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

Armando L. Caro Jr., Paul D. Amer
Protocol Engineering Lab
Computer and Information Sciences
University of Delaware
{acaro, amer}@cis.udel.edu

Randall R. Stewart
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 [2], 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) [7], [11] and the Datagram Congestion Control Protocol (DCCP) [9] 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.

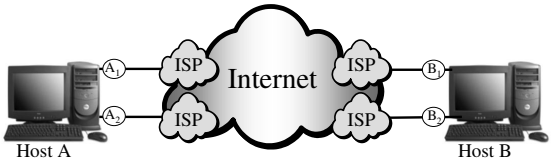

Fig. 1. Example multihoming topology

Currently, SCTP uses multihoming for redundancy purposes only and not for concurrent multipath transfer [8]. 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 [11] states in Section 6.4 that "when its peer is multihomed, an endpoint SHOULD try to retransmit [data] to

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.

This research supported, in part, by the University Research Program of Cisco Systems, Inc.

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 [10]. 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 policy in RFC2960 actually degrades performance in some circumstances [5]. 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 [4].

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 [5].

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 [5]. 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 [4].

III. Methodology

We evaluate the ten retransmission schemes described in Section II using University of Delaware's SCTP module [6] 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.

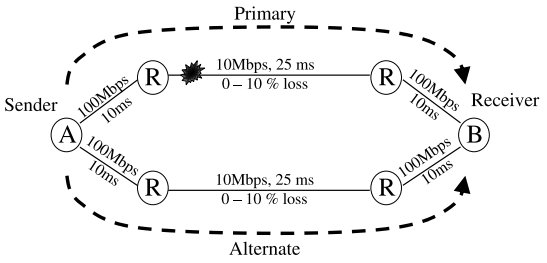

Fig. 2. Simulation network topology

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 [4], [5]. 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 completed 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 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 lossy conditions.

IV. Non-failure Scenarios

We collected results for 0-10% loss rates on the primary and alternate path. Due to space constraints in this paper, we do not include results for all the different primary path loss rates in non-failure scenarios. Since the retransmission policy is the most influential factor in a scheme's performance, we first compare the three policies without any of the algorithms from Section II-B. Then, we compare the results for all ten schemes.

A. Retransmission Policies Only

Figure 3 illustrates our results that have 1%, 5%, and 10% primary path loss rates. The alternate path's loss rate is varied on the x-axis, ranging from 0% to 10%. The graphs in Figure 3 compare the time to transfer a 4MB file using SCTP's current retransmission policy (AllRtxAlt) versus the other two policies, AllRtxSame and FrSameRtoAlt.

The graphs depict the mean file transfer time at varying alternate path loss rates for each retransmission scheme. 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; larger confidence intervals tend to be for higher loss rates.

Figure 3 shows that FrSameToAlt always does about the same or better than AllRtxSame. Also, AllRtxAlt only outperforms FrSameToAlt when the alternate path's loss rate is less than half of the primary's.

At low primary path loss rates, most of the losses are detected by fast retransmit. Hence, FrSameToAlt will send most of its lost packets to the same destination as AllRtxSame, thus experiencing similar results. Furthermore, they do not suffer, as AllRtxAlt does, from overly conservative RTOs for the alternate destination (see Section II-A).

As the loss rate on 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. In contrast, FrSameRtoAlt is able to alleviate severe loss conditions by sending timeout retransmissions to the alternate path. AllRtxAlt performs best when loss conditions on the primary are significantly worse than the alternate. The alternate path's loss rate must be fairly low to masque the influence of overly conservative RTOs.

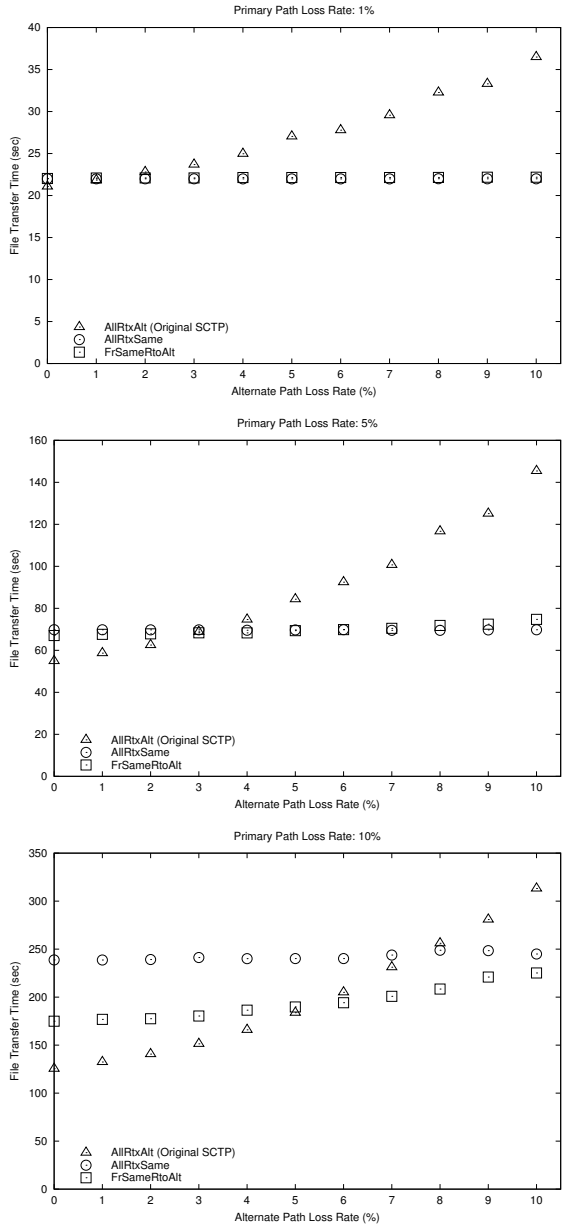

Fig. 3. 4MB file transfer time for (a) 1%, (b) 5%, and (c) 10% primary path loss with no failure

TABLE I

IMPROVEMENT FOR 5% PRIMARY PATH LOSS

Alt Path Loss	Policy	Algorithm: None	HAR	TS	MFR	MFR+ HAR
0%	AllRtxAlt	**54.9**	54.9	55.7	–	–
	AllRtxSame	69.8	–	–	**62.3**	–
	FrSameRtoAlt	67.1	67.5	68.1	**60.8**	61.6
1%	AllRtxAlt	58.7	57.6	**57.4**	–	–
	AllRtxSame	69.8	–	–	**62.3**	–
	FrSameRtoAlt	67.6	68.1	68.6	**61.4**	61.6
2%	AllRtxAlt	62.5	60.1	**58.8**	–	–
	AllRtxSame	69.7	–	–	**62.3**	–
	FrSameRtoAlt	67.9	68.6	68.9	**61.6**	61.8
3%	AllRtxAlt	68.8	61.9	**60.3**	–	–
	AllRtxSame	69.8	–	–	**62.4**	–
	FrSameRtoAlt	68.4	68.9	69.2	**61.6**	61.9
4%	AllRtxAlt	74.7	63.4	**61.5**	–	–
	AllRtxSame	69.6	–	–	**62.4**	–
	FrSameRtoAlt	68.3	69.3	69.3	**61.7**	62.0
5%	AllRtxAlt	84.4	65.4	**62.8**	–	–
	AllRtxSame	69.7	–	–	**62.4**	–
	FrSameRtoAlt	69.4	69.3	70.0	**61.8**	62.2
6%	AllRtxAlt	92.4	66.8	**64.2**	–	–
	AllRtxSame	69.8	–	–	**62.4**	–
	FrSameRtoAlt	69.9	69.6	69.7	**62.2**	62.2
7%	AllRtxAlt	100.7	71.6	**66.1**	–	–
	AllRtxSame	69.5	–	–	**62.3**	–
	FrSameRtoAlt	70.5	70.1	70.3	62.3	**62.0**
8%	AllRtxAlt	116.7	75.8	**68.6**	–	–
	AllRtxSame	69.5	–	–	**62.4**	–
	FrSameRtoAlt	71.9	70.3	70.6	62.2	**61.9**
9%	AllRtxAlt	125.1	78.5	**69.3**	–	–
	AllRtxSame	69.7	–	–	**62.5**	–
	FrSameRtoAlt	72.4	70.8	71.0	62.6	**62.3**
10%	AllRtxAlt	145.4	82.8	**70.7**	–	–
	AllRtxSame	69.8	–	–	**62.5**	–
	FrSameRtoAlt	74.7	70.8	71.4	62.8	**62.6**

B. All Retransmission Schemes

We now present the results of these policies when the three algorithms in Section II-B are introduced. Table I details the results for all ten schemes when the primary path's loss rate is 5%. The *None* column represents the graph shown in Figure 3b. For each alternate path loss rate and retransmission policy, the best performing scheme is in bold text (ties are broken by the simplist scheme).

As Table I shows, the algorithms generally improve performance significantly – enough to close most of the performance gaps between the three policies. Sometimes, however, introducing a retransmission algorithm degrades performance. For example, the packet overhead of TS sometimes outweighs the potential benefits for FrSameRtoAlt, but AllRtxAlt almost always benefits with TS.

Both HAR and TS succeed in addressing AllRtxAlt's issue with overly conservative RTOs on the alternate path. However, since TS provides more alternate path RTT measurements, AllRtxAlt+TS outperforms AllRtxAlt+HAR at higher alternate path loss rates.

On the other hand, HAR and TS only improve FrSameRtoAlt's performance at higher alternate path loss rates. Since FrSameRtoAlt sends much fewer retransmissions to the al-

ternate path, the alternate path loss rate must be high before overly conservative RTOs becomes an issue. MFR reduces the number of timeouts, and thus further reduces the number of retransmissions that FrSameRtoAlt sends on the alternate path. Hence, the alternate path's overly conservative RTO becomes less of an issue. As a result, MFR is the best algorithm for FrSameRtoAlt.

Lastly, MFR is the only algorithm which applies to AllRtxSame. By reducing the number of timeouts, MFR improves AllRtxSame's performance significantly.

At this point, we will not conclude which policy is best. They first need to be evaluated in failure scenarios. We will, however, conclude which is the best retransmission scheme for each policy in non-failure scenarios: AllRtxAlt+TS, AllRtxSame+MFR, and FrSameRtoAlt+MFR.

V. Failure Scenarios

Again, we collected results for 0-10% loss rates on the primary and alternate path, but due to space constraints, we only include failure scenario results for primary path loss rates 0%, 1%, and 5%. 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. To detect failure, SCTP uses a *Path.Max.Retrans* parameter. Each timeout (for data or heartbeats) on a particular destination increments an error count for that destination. The error count for a 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 *Path.Max.Retrans*. If a failed destination is the primary, the sender fails over to an alternate destination address.

All of our simulations use the parameter settings recommended in RFC2960 [11]: *RTO.Min* $= 1$ second, *RTO.Max* $= 60$ seconds, and *Path.Max.Retrans* $= 5$ attempts. Since our simulation topology has a 90ms RTT on both paths, we expect the primary path's RTO to be 1 second (assuming 0% loss) at the time a failure occurs. The exponential back-off procedure causes the RTO to be doubled on each timeout. Thus, we expect, in the best case, the six consecutive timeouts needed to detect failure to take $1+2+4+8+16+32 = 63$ seconds.

We omit results for schemes that do not employ one of the three retransmission algorithms, because Section IV shows that they are suboptimal for non-failures. We also omit results for the three schemes which use HAR, because the algorithm interferes with the exponential backoff procedure in the failure detection mechanism. Many 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 behavior can potentially cause spurious failovers. Furthermore, we can simply reduce *Path.Max.Retrans* to achieve similar behavior. Therefore, we present results for the remaining four schemes: AllRtxAlt+TS, AllRtxSame+MFR, FrSameRtoAlt+TS, and FrSameRtoAlt+MFR.

Figure 4 plots the average failure detection times for primary path loss rates of 0%, 1%, and 5%. Again, 90% confidence intervals were measured, but are not shown. The results show that the four retransmission schemes presented in Figure 4 are able to detect failure in *roughly* the same time at low primary path loss rates. However, some subtle differences do exist between them.

AllRtxSame+MFR is the only scheme able to achieve the expected failure detection time of 63 seconds at 0% primary path loss (see Figure 4a). As expected, AllRtxSame+MFR's failure detection is independent of the alternate path's loss rate. Also, as the primary path loss rate increases from 0% to 5%, AllRtxSame+MFR's failure detection time increases negligibly. On the other hand, the failure time increases more drastically (from 63 to 71 seconds) as the primary path loss rate increases from 5% to 10%.

The sudden drastic increase beginning at 5% primary loss 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 another 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!

The failure detection times for AllRtxAlt+TS, FrSameRtoAlt+FS, and FrSameRtoAlt+MFR increase with the loss rates on both the primary and alternate path. The cause for performance dependence on the primary path is similar to that of AllRtxSame+MFR, but the scenarios are slightly different. Due to space constraints, we do not describe these scenarios. Figure 4a shows that in the best case (0% loss on primary and alternate paths), failure is detected in 64 seconds – only slightly longer than expected. While the primary path's loss rate remains 0%, the average failure detection times reach as high as 66, 66, and 71 seconds for AllRtxAlt+TS, FrSameRtoAlt+TS, and FrSameRtoAlt+MFR, respectively. Since new data may not be transmitted to the primary destination until all queued retransmissions have been sent, the failure detection times for these schemes depend on the quality of the alternate path. If the alternate path's loss

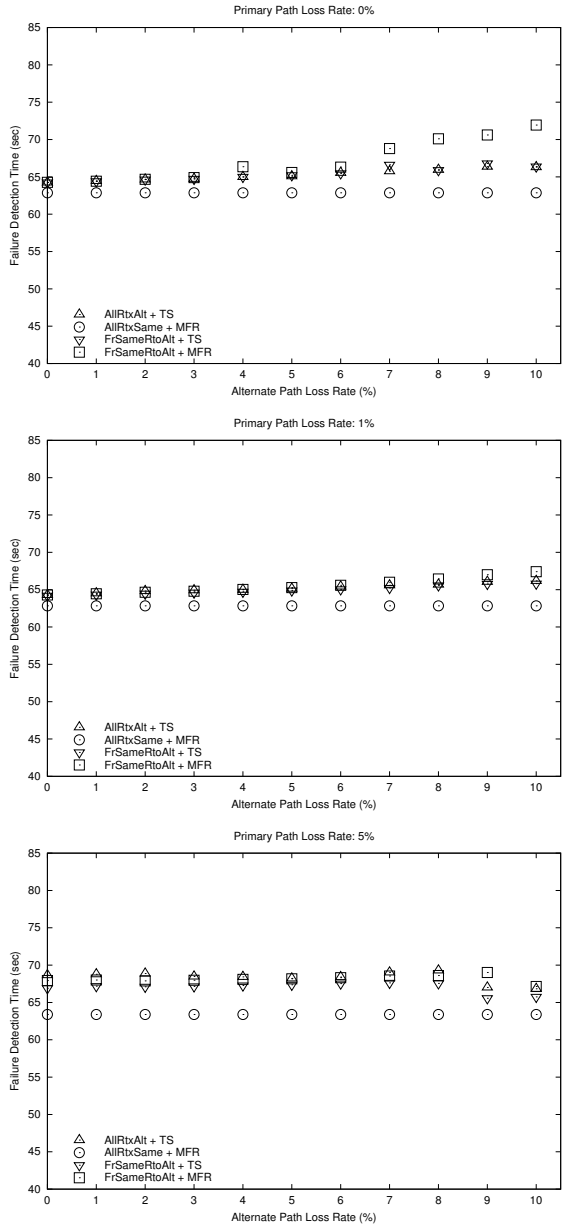

Fig. 4. Failure detection time for (a) 0%, (b) 1%, and (c) 5% primary path loss

rate is high, it will take more time to send the retransmissions.

The exception to this trend is that the failure detection time actually decreases when both the primary and alternate path loss rates are high (see Figure 4c). This anomaly is caused by the interaction of the two scenarios described above for dependency on primary and alternate path loss rates. The higher 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 to the primary destination. These re-retransmissions get lost and timeout, causing the primary destination's error count to be incremented sooner than if the retransmissions on the alternate path were not lost.

When the primary path loss rate is low and the alternate path loss rate is high, FrSameRtoAlt+MFR does worse than AllRtxAlt+TS and FrSameRtoAlt+TS. These loss conditions cause some ambiguous situations that TS is able to resolve. Without resolving the ambiguities, the primary destination's error count may be cleared without an RTT measurement, causing the failure detection time to increase.[1] This behavior occurs more often at 0% primary path loss instead of 1%, because the cwnd is able to grow larger up to the time of failure. Therefore, with a greater number of outstanding packets at the time of failure, the alternate path has a higher chance of losing a packet and causing the ambiguous behavior.

Although AllRtxSame+MFR detects failure faster and more consistently than the other three schemes, there is a tradeoff between them. A drawback to AllRtxSame+MFR is that the sender will not successfully deliver any data until the entire failure detection process has completed, and the sender fails over to an alternate destination. In our simulations with 0% primary loss, the sender has 30 lost data packets outstanding when failure occurs. AllRtxAlt+TS, FrSameRtoAlt+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 and delays them by 63 seconds!

B. File Transfer Time

Figure 5 plots the transfer times for failure scenarios with primary path loss rates of 0%, 1%, and 5%. 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 all of the schemes provide about the same transfer time with failure. Note, however, that in our simulations, at least half of the 4MB file is left to be transferred after the failure. With such a large amount remaining to be sent, plenty of time is left to close any gaps in performance. Also, once failover occurs, there is only one active destination address, which eliminates the difference between the three retransmission policies. However, the results may differ for multihomed hosts with more than two interfaces.

As discussed in Section V-A, AllRtxSame+MFR delays the outstanding packets until after the failure detection completes.

[1]RFC2960 allows implementations to choose how to handle ambiguous situations during failure detection. Other implementations may not experience this effect, but handling ambiguous situations differently may cause other side-effects unknown to the authors.

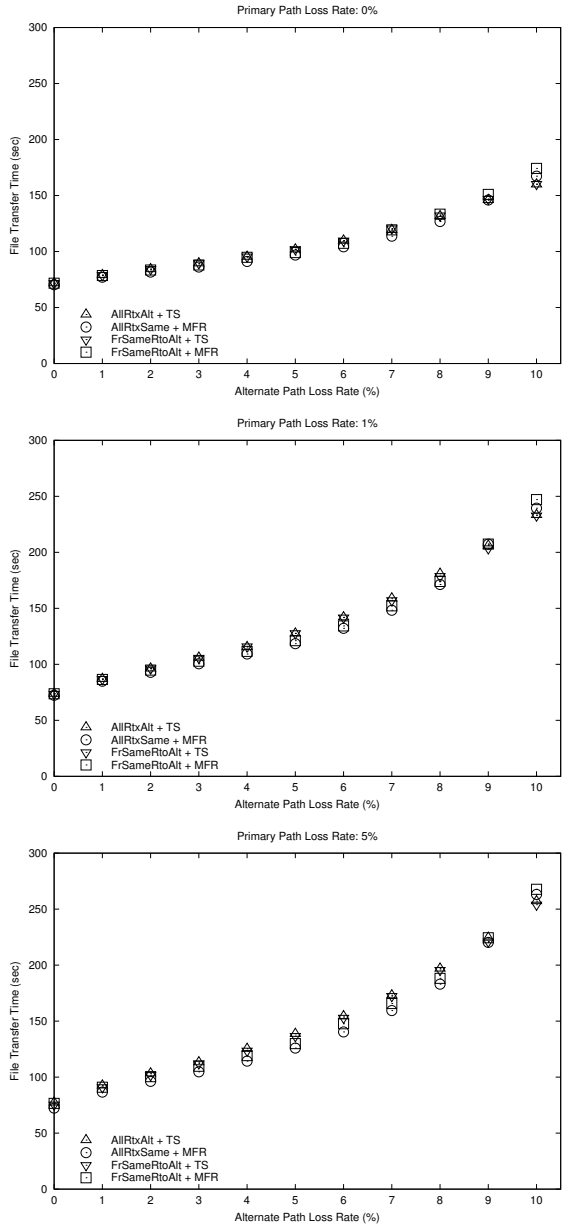

Fig. 5. 4MB file transfer time for (a) 0%, (b) 1%, and (c) 5% primary path loss with failure

Thus, the throughput of small transfers will suffer significantly with AllRtxSame+MFR.

VI. Conclusion and Future Work

Our results show that AllRtxAlt+TS and FrSameRtoAlt+MFR perform the best overall in non-failure and failure scenarios. Of these two schemes, we argue that FrSameRtoAlt+MFR is the scheme that should be adopted. FrSameRtoAlt+MFR minimizes the number of timeouts, which signficantly benefits small transfers. Also, FrSameRtoAlt+MFR's fairly consistent performance across different loss rates is beneficial when the sender does not have prior knowledge about the paths' characteristics.

However, a strong conclusion cannot be made without further investigation. These retransmission schemes should be evaluated with small transfers and under uni-directional failures. Network topologies that have different primary and alternate path bandwidth-delay products should be evaluated. These schemes should also be tested for resilience to spurious failovers with more aggressive failover thresholds. Finally, the degree of multihoming should be increased beyond two per endpoint to ensure that the trends remain the same.

Acknowledgements

The authors acknowledge Janardhan Iyengar, Sourabh Ladha, and Ryan Bickhart 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] R. Braden. Requirements for Internet hosts–communication layers. RFC1122, Internet Engineering Task Force (IETF), October 1989.

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

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

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

[6] A. Caro and J. Iyengar. ns-2 SCTP module, Version 3.4, August 2003. http://pel.cis.udel.edu.

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

[8] J. Iyengar, K. Shah, P. Amer, and R. Stewart. Concurrent Multipath Transport Using SCTP Multihoming. Tech Report TR2004-02, CIS Dept, University of Delaware, September 2003.

[9] E. Kholer, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). draft-ietf-dccp-spec-06.txt, Internet Draft (work in progress), Internet Engineering Task Force (IETF), February 2004.

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

[11] 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, Internet Engineering Task Force (IETF), October 2000.