# Retransmission Schemes for End-to-end Failover with Transport Layer Multihoming

**Armando L. Caro Jr.**
Protocol Engineering Lab
Computer and Information Sciences
University of Delaware
acaro@cis.udel.edu

**Paul D. Amer**
Protocol Engineering Lab
Computer and Information Sciences
University of Delaware
amer@cis.udel.edu

**Randall R. Stewart**
Transport Technologies
Internet Technologies Division
Cisco Systems, Inc.
rrs@cisco.com

***Abstract*— We previously evaluated five retransmission schemes in non-failure scenarios for transport protocols that support multihoming. In this paper, we introduce five additional retransmission schemes, and evaluate all ten schemes under both non-failure and failure scenarios. We show that the best retransmission policy dictates that (a) new data transmissions and fast retransmissions should be sent to the same peer IP address, and (b) timeout retransmissions should be sent to an alternate peer IP address. This policy performs best if combined with our Multiple Fast Retransmit algorithm.**

## I. Introduction

Multihoming among networked machines is a technologically feasible and increasingly economical proposition. A host is multihomed if it can be addressed by multiple IP addresses, as is the case when the host has multiple network interfaces. Though feasibility alone does not determine adoption of an idea, multihoming can be expected to be the rule rather than the exception in the near future. For instance, cheaper access to the Internet will motivate content providers to have simultaneous connectivity through multiple ISPs. More and more home users will have wired and wireless connections. Furthermore, wireless devices may be simultaneously connected through multiple access technologies. Multihoming is improving a host's fault tolerance at an increasingly economical cost.

The current transport protocol workhorses, TCP and UDP, are ignorant of multihoming; TCP allows binding to only one network address at each end of a connection. When TCP was designed, network interfaces were expensive components, and hence multihoming was beyond the ken of research. Lowering interface costs and a desire for networked applications to be fault tolerant at an end-to-end level have brought multihoming within the purview of the transport layer.

Two recent transport layer protocols, the Stream Control Transmission Protocol (SCTP) [6], [10] and the Datagram Congestion Control Protocol (DCCP) [8] support multihoming at the transport layer. The motivation for multihoming in DCCP is mobility, while SCTP is driven by a broader and more generic application base, which includes fault tolerance and mobility. Of the two, we use SCTP primarily because our focus is on fault tolerance, but the results and conclusions presented in this paper should be applicable in general to reliable SACK-based transport protocols that support multihoming.

SCTP, an IETF standards track transport layer protocol, allows binding of one transport layer association (SCTP's term for a connection) to multiple IP addresses at each end of the association. This $n$ to $m$ binding allows an SCTP sender to send data to a multihomed receiver via different destination addresses. For example, an SCTP 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 would allow data transmission from host $A$ to host $B$ to be sent to either $B_1$ or $B_2$.

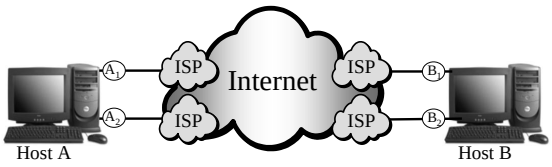


Fig. 1. Example multihoming topology

Currently, SCTP uses multihoming for redundancy purposes only and not for concurrent multipath transfer [7]. Each endpoint chooses a single peer IP address as the primary destination address, which is used for transmission of new data. Retransmitted data are sent to an alternate peer IP address(es). RFC2960 [10] states in Section 6.4 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."

SCTP's current retransmission policy attempts to improve the chance of success by sending all retransmissions to an alternate peer IP address [9]. The underlying assumption is that loss indicates either that the network path to the primary destination is congested, or the peer IP address used is unreachable. Hence, SCTP retransmits to an alternate peer IP address in attempt to avoid another loss of the same data. We have shown previously that SCTP's current retransmission

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.

policy in RFC2960 actually degrades performance in some circumstances [4]. We also explored alternative retransmission schemes, and concluded that best performance occurs if our Multiple Fast Retransmit algorithm is used and lost data are retransmitted to the same peer IP address to which they were originally sent [3].

However, our previous results assume reachability of all peer IP addresses at all times (i.e., no failures). In this paper, we evaluate retransmission schemes during failure scenarios. We also introduce five additional schemes, resulting in a total of ten retransmission schemes that we evaluate in both non-failure and failure scenarios. We show that the best retransmission policy dictates that (a) new data transmissions and fast retransmissions should be sent to the same peer IP address, and (b) timeout retransmissions should be sent to an alternate peer IP address. We find this policy to perform best if combined with our Multiple Fast Retransmit algorithm.

We begin in Section II by describing the retransmission schemes evaluated in this paper. We comparatively evaluate these schemes using ns-2 simulation as described in Section III. The results and analysis of non-failure and failure scenarios are presented in Section IV and Section V, respectively. Section VI concludes the paper and discusses future work.

## II. Retransmission Schemes

The ten retransmission schemes evaluated are combinations of the following three policies and three algorithms.

### A. Policies

1) AllRtxAlt - All retransmissions are sent to an alternate destination. This policy represents SCTP RFC2960 and attempts to bypass transient network congestion and path failures. A drawback is that alternate destinations often have overly conservative (i.e., too large) RTOs, which significantly degrades performance when retransmissions of lost packets themselves are lost [4].

2) AllRtxSame - All retransmissions are sent to the same destination. This policy often improves performance in non-failure scenarios by using the destination with the most accurate RTO [4]. However, if the primary destination becomes unreachable, this policy will not successfully deliver any data until the sender detects failure and fails over to an alternate destination.

3) FrSameRtoAlt - Fast retransmissions are sent to the same destination, and timeout retransmissions are sent to an alternate destination. This policy is introduced in this paper as a compromise between the two policies above. Fast retransmissions are generally caused by network congestion, whereas timeouts may be caused by either severe congestion or path failure.

### B. Algorithms

1) Heartbeat After RTO (HAR) - In addition to normal timeout behavior, a *heartbeat* (control probe normally sent to each idle destination for RTT measurement and reachability status) is sent immediately to the destination on which a timeout occurred. This algorithm is useful to obtain more RTT measurements and hence a more accurate RTO setting for alternate destinations that experience timeouts. This algorithm applies only to AllRtxAlt and FrSameRtoAlt policies, because it offers no benefits to AllRtxSame.

2) Timestamps (TS) - Similar to TCP's timestamp option, each packet includes a 12-byte timestamp to eliminate the retransmission ambiguity. Thus, Karn's algorithm can be eliminated, and successful retransmissions on alternate paths can be used to obtain RTT measurements. This algorithm applies only to AllRtxAlt and FrSameRtoAlt policies, because the packet overhead is not worth the limited performance gain (if any) for AllRtxSame.

3) Multiple Fast Retransmit (MFR) - The sender maintains extra recovery state to allow lost fast retransmissions to be fast retransmitted again. Thus, MFR reduces the number of timeouts. This algorithm applies only to AllRtxSame and FrSameRtoAlt policies, because using it with AllRtxAlt may often generate spurious fast retransmissions.

### C. Schemes

1) AllRtxAlt (i.e., original SCTP)
2) AllRtxAlt+HAR
3) AllRtxAlt+TS
4) AllRtxSame
5) AllRtxSame+MFR
6) FrSameRtoAlt
7) FrSameRtoAlt+HAR
8) FrSameRtoAlt+TS
9) FrSameRtoAlt+MFR
10) FrSameRtoAlt+MFR+HAR

Further motivation and details about these retransmission policies and algorithms can be found in [3].

## III. Methodology

We evaluate the ten retransmission schemes described in Section II using University of Delaware's SCTP module [5] for the ns-2 network simulator [1]. Figure 2 illustrates the network topology used. The core links have a 10Mbps bandwidth

and a 25ms one-way delay. Each router, $R$, 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 one-way delay is 45ms, which approximates reasonable Internet delays for distances such as coast-to-coast of the continental US, and eastern US to/from western Europe.

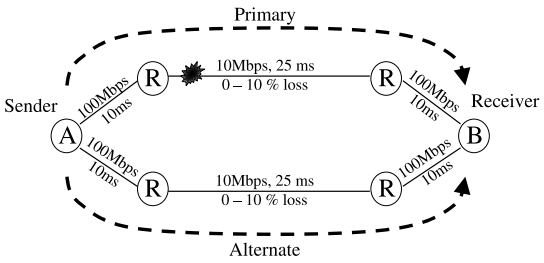


Fig. 2. Simulation network topology

The SCTP sender, $A$, has two paths (labeled *Primary* and *Alternate*) to the SCTP receiver, $B$. We introduce uniform loss on these paths (0-10% each way) at the core links. We realize that a more realistic approach would be to introduce only congestion induced loss by simulating self-similar cross-traffic. Our previous results were gathered using this technique with a dual-dumbbell topology [3], [4]. However, the simulation time for this technique was too time consuming and impractical. We did compare representative simulations using uniform loss versus simulations using cross-traffic based loss, and our analysis remains unchanged. We therefore proceeded with uniform loss on the paths instead of cross-traffic based loss.

We simulate a 4MB file transfer with and without failure. The failure scenario has a bi-directional failure on the primary path occurring at time = 4 seconds into the transfer (with 0% loss on the primary path, about 53% of the file transfer is complete by this time), and remaining until the end of the simulation. The failure is simulated by a link breakage between the routers on the primary path. The three input parameters for each simulation are the primary path's loss rate, the alternate path's loss rate, and one of the ten retransmission schemes. Each parameter set is simulated with 120 different seeds. Our results exclude the few simulations that were unable to successfully establish an association due to loss conditions.

## IV. Non-failure Scenarios

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 for non-failure scenarios. Figure 3 illustrates the results for 1-9% primary path loss rates. For each graph in Figure 3, the alternate path's loss rate is varied on the $x$-axis, ranging from 0-10%. The graphs in Figure 3 compare the average transfer time of a 4MB file using SCTP's current retransmission scheme (AllRtxAlt) versus five of the remaining nine schemes. The schemes compared in Figure 3 are the best performing schemes of each retransmission policy: AllRtxAlt+HAR, AllRtxAlt+TS, AllRtxSame+MFR, FrSameRtoAlt+MFR, FrSameRtoAlt+MFR+HAR.

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 schemes, but on average the 90% confidence interval is about +/- 2 seconds around the mean. The largest 90% confidence interval is about +/- 13 seconds around the mean; as expected, larger confidence intervals tend to be for higher loss rates.

Figure 3 clearly shows that AllRtxAlt performs the worst overall. This scheme does well when the alternate path loss rate is very low, but its performance degrades rapidly as the alternate path loss rate increases. We have previously shown that this behavior is due to overly conservative RTOs for the alternate destination. In other words, the alternate destination does not have enough traffic that can gather RTT measurements. Thus, if a retransmission on the alternate path is lost, it will eventually timeout, double the alternate destination's RTO, and be retransmitted on the primary path. The doubled RTO is only reduced when a new RTT measurement is obtained for the alternate destination, but heartbeats (normally sent approximately every 30 seconds) are the only traffic on the alternate path that can measure the RTT [4].

The HAR and TS algorithms dramatically improve AllRtxAlt's performance, but AltRtxAlt+HAR and AltRtxAlt+TS do not perform well when the primary path loss rate is 1-5% and the alternate path loss rate is greater than the primary's. Increasing the number of RTT measurements may alleviate the overly conservative RTO issue, but it cannot avoid the drawback of retransmitting on an alternate path with a higher loss rate than the primary path.

AltRtxSame (not shown) and AltRtxSame+MFR also improve performance, but suffer nearly the opposite problem. They do not perform well when the primary path loss rate is larger than the alternate's. Furthermore, when the primary path loss rate is greater than 8%, they perform poorly for all alternate path loss rates.

The FrSameRtoAlt policy is intended to be a compromise between the other two policies. Figure 3 shows that FrSameRtoAlt+MFR and FrSameRtoAlt+MFR+HAR almost always performs about the same or better than AltRtxSame+MFR. Most of the time, FrSameRtoAlt+MFR and FrSameRtoAlt+MFR+HAR also perform the same or better than AllRtxAlt+HAR and AllRtxAlt+TS. AllRtxAlt+HAR and AllRtxAlt+TS perform the best only when the alternate path loss rate is significantly less (generally half) than the primary's.

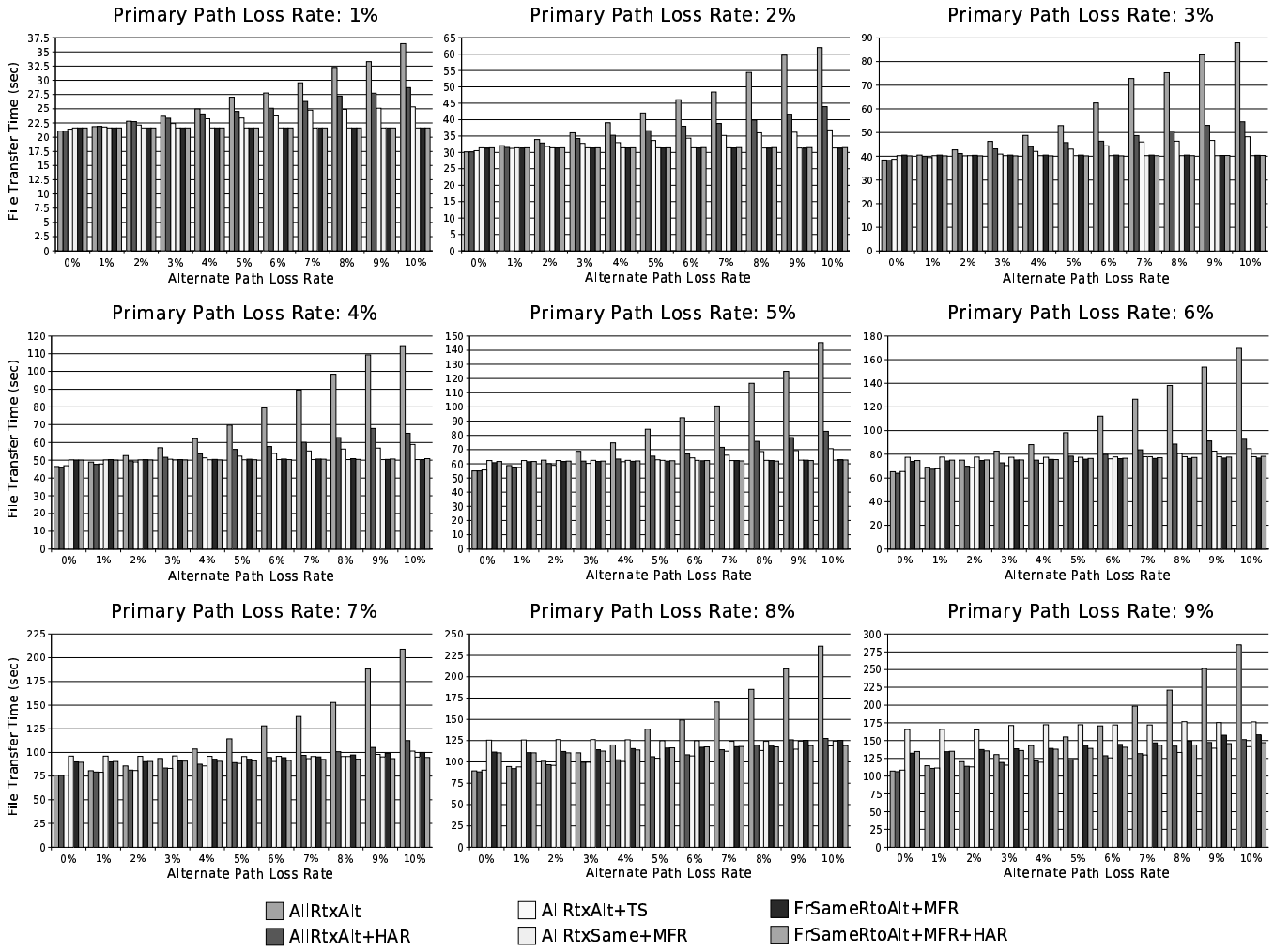


Fig. 3. File transfer time with no failure for primary path loss rates 1-9%

At low primary path loss rates, most of the losses are detected by fast retransmit. Hence, FrSameRtoAlt will send most of its lost packets to the same destination as AllRtxSame, thus experiencing similar results. Furthermore, the fast retransmissions do not suffer, as they would with AllRtxAlt, from overly conservative RTOs for the alternate destination.

As the loss rate on the primary path increases relative to the alternate path, it becomes more sensible to alleviate the loss conditions by retransmitting to the alternate path. As a result, AllRtxSame suffers at higher primary path loss rates by not using the alternate path. In contrast, FrSameRtoAlt is able to alleviate severe loss conditions by sending timeout retransmissions to the alternate path. Since AllRtxAlt sends all retransmissions to the alternate path, it performs best when loss conditions on the primary are significantly worse than the alternate.

We conclude that FrSameRtoAlt is the best overall policy in non-failure scenarios. The MFR and MFR+HAR algorithms provide the best improvement to the policy, but the benefits of MFR+HAR are only visible at 7-10% primary path loss.

## V. Failure Scenarios

We use two metrics to evaluate the retransmission schemes in failure scenarios: failure detection time and file transfer time.

### A. Failure Detection Time

Failure detection time is the time period from when a failure occurs to when the SCTP sender detects the failure. In the current RFC2960, each SCTP endpoint uses both implicit and explicit probes to dynamically determine 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 probe idle destinations. Each timeout (for data or heartbeats) on a

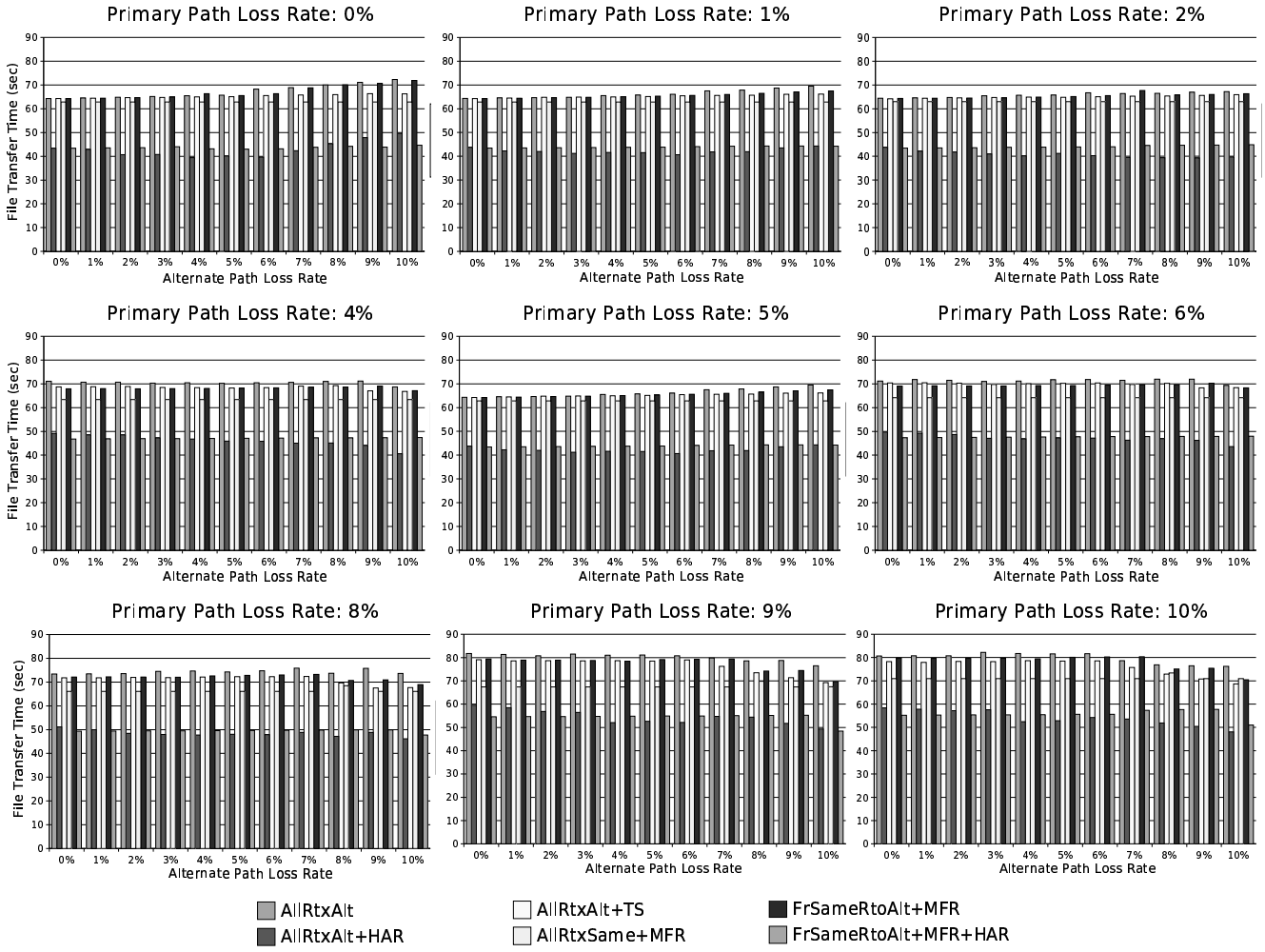


Fig. 4. Failure detection time for primary path loss rates 0-2%, 4-6%, and 8-10%

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 when its error count *exceeds* the *Path.Max.Retrans (PMR)* parameter. If the primary destination fails, the sender fails over to an alternate destination address and continues probing the primary destination with heartbeats. This alternate destination, however, does not become the new primary destination. The primary destination remains unchanged to allow a sender to resume sending new data to the primary destination if and when a future probe to the primary destination is successfully acked.

Our simulations use the parameter settings recommended in RFC2960: minimum RTO $= 1$ second, maximum RTO $= 60$ seconds, and PMR $= 5$. Using these defaults, the first timeout towards failure detection takes 1 second 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 $1 + 2 + 4 + 8 + 16 + 32 = 63$ seconds. In the worst case, the first timeout takes the maximum of 60 seconds, and the failure detection time takes $6 * 60 = 360$ seconds.

Figure 4 plots the average failure detection times for primary path loss rates of 0-2%, 4-6%, and 8-10% (the others were omitted due to space constraints). Again, 90% confidence intervals were measured, but are not shown. The results show that the HAR algorithm lowers the average failure detection time below the theoretical best case time. The HAR algorithm interferes with the exponential backoff procedure in the failure detection mechanism. Some RTO periods experience more than one timeout: one for data and one for a heartbeat. These RTO periods double count the errors for the failed destination, causing failure detection to occur in fewer RTO periods and sooner than the expected 63 seconds. Although faster failure detection is desirable, this algorithm does not correctly follow the conservative nature of the exponential backoff procedure.

AllRtxSame (not shown) and AllRtxSame+MFR are the only schemes able to achieve the theoretical best case detection time of 63 seconds at 0% primary path loss. For these schemes, the average failure detection time is independent of the alternate path loss rate, but increases as the primary path loss increases. The reason for the increase is due to the increased possibility of a timeout immediately before a failure occurs. In such a case, the failure detection may increase as follows. The timeout causes the primary destination's RTO to be doubled and the lost packet to be retransmitted to the primary destination. Then the ack for the retransmission clears the primary destination's error count, but does not provide an RTT measurement to reduce the RTO. If the failure occurs before an RTT measurement is obtained for the primary destination, then the six consecutive timeouts needed to detect failure will now take $2+4+8+16+32+60 = 122$ seconds!

On the other hand, the other schemes' failure detection times are influenced by the loss rates on both the primary and alternate paths. The primary path's influence on failure detection time is similar to that of AllRtxSame and AllRtxSame+MFR (explained above). Figure 4 shows that for AllRtxAlt, AllRtxAlt+TS, and FrSameRtoAlt+MFR the best case failure detection time is 64 seconds – only slightly longer than the theoretical minimum. This best case occurs when both the primary and alternate path loss rates are 0%. While the primary path's loss rate remains 0%, the average failure detection times reach as high as 72, 64, and 72 seconds for AllRtxAlt, AllRtxAlt+TS, and FrSameRtoAlt+MFR, respectively. Since new data may not be transmitted to the primary destination until all queued retransmissions on the alternate path have been sent, the failure detection times for these schemes depend on the quality of the alternate path. If the alternate path's loss rate is high, it will take more time to send the retransmissions, and thus will increase failure detection time.

The exception to this trend for AllRtxAlt, AllRtxAlt+TS, and FrSameRtoAlt+MFR is that the failure detection time actually decreases when both the primary and alternate path loss rates are high (see Figure 4's graph for 10% primary path loss). This anomaly is caused by the interaction of the two scenarios described above for dependency on primary and alternate path loss rates. The longer detection times at higher primary path loss rates are offset when losses on the alternate path causes a significant number of lost retransmissions to be re-retransmitted on the primary path. If these re-retransmissions get lost and timeout on the primary path, the primary destination's error count is incremented again. As a result, failure detection happens sooner.

Overall, AllRtxSame+MFR detects failure faster and more consistently than AllRtxAlt, AllRtxAlt+TS, and FrSameRtoAlt+MFR, but the drawback is that the sender does not successfully deliver any data until the entire failure detection process completes and failover occurs. In our simulations with 0% primary loss, the sender has 30 lost data packets outstanding when failure occurs. AllRtxAlt, AllRtxAlt+TS, and FrSameRtoAlt+MFR all successfully retransmit these 30 packets after the first timeout in the failure detection process, thus delaying them by only 1 second. On the other hand, AllRtxSame+MFR successfully retransmits the 30 packets after the failure detection completes, delaying them by at least 63 seconds!

### *B. File Transfer Time*

Figure 5 plots the transfer times for failure scenarios with primary path loss rates of 0-2%, 4-6%, and 8-10%. In these transfers, the sender transmits data to the primary for the first 4 seconds and then a failure occurs on the primary path. Eventually, the sender fails over to the alternate destination address, and resumes sending until the 4MB file transfer completes.

The results show that AllRtxAlt+HAR and FrSameRtoAlt+MFR+HAR provide the best throughput in failure scenarios, but this is due to the HAR algorithm. As explained in Section V-A, the HAR algorithm detects failure faster, but interferes with the conservative failure detection process.

AllRtxAlt and AllRtxAlt+TS perform the worst. Their poor performance is attributed to lower throughput during the non-failure portion of the transfer (see Figure 3) and longer failure detection times (see Figure 4).

As Figure 5 shows, AllRtxSame+MFR performs slightly better than FrSameRtoAlt+MFR. Note, however, that in our simulations, at least half of the 4MB file is left to be transferred after the failure occurs. With such a large amount remaining to be sent, plenty of time is left to close any gaps in performance. Considering AllRtxSame+MFR's lack of transfer progress during failure detection, we argue that FrSameRtoAlt+MFR would provide better performance if the transfer were closer to completion at the time of failure. Therefore, we conclude that FrSameRtoAlt+MFR provides the best overall performance during failure scenarios.

## VI. Conclusion and Future Work

We have concluded that FrSameRtoAlt+MFR and FrSameRtoAlt+MFR+HAR are the best performing retransmission schemes in non-failure scenarios, and FrSameRtoAlt+MFR is the best in failure scenarios. Therefore, we conclude that FrSameRtoAlt+MFR provides the best overall performance under the conditions evaluated.

Future work is to evaluate the retransmission schemes in network topologies that have different bandwidth-delay products on the primary and alternate paths. Multihoming with two interfaces per endpoint, as done in this paper, is a special case of mulithoming. The degree of multihoming should be increased beyond two per endpoint to ensure that the trends remain the same for $n$ interfaces per endpoint.

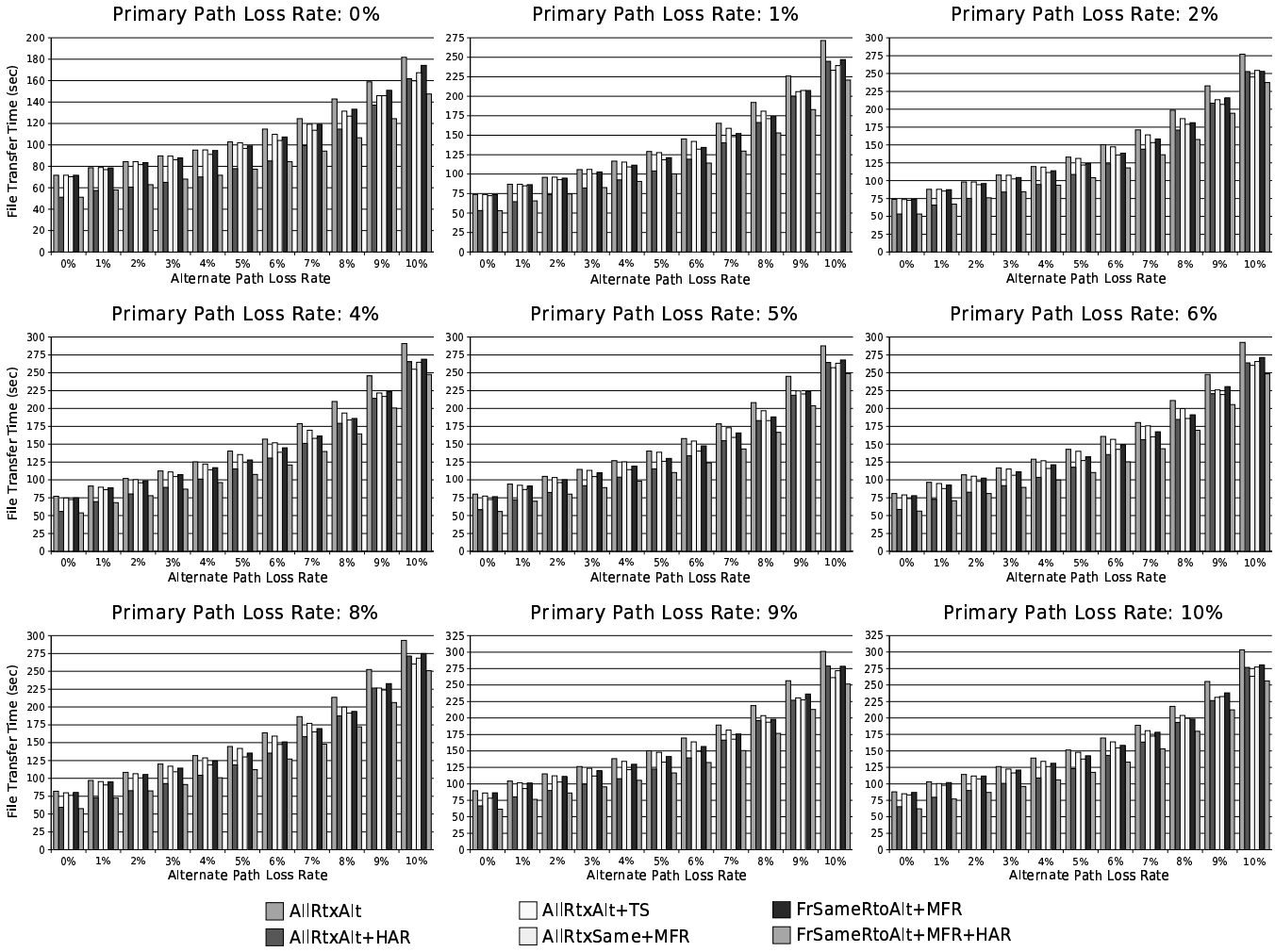


Fig. 5. File transfer time with failure for primary path loss rates 0-2%, 4-6%, and 8-10%


## ACKNOWLEDGEMENTS

The authors acknowledge Ryan Bickhart, Janardhan Iyengar, and Sourabh Ladha of University of Delaware's Protocol Engineering Lab 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] UC Berkeley, LBL, USC/ISI, and Xerox Parc. ns-2 documentation and software, Version 2.26, 2003. http://www.isi.edu/nsnam/ns.

[2] A. Caro. *End-to-End Fault Tolerance Using Transport Layer Multihoming*. PhD Dissertation, CISC Dept, University of Delaware. (in progress).

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

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

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

[6] 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.

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

[8] E. Kholer, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). draft-ietf-dccp-spec-07.txt, July 2004. (work in progress).

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

[10] 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.