomly chosen from a uniform distribution between 5-20ms. The end-to-end one-way propagation delays range between 35-65ms. These delays roughly approximate reasonable Internet delays for distances such as coast-to-coast of the continental US, and eastern US to/from western Europe. Also, each link (both edge and core) has a buffer size twice the link's bandwidth-delay product.

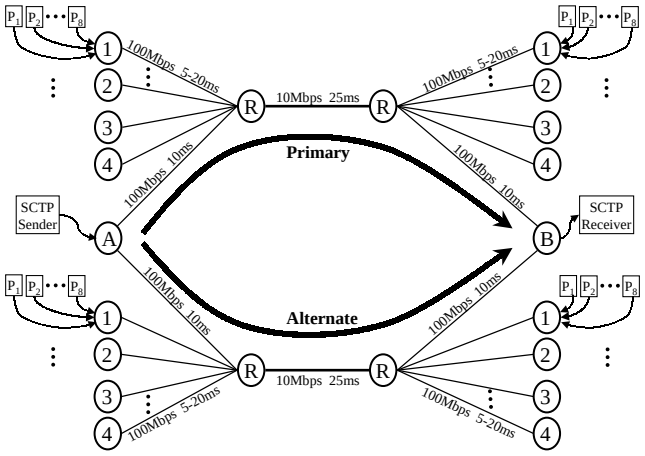

Fig. 2. Simulation network topology with cross-traffic, congestion-based loss, and no failures

Our configuration has two SCTP endpoints (sender A, receiver B) on either side of the network, which are attached to the dual-homed edge nodes. A has two paths, labeled primary and alternate, to B. Each single-homed edge node has eight traffic generators (labeled $P_1 - P_8$), each introducing cross-traffic based on a Pareto distribution. The cross-traffic packet sizes are chosen to *roughly* resemble the distribution found on the Internet: 50% are 44 bytes, 25% are 576 bytes, and 25% are 1500 bytes [1], [10]. The aim is to simulate an SCTP data transfer over a network with self-similar cross-traffic, which resembles the observed nature of traffic on data networks [20].

We simulate a 4MB file transfer with different network conditions, controlled by varying the load introduced by cross-traffic. All loss experienced is due to congestion at the routers; no loss is due to bit errors. The aggregate levels of cross-traffic on each path range from 5Mbps to 11Mbps. Although we independently control the levels of cross-traffic on each of the core links, the controls for the cross-traffic on each forward-return path pair are set the same. Each simulation has three parameters:

1) level of cross-traffic (in Mbps) on the primary path
2) level of cross-traffic (in Mbps) on the alternate path
3) AllRtxAlt vs AllRtxSame policy

B. Results

We compare the transfer times using AllRtxAlt versus AllRtxSame under various loss rates, with all else being equal (bandwidth, delay, etc). Since loss in our simulations only occurs due to congestion, we do not set the loss rate. Instead, we calculate the observed loss rate for a transfer after the simulation has completed. The loss rate is calculated as the number of SCTP packets dropped divided by the number of SCTP packets transmitted.

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 (more detailed results appear in [4]). Figure 3 presents the results for transfers with $\{3, 5, 8\}\%$ primary path loss. The graphs compare the file transfer time using AllRtxAlt versus AllRtxSame at various loss rates on the alternate path. Without failures, AllRtxSame never uses the alternate path, and therefore is unaffected by the alternate path's loss rate. Thus, AllRtxSame's transfer times are represented as a band parallel to the x-axis. This band outlines the upper and lower bounds of the 90% confidence interval. For example, we are 90% confident that the average 4MB file transfer time at 3% primary path loss lies between 34.3 and 35.1 seconds.

AllRtxAlt's transfer times are grouped by ranges of alternate path loss rates. The graph depicts the mean and the 90% confidence interval for each of these groups. The 90% confidence interval is calculated using an acceptable error of 10% of the mean. That is, we ran enough simulations to estimate the mean and 90% confidence interval with an acceptable error of at most 10% of the mean. For example, when the primary path's loss rate is 3% and the alternate path's loss rate is 1.5-2.5%, the 4MB file transfer time is on average 42.8 seconds with a 90% confidence interval between 41.1 and 44.5 seconds.

The graphs show that for $\{3, 5\}\%$ primary path loss, AllRtxSame outperforms AllRtxAlt for all alternate path loss rates (except 0%). Even when the alternate path's loss rate is better (i.e., lower) than the primary's, retransmitting on the alternate path degrades performance. This trend remains for all loss rates. Consider the results for 8% primary path loss. The anticipated benefits of AllRtxAlt only appear for alternate path loss rates of 0-3%. In other words, even if the alternate path's loss rate is up to 3% better (5-8%), it is better to retransmit data on the primary path with its an 8% loss rate. Clearly, this behavior is not what the SCTP authors expected when specifying the current retransmission policy.

Intuition tells us that when an alternate path's conditions are better than the primary's, then AllRtxAlt should improve performance, and when the conditions are worse on the alternate path, then AllRtxAlt should degrade performance. However, our results show that often the former expectation does not hold. Furthermore, independent results by other researchers confirm that AllRtxAlt degrades performance [11].

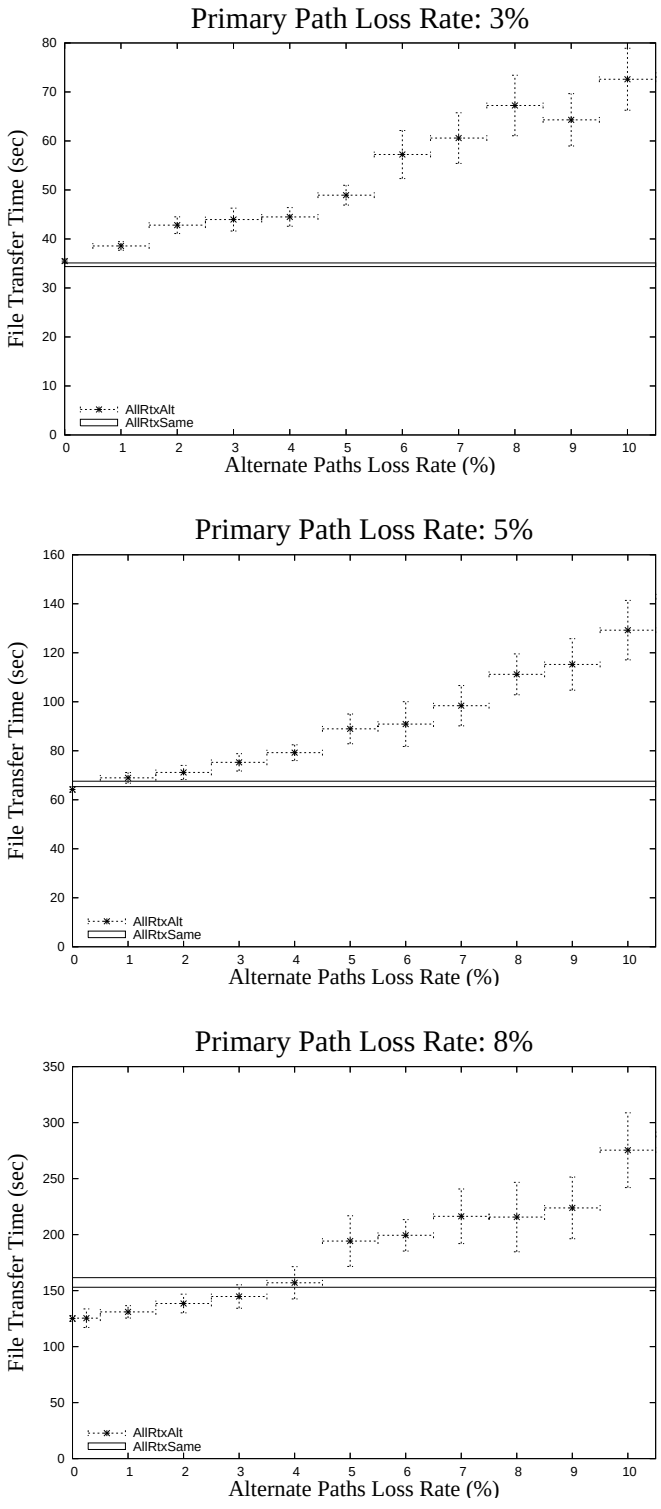

Fig. 3. AllRtxAlt vs AllRtxSame at $\{3, 5, 8\}\%$ primary path loss

C. Stale RTOs

Following analysis of several experiment traces, we attribute AllRtxAlt's poor performance to stale RTO values for the

alternate path. Due to Karn's algorithm [16], successful retransmissions on the alternate path cannot be used to update the RTT estimation of the alternate path. Timeouts on retransmissions, however, exponentially increase the RTO. The only traffic on the alternate path which updates the RTT estimate are the periodic heartbeat probes used to determine destination reachability, but these heartbeats are transmitted relatively infrequently (approximately every 30 seconds [23]). In many cases the RTO is exponentially increased more frequently than it can be reduced by an RTT estimate. The result is an overly conservative (i.e., too large) RTO on the alternate path for the majority of the association. Thus, anytime a retransmission on the alternate path is lost, a timeout occurs and the timeout is likely to be unnecessarily long. In addition, each timeout further contributes to the problem by doubling the RTO value.

Figure 4 illustrates the dynamics of the RTO values for the primary path (8% loss rate) and the alternate path (5% loss rate) during a 4MB file transfer using AllRtxAlt. This specific transfer sent a total of 2,889 original transmissions on the primary path, of which 229 had to be retransmitted on the alternate path, and of those retransmissions, 14 were lost and re-retransmitted on the primary path. The RTO value of the primary path stays relatively low (average is 2.3 seconds) during most of the transfer, because successful new data transmission on the primary path updates the RTT estimation and reduces the RTO value (most likely back to the minimum of 1 second). On the other hand, the alternate path with a lower loss rate maintains an average RTO value of 5.9 seconds – more than double the primary's. Figure 4's graph for the alternate path shows that the alternate path's RTO reduces only three times. In other words, only three heartbeats are successfully acked and used to measure the alternate path's RTT. The graph also shows seven timeouts exponentially increasing the RTO value of the alternate path.

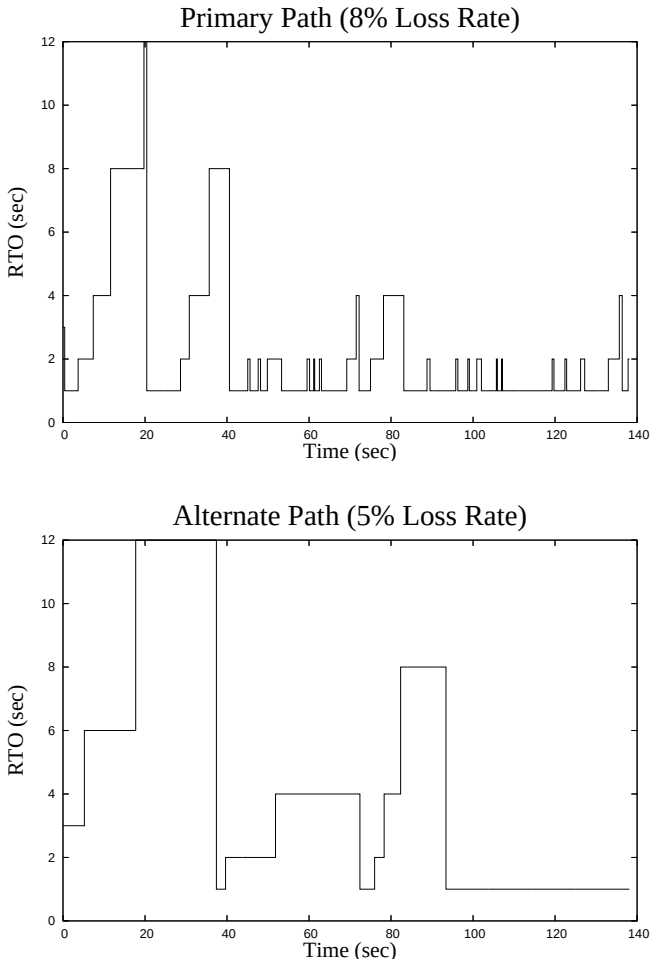

Fig. 4. Example RTO dynamics with 8% primary path loss and 5% alternate path loss

III. Balancing The Tradeoffs

We have demonstrated the tradeoffs between AllRtxAlt and AllRtxSame. AllRtxSame generally provides better performance, but AllRtxAlt may improve performance if the alternate path's loss rate is low enough to overcome the stale RTO problem. The difficulty in practice is that a sender generally has no prior knowledge about the paths' conditions. Without such information, the best a sender can do is balance the tradeoffs. To do so, we introduce the FrSameRtoAlt policy – a hybrid of AllRtxAlt and AllRtxSame. FrSameRtoAlt sends (a) fast retransmissions to the same destination as their original transmissions, and (b) timeout retransmissions to an alternate destination [7]. Since timeouts tend to occur more often at higher loss rates, this policy increases the use of the alternate path as the primary path's loss rate increases. This section evaluates FrSameRtoAlt against AllRtxAlt and AllRtSame. We determine whether FrSameRtoAlt does indeed balance the tradeoffs between the other two policies.

A. Analysis Methodology

Figure 5 illustrates the network topology used, which is based on the topology previously presented in Figure 2. But instead of using cross-traffic to induce congestion-based loss, we introduce uniform loss on these paths (0-10% each way) at the core links. We realize that the cross-traffic approach used in Figure 2 is a more realistic approach, but the simulation time for this technique became impractical. To evaluate if Figure 5's simplified model could still provide meaningful results, we compared representative simulations using the cross-traffic model from Figure 2 and the simpler uniform loss model from Figure 5. Although the absolute results differed for those examples compared, relative relationships remained consistent – leading to the same conclusions. We therefore proceeded with the simpler uniform loss model.

The topology in Figure 5 maintains the same link bandwidths and delays used in Figure 2. The core links have 10Mbps bandwidth and 25ms one-way delay, and the edge links have 100Mbps bandwidth and 10ms one-way delay. Thus, the end-to-end one-way delay on either path is 45ms, which is a reasonable delay within the continental US.

We simulate a 4MB file transfer with three input parameters for each simulation: (1) the primary path's loss rate, (2) the alternate path's loss rate, and (3) one of the three retransmission policies. Each parameter set is simulated with 60 different seeds.

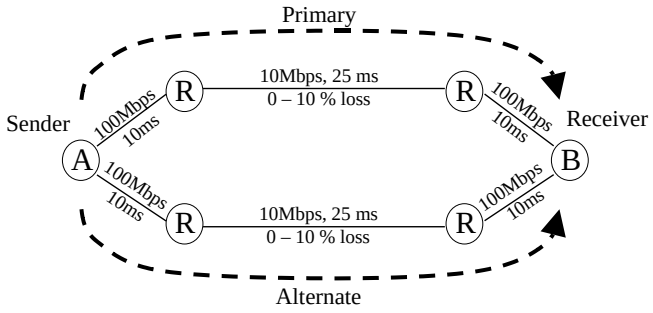

Fig. 5. Simulation network topology with random loss

B. Results

Figure 6 illustrates the results for $\{3, 5, 8\}\%$ primary path loss rates. For each graph in Figure 6, the alternate path's loss rate is varied on the x-axis, ranging from 0-10%. The graphs in Figure 6 compare the average transfer time of a 4MB file using one of the three policies: AllRtxAlt, AllRtxSame, FrSameRtoAlt.

We ensure statistical confidence by calculating the 90% confidence interval with an acceptable error of 10% of the mean. The 90% confidence intervals are not shown in the graphs for clarity. These intervals vary for different loss rates and retransmission policies, but on average the 90% confidence interval is about +/- 2-5 seconds around the mean. The largest 90% confidence interval is about +/- 13 seconds around the mean; as expected, larger confidence intervals tend to occur for higher loss rates and policies that use the alternate path more often.

Figure 6 clearly shows, as expected, that AllRtxSame's performance is uninfluenced by the alternate path's loss rate or by the stale RTO problem. Following the same trends observed in Section II-B, the graphs in Figure 6 also show that AllRtxAlt may improve performance when the alternate path's loss rate is lower than the primary's, but the stale RTO problem dominates performance. First, AllRtxAlt does worse than AllRtxSame when both paths have the same bandwidth, delay, and loss rate. Second, AllRtxAlt degrades performance more often than it improves performance, and the degree to which AllRtxAlt degrades performance is significantly higher than the degree to which it improves performance. For example, when the primary path loss rate is 5%, AllRtxAlt improves performance over AllRtxSame by 21% when the alternate path loss rate is 0%, but degrades performance by more than double (108%) when the alternate path loss rate is 10%.

FrSameRtoAlt, a hybrid policy, compromises between the advantages and disadvantages of AllRtxAlt and AllRtxSame. At low primary path loss rates (e.g., top graph in Figure 6), FrSameRtoAlt and AllRtxSame perform similarly. Most lost packets at such loss rates are detected by the fast retransmit algorithm, and thus are retransmitted to the same destination. The relatively few timeouts that occur in these conditions are not enough to significantly influence the results.

As the primary path loss rate increases, AllRtxSame and FrSameRtoAlt begin to perform differently. An increase in the number of timeouts causes FrSameRtoAlt to send more

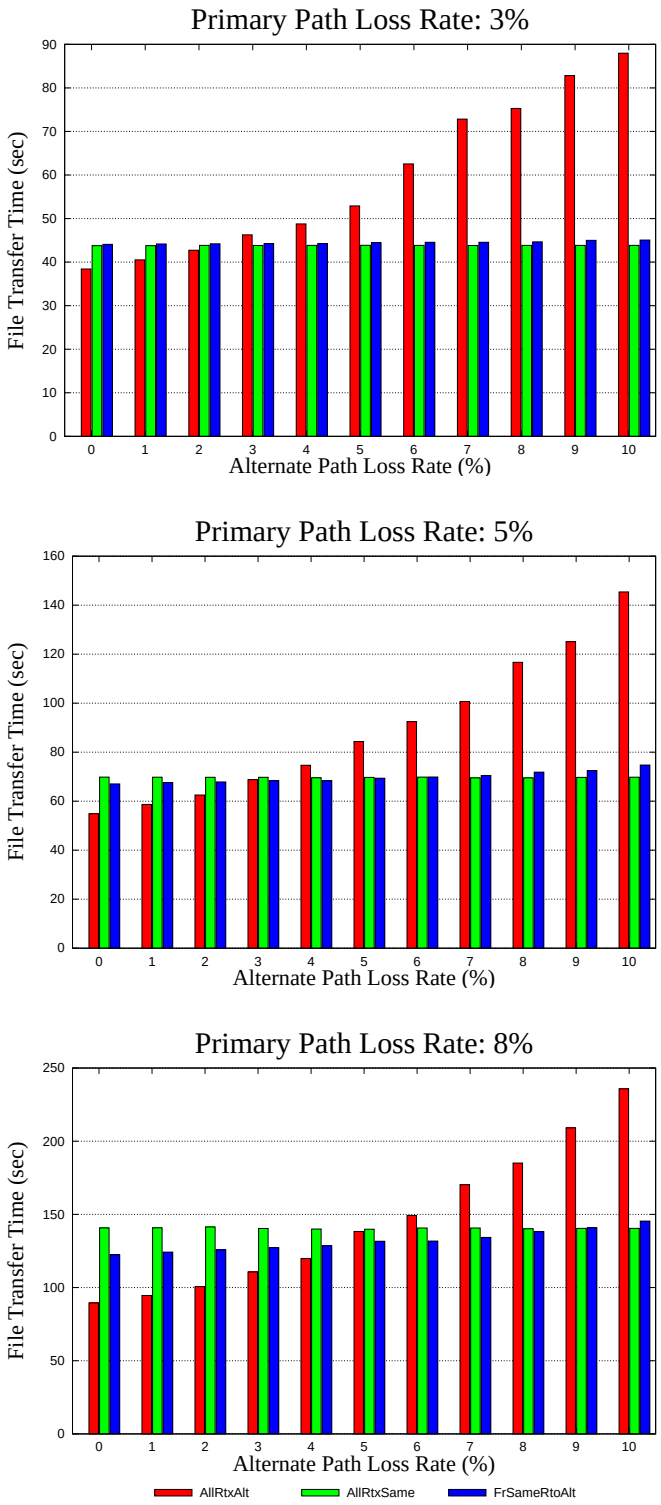

Fig. 6. AllRtxAlt, AllRtxSame, and FrSameRtoAlt at $\{3, 5, 8\}\%$ primary path loss

traffic to the alternate destination. As a result, FrSameRtoAlt's performance depends more on the alternate path's loss rate. However, since FrSameRtoAlt does not send fast retransmissions to the alternate destination, the alternate path's loss rate influences FrSameRtoAlt's performance less than AllRtxAlt's. FrSameRtoAlt's improvements are not as great as AllRtxAlt's, but neither are the degradations. Furthermore, FrSameRtoAlt improves performance to a greater extent than it degrades performance. For example, when the primary path loss rate is 8%, FrSameRtoAlt improves performance over AllRtxSame by 13% when the alternate path loss rate is 0%, but degrades performance by only 3% when the alternate path loss rate is 10%. To contrast, AllRtxAlt offers a 36% improvement and 68% degradation under the same conditions.

Since loss conditions of paths are unknown *a priori*, we need to consider overall performance. From the results in this section, we conclude that AllRtxAlt is the worst policy. AllRtxSame and FrSameRtoAlt perform about the same with FrSameRtoAlt offering a slight advantage when primary path loss rates are high.

IV. Performance Enhancing Extensions

We now introduce three performance enhancing policy extensions. The motivation behind these extensions is to determine if the relative relationships between the retransmission policies remain unchanged even after improving each of one's performance.

A. Heartbeat After RTO (HAR)

When a timeout occurs, the Heartbeat After RTO (HAR) mechanism sends a heartbeat immediately to the destination on which a timeout occurred. This behavior is in addition to the normal data retransmission behavior (specified by the retransmission policy) that remains unchanged. Since AllRtxSame sends timeout retransmissions to the same destination, HAR is not applicable (see Figure 7). The extra heartbeats introduced by HAR try to alleviate the stale RTO problem of AllRtxAlt and FrSameRtoAlt. With HAR, a sender updates an alternate destination's RTT estimate more frequently, thus resulting in a better RTT estimate on which to base the RTO value [5].

For example, suppose a packet is lost in transit to the primary destination, and later gets retransmitted to an alternate destination. Also suppose that the retransmission times out. The lost packet is retransmitted again to yet another alternate destination (if one exists; otherwise, the primary). More importantly, a heartbeat is also sent to the alternate destination which timed out. If the heartbeat is successfully acked, that destination acquires an additional RTT measurement to help reduce its recently doubled RTO.

B. Timestamps (TS)

The timestamp (TS) mechanism is similar to TCP's timestamp mechanism. By including timestamps in each packet, the retransmission ambiguity problem is resolved. That is, the sender can distinguish between acks for original transmissions and acks for retransmissions. Thus, Karn's algorithm can be eliminated, and successful retransmissions can be used to update the RTT estimate and maintain a more accurate RTO value. This feature is especially useful in alleviating the stale RTO problem of AllRtxAlt and FrSameRtoAlt [5].

Note that this extension's motivation is to evaluate how much performance can be improved by eliminating the retransmission ambiguity problem. One alternative solution, incurring less packet overhead, may be to use flag(s) in the data and sack headers to signal whether the data/sack is for an original transmission or retransmission.

C. Multiple Fast Retransmit (MFR)

The Multiple Fast Retransmit (MFR) algorithm introduces extra state at the sender to allow lost fast retransmissions to be fast retransmitted again instead of incurring a timeout. For example, suppose a sender has a window of data in flight to the receiver, and packet x is lost. Data successfully received at the receiver are sacked, and any sacks for packets sent after x serve as missing reports for packet x. When the sender receives four such missing reports, the standard fast retransmit algorithm is triggered and packet x is retransmitted.[1] At this point, MFR state stores the highest packet currently outstanding, n. This way, if the retransmission of x is also lost, the sender can detect the loss with another four missing reports. However, this time only sacks for packets greater than n can serve as missing reports, because the sacks up to n were already in flight when x was fast retransmitted the first time [5].

MFR applies to AllRtxSame and FrSameRtoAlt. Since AllRtxAlt sends fast retransmissions to an alternate path, MFR could cause spurious fast retransmissions when path delays are different. For example, imagine a fast retransmission scenario where the primary path's RTT is shorter than the alternate path's. After a fast retransmission is sent on the alternate path, new data sent on the primary path may arrive at the receiver first. If so, the receiver uses sacks to convey this reordering to the sender. However, the sender's MFR algorithm will mistakenly interpret the reordering as loss of the fast retransmitted data, and incorrectly trigger another fast retransmission of the same data.

Although MFR prevents some timeouts, it does not provide additional RTT samples for alternate destinations, and thus inevitable timeouts continue to suffer from the stale RTO problem. MFR may be combined with HAR or timestamps to address stale RTOs. Figure 7 illustrates all ten policy-extension combinations.

D. Performance Evaluation

This section independently examines each policy, with its possible extensions. We determine which extension(s) provides the best improvement to each policy. For our evaluation, we use the methodology presented in Section III-A.

[1] SCTP [23] requires four missing reports to trigger a fast retransmit, whereas TCP requires only three analogous dupacks [2].

Rtx Policy	HAR	TS	MFR	MFR + HAR	MFR + TS
		Policy Extension(s)			
AllRtxAlt	✓	✓			
AllRtxSame		✓	✓		✓
FrSameRtoAlt	✓	✓	✓	✓	✓

Fig. 7. Possible policy-extension combinations

1) AllRtxAlt's Extensions: Figure 8 presents the results for AllRtxAlt and its extensions with $\{3, 5, 8\}\%$ primary path loss rates. As the graphs show, both the Heartbeat After RTO (HAR) and Timestamps (TS) extensions drastically improve AllRtxAlt's performance. HAR improves performance by as much as 38%, 43%, and 45% for primary path loss rates of 3%, 5%, and 8%, respectively. TS improves performance by slightly larger margins – as much as 45%, 51%, and 50%.

Both HAR and TS provide more RTT measurements of the alternate destination and reduce the occurrence of stale RTOs. Since HAR is a reactive mechanism that only obtains an extra measurement when timeouts occur, TS has an advantage of HAR. TS is proactive and offers more opportunities to measure the alternate path's RTT. Although TS adds a 12-byte overhead into each packet, the overhead does not adversely impact performance. We conclude TS is the better extension for AllRtxAlt.

2) AllRtxSame's Extensions: Figure 9 presents the results for AllRtxSame and its extensions. Since AllRtxSame's performance is independent of the alternate path's conditions, we plot all the results in a single graph with the primary path's loss rate on the x-axis.

The graph shows that the Multiple Fast Retransmit (MFR) extension is able to avoid timeouts and increase AllRtxSame's performance. For example, MFR improves AllRtxSame's performance by 8%, 10%, and 11% under 3%, 5%, and 8% primary path loss rates. TS only improves performance when the primary path's loss rate is high. For example, including TS improves performance by 6-8% when the primary path's loss rate is 8%, but provides no benefit at 3% and 5% primary path loss. At high loss rates, timeouts may occur frequently enough that no RTT measurement is obtained between timeouts. Thus, TS improves performance by allowing a successful timeout retransmission to be used for measuring the RTT, which in turn decreases the exponentially backed-off RTO. Combining MFR and TS provides the best performance for AllRtxSame.

3) FrSameRtoAlt's Extensions: FrSameRtoAlt qualifies for five extension combinations, three of which include the Multiple Fast Retransmit (MFR) extension. Figure 10 shows that individually, MFR provides greater improvement than either the Heartbeat After RTO (HAR) or Timestamps (TS) extensiosn. Using HAR or TS alone, at best, provides 2%, 5%, and 9% improvement at 3%, 5%, and 8% primary path loss, respectively. MFR alone, on the other hand, improves perfor-

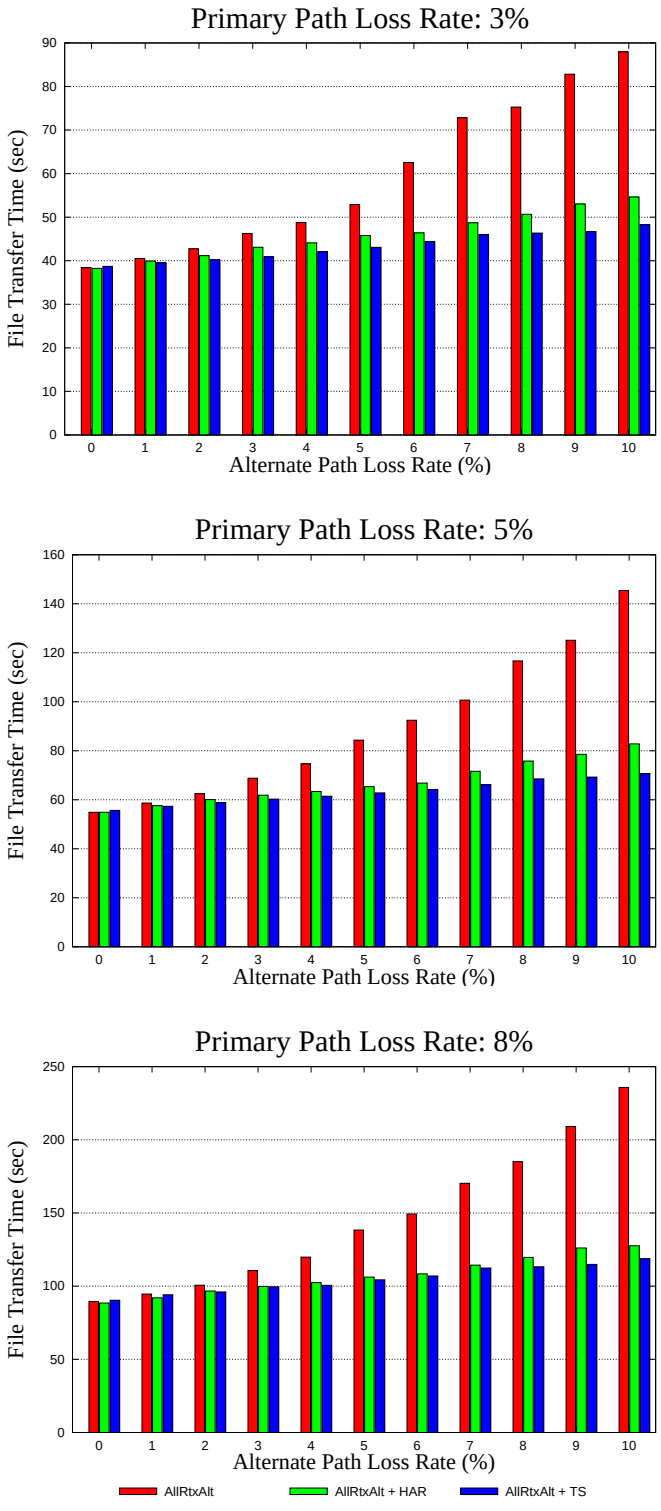

Fig. 8. AllRtxAlt and its extensions at $\{3, 5, 8\}\%$ primary path loss

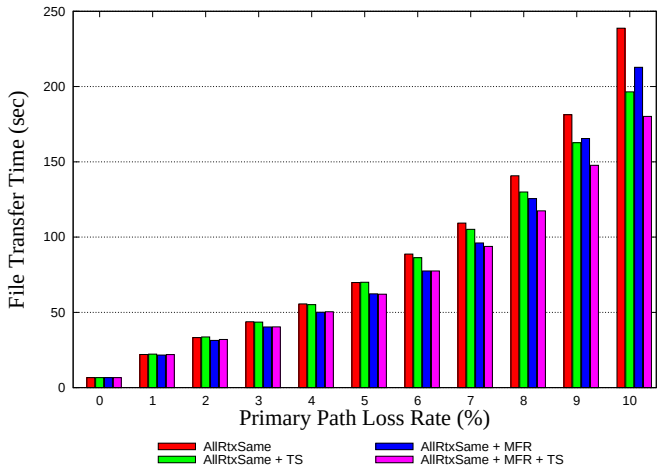

Fig. 9. AllRtxSame and its extensions across all primary path loss rates

mance by as much as 10%, 16%, and 14%. MFR's ability to avoid some timeouts has dramatic effects on FrSameRtoAlt's performance, because the stale RTO problem on the alternate path is also avoided.

Combining HAR or TS with MFR in general provides no added improvement; some marginal improvement occurs when the loss rate is high on the primary and alternate paths. For example, with 8% primary path loss and 10% alternate path loss, MFR+HAR and MFR+TS perform similarly and provide an additional 4-5% improvement over MFR alone. Thus, FrSameRtoAlt performs best when combined with either MFR+HAR or MFR+TS. However, we recommend that MFR+TS be used, since TS (or any mechanism that eliminates the retransmission ambiguity) has other orthogonal applications, such as the Eifel algorithm [19], [21].

V. Non-Failure Scenarios

This section revisits our performance comparison of the three policies in non-failure scenarios, but this time each policy is combined with our recommended extension(s):

- AllRtxAlt with Timestamps (AllRtxAlt+TS)
- AllRtxSame with Multiple Fast Retransmit and Timestamps (AllRtxSame+MFR+TS)
- FrSameRtoAlt with Multiple Fast Retransmit and Timestamps (FrSameRtoAlt+MFR+TS)

First, we evaluate their performance when both the primary and alternate paths have equal RTTs. Then, we assess the influence of the alternate path's delay. Finally, we consider three paths to determine if relative performance of the retransmission policies is influenced by the degree of multihoming. For readability throughout the remainder of this paper, we refer to AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS as simply AllRtxAlt, AllRtxSame, and FrSameRtoAlt, respectively.

A. Analysis Methodology

We again use the methodology presented in Section III-A for our evaluation, but in this section we investigate alternate path RTTs. The primary path remains unchanged

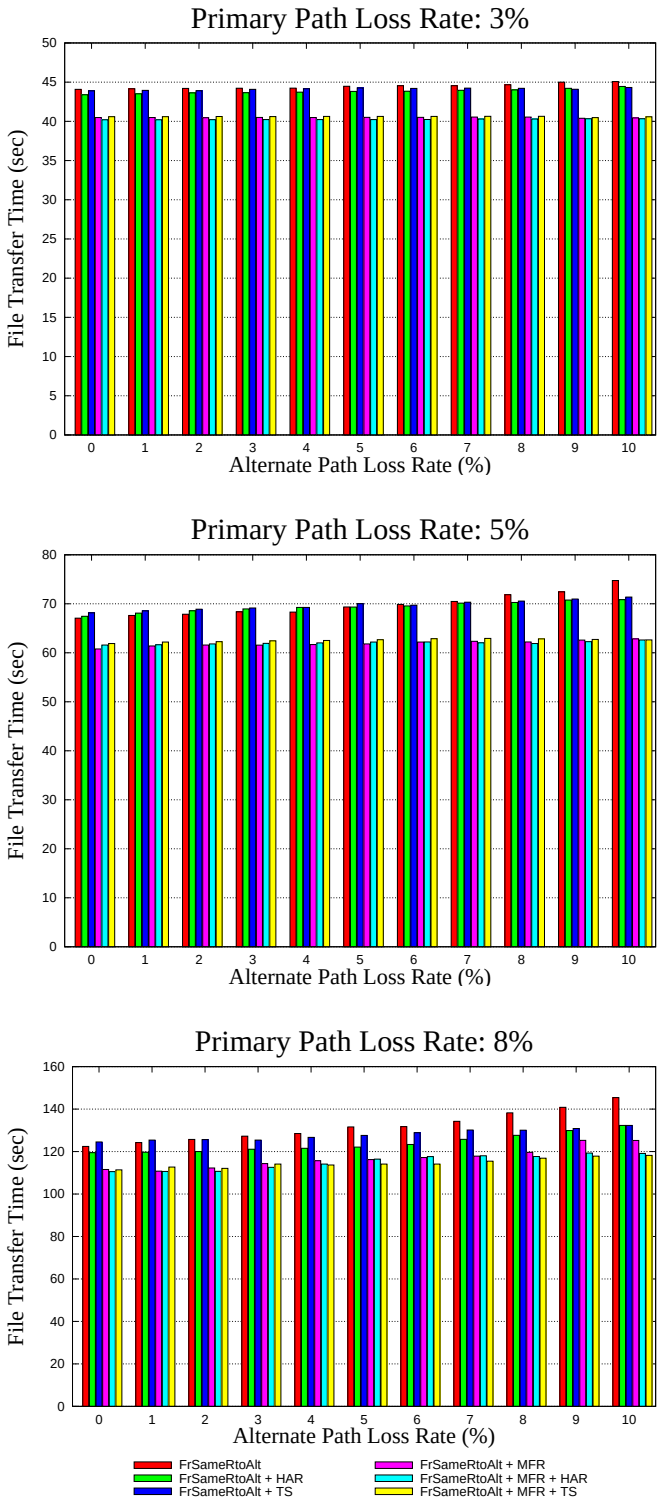

Fig. 10. FrSameRtoAlt with its extensions at $\{3, 5, 8\}\%$ primary path loss

(see Figure 11). However, the alternate path's core link has three possible one-way delays: 25ms, 85ms, and 500ms (i.e., end-to-end RTTs of 90ms, 210ms, and 1040ms). These values sample reasonable RTTs experienced on the Internet. Although 1040ms may seem large, flows passing through cellular networks often experience RTTs as high as 1 or more seconds [12], [13], [15].

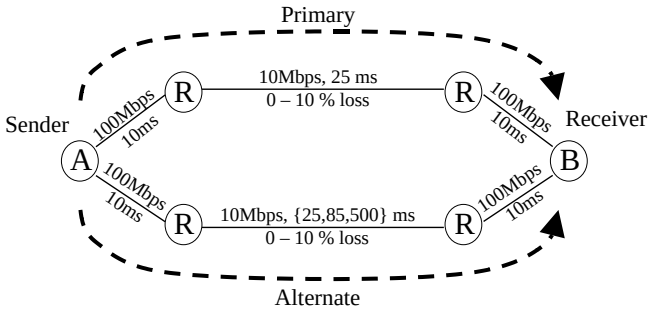

Fig. 11. Simulation network topology with random loss, 90ms primary path RTT, and $\{90, 210, 1040\}$ms alternate path RTT

Note that we do not simulate different link bandwidths. Lowering the alternate path's bandwidth simply increases the RTT, which we already independently control. Thus, the bandwidths remain constant in all our simulations.

B. Symmetric Path Delays

Figure 12 illustrates the results for $\{3, 5, 8\}\%$ primary path loss rates, a 90ms primary path RTT, and a 90ms alternate path RTT. Our first observation is that the extensions reduced the performance gap between the three retransmission policies (compare Figure 12 with Figure 6). For 3% primary path loss, the three policies perform relatively the same (less than 5% difference) for 0-4% alternate path loss. Higher alternate path loss rates cause AllRtxAlt to degrade performance by as much as 20%, while the results for AllRtxSame and FrSameRtoAlt remain unchanged.

When the primary path loss rate is 5%, AllRtxSame and FrSameRtoAlt again perform similarly. AllRtxAlt, on the other hand, improves performance by as much as 10% and degrades performance by as much as 14%, depending on the alternate path's loss rate (generally an unknown metric). Comparing this relatively low degradation to the degradation of 108% presented in Section III-B for the same network conditions, the stale RTO problem seems to have been completely eliminated.

The results for 8% primary path loss further confirm this observation. AllRtxAlt and FrSameRtoAlt outperform AllRtxSame across nearly all alternate path loss rates, and do about the same (within 2% of each other) when that alternate path loss rate is higher than 8%.

Overall, the results in Figure 12 do not present a strong argument for a single best policy. FrSameRtoAlt outperforms AllRtxSame, but deciding between AllRtxAlt and FrSameRtoAlt is not straight forward. FrSameRtoAlt provides only conservative gains, but does not degrade performance at all. AllRtxAlt may provide more significant gains, but risks the potential of degradation of the same magnitude.

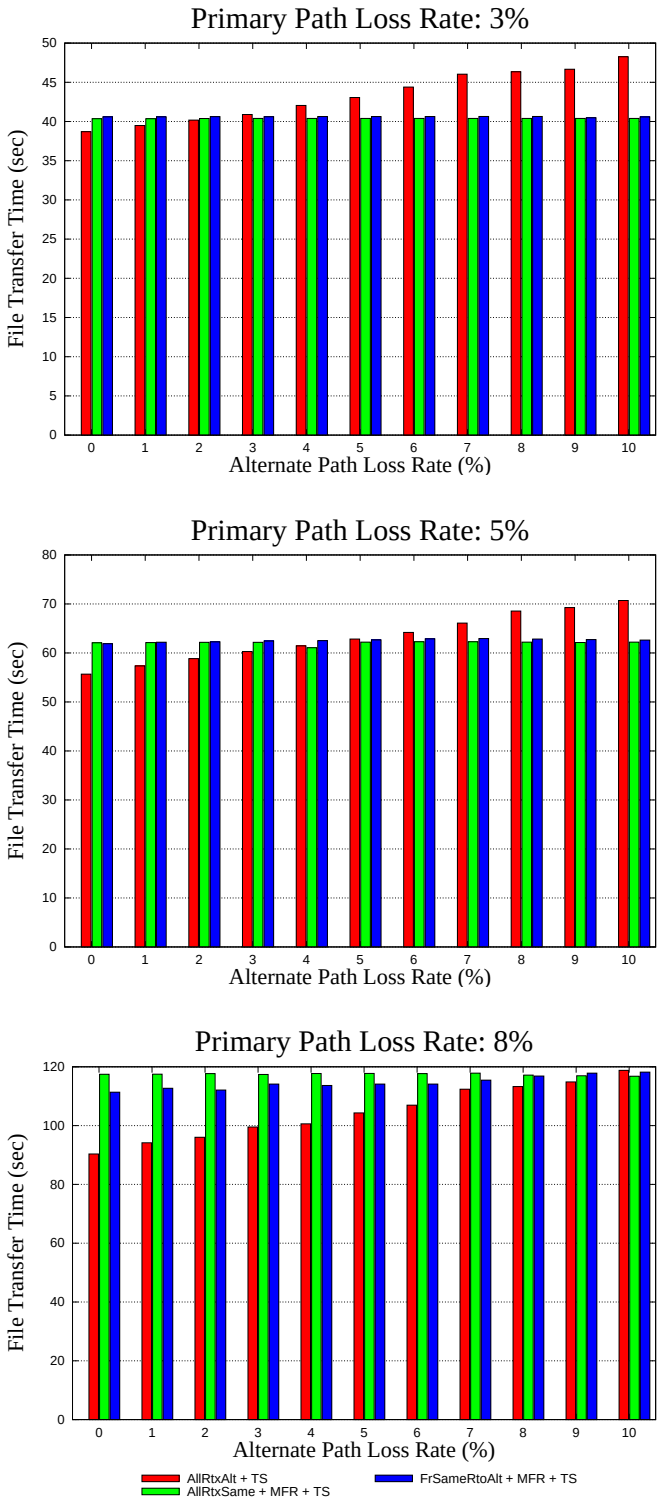

Fig. 12. AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS at $\{3, 5, 8\}\%$ primary path loss, 90ms primary path RTT, and 90ms alternate path RTT

C. Asymmetric Path Delays

We find that increasing the alternate path's RTT to slightly more than double (210ms) does not significantly affect performance. Although the results are not shown, the graphs are similar to those in Figure 12. Hence, we push the limits further and present the performance of a 1040ms alternate path RTT in Figure 13.

The most obvious result is that AllRtxAlt's heavy use of the alternate path significantly degrades performance when the alternate path delay is large (no surprise). AllRtxSame's performance remains unchanged, as expected. FrSameRtoAlt's results, however, prove interesting. At 3% primary path loss, few timeouts occur. Hence, the alternate path is rarely used and FrSameRtoAlt's results remain unchanged. With a 5% and 8% primary path loss rate, FrSameRtoAlt degrades performance compared to AllRtxSame, but given the large difference in path delays, this degradation is minor. The alternate path's delay is more than ten times that of the primary, but in the worst case, FrSameRtoAlt degrades performance by only 9% and 24% for primary path loss rates of 5% and 8%, respectively.

D. 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 11. We configure both alternate paths to have the same properties (bandwidths, delays, and loss rates). Otherwise, the number of simulation parameters would quickly become unmanageable. The results (not shown) are similar to those for two paths. That is, the relationships between the policies remain the same. We expect that the trends will remain the same for configurations with more than three paths between endpoints.

VI. Failure Scenarios

We again evaluate the performance of the three policies with their best performing extension(s), this time focusing on failure scenarios, an important criteria in the overall evaluation. After all, a key motivation for supporting multihoming at the transport layer is improved failure resilience. Hence, a multihomed transport layer should use a retransmission policy that performs well when the primary destination becomes unreachable.

A. Failover Algorithm

In our evaluation, we assume a failover algorithm similar to that of SCTP. 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 heartbeats) on a particular destination increments an error count for that destination. The error count per destination is cleared whenever data or a heartbeat sent to that destination is acked. A destination is marked as failed

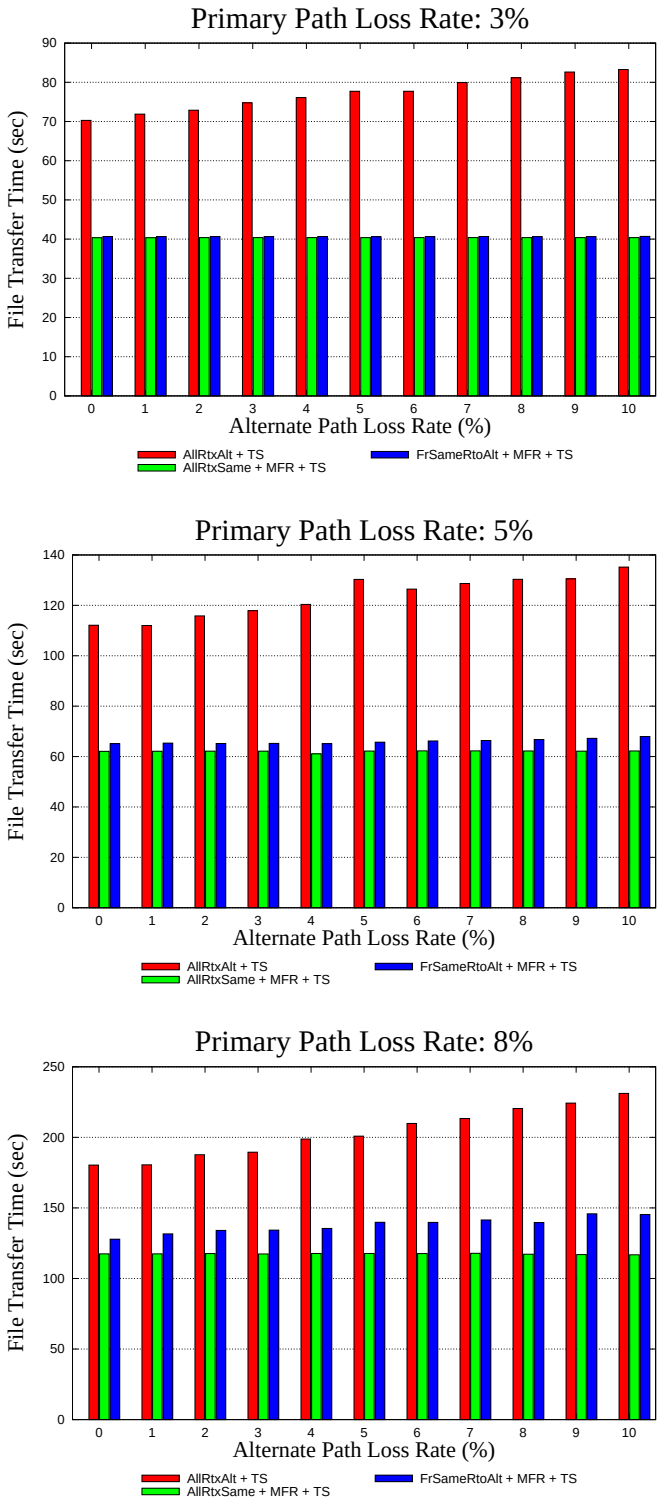

Fig. 13. AllRtxAlt+TS, AllRtxSame+MFR, and FrSameRtoAlt+MFR+TS at $\{3, 5, 8\}\%$ primary path loss, 90ms primary path RTT, and 1040ms alternate path RTT

when its error count *exceeds* the failover threshold (called Path.Max.Retrans in SCTP).

If the primary destination fails, the sender fails over to an alternate destination address and continues probing the primary destination with heartbeats. Failover is temporary in that a sender resumes sending new data to the primary destination if and when a future probe to the primary destination is successfully acked. If more than one alternate destination address exists, RFC2960 [23] leaves the alternate destination selection method unspecified. We assume a round-robin selection method.

RFC2960 [23] recommends default settings of: minimum RTO $= 1$s, maximum RTO $= 60$s, and Path.Max.Retrans (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 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 the failure detection time requires $6 * 60 = 360$s.

The failover details in this section are important for our analysis of the retransmission policies' performance in failure scenarios. However, we believe that the conclusions are independent of the actual failure detection method and/or parameters.

B. Analysis Methodology

We use the same methodology described in Section III-A, but in this section we introduce failure scenarios. The topology in Figure 14 shows that the both paths have the same characteristics, except that the primary path's core link experiences a bi-directional failure. We simulate a link breakage between the routers on the core path at two different times. The first set of failure experiments experience the link breakage at time $= 4$s into the transfer. With 0% loss on the primary path, about half of the 4MB file transfer is complete by this time. The second set's link breakage occurs at time $= 6.8$s into the transfer, specifically chosen to occur during the last RTT of the 4MB file transfer when the primary path's loss rate is 0%. In both failure scenarios, the link remains down until the end of the simulation.

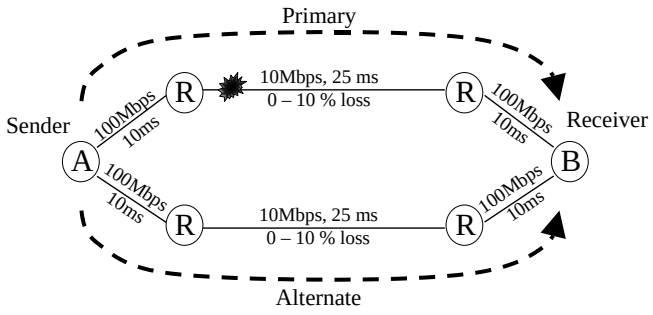

Fig. 14. Simulation network topology with random loss, equal delays, and primary path failure

C. Results

To gauge the performance during failure scenarios, we not only measure the file transfer time, we also consider the timeliness of data. File transfer in a failure scenario can be divided into three periods: (1) before failure, (2) during failure detection, (3) after failover. The first period has been covered in Section V.

The second period, failure detection, is important for both file transfer time and data timeliness. Fast failover time improves file transfer time, because the sender is able to resume "normal" transmission more quickly. As expected, we find that the failure detection time is similar for the three retransmission policies.

The retransmission policy affects timeliness of data in that it determines whether a transfer is blocked during the failure detection process. AllRtxSame delivers no data to the peer until the entire failure detection process completes and failover occurs. For example, with 0% primary path loss, the sender has 30 lost data packets outstanding when failure occurs in our first failure scenario (link breakage at time $= 4$s). AllRtxAlt and FrSameRtoAlt successfully retransmit these 30 packets after the first timeout in the failure detection process, thus delaying them by only 1s (or whatever the primary path's RTO is at that point). Furthermore, during each subsequent timeout that contributes to failure detection, the sender successfully retransmits one packet to the alternate destination. On the other hand, with AllRtxSame the sender successfully retransmits the initial 30 lost packets only after the failure detection completes, delaying them by at least 63s! This delay may be unacceptable to applications requiring timely data delivery.

During the third period, the sender has only one available path for transmission in our simulations. (The results in Section V apply to scenarios where more than one path are available during the third period.) Figure 15 presents the final transfer times for failure at time $= 4$s. As the graphs show, the primary path's loss rate has minimal influence on the file transfer time. Comparing these results with those in Figure 12 suggests that the third period has heaviest influence on file transfer time. Since failure occurs relatively early in the file transfer, the remaining portion of the transfer is large enough that its sole use of the alternate path is the most influential factor on file transfer time. Even the policies themselves do not provide much difference (at most 9%) in performance. Since there is only one available path in the third period, all three retransmission policies perform similarly, differing only by the extensions used.

As a worst case example, the file transfer times for 0% primary path loss and failure at time $= 6.8$s are shown in Figure 16. Since this failure scenario has a link breakage in the last RTT of the data transfer, the second period (i.e., failure detection) is the most influential factor on file transfer time. Figure 16 shows that AllRtxSame's blocking behavior during failure detection has drastic effects on the results. The file transfer with AllRtxSame takes about 70s to complete, whereas it only takes about 8-18s (depending on the alternate path's loss rate) with AllRtxAlt and FrSameRtoAlt. The reason is that AllRtxSame is unable to complete the transfer until

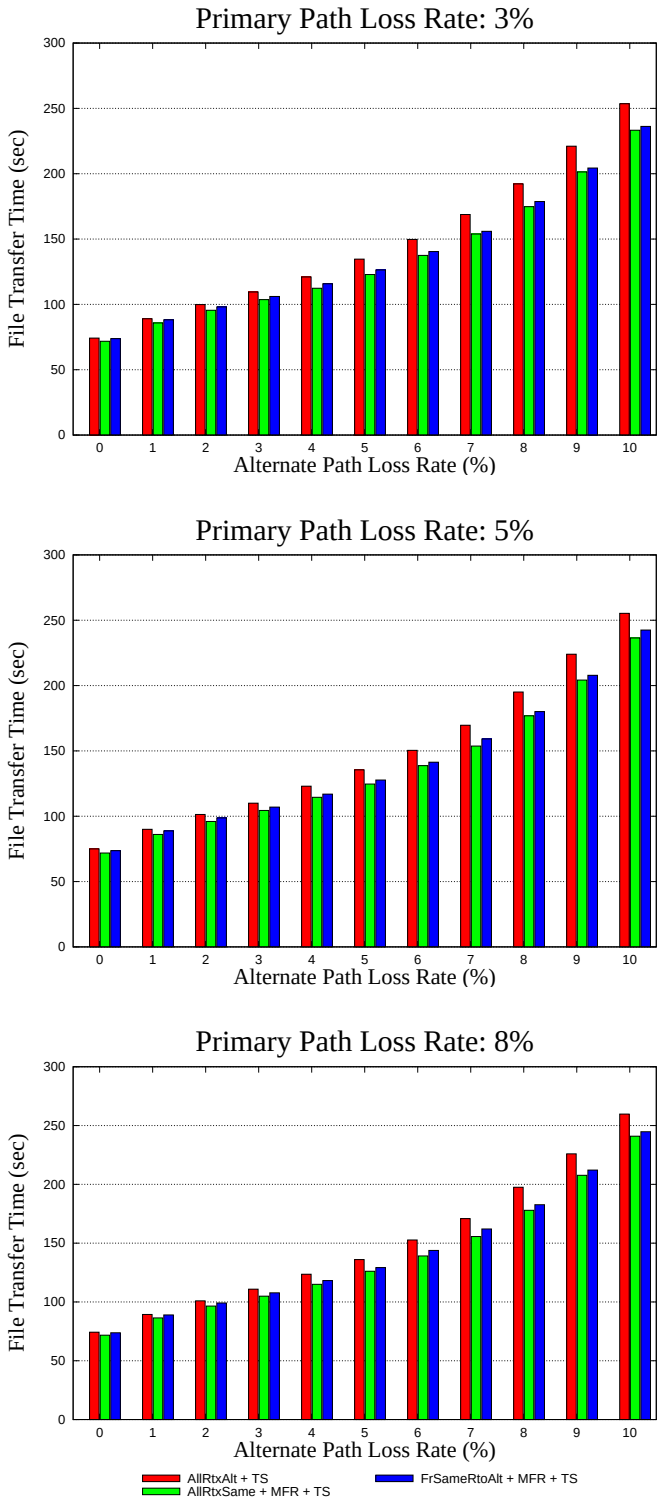

Fig. 15. AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS with primary path failure at time = 4s

after failover occurs, but AllRtxAlt and FrSameRtoAlt are able to finish the transfer during failure detection. Note that this example indeed represents a worst case situation for AllRtxSame, and was diabolically conceived.

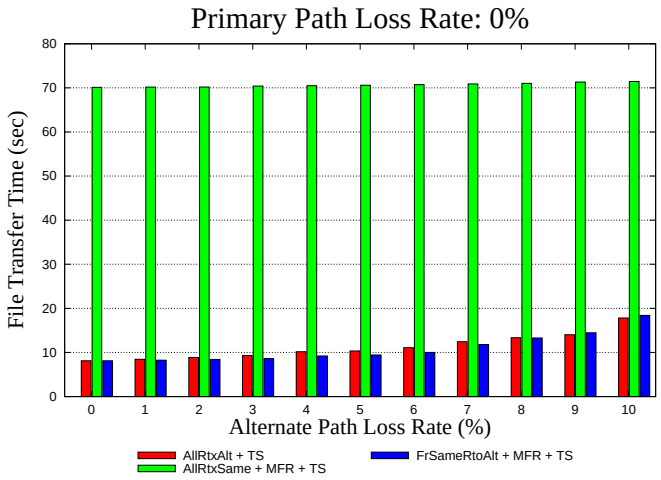

Fig. 16. AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS with primary path failure at time = 6.8s

In summary, all three policies provide similar throughput performance for large transfers during failure scenarios. However, AllRtxSame's blocking failure detection behavior degrades performance if the failure coincidentally occurs near the end of the transfer and/or data timeliness is important. Hence, AllRtxAlt and FrSameRtoAlt are recommended for failure scenarios.

VII. Conclusion

We have evaluated three retransmission policies for multihomed transport protocols, using SCTP to demonstrate the concepts. Without *a priori* knowledge about the available paths, a sender cannot have a static policy that decides where to retransmit lost data and expect to guarantee the best performance. Through simulation, we have measured and demonstrated the tradeoffs of three policies in non-failure and failure conditions. Our results show that the retransmission policy which best balances the tradeoffs is (1) send fast retransmissions to the same peer IP address as the original transmission, and (2) send timeout retransmissions to an alternate peer IP address. We have shown that this hybrid policy performs best when combined with two enhancements: our Multiple Fast Retransmit algorithm, and either timestamps or our Heartbeat After RTO mechanism. The Multiple Fast Retransmit algorithm reduces the number of timeouts. Timestamps and the Heartbeat After RTO mechanism both improve performance when timeouts are common by providing extra RTT measurements and maintaining low RTO values – an important feature for alternate paths that are mostly idle.

Acknowledgements

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

DISCLAIMER

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U. S. Government.

REFERENCES

[1] CAIDA: Packet Sizes and Sequencing, Mar 1998. http://traffic.caida.org.

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

[3] UC Berkeley, LBL, USC/ISI, and Xerox Parc. ns-2 documentation and software, Version 2.27, January 2004. www.isi.edu/nsnam/ns.

[4] A. Caro. End-to-End Fault Tolerance Using Transport Layer Multihoming. Phd dissertation, CIS Dept, University of Delaware, August 2005.

[5] A. Caro, P. Amer, J. Iyengar, and R. Stewart. Retransmission Policies with Transport Layer Multihoming. In *ICON 2003*, Sydney, Australia, September 2003.

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

[7] A. Caro, P. Amer, and R. Stewart. Retransmission Schemes for End-to-End Failover with Transport Layer Multihoming. In *GLOBECOM 2004*, Dallas, TX, November 2004.

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

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

[10] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. *INET 1998*, April 1998.

[11] M. Duke, T. Henderson, P. Spagnolo, J. Kim, and G. Michael. Stream Control Transport Protocol (SCTP) Performance Over the Land Mobile Satellite Channel. In *MILCOM 2003*, Boston, MA, October 2003.

[12] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola. Multi-Layer Protocol Tracing in a GPRS Network. In *International Conference on Ubiquitous Computing*, September 2002.

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

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

[15] R. Jayaram and I. Rhee. A Case for Delay-based Congestion Control for CDMA 2.5G Networks. In *International Conference on Ubiquitous Computing*, October 2003.

[16] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *ACM SIGCOMM 1987*, August 1987.

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

[18] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). draft-ietf-dccp-spec-09.txt, November 2004. (work in progress).

[19] S. Ladha, S. Baucke, R. Ludwig, and P. Amer. On Making SCTP Robust to Spurious Retransmissions. *ACM SIGCOMM Computer Communication Review*, 34(2):123–135, April 2004.

[20] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-similar Nature of Ethernet Traffic. In *ACM SIGCOMM 1993*, San Francisco, CA, September 1993.

[21] R. Ludwig and R. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communications Review*, 30(21):30–36, January 2000.

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

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