END-TO-END FAULT TOLERANCE
USING TRANSPORT LAYER MULTIHOMING

by

Armando L. Caro, Jr.

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer and Information Sciences

Summer 2005

END-TO-END FAULT TOLERANCE
USING TRANSPORT LAYER MULTIHOMING

by

Armando L. Caro, Jr.

Approved: _____
Henry Glyde, Ph.D.
Interim Chair of the Department of Computer and Information Sciences

Approved: _____
Conrado M. Gempesaw II, Ph.D.
Interim Dean of the College of Arts and Sciences

Approved: _____
Conrado M. Gempesaw II, Ph.D.
Vice Provost for Academic and International Programs

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Paul D. Amer, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Adarshpal S. Sethi, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Phillip T. Conrad, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Stephan Bohacek, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Randall R. Stewart
Member of dissertation committee

ACKNOWLEDGEMENTS

I have to first thank my advisor, Prof. Paul Amer, for being an exceptional advisor. His "hands off" approach to mentoring made the beginning stages of my dissertation an uphill battle, but I have since realized that the benefits do pay off in the end. He never seizes to amaze me with his incredible patience and attention to detail. Even when I began to grow tired of revising the same text over and over, he always showed enthusiasm for helping me improve its presentation style and pointing out the important contributions. I cannot even imagine how I could have completed this dissertation without his support.

I am grateful to my dissertation committee: Prof. Adarshpal Sethi, Prof. Phillip Conrad, Prof. Stephan Bohacek, and especially Randall Stewart for the many questions, suggestions, and insights that were invaluable in forming my dissertation. Randall was generous enough to participate on my committee in spite of the distance between us.

I would like to also thank my colleagues at the Protocol Engineering Lab: Phill Conrad and Sami Iren from my early days, and more recently, Len Armstrong, Ryan Bickhart, Jerry Heinz, Mark Hufe, Jana Iyengar, Sourabh Ladha, Preethi Natarajan, and Keyur Shah, for their valuable suggestions, discussions, and last minute proofreads before a paper deadline. I would like to especially thank Phill Conrad, Jana Iyengar, and Sourabh Ladha. When I joined the lab (*many* years ago) as an undergraduate researcher with only very basic coding skills, Phill took me under his wing and taught me many of the hands-on skills needed when doing research. Sourabh's time in the lab felt like a whirlwind, but remains very memorable. He managed to spend only one and a half years in the lab, yet still rival us in the number of projects and publications. He always kept us on our toes, and I will always admire his ability to tackle so many tasks in parallel. From day one,

Jana brought to the lab such incredible enthusiasm and passion for doing good research that it became contagious. I can honestly say that he, without knowing it, somehow made me realize *why* I actually wanted to get a PhD. I definitely benefitted professionally and personally from all the *many* long discussions we had over the last four years.

I would like to acknowledge the financial support that made this dissertation possible. This research was supported in part by the University Research Program of Cisco Systems, Inc., and by the U.S. Army Research Laboratory under two programs: (1) Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002, and (2) Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. I thank Prof. Kevin Almeroth of UC in Santa Barbara and the IETF budget for funding my trips to IETF meetings. I also thank John Loughney and the Nokia Research Center for funding me for a summer in Helsinki, Finland to work on the SCTP ns-2 module.

My family and friends have offered more support and encouragement than anyone could ever expect. Over the years, they have learned to recognize my disappearing act as a sign of an upcoming deadline. I thank them for understanding and letting me bounce back into their lives when I had more time on my hands. I would like to give a special thanks to my girlfriend, Roxana García, for putting up with my irregular working habits and occasional workaholic tendencies. I am also grateful to her for showing genuine interest in my field whenever I rambled on about techie stuff, even though she thinks I'm a geek for reading "News for Nerds".

I owe my deepest debt of gratitude to my parents, Armando and Ruby Caro, for providing me with the resources to succeed in life. I thank my mom for teaching me that time management, organization, and diligence are the keys to success. I thank my dad for turning me on to computers by buying our first computer and teaching me to code in BASIC when I was seven. He showed me a simple three-line program to display my name blinking in the middle of the screen, and I was hooked!

Finally, I would like to thank "the academy"... ☺

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This dissertation investigates the use of *transport layer multihoming* for providing end-to-end network fault tolerance and improved application performance. Transport layer multihoming is a feature that binds a single transport layer association to multiple network addresses at each endpoint, thus allowing the two end hosts to communicate over multiple network paths. Such path redundancy is useful for fault tolerance in that traffic of existing connections can be redirected (i.e., failover) to a peer's alternate network address without the need for applications (or users) to abort and re-establish connections. Considering the prevalence of path outages in the Internet today, multihoming support at the transport layer can improve resilience of established connections.

Using the Stream Control Transmission Protocol (SCTP), we investigate possible design decisions of a multihomed transport protocol, and provide insight for future transport protocols that support multihoming. In particular, we investigate retransmission policies and failover mechanisms in two contexts: proactive (for fixed infrastructure networks), and reactive routing (for mobile ad-hoc networks) protocols. Retransmission policies control the behavior when a transport sender fails to receive acks for sent data. Failover mechanisms determine under which conditions a path is presumed failed, when a sender migrates to a new path, and if/when a sender resumes new data transmission on the original path. We provide a decision tree to suggest a retransmission policy and failover mechanism based on expected network conditions.

For topologies with proactive routing and *roughly* symmetric path delays, our results show that the best retransmission policy is a hybrid retransmission policy introduced by this author: (a) send fast retransmissions to the primary destination address, and (b)

send timeout retransmissions to an alternate destination address. We formally specify and evaluate variations of SCTP's current failover mechanism to show that using the most aggressive failover threshold performs best for these topologies. As expected, an agressive failover threshold does trigger additional spurious failovers; however, we show evidence that with the hybrid retransmission policy, spurious failovers do not degrade performance, and counter-intuitively often improve goodput, regardless of the paths' characteristics. These results also hold for reactive routing environments with either short route discovery delays or large route cache lifetimes.

In reactive routing environments with short route cache lifetimes and long route discovery delays, or when path delays are highly asymmetric, our results show a *slightly* more conservative approach performing best: (a) send all retransmissions to the primary destination address, and (b) failover after *two* consecutive timeouts.

We also investigate permanent failovers, a new concept introduced by this author and not existing in current transport layer multihoming protocols. We find permanent failovers beneficial when a sender can estimate each path's RTT and loss rate.

To evaluate these design decisions, this author co-developed an SCTP module for the ns-2 network simulator. This module is now an official component of ns-2's distribution (version 2.27), and is widely used by the SCTP research community.

Our results have uncovered an important design principle for multihomed transport protocols: traditional conservative failover techniques used in routing do not apply when path redundancy begins at the end hosts and is handled by the transport layer. Since failovers at the routing layer are transparent to the transport layer, the failover thresholds must be conservative to avoid oscillations that could cause the transport layer to maintain inaccurate path metrics (RTT, cwnd, ssthresh). On the other hand, a multihomed transport layer is completely aware of failover events and is able to maintain separate metrics per path. As a result, transport layer multihoming can improve performance by providing aggressive failovers that reduce stalls during network congestion and failure events.

Chapter 1

INTRODUCTION

1.1 Problem Statement

This dissertation focuses on transport layer techniques that exploit host multihoming to provide end-to-end fault tolerance and improved application performance. Generally, mission critical systems rely on redundancy at multiple levels to provide uninterrupted service during resource failures. Following the redundancy approach, mission critical systems can multihome their hosts to improve availability. A host is multihomed if it can be addressed by multiple IP addresses [23]. Redundancy at the network layer allows a host to be accessible even if one of its IP addresses becomes unreachable for an extended period of time (assuming the paths to the multiple interfaces do not share the same failed link). Transport layers that support multihoming allow traffic of existing connections to be redirected to a peer's alternate IP address without the need for applications (or users) to abort and re-establish connections. Considering the prevalence of path outages on the Internet today, multihoming support at the transport layer can improve resilience of established connections, and thus improve application performance. While fault tolerance can be addressed at other layers, we argue that the transport layer is in the best position to detect failure and make end-to-end failover decisions. After all, the transport layer is the lowest layer responsible for both end-to-end quality of service and having knowledge about end-to-end path characteristics.

Wide spread use of multihoming was infeasible during the early days of the Internet due to cost constraints; today, network interfaces have become commodity items.

Cheaper network interfaces and cheaper Internet access motivate content providers to have simultaneous connectivity through multiple ISPs, and more home users are installing wired and wireless connections for added flexibility and fault tolerance. Furthermore, wireless devices are being simultaneously connected through multiple access technologies, such as wireless LANs (e.g., 802.11) and cellular networks (e.g., GPRS, CDMA).

1.2 Motivation for Multihoming

We begin with an argument for why host multihoming is important today and is expected to further increase in importance in the short and long terms. Today, content providers and server operators on the Internet face a significant challenge of ensuring that services are readily available and highly reliable. Servers on the Internet today are recognized as being much less reliable than mission-critical services provided today, such as the telephone network's "five 9's" expected level of reliable operation. Much of the gap in reliability is attributed to network availability and host reachability; unfortunately, network path outages are not rare on the Internet. Sections 1.2.1 and 1.2.2 present two causes of path outages on the Internet: link failures and overloaded links.

1.2.1 Link Failures

The Internet's backbone routing is designed to be robust against link failures. Link failures occur when a router or a link connecting two routers fails due to link disconnection, hardware malfunction, or software error. The routing system has the responsibility to detect such failures, and in response, reconfigure routing tables to bypass the failure.

One problem with routing recovery is that the Internet's backbone routing, which is based on Border Gateway Protocol (BGP) [94, 95], has been optimized for scalability and simplicity. As a result, BGP can take a long time to converge on a new route after a link failure is detected. Labovitz et al. [72] show that the Internet's interdomain routers may take tens of minutes to converge after a failure. They find that during these "delayed convergences," end-to-end Internet paths experience intermittent loss of connectivity in

addition to increased packet loss, latency, and reordering. Although they demonstrate that much of the convergence delays are due to BGP specification ambiguity and specific router vendor implementation decisions, long convergence times are inherent to the path vector routing protocol. Their analysis shows that in the best case, convergence times will grow linearly with the addition of new autonomous systems; in the worst case, the growth is exponential.

Paxson [91] uses probes to find that "significant routing pathologies" prevent selected pairs of hosts from communicating about 1.5% to 3.3% of the time. Importantly, he also finds that this trend has not improved with time. Labovitz et al. [73] examine routing table logs of Internet backbones to find that 10% of all considered routes were available less than 95% of the time, and more than 65% of all routes were available less than 99.99% of the time. They also find that the duration of path outages are heavy-tailed, and about 40% of path outages take more than 30 minutes to repair. Chandra et al. [37] use probes to confirm that failure durations are heavy-tailed, and report that 5% of detected failures last more than 2.75 hours, and as long as 27.75 hours.

1.2.2 Overloaded Links

Another problem with BGP is that it is not designed to detect performance failures due to overloaded network links. Flash crowds and denial-of-service (DoS) attacks are two situations that can overload a path to a host. Such scenarios drastically degrade the end-to-end communication between peer hosts, but BGP is unaware of such performance failures. Even if alternate paths exist, BGP will not use them to route traffic around overloaded links.

A flash crowd is a sudden, large surge of legitimate traffic to a particular site. Flash crowds resulted from the Ken Starr Report on September 11, 1998, Victoria's Secret Fashion Show webcast on May 18, 2000, and news reports of the terrorist attacks of September 11, 2001. Also, the *Slashdot Effect* is a commonly experienced phenomenon

that occurs when a high volume news web site (www.slashdot.org) posts a web site link, and triggers a spontaneous high hit rate upon the server of that link.

A DoS attack floods a network with useless traffic to overwhelm the network and/or system resources of the victim host. Moore et al. [86] study three, one-week datasets to find that DoS activity on the Internet is widespread, and distributed among many different domains and ISPs. During the three one-week periods, they observe more than 12,800 attacks against more than 5,000 distinct targets belonging to 2,000 distinct DNS domains. They report attack rates and durations that would disrupt most (if not all) communication to the victim IP addresses. They report that 50% of attacks are less than 10 minutes in duration, 80% are less than 30 minutes, and 90% less than an hour. At the tail of the distribution, 2% of attacks are greater than 5 hours, 1% are greater than 10 hours, and dozens last multiple days. They also find that the range of targets is surprisingly more encompassing than expected. Targets include well known sites, such as Amazon and Hotmail, but a significant fraction of attacks is directed against small and medium sized businesses, dialup and broadband home machines, and network infrastructure, such as name servers and routers.

These statistics are not promising for mission critical systems which require high availability. Many commercial services advertise 99.99% or 99.999% (known as "four 9's" or "five 9's) server availability, but highly available servers do not guarantee highly available services for their clients. The end-to-end paths from the clients to the server must also be highly available. A system which advertises 99.999% server availability expects to have a down time of at most 5 minutes a year. However, a typical client may find the server available much less often; even 99% availability translates to approximately 15 minutes of down time a day.

Previous research has attempted to improve host availability on the Internet by various means, such as server replication [1, 43, 46, 49, 109, 114], site multihoming [2–6, 8], or overlay routing networks [12, 13, 54, 113]. Each of these approaches can be effective,

but they have limitations (see Section 6.1). Server replication's costly infrastructure limits this solution to high-end Web sites that can afford the expense. Site multihoming (provisioning a site with multiple ISP links) protects against a single access link failure, but it cannot avoid the long convergence times of BGP within the Internet. Overlay routing networks require extra infrastructure within the network that does not scale well beyond a small set of nodes. Furthermore, each of these approaches are unable to route around last-hop failures, which Gummadi et al. [54] show to occur in 16% of path failures to servers and 60% of path failures to broadband hosts. Host multihoming, on the other hand, can route around last-hop failures by allowing a host to be accessible even if one of its IP addresses becomes unreachable.

1.3 Transport Layer Multihoming

Transport layer multihoming is a feature that binds a single transport layer connection to multiple network addresses at each endpoint. This transport protocol feature is not a new concept; it is actually an old concept disguised under a new name. Historically, transport layer multihoming was referred to as splitting/recombining or downward-multiplexing, and was for providing added resilience against network failure and/or potentially increasing throughput [60, 106, 110]. Although transport layer multihoming is an old concept, neither of the Internet's current transport protocol workhorses, TCP or UDP, support multihoming. UDP's connectionless nature is incompatible with transport layer multihoming, and TCP allows applications to bind to only one network address at each end of a connection. Furthermore, we are unaware of any other historical transport protocols that support multihoming [60]. Network interfaces were expensive components in the early days of the Internet, which meant that transport layer multihoming was beyond the ken of research.

Now that network interfaces have become commodity items, and multiple ISP access (e.g., dialup/broadband home connection, WiFi hotspots, and cellular provider) is

affordable, transport layer multihoming has become a feature worth supporting and investigating. Two recent IETF transport layer protocols, the Stream Control Transmission Protocol (SCTP) [35, 104, 105] and the Datagram Congestion Control Protocol (DCCP) [71] support multihoming at the transport layer. DCCP is currently in its formative stages, and transport layer mobility is its driving motivation for multihoming support. SCTP, on the other hand, is more mature (RFC2960 was published in October 2000), and supports multihoming primarily for network fault tolerance.[1] Therefore, we use SCTP for our experiments, but we believe the results and conclusions presented in this dissertation apply in general to reliable transport protocols that support multihoming.

SCTP was originally developed to carry telephony signaling messages over IP networks. With continued work, SCTP evolved into a general purpose transport protocol that includes advanced delivery options. SCTP is a message-oriented protocol that provides a reliable, full-duplex connection, called an *association*. Its key features are multihoming, multistreaming, and extra protection against SYN-flooding attacks, blind masquerade attacks, and stale SCTP packets from previous associations. *Multistreaming* allows for independent delivery among streams, and reduces the risk of head-of-line blocking. SCTP resists SYN-flooding attacks by using a four-way handshake that verifies the legitimacy of an association initialization request before allocating server-side resources. Blind masquerade attacks and stale packets are avoided by using a 32-bit verification tag to validate the sender of an SCTP packet. Refer to Table 1.1 for a complete list of SCTP's services/features, compared and contrasted with TCP and UDP.

To explain SCTP's multihoming feature, we contrast SCTP and TCP in a multihomed topology (see Figure 1.1). Since TCP can only bind to a single IP address at each endpoint, four distinct TCP connections are possible between Hosts A and B: (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2). SCTP's binding, on the other hand, is not limited to a single

[1] Transport layer mobility is a special case of network fault tolerance that SCTP supports with some proposed extensions [96, 103], but mobility is not the focus of this dissertation.

Services / Features	SCTP	TCP	UDP
Connection-oriented	yes	yes	no
Full duplex	yes	yes	yes
Reliable data transfer	yes	yes	no
Partial-reliable data transfer	optional	no	no
Flow control	yes	yes	no
TCP-friendly congestion control	yes	yes	no
ECN capable	yes	yes	no
Ordered data delivery	yes	yes	no
Unordered data delivery	yes	no	yes
Uses selective ACKs	yes	optional	no
Preservation of Application PDU Boundaries	yes	no	yes
Application PDU fragmentation	yes	yes	no
Application PDU bundling	yes	yes	no
Path MTU discovery	yes	yes	no
Multistreaming	yes	no	no
Multihoming	yes	no	no
Protection against SYN flooding attack	yes	no	n/a
Allows half-closed connections	no	yes	n/a
Reachability check	yes	yes	no
Pseudo-header for checksum	no (uses vtags)	yes	yes
Time wait state	for vtags	for 4-tuple	n/a

Table 1.1: Compare and contrast SCTP, TCP, UDP

IP address on each end. Instead, a single SCTP association may consist of all IP addresses, which in our example would be: $(\{A_1, A_2\}, \{B_1, B_2\})$.[2] Currently, SCTP uses multihoming for redundancy purposes only and not for concurrent multipath transfers [63–65]. Each endpoint chooses a single destination address as the primary destination address, which is used for all data traffic during "normal transmission". Note that a single port number is used at each endpoint regardless of the number of IP addresses.

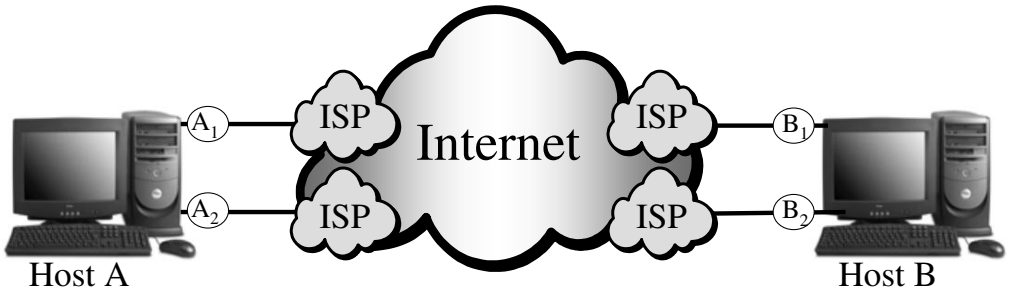

Figure 1.1: Example multihomed topology

Now suppose that there exists a TCP connection, (A_1, B_1), and an SCTP association, $(\{A_1, A_2\}, \{B_1, B_2\})$, with A_1 and B_1 as the primary destinations. If B_1 becomes unreachable, TCP and SCTP behave quite differently. Since TCP does not support multihoming, the TCP connection awaits a predefined maximum number of retransmission timeouts and then aborts; thus forcing the application layer (or user) to recover. This "wait, abort, and reconnect" behavior may be unacceptable for mission critical applications. An SCTP association, however, remains alive even when B_1 becomes unreachable. SCTP's built-in failure detection and recovery system, known as *failover*, allows endpoints to dynamically send traffic to an alternate peer IP address when needed. Hence, in this example, the SCTP association temporarily redirects traffic to B_2 until B_1 becomes reachable again.

[2] Although less useful for network failure recovery, an SCTP endpoint may bind to a proper subset of its available IP addresses.

1.4 Research Overview

Transport layer multihoming is an old concept finally making its way into standardized transport protocols, but it remains a feature left mostly unexplored. With multiple paths at its disposal, a sending transport layer has several actions that can be taken to handle normal data transmission, loss recovery, path failure detection, failover, and path recovery. The authors of SCTP designed the fault tolerance mechanisms based on intuition and previous experience in single homed networking, but much of the multihoming functionality does not have research data to justify its design. This dissertation investigates and challenges some of the design decisions of SCTP to determine if their predicted benefits are indeed realized. This research provides insight for future transport protocols that support multihoming.

Without complete knowledge about the network, a sender is faced with the dilemma of blindly deciding which path to use for loss recovery. A sender has no way of knowing if lost data is attributed to noise, minor transient congestion, major long term congestion, or path failure. Furthermore, endpoints are generally ignorant of the bandwidth-delay product of the available paths to their peer. Instead of gambling on a case-by-case basis, SCTP uses a fixed retransmission policy that attempts to maximize long term benefits. We challenge SCTP's retransmission policy by investigating a few policies for failure and non-failure scenarios. The results of this study are available in Chapter 2 of this dissertation.

Path failure detection time is important for swift failovers that minimize stall time of transfers. Failure detection time can be improved by lowering the failover threshold, but doing so increases the number of false alarms (i.e., spurious failovers). We formally specify and evaluate a few variations of SCTP's failover mechanism to investigate the tradeoff between more aggressive failover and more frequent spurious failovers to determine appropriate failover thresholds for fast failovers that do not degrade goodput performance of transfers. The results of this study are available in Chapter 3 of this dissertation.

We also challenge SCTP's failover mechanism for its temporary nature. As currently specified in RFC2960, failovers are never permanent; if the primary path recovers, a sender resumes sending new data to the primary destination. When changing destinations, a sender throttles its sending rate back to slow start to regain its ack clock for the new path. Consequently, the transfer is actually penalized for a path recovering. We formally specify and investigate extending the current state-of-the-art in multihoming to support *permanent failover* for avoiding such penalization. The results of this study are also available in Chapter 3 of this dissertation.

As wireless devices become more prevalent, more mesh and infrastructure-less networks are emerging. In particular, mobile ad-hoc networks (MANETs) are a promising future direction in networking. The cooperative, yet mobile and volatile nature of nodes in MANETs make transport layer multihoming appealing. However, some MANET routing protocols (e.g., AODV, DSR) are reactive (instead of proactive as in traditional fixed infrastructure networks). Taking into account the interaction of transport and network layer, reactive routing can alter the results and conclusions of our investigations of retransmission policies and failover thresholds. Therefore, we reevaluate our results in the context of reactive routing protocols. The results of this study are available in Chapter 4 of this dissertation.

To perform the simulation work used to evaluate these design decisions, we needed a network simulator that supported SCTP. Thus, we developed an SCTP module for the ns-2 network simulator. This module is now an official component of ns-2, and is used significantly by the SCTP research community. Details about the module are available in Chapter 5 of this dissertation.

Chapter 6 concludes this dissertation. In this chapter, we first present a summary of related work that addresses end-to-end network fault tolerance. These approaches include dynamic host routing, site multihoming, server replication, overlay routing networks, and interrupted connection re-establishment. Then, we present a summary of key results in

this thesis. Finally, we end with suggestions for future study.

Figure 1.2 shows a structural outline of the dissertation and the author's related publications for each topic.

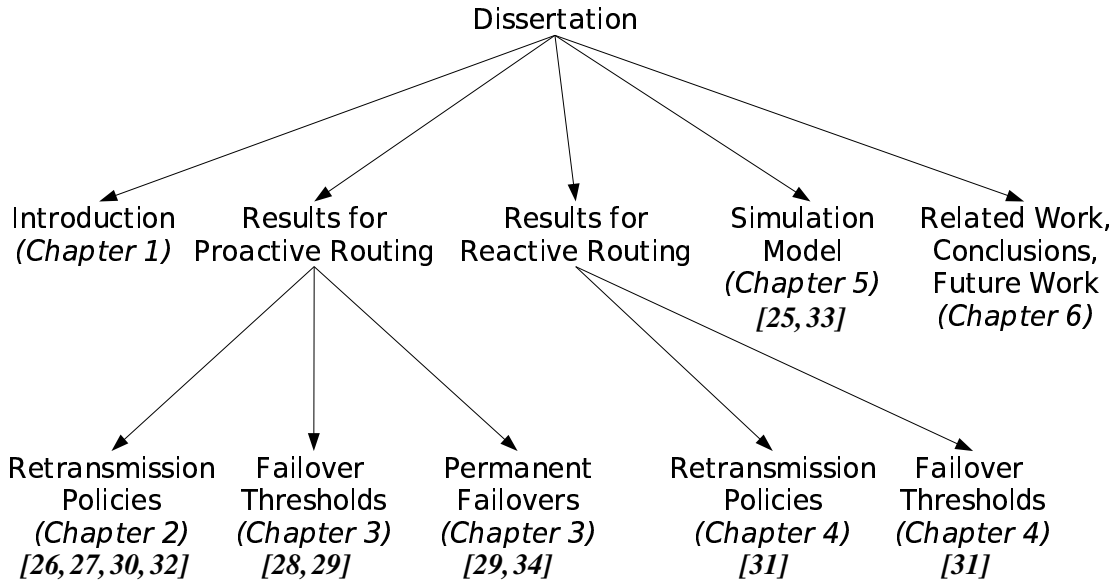

Figure 1.2: Dissertation structure

Chapter 2

RETRANSMISSION POLICIES

2.1 Introduction

Currently, an SCTP sender uses an alternate destination address for retransmissions when data sent to the primary destination is lost. SCTP's current retransmission policy [105] 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." According to the authors of SCTP, this policy, which we refer to as *AllRtxAlt (All Retransmissions to Alternate)* attempts to improve the chance of success by sending retransmissions to alternate destinations [104]. The 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 will more likely avoid a repeated loss of the same data.

We show that this policy actually degrades performance in many circumstances. We explore two alternative retransmission policies that were introduced by this author and find that the best policy, for both failure and non-failure scenarios, is a hybrid policy: send (a) fast retransmissions to the primary destination, and (b) timeout retransmissions to an alternate destination.[1] To this author's best knowledge, no transport protocol (necessarily multihomed) has ever explored such a hybrid policy. We show that this hybrid policy performs best when combined with two enhancements: a Multiple Fast Retransmit algorithm (also introduced by this author), and either timestamps or our Heartbeat After RTO

[1] The hybrid policy was inspired by a conversation with a student from CCNY about my work on retransmission policies.

mechanism (suggested by Randall Stewart). 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 to increase a sender's accuracy of its RTT estimates and therefore RTO values.

Section 2.2 demonstrates a problem with SCTP's current retransmission policy (AllRtxAlt) by comparing it to an alternative policy, *AllRtxSame (All Retransmissions to Same)*. Section 2.3 introduces and evaluates our hybrid policy, *FrSameRtoAlt (Fast Retransmissions to Same, Timeouts to Alternate)*, which combines the good points of AllRtxAlt and AllRtxSame. Section 2.4 introduces and evaluates three extensions to further improve the performance of the three policies. Section 2.5 compares the policies' performance with their best extensions in non-failure scenarios, and Section 2.6 compares them in failure scenarios. Section 2.7 concludes this chapter.

2.2 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, the author of this policy expected 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, one would 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.

2.2.1 Analysis Methodology

We evaluate the retransmission policies using University of Delaware's SCTP module [33] for the ns-2 network simulator [19]. This module is further discussed in

Chapter 5. Figure 2.1 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, which is a reasonable setting in practice.

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, each exhibiting ON/OFF patterns with ON-periods and OFF-periods drawn from 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 [7,41]. 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 [75]. We chose this model for simulating self-similar cross-traffic based on results by Willinger et al. [111], which show that self-similar traffic can be modeled as an aggregation of ON/OFF sources with durations drawn from distributions with heavy tails (e.g., Pareto).

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

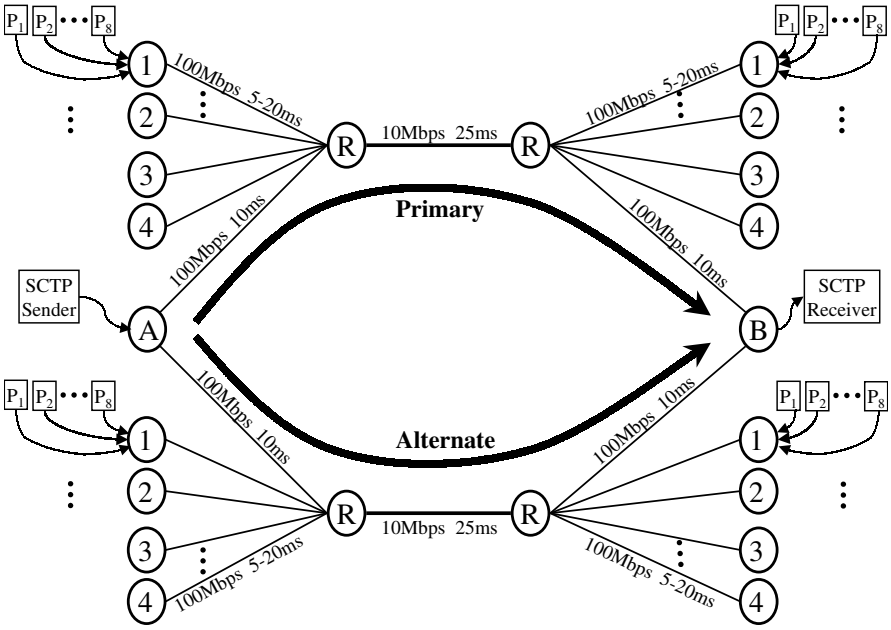

Figure 2.1: Simulation network topology with cross-traffic, congestion-based loss, and no failures

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

2.2.2 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 set various levels of cross-traffic and 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 by the routers (labelled R) divided by the number of SCTP packets transmitted by sender A.

We collected results for 0-10% loss on the primary and alternate paths, but for readability we do not include all results. Figure 2.2 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

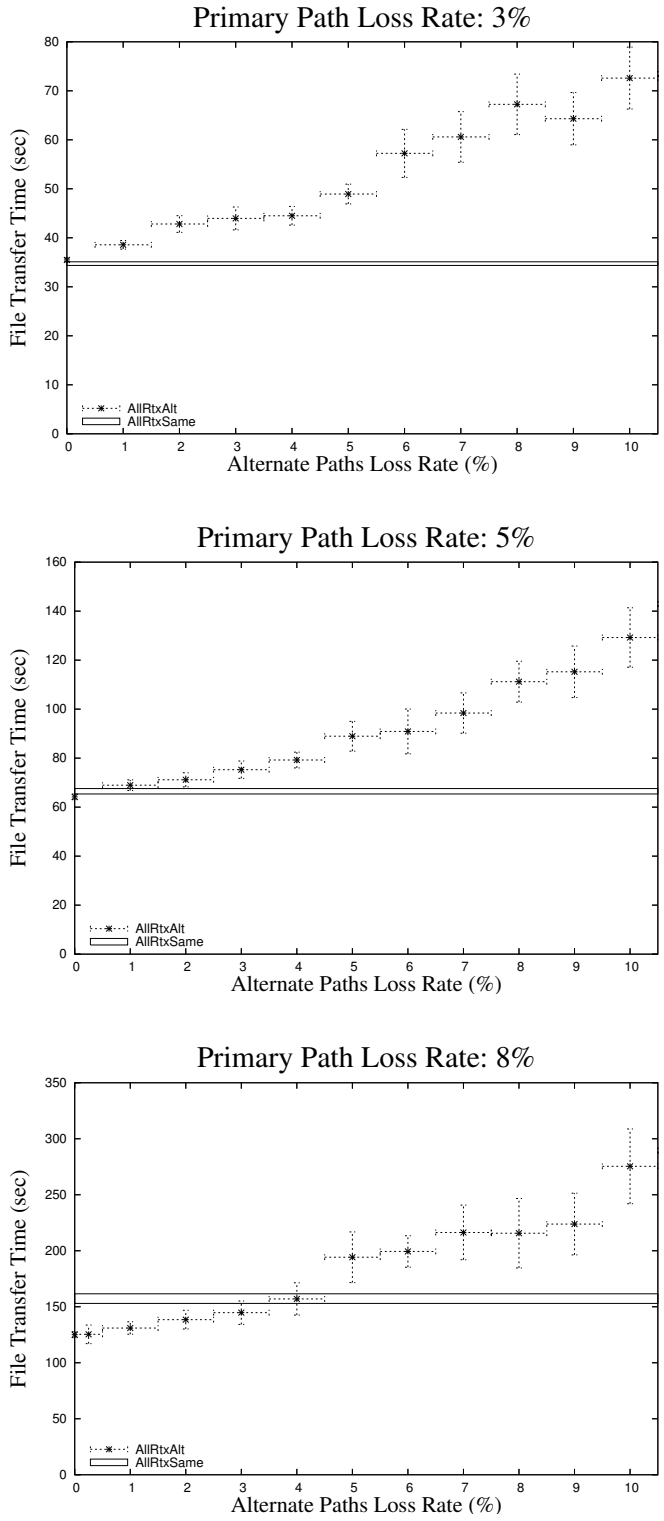

Figure 2.2: AllRtxAlt vs AllRtxSame at $\{3, 5, 8\}\%$ primary path loss

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 at times 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 [45]. The next section explains why.

2.2.3 Stale RTOs

After analyzing several experiment traces in detail, we attribute AllRtxAlt's poor performance to stale RTO values for the alternate path. Due to Karn's algorithm [68], successful retransmissions on the alternate path cannot be used to update the sender's 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 [105]).

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 2.3 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 back 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, even with a lower loss rate, maintains an average RTO value of 5.9 seconds – more than double the primary's. Figure 2.3'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 and improve the alternate path's RTT estimate. The graph also shows seven timeouts exponentially increasing the RTO value of the alternate path.

2.3 Best of Both Worlds

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 may or may not have *a priori* knowledge about the paths' conditions. Without such information, the best a sender can do is combine the good points of both policies. To do so, this author introduced a FrSameRtoAlt policy – a hybrid of AllRtxAlt and AllRtxSame. FrSameRtoAlt sends (a) fast retransmissions to the same

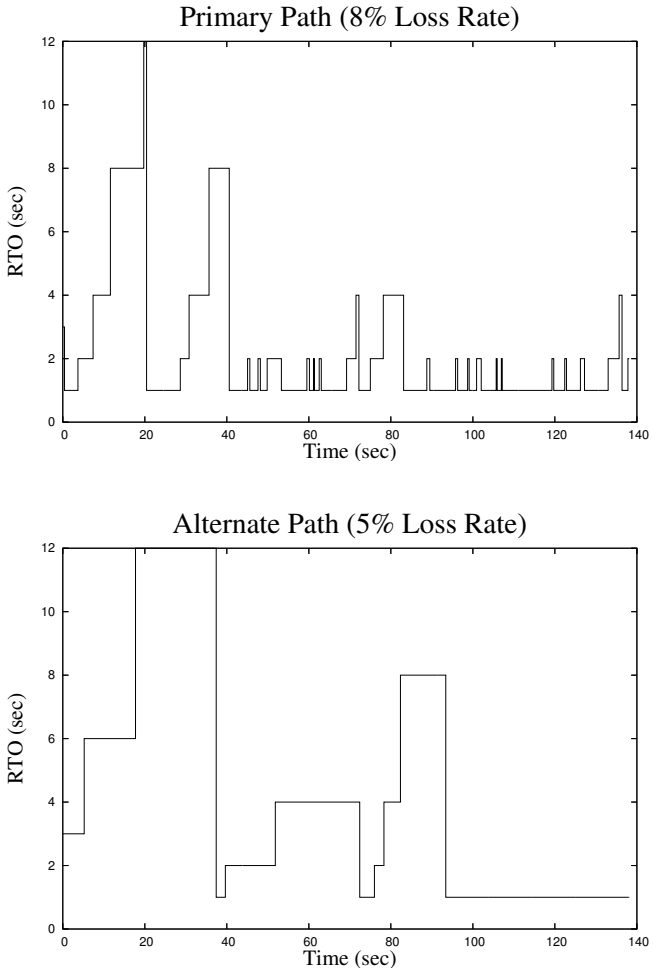

Figure 2.3: Example RTO dynamics with 8% primary path loss and 5% alternate path loss

destination as their original transmissions, and (b) timeout retransmissions to an alternate destination. 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 will determine whether FrSameRtoAlt does indeed effectively combine the good points of the other two policies.

2.3.1 Analysis Methodology

Figure 2.4 illustrates the network topology used, which is based on the topology previously presented in Figure 2.1. 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 2.1 more realistically simulates Internet traffic, but the simulation execution time for this technique became impractical. To evaluate if Figure 2.4's simplified model could still provide meaningful results, we compared representative simulations using the cross-traffic model from Figure 2.1 and the simpler uniform loss model from Figure 2.4. 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 2.4 maintains the same link bandwidths and delays used in Figure 2.1. 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 RTT on either path is 90ms, which is a reasonable RTT 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. We found 60 seeds to be sufficient for obtaining a 90% confidence interval.

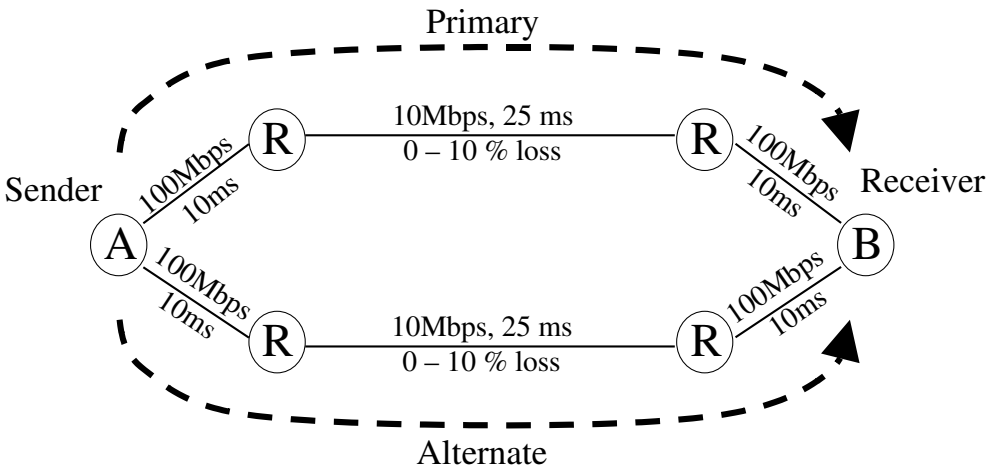

Figure 2.4: Simulation network topology with random loss

2.3.2 Results

Figure 2.5 illustrates the results for $\{3, 5, 8\}\%$ primary path loss rates. For each graph in Figure 2.5, the alternate path's loss rate is varied on the x-axis, ranging from 0-10%. The graphs in Figure 2.5 compare the average transfer time of a 4MB file using each 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 occurred for higher loss rates and policies that use the alternate path more often.

Figure 2.5 clearly shows that (obviously) 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 2.2.2, the graphs in Figure 2.5 also show that AllRtxAlt may improve performance when the alternate path's loss rate is lower than the primary's, but

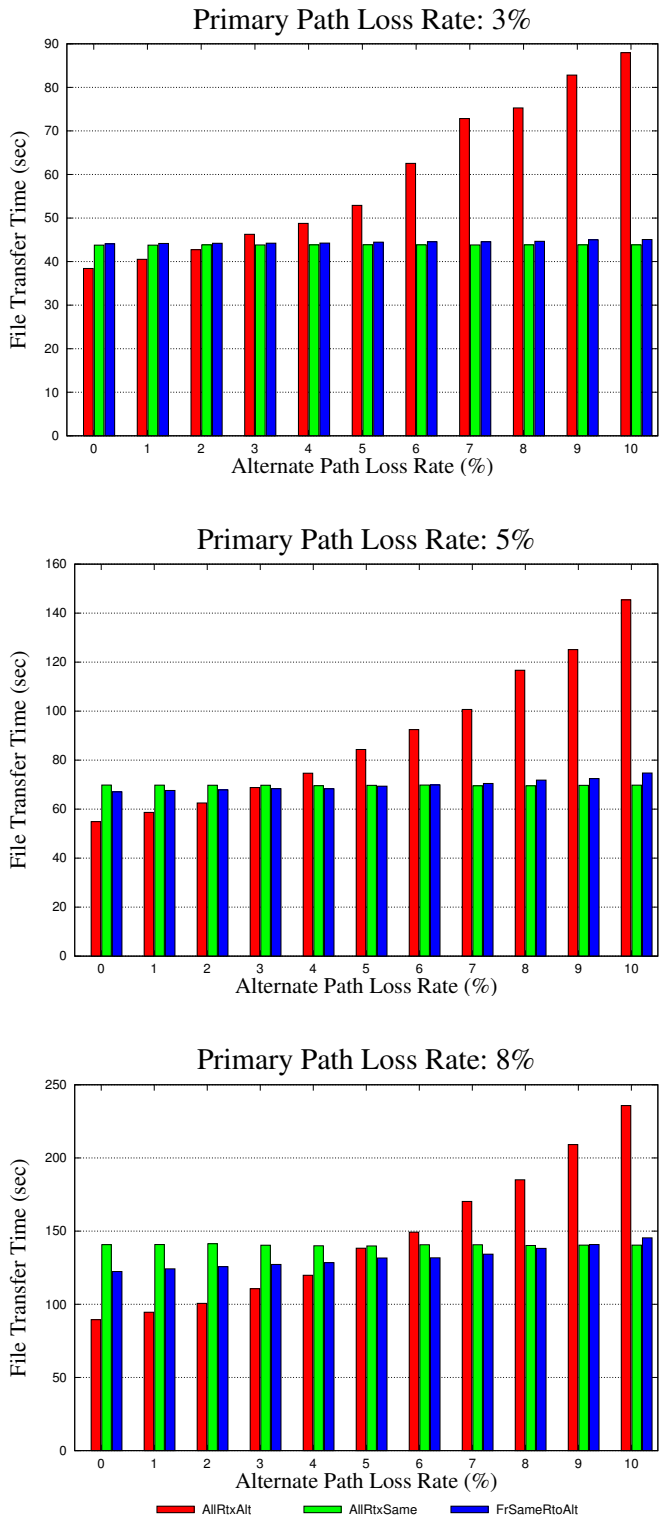

Figure 2.5: AllRtxAlt, AllRtxSame, and FrSameRtoAlt at $\{3, 5, 8\}\%$ primary path loss

the stale RTO problem dominates performance. First, AllRtxAlt does worse than AllRtxSame when both paths have the same bandwidth, delay, and loss rate. Second, 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 2.5), FrSameRtoAlt and AllRtxSame perform similarly. Most lost TPDUs 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 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.

When 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.

2.4 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 trying to improve each one's performance.

2.4.1 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, suggested by Randall Stewart, is in addition to the normal data retransmission behavior (specified by the retransmission policy) that remains unchanged. Since AllRtxSame only sends timeout retransmissions to the same destination, HAR is not applicable (see Figure 2.6). The extra heartbeats introduced by HAR try to eliminate 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, at the expense of extra traffic.

For example, suppose a TPDU 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 TPDU 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 undo the exponentially backed off RTO.

2.4.2 Timestamps (TS)

The timestamp (TS) mechanism is similar to TCP's timestamp mechanism. By including timestamps in each TPDU, all retransmission ambiguity is resolved. That is, the sender can always determine which transmission (original or retransmission) an ack belongs to. Thus, Karn's algorithm can be eliminated, and successful retransmissions can be used to update a sender's RTT estimate, and in turn facilitate a more accurate RTO value. Using a timestamp mechanism in a multihomed transport protocol (introduced by this author) is especially useful in alleviating the stale RTO problem of AllRtxAlt and FrSameRtoAlt, but at the expense of a 12 byte overhead in each TPDU.

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 TPDU 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.

2.4.3 Multiple Fast Retransmit (MFR)

In TCP and the current SCTP specification, a TPDU is "fast retransmitted" after a sender receives a specified number of missing reports (known as duplicate acks, or dupacks for short, in TCP terminology). Currently, however, a TPDU can be fast retransmitted only once. The Multiple Fast Retransmit (MFR) algorithm, introduced by this author, introduces extra state at the sender to allow lost fast retransmissions, in some cases, 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 TPDU x is lost. Data successfully received at the receiver are sacked, and any sacks for TPDUs sent after x serve as missing reports for TPDU x. When the sender receives four such missing reports, the standard fast retransmit algorithm is triggered and TPDU x is retransmitted.[2] At this point, MFR state stores the highest TPDU currently outstanding, n. This way, if the retransmission of

[2] SCTP [105] requires four missing reports to trigger a fast retransmit, whereas TCP requires only three analogous dupacks [11].

x is also lost, the sender can detect the loss with another four missing reports. Only sacks for TPDUs greater than n can serve as missing reports for another fast retransmission, because the sacks up to n were already in flight when x was fast retransmitted the first time.

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 can prevent 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 2.6 illustrates all ten policy-extension combinations.

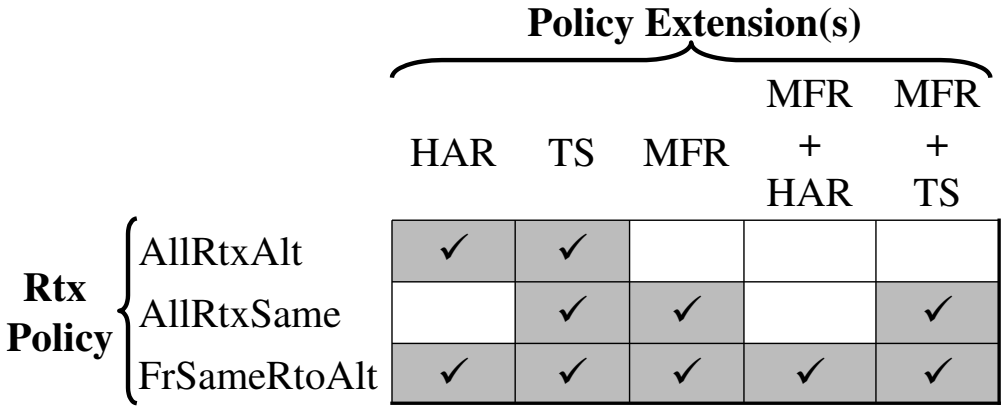

Figure 2.6: Possible policy-extension combinations

2.4.4 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 2.3.1.

2.4.4.1 AllRtxAlt's Extensions

Figure 2.7 presents the results for AllRtxAlt and its extensions with $\{3, 5, 8\}\%$ primary path loss rates. As the graphs show, both HAR and TS 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 over HAR. TS is proactive and offers more opportunities to measure the alternate path's RTT. Although TS adds a 12-byte overhead into each TPDU, the overhead does not significantly adversely impact performance. We conclude TS is the better extension for AllRtxAlt.

2.4.4.2 AllRtxSame's Extensions

Figure 2.8 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 MFR 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

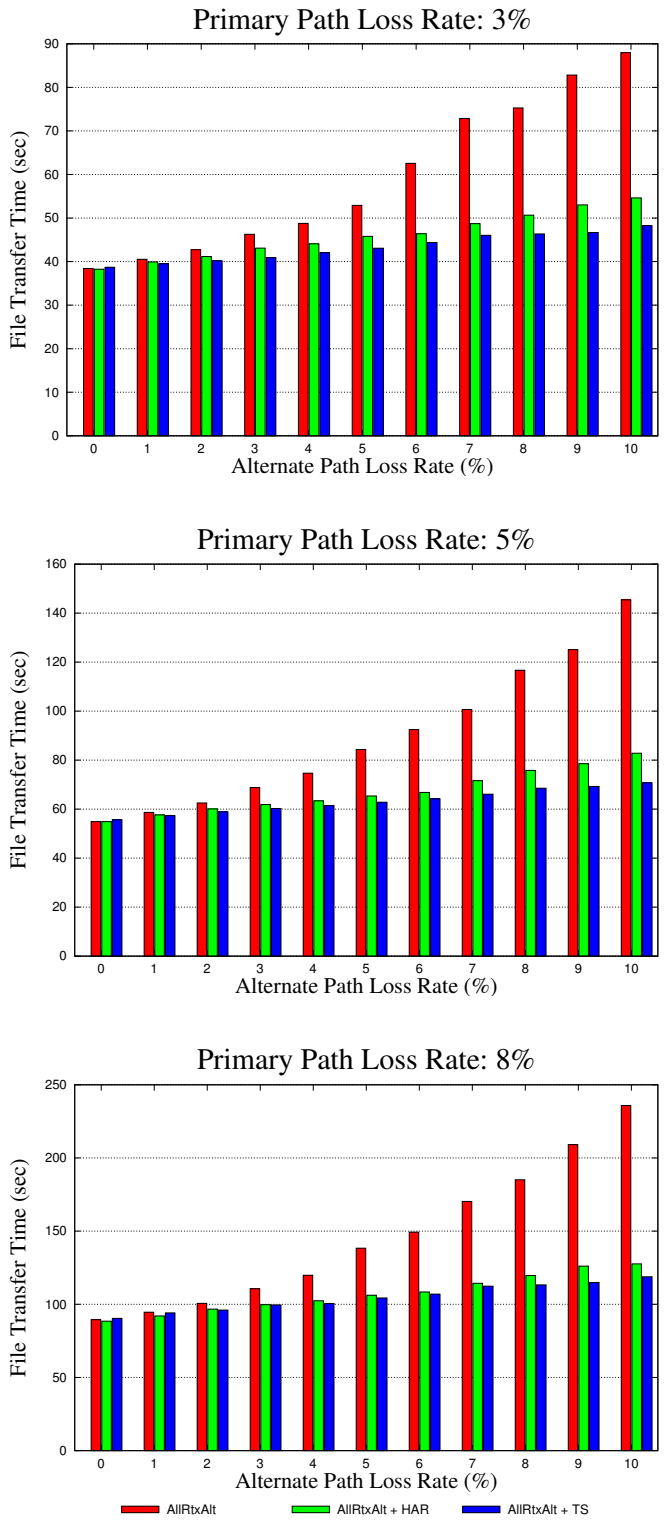

Figure 2.7: AllRtxAlt and its extensions at $\{3, 5, 8\}\%$ primary path loss

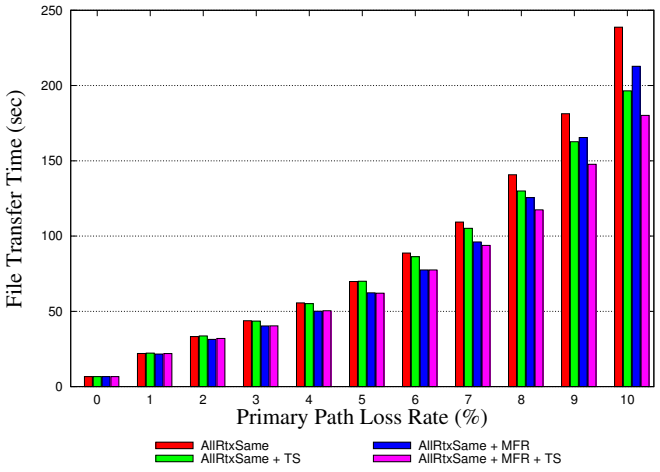

Figure 2.8: AllRtxSame and its extensions across all primary path loss rates

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.

2.4.4.3 FrSameRtoAlt's Extensions

FrSameRtoAlt qualifies for five extension combinations, three of which include MFR. Figure 2.9 shows that individually, MFR provides greater improvement than either HAR or TS. 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 performance 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.

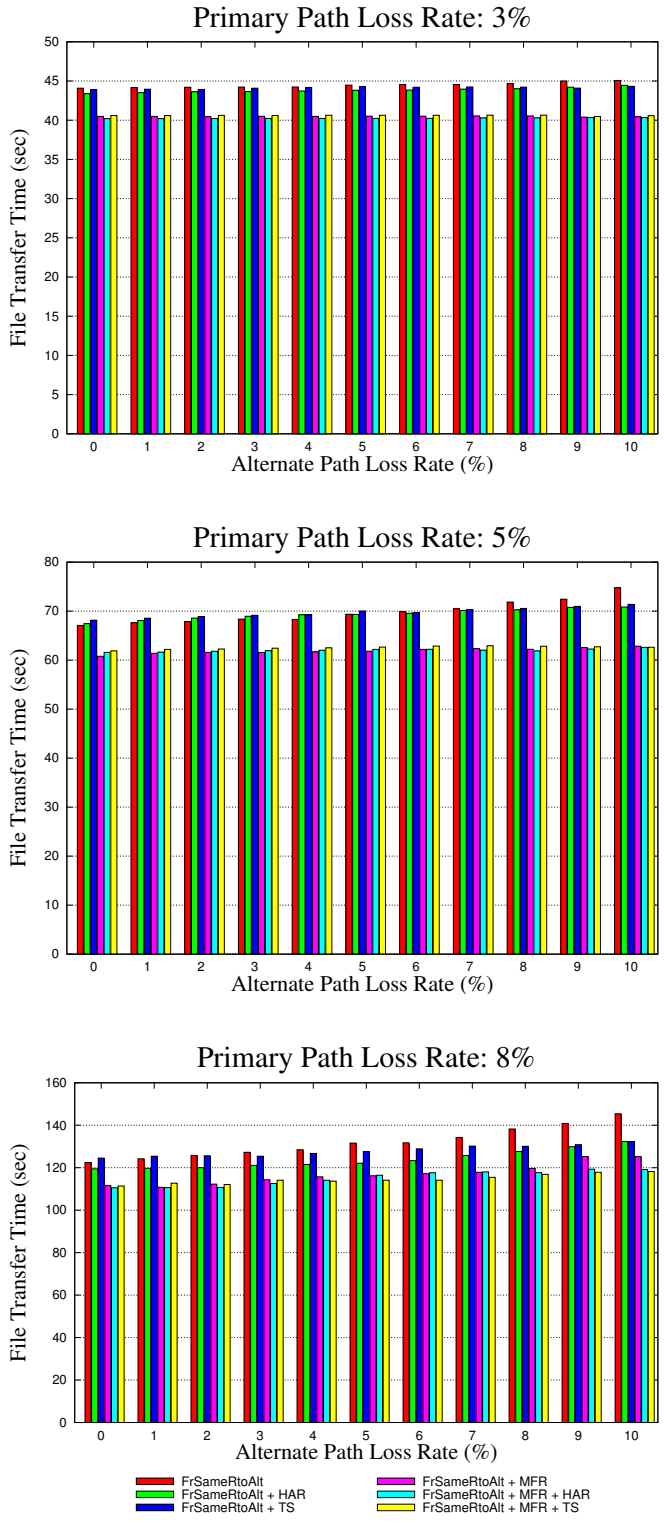

Figure 2.9: FrSameRtoAlt with its extensions at $\{3, 5, 8\}\%$ primary path loss

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 uses that can improve performance, such as the Eifel algorithm [74, 78].

2.5 Non-Failure Scenarios

This section revisits our performance comparison of the three policies in non-failure scenarios (done in Sections 2.2 and 2.3), but this time each policy is combined with our recommended extension(s): AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS. First, we evaluate their performance with symmetric path delays (i.e., both the primary and alternate paths have equal RTTs. Then, we assess the influence of the alternate path's delay in asymmetric trials. 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 chapter, we refer to AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS as simply AllRtxAlt, AllRtxSame, and FrSameRtoAlt, respectively.

2.5.1 Analysis Methodology

We again use the methodology presented in Section 2.3.1 for our evaluation, but in this section we investigate alternate path RTTs. The primary path remains unchanged (see Figure 2.10). 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 [55, 59, 66].

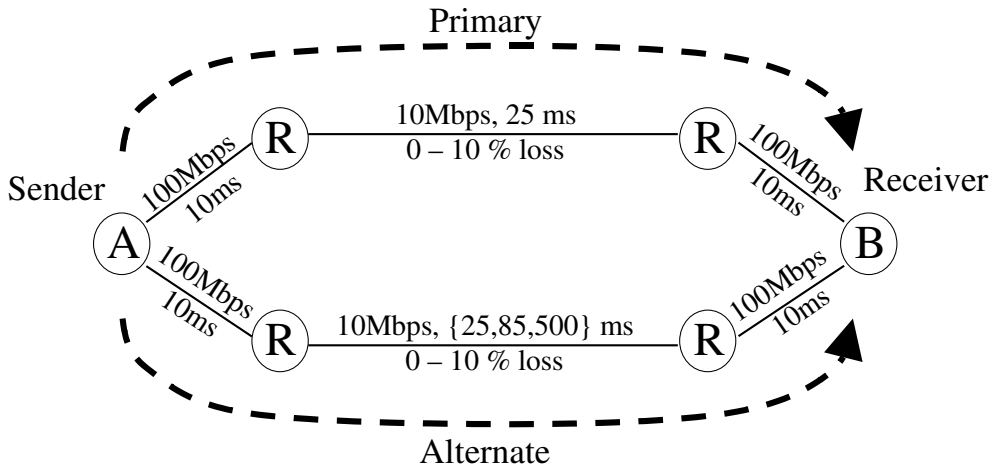

Figure 2.10: Simulation network topology with random loss, 90ms primary path RTT, and $\{90, 210, 1040\}$ms alternate path RTT

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.

2.5.2 Symmetric Path Delays

Figure 2.11 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 2.11 with Figure 2.5). 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.

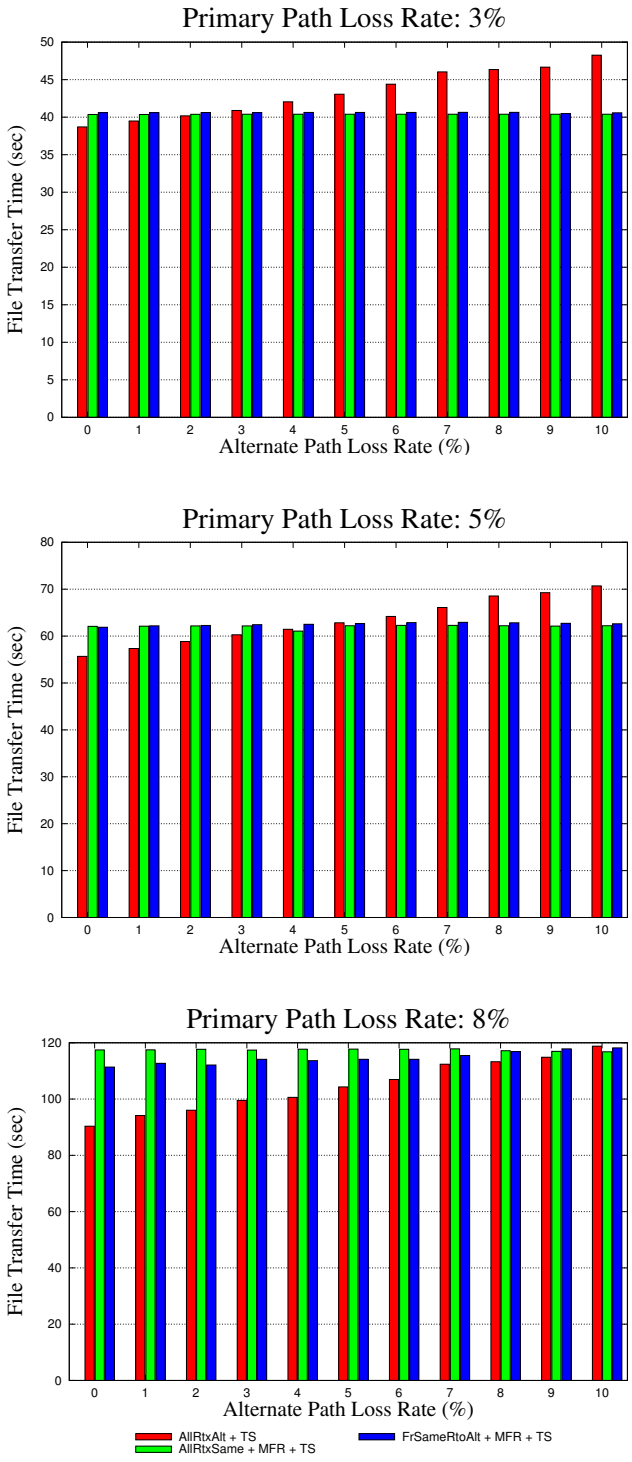

Figure 2.11: AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS at $\{3, 5, 8\}\%$ primary path loss, 90ms primary path RTT, and 90ms alternate path RTT

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 2.3.2 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 2.11 do not present a decisive argument for a particular best policy. FrSameRtoAlt outperforms AllRtxSame, but deciding between AllRtxAlt and FrSameRtoAlt is not straightforward. 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.

2.5.3 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 2.11. Hence, we push the limits further and present in Figure 2.12 the performance when the alternate path's RTT is more than ten times longer than the primary.

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

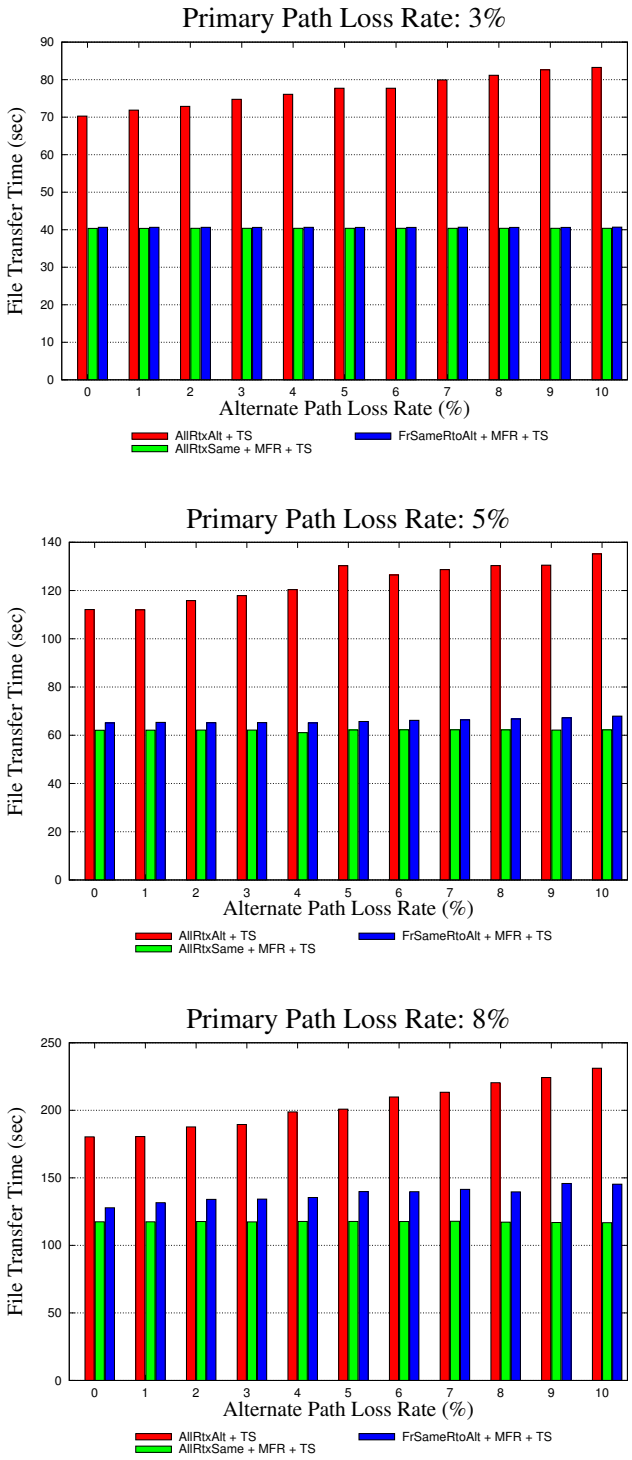

Figure 2.12: AllRtxAlt+TS, AllRtxSame+MFR, and FrSameRtoAlt+MFR+TS at $\{3, 5, 8\}\%$ primary path loss, 90ms primary path RTT, and 1040ms alternate path RTT

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.

Therefore, if the alternate path delay is *known* to be an order of magnitude longer than the primary's, we suggest using AllRtxSame. Otherwise, the conclusions from Section 2.5.2 remain.

2.5.4 Three Paths

To determine if our conclusions hold when the number of paths between the endpoints increases, we add an additional alternate path to the topology in Figure 2.10. We configure both alternate paths to have the same properties (bandwidths, delays, and loss rates) as each other. 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.

2.6 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.

2.6.1 Failover Algorithm

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 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 leaves the alternate destination selection method unspecified. We assume a round-robin selection method.

RFC2960 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.

2.6.2 Analysis Methodology

We use the same methodology described in Section 2.3.1, but in this section we introduce failure scenarios. The topology in Figure 2.13 shows that the both paths have the same characteristics, except that some time during the file transfer, 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, roughly 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, diabolically 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.

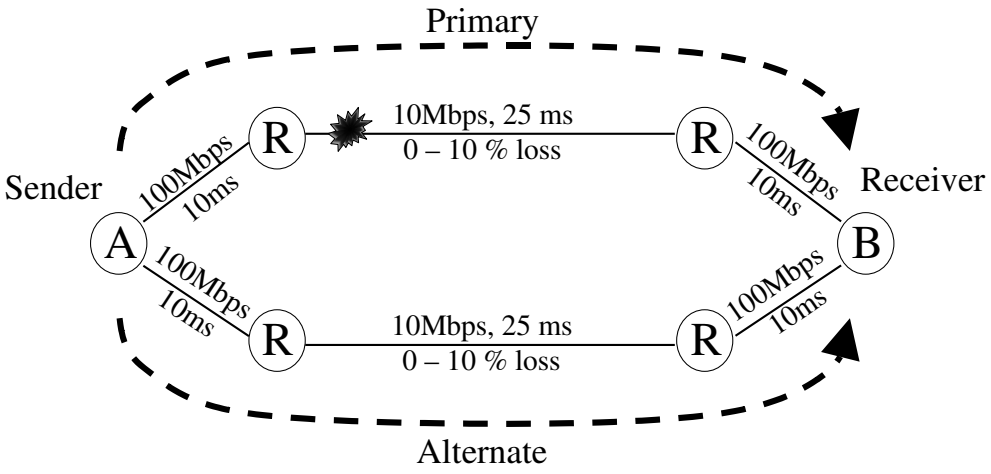

Figure 2.13: Simulation network topology with random loss, equal delays, and primary path failure

2.6.3 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 2.5.

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 on an alternate path 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 stalled 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 TPDUs outstanding when

failure occurs in our first failure scenario (link breakage at time $= 4$s). AllRtxAlt and FrSameRtoAlt successfully retransmit these 30 TPDUs 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 TPDU to the alternate destination. On the other hand, with AllRtxSame the sender successfully retransmits the initial 30 lost TPDUs only after the failure detection completes, delaying them by at least 63s! While perhaps not an issue for a file transfer, as being simulated in our experiments, this delay may be unacceptable to applications requiring timely or consistent rate-based data delivery.

During the third period, the sender has only one available path for transmission in our simulations. (The results in Section 2.5 apply to scenarios where more than one path are available during the third period.) Figure 2.14 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 2.11 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 2.15. 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 2.15 shows that AllRtxSame's transfer stalls during failure detection has drastic effects on the results. The file transfer with AllRtxSame requires complete failure detection and takes about 70s to complete,

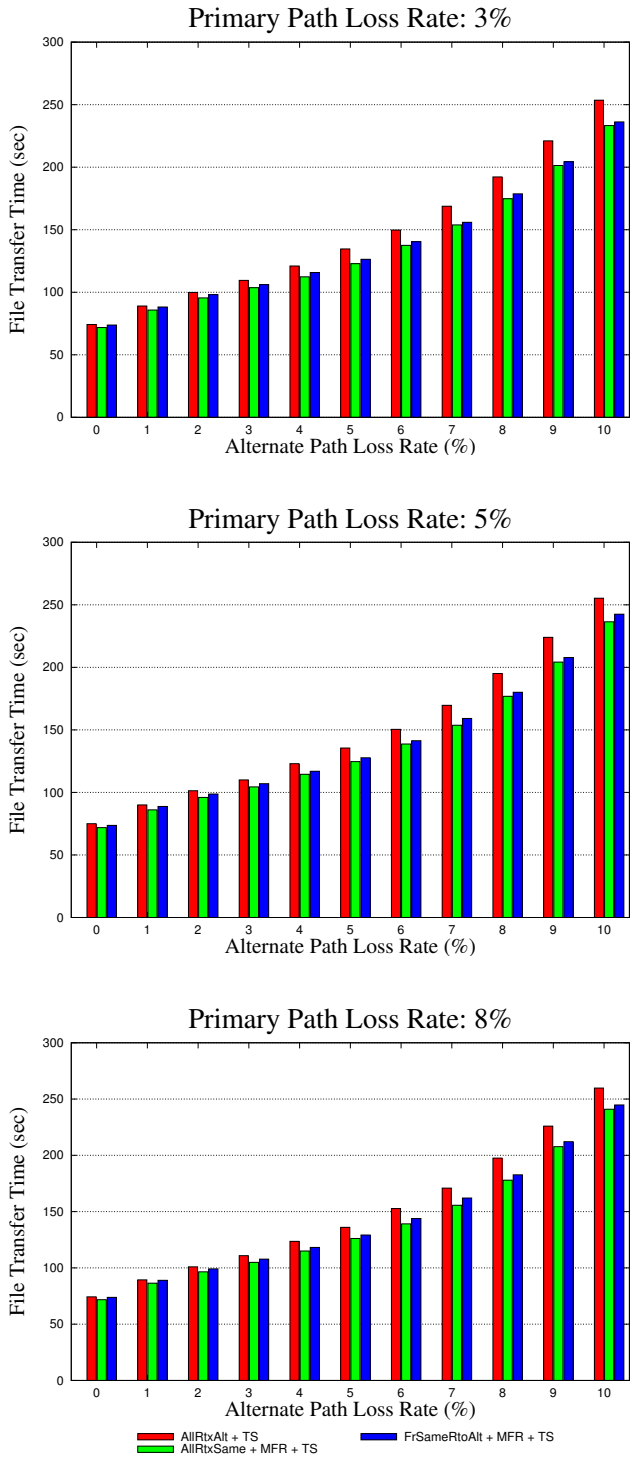

Figure 2.14: AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS with primary path failure at time $= 4$s

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 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.

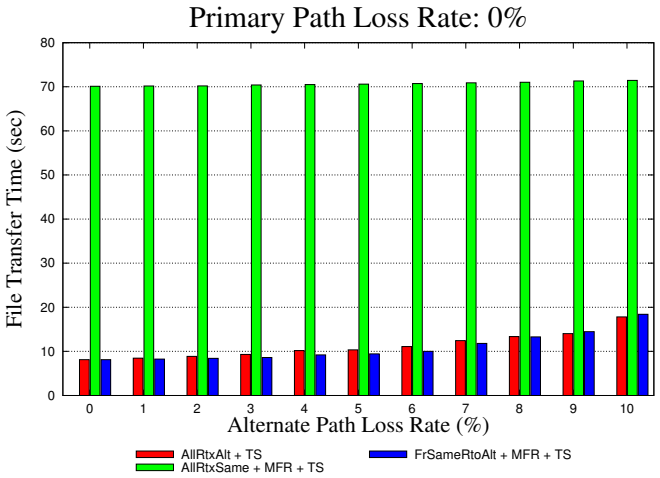

Figure 2.15: AllRtxAlt+TS, AllRtxSame+MFR+TS, and FrSameRtoAlt+MFR+TS with primary path failure at time $= 6.8$s

In summary, all three policies provide similar throughput performance for large transfers during failure scenarios. However, AllRtxSame's transfer stalls during failure detection 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.

2.7 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 this author's hybrid policy: (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.[3] 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.

[3] This policy has been proposed to the IETF as a change to SCTP [101].

Chapter 3

FAILOVER THRESHOLDS

3.1 Introduction

SCTP has a tunable failover threshold that RFC2960 recommends should be set to a conservative value of six consecutive timeouts, which translates to a failure detection time of at least 63 seconds – unacceptable for many applications. In this chapter, we measure the tradeoff between more aggressive failover (i.e., lower thresholds) and spurious failovers. Lower thresholds provide faster failure detection when failure has occurred, but cause spurious failovers in non-failure lossy conditions. However, we surprisingly find that spurious failovers do not degrade performance, and often actually improve goodput regardless of the paths' characteristics (bandwidth, delay, and loss rate).

Currently, traffic migrates back to the primary path when the primary path recovers (i.e., failovers are temporary). This temporary nature throttles the sending rate, because upon returning to using the primary path, the sender must enter slow start with a cwnd of one MTU. To avoid this slowdown, this author introduces the concept of permanent failovers in multihoming protocols, where a sender makes the failover permanent if the primary path does not respond within some threshold amount of time. We find that permanent failovers can improve performance when a sender has estimates of each path's RTT and loss rate to make an informed decision. In the case a sender does not have such information, we recommend that permanent failovers not be used.

Section 3.2 describes SCTP's current failover mechanism. Section 3.3 presents the tradeoffs between more aggressive failover and spurious failovers. Section 3.4 introduces

and evaluates a modified failover mechanism that allows failovers to become permanent. We conclude the chapter in Section 3.5.

3.2 Formal Specification of SCTP's Failover Mechanism

Section 2.6.1 presented the details of the failover algorithm, but here we present its formal specification, as shown in Figure 3.1 for n destinations. The association begins in Phase I, where destination D_i is the primary destination, D_i is in the active state, and all new data are sent to D_i. When D_i fails, "failover" occurs and the association moves into Phase II.

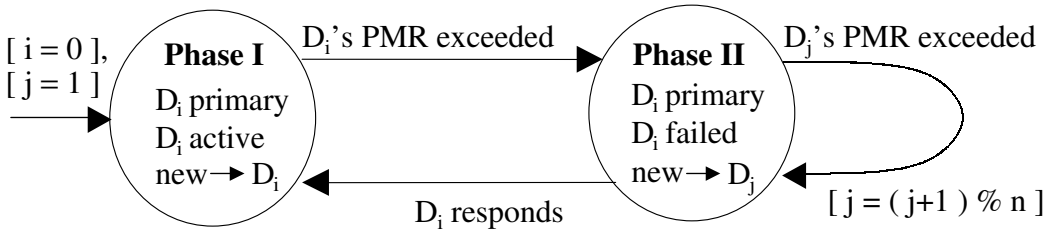

Figure 3.1: FSM for current failover mechanism

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

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

3.3 Reducing PMR

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

3.3.1 Methodology

We evaluate different PMR settings using the University of Delaware's SCTP module [33] for the ns-2 network simulator [19] (see Chapter 5). Figure 3.2 illustrates the network topology. The multihomed sender, A, has two paths (labeled *Primary* and *Alternate*) to the multihomed receiver, B. The primary path's core link has a 10Mbps bandwidth and a 25ms one-way delay. The alternate path's core link has a 10Mbps bandwidth and one-way delays of 25ms, 85ms, and 500ms. Each router, R, is attached to a dual-homed node (A or B) via an edge link with 100Mbps bandwidth and 10ms one-way delay.

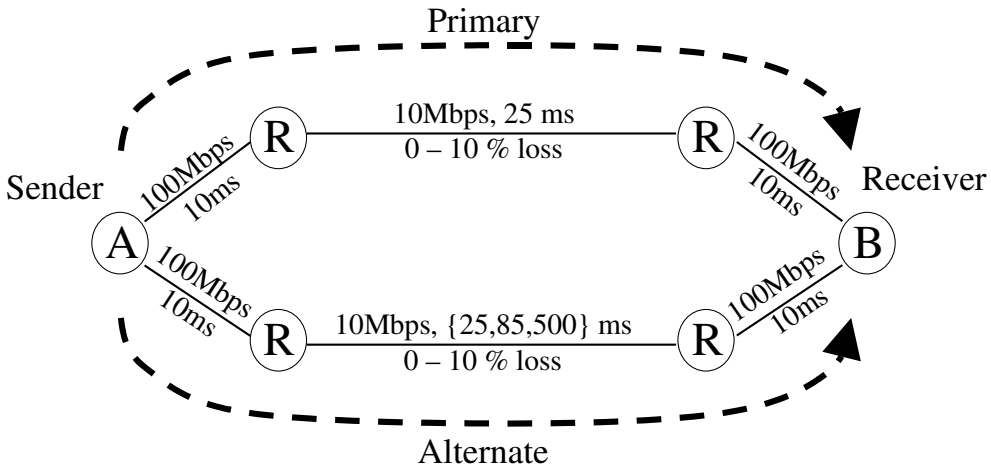

Figure 3.2: Simulation network topology

The end-to-end RTTs are 90ms, 210ms, and 1040ms, which sample reasonable delays on the Internet today. An RTT of 1040ms may seem large, but as state in Section 2.5.1, flows passing through cellular networks often experience RTTs as high as 1 or more seconds [55, 59, 66]. In any case, the delays are selected to demonstrate relative performance, and we believe our results and conclusions are independent of the actual bandwidth and delay configurations.

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

We introduce uniform loss on these paths (0-10% each way) at the core links. Again, we realize that using cross-traffic to cause congestion would more realistically simulate loss, but we found the simulation time for such a technique became impractical. On the other hand, uniform loss is a simple, yet sufficient model to provide insight about the effectiveness of different PMR settings accurately detecting failure. To evaluate if Figure 3.2's loss model was reasonable, we compared representative simulations using a cross-traffic model, shown in Figure 2.1, to produce self-similar, bursty traffic. Although the absolute results differed for those examples compared, relative relationships remained consistent – leading to the same conclusions. We therefore proceeded with the simpler uniform loss model.

In our simulations, the sender uses a different retransmission policy than specified in RFC2960. Instead, the sender transmits using this author's hybrid policy, FrSameRtoAlt, which was shown in Chapter 2 to perform better for *roughly* asymmetric path delays (i.e., no path delay is more than double of another). In our simulations, this author's Multiple Fast Retransmit algorithm (see Section 2.4.3) is also used to reduce the number of timeouts.

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

1. primary path's loss rate

2. alternate path's loss rate

3. alternate path's core link delay

4. PMR setting

3.3.2 Spurious Failovers

Figure 3.3 plots, for each PMR setting, the fraction of 80MB file transfers that experience at least one spurious failover at primary path loss rates 0-10%. The graph aggregates all alternate path loss rates for each particular primary path loss rate.

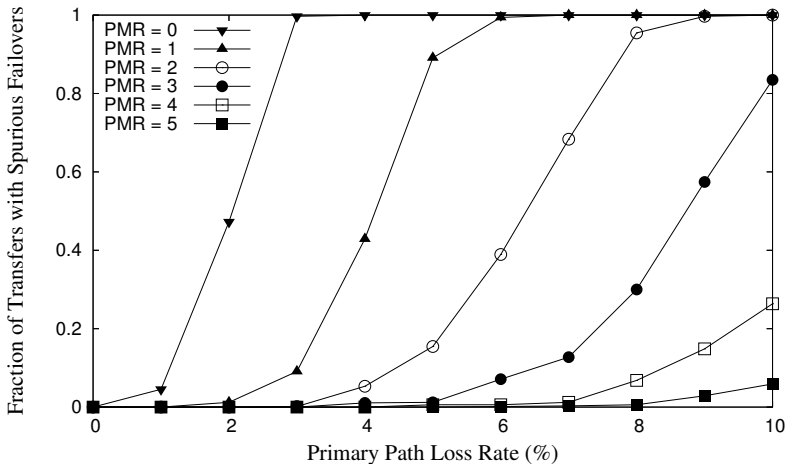

Figure 3.3: Fraction of transfers with spurious failovers

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

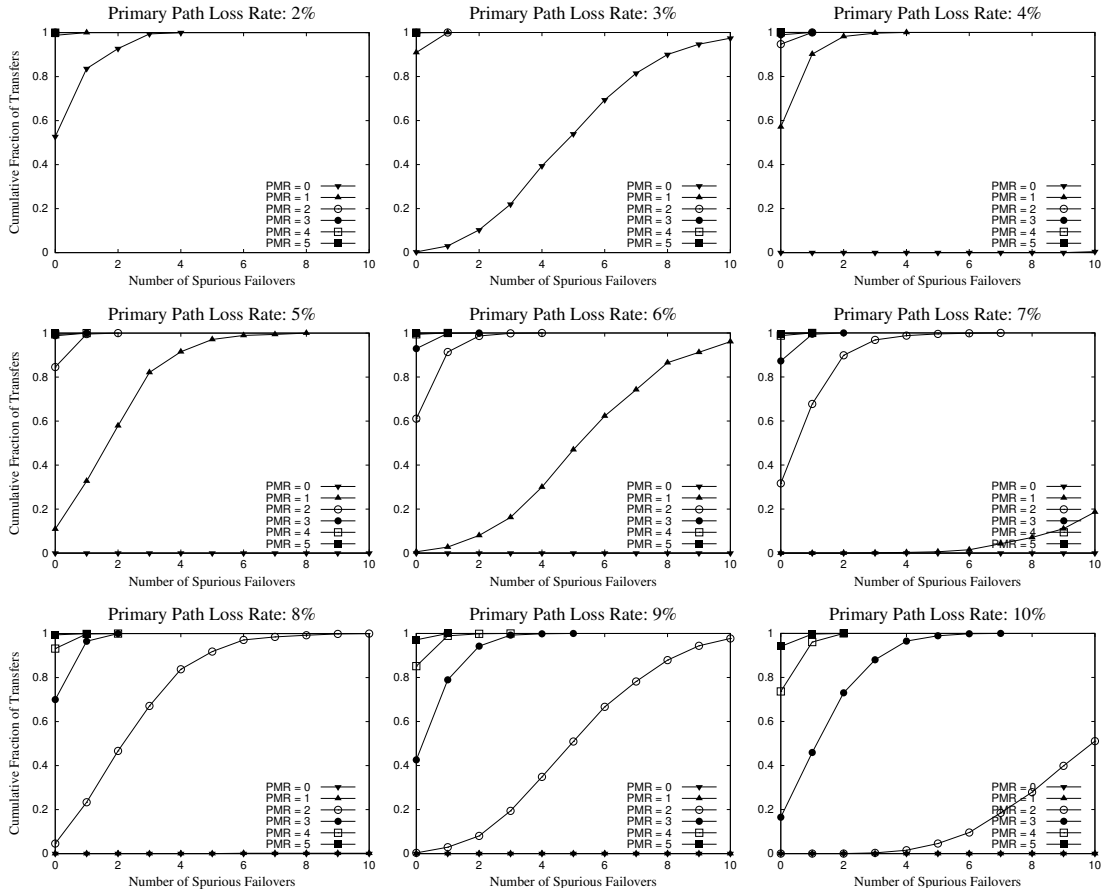

Figure 3.4: CDF of the number of spurious failovers for primary path loss rates 2-10%

the PMR increases, which argues that the robustness increases by more than a constant for each PMR setting.

The frequency of spurious failovers is also important when considering the robustness of various PMR settings. Figure 3.4 plots the cumulative distribution function (CDF) of the number of spurious failovers for primary path loss rates 2-10%. The CDFs for 1% primary path loss rate are omitted, because PMR $= \{1, 2, 3, 4, 5\}$ experience no spurious failovers, and PMR $= 0$ experiences spurious failovers in only 5% of the transfers. Again, each graph in Figure 3.4 aggregates all alternate path loss rates for each primary path loss rate.

At a 2% primary path loss rate, 53% of transfers with PMR $= 0$ experience no

spurious failovers, and 84% of transfers spuriously failover at most once. When the loss rate increases to 3%, less than 1% of transfers with PMR $= 0$ experience no spurious failovers. Then with 4% loss, only 1% of transfers experience less than ten spurious failovers.

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

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

To conclude, determining which failover threshold is "robust enough" largely depends on the networking environment. For example, Zhang et al. [116] use end-to-end Internet measurements to report that 84% of their traces experienced less than a 1% loss rate (i.e., essentially "lossless"), and 15% of their traces had loss rates of 1-10% (with an average of 4%). Thus, to be completely robust against spurious failovers on 99% of Internet paths, PMR should be set to 6 (even PMR $= 5$ spuriously fails over 6% of the time at 10% loss), but that translates to a failover time of 123 seconds! *However, we conclude that PMR $= 3$ is robust enough for the Internet. This setting translates to a 15 second failover time, and is robust for all "lossless" paths and the average "lossy" path.*

3.3.3 Symmetric Path Delays

While the frequency of spurious failovers is important in providing intuition about overall behavior, of greater importance is how these spurious failovers affect performance. We collected results for 0-10% loss on the primary and alternate paths, but we do not include all results. Figure 3.5 plots the average 80MB file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 90ms alternate path RTT. Each graph

has a fixed primary path loss rate, and varies the alternate path loss rate on the x-axis from 0-10%.

Counter to our intuition, we observe that the PMR setting has little effect on the goodput for primary path loss rates less than 8%. Above 8%, the results show that lower (!) PMR settings begin to improve performance, with PMR $= 0$ providing the most improvement. That is, surprisingly, being more aggressive with failover often provides improved performance, even when the alternate path loss rate is higher than that of the primary path. For example, reducing the PMR from 5 to 0 improves the performance by 4% when the primary and alternate path loss rates are 8% and 10%, respectively. These counter-intuitive results will be explained later in Section 3.3.7.

3.3.4 Asymmetric Path Delays

We are also surprised to find that being aggressive with failover does not change with asymmetric path delays. We expected lower PMR settings to perform relatively worse when the alternate path has a larger RTT than the primary path. However, we find that the results remain nearly constant regardless of the alternate path delay. Figure 3.6 plots the results for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 1040ms alternate path RTT. Comparing these results with those in Figure 3.5 shows that the alternate path's longer RTT does not affect the performance. Even when the alternate path's RTT is more than ten times longer than the primary path's, PMR $= 0$ outperforms other PMR settings. Again, these results were unexpected, and will be explained in Section 3.3.7.

3.3.5 Three Paths

To determine if our conclusions hold when the number of paths between the endpoints increases, we add an additional alternate path to the topology in Figure 3.2. We configure both alternate paths to have the same properties (bandwidths, delays, and loss rates) as each other. Again, we want to avoid an unmanageable number of simulation

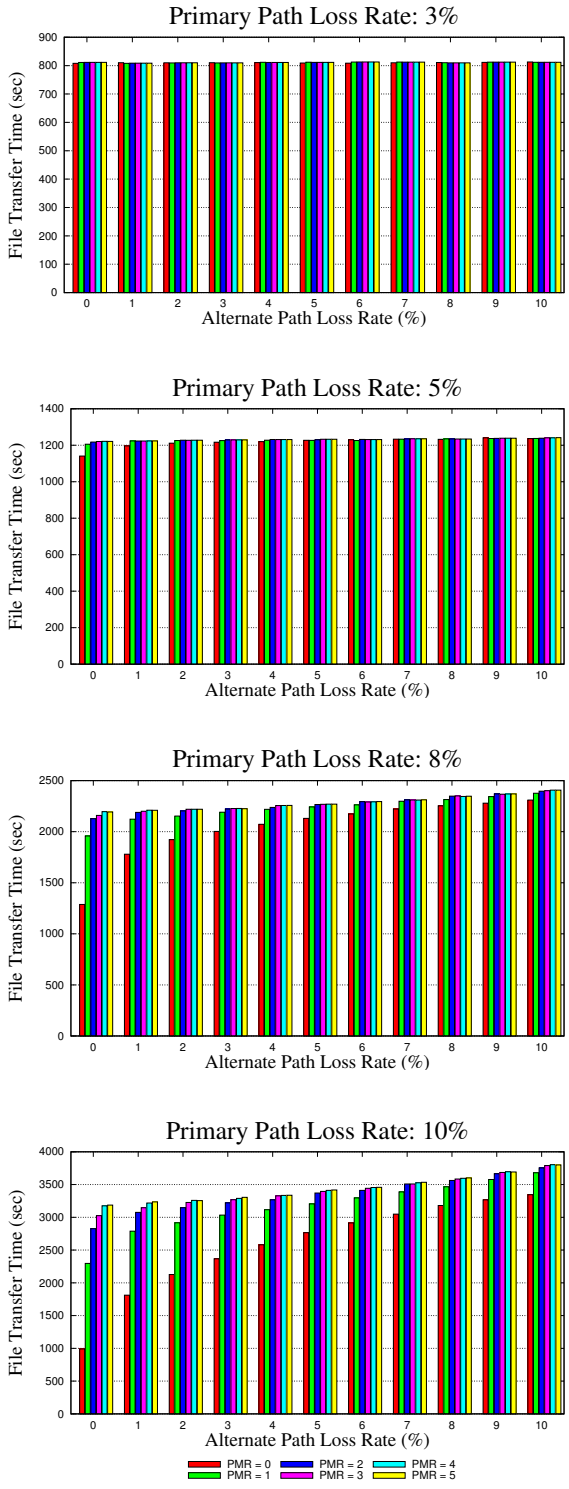

Figure 3.5: PMR evaluation: 90ms primary path RTT and 90ms alternate path RTT

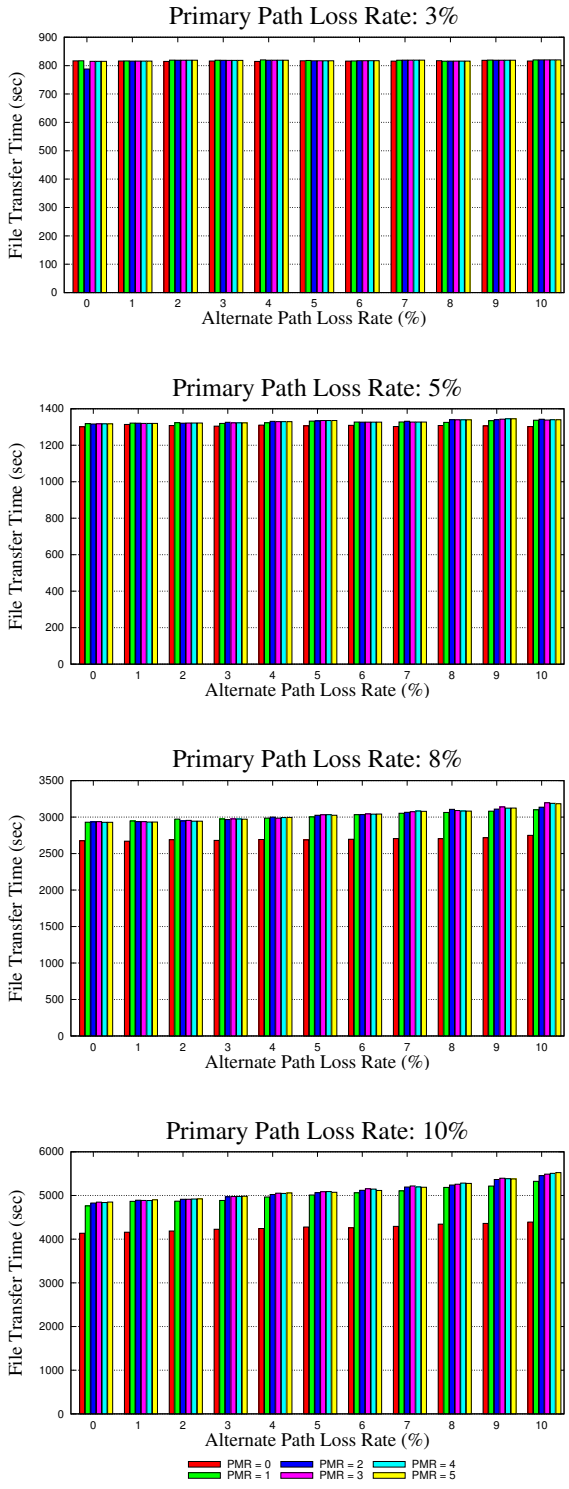

Figure 3.6: PMR evaluation: 90ms primary path RTT and 1040ms alternate path RTT

parameters. The results (not shown) are consistent with those for two paths. That is, the relationships between the different PMR settings remain the same. We expect that the trends will remain the same for configurations with more than three paths between endpoints.

3.3.6 Dormant State Behavior

As the finite state machine in Figure 3.1 shows, if a sender fails over to an alternate destination that in turn fails, the sender will failover to yet another alternate destination. If needed, the sender continues to failover to other alternate destinations until all alternate destinations are exhausted. When all destinations have failed, the association enters the *dormant state* [104] (not represented in Figure 3.1).

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

In the current SCTP is specification, dormant state behavior is considered unimportant, because high PMR settings make the dormant state unlikely. However, if PMR is lowered to 0, as our results thus far argue should be done, entering the dormant state becomes more likely. Thus, this author has identified three different dormant state behaviors to evaluate how they might impact performance: (1) Dormant LastDest, (2) Dormant Primary, and (3) Dormant Hop.

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

Phase I (permanent failover will be introduced in Section 3.4). If an alternate destination replies, the association returns to Phase II with the destination that replied as D_j.

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

The Dormant Hop behavior, shown in Figure 3.7, attempts to be more aggressive in finding an active destination. While in the dormant state, the sender transmits new data to a different destination after each timeout. The sender cycles through all the destinations in a round-robin fashion until either a destination responds, or the association aborts.

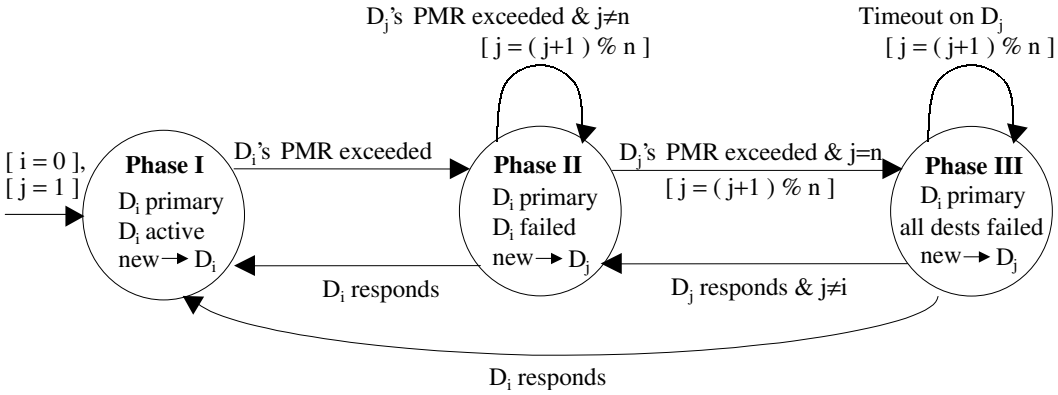

Figure 3.7: FSM with Dormant Hop behavior

The results in Sections 3.3.2 through 3.3.5 use the Dormant Hop behavior, but we also evaluate the performance of the other two dormant state behaviors. We find that dormant state behavior does not affect goodput, and the trend reported in those sections remains consistent for all dormant state behaviors (results not shown).

3.3.7 Explaining the Results

Our results document that aggressive failover settings (in particular, PMR $= 0$) improve performance regardless of the path loss rates, path delays, and/or dormant state

behavior – a result counter to our intuition. We spent considerable time investigating this surprising conclusion, which we now explain.

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

The remainder of this section presents four detailed timeout scenarios (shown in Figure 3.8) for PMR $= \{0, 1\}$ to demonstrate the merits of more aggressive failover. They all begin with TSN 1 (i.e., TPDU 1) being lost in transit to the primary destination and subsequently timing out. For PMR $= 0$, the sender immediately fails over, retransmits TSN 1 to the alternate destination, and sends a heartbeat to the primary destination. For PMR $= 1$, the sender retransmits TSN 1 to the alternate destination and sends TSN 2 to the primary destination. We compare the behavior of these two PMR settings by following the details of four (of many) possible scenarios beyond this point.

3.3.7.1 Scenario 1

The first TPDU sent to the primary destination and the first TPDU sent to the alternate destination following TSN 1's timeout are both delivered successfully.

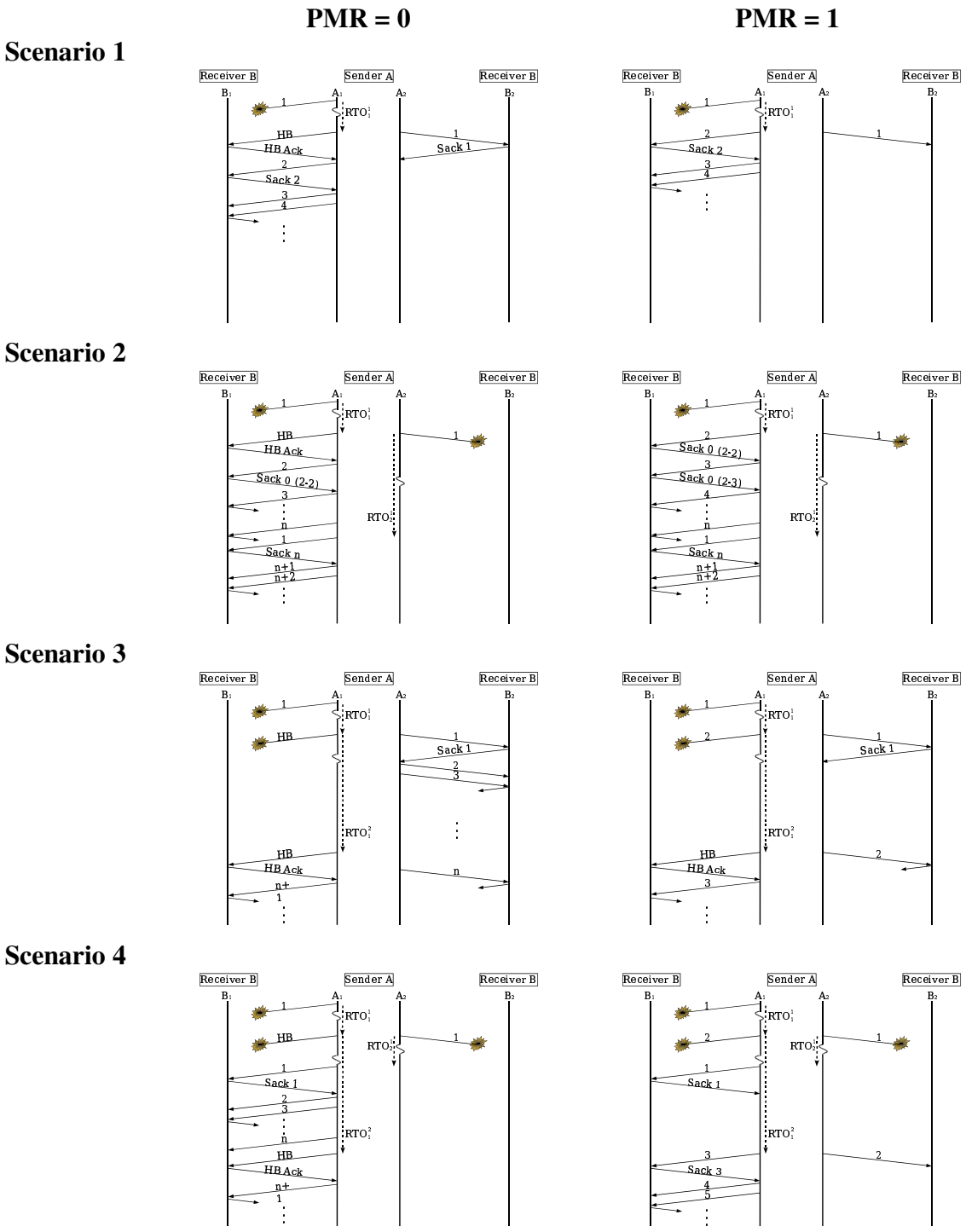

Figure 3.8: Timeout scenarios

- **PMR = 0** The failover is cancelled when the heartbeat is acked. Although the figure shows both TSN 1 and the heartbeat are acked at the same time, it is a race condition. If the heartbeat gets acked first (as shown in Figure 3.8's Scenario 1), then TSN 2 is sent on the primary and normal data transfer continues from this point. If TSN 1 gets acked first (not shown), then TSNs 2-3 are sent to the alternate destination, TSN 4 is sent to the primary destination when the heartbeat is acked, and normal data transfer continues to the primary destination.

- **PMR = 1** As both TSN 1 and 2 are sent at about the same time, again a race condition occurs. If TSN 1 arrives at the receiver first, the receiver's delayed ack algorithm causes a single cumulative ack (denoted SACK 2) to be generated for both TSN 1 and 2 (as shown in Figure 3.8). When this ack arrives, TSNs 3-4 are sent to the primary destination and normal data transfer continues to the primary destination. If TSN 2 arrives at the receiver first, the receiver generates two acks (not shown). The first selectively acks TSN 2 with a missing report for TSN 1, and the second cumulatively acks TSN 2. Upon receiving the first, the sender sends TSN 3 to the primary destination and normal data transfer continues to the primary destination.

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

On the other hand, if $RTT_1 > RTT_2$ (not shown in Figure 3.8 and not the case in our experiments), then PMR $= 0$ gets ahead of PMR $= 1$ in the overall transfer. The amount by which PMR $= 0$ gets ahead depends on the ratio of the two paths' RTTs. However, since $RTT_1 \leq RTT_2$ in our experiments, we omit detailed analysis of PMR $= 0$'s performance gain when $RTT_1 > RTT_2$.

3.3.7.2 Scenario 2

The first TPDU sent to the primary destination following TSN 1's timeout is successfully delivered, and the first TPDU sent to the alternate destination is lost.

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

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

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

3.3.7.3 Scenario 3

The first TPDU sent to the primary destination following TSN 1's timeout is lost, and the first TPDU sent to the alternate destination is delivered successfully.

- **PMR = 0** When TSN 1 is acked, TSNs 2-3 are sent to the alternate destination, and normal data transfer continues temporarily to the alternate destination. Eventually, the heartbeat times out, and another heartbeat is then sent to the primary destination. Since this timeout is the second consecutive timeout on the primary destination, it

will take at least 2 seconds to expire (assuming RTO.Min is 1 second). Once the second heartbeat is successfully acked, the sender cancels the failover, and resumes normal data transmission to the primary destination.

- **PMR = 1** When TSN 1 is acked, the sender is temporarily stalled and does not send any new data. When TSN 2 times out (again, at least 2 seconds later), the sender fails over to the alternate destination, retransmits TSN 2 to the alternate destination, and sends a heartbeat to the primary destination. From this point, normal data transfer continues to the alternate destination until the heartbeat is acked and the failover is cancelled. Then the sender resumes normal data transfer to the primary destination.

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

$$r = min\left[1, \frac{RTO_1^2 + RTT_1}{RTT_2}\right] \tag{3.1}$$

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

To estimate the number of TPDUs, d, sent to the alternate destination during r alternate path round trips, we first assume that no loss occurs on the alternate path during this period. Hence, the transfer on the alternate path exits slow start when cwnd exceeds

ssthresh. Using the slow start cwnd growth model from [24], the last alternate path cwnd before exiting slow start is

$$cwnd = ssthresh = init\_cwnd \cdot \left(1 + \frac{1}{b}\right)^{r_{ss}-1} \tag{3.2}$$

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

$$r_{ss} = max\left[r, 1 + log_{\frac{3}{2}}\left(ssthresh\right)\right] \tag{3.3}$$

Using a component of the slow start data transfer model from [24], the number of TPDUs sent during the first r_{ss} round trips on the alternate path is

$$\begin{aligned} d_{ss} &= 1 \cdot \frac{\left(\frac{3}{2}\right)^{r_{ss}} - 1}{\frac{3}{2} - 1} \\ &= 2\left[\left(\frac{3}{2}\right)^{r_{ss}} - 1\right] \end{aligned} \tag{3.4}$$

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

$$r_{ca} = r - r_{ss} \tag{3.5}$$

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

$$cwnd_{i+1} = cwnd_i + 1 \tag{3.6}$$

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

$$cwnd_{i+1} = ssthresh + i \tag{3.7}$$

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

$$d_{ca} = \sum_{i=1}^{r_{ca}} (ssthresh + i)$$

$$= (r_{ca} \cdot ssthresh) + \sum_{i=1}^{r_{ca}} i$$

$$= (r_{ca} \cdot ssthresh) + \frac{r_{ca} + (r_{ca})^2}{2} \quad (3.8)$$

Combining (3.4) and (3.8), we estimate the number of successful data TPDUs that PMR $= 0$ sends to the alternate destination in r alternate path round trips as

$$d = d_{ss} + d_{ca} \quad (3.9)$$

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

This analysis assumes that the alternate path does not experience loss, but we now relax this constraint by considering alternate path losses after TSN 1 (the case where TSN 1 is lost is presented next in Scenario 4). Without getting into the details of such scenarios (there are an infinite number), it suffices to say that our estimate of d in (3.9) is an overestimate when loss is introduced. However, the fact that $d \geq 1$ remains true. Therefore, it remains that, in this scenario, PMR $= 0$ performs no worse than PMR $= 1$, and may outperform PMR $= 1$ by as much as $(d - 1)$ TPDUs, depending on r and the loss conditions on the alternate path.

3.3.7.4 Scenario 4

The first TPDU sent to the primary destination and the first TPDU sent to the alternate destination following TSN 1's timeout are both lost.

- **PMR = 0** TSN 1's retransmission times out first, and TSN 1 is re-retransmitted to the primary destination. When TSN 1 is acked, the failover is cancelled and normal

data transfer continues to the primary destination from this point. Note that the heartbeat times out later, but does not affect the data transfer.

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

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

$$r = min\left[1, \frac{RTO_1^2 - RTO_2^1}{RTT_1}\right] \tag{3.10}$$

where RTO_1^2 is the primary path's RTO for the second consecutive timeout, RTO_2^1 is the alternate path's RTO, and RTT_1 is the primary path's RTT. Using this value of r in (3.3) and (3.5), we can use (3.9) to estimate the number of successful data TPDUs, d, that PMR $= 0$ sends to the primary destination by the time RTO_2^1 expires. Therefore, this scenario also shows PMR $= 0$ performing no worse than PMR $= 1$, and possibly outperforming PMR $= 1$ by as much as $(d - 1)$ TPDUs.

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

3.4 Permanent Failovers

When failovers are temporary, traffic migrates back to the primary path when it recovers. This migration can potentially throttle the sending rate because the sender returns to slow start's cwnd of one MTU. To avoid this slowdown, this author introduces a major potential change to SCTP – the concept of "permanent failover" using a *Change Primary Threshold (CPT)*. Permanent failover is based on a two-level threshold failover mechanism proposed in [34]. Once failover occurs, the sender can make the failover permanent (i.e., change the primary destination) if more than CPT heartbeat probes sent to the primary destination time out.

The specification for permanent failovers, shown in Figure 3.9, adds two new transitions to the finite state machine in Figure 3.7. While the association is in Phase II or III, if the primary destination's CPT threshold is exceeded, the primary destination is changed to the alternate destination currently in use. In Phase II, the association returns to Phase I with the new primary destination. In Phase III, however, the association remains in Phase III when a new primary destination is set; that is, changing the primary destination does not change the status of any destination, and thus the association remains in the dormant state.

We evaluate different CPT settings using the same methodology explained in Section 3.3, except here we only focus on PMR $= 0$ and the Dormant Hop behavior. We use these settings, because we have shown in Section 3.3 that PMR $= 0$ performs best and the dormant state behavior behavior is insignificant.

3.4.1 Symmetric Path Delays

Figure 3.10 plots the average 80MB file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss, a 90ms primary path RTT, and a 90ms alternate path RTT. When the alternate path loss rate is lower than the primary path loss rate, more aggressive permanent failover (i.e., lower CPT settings) dramatically improve performance. On the flip side, the performance is degraded relatively little when the alternate path loss rate is higher than that

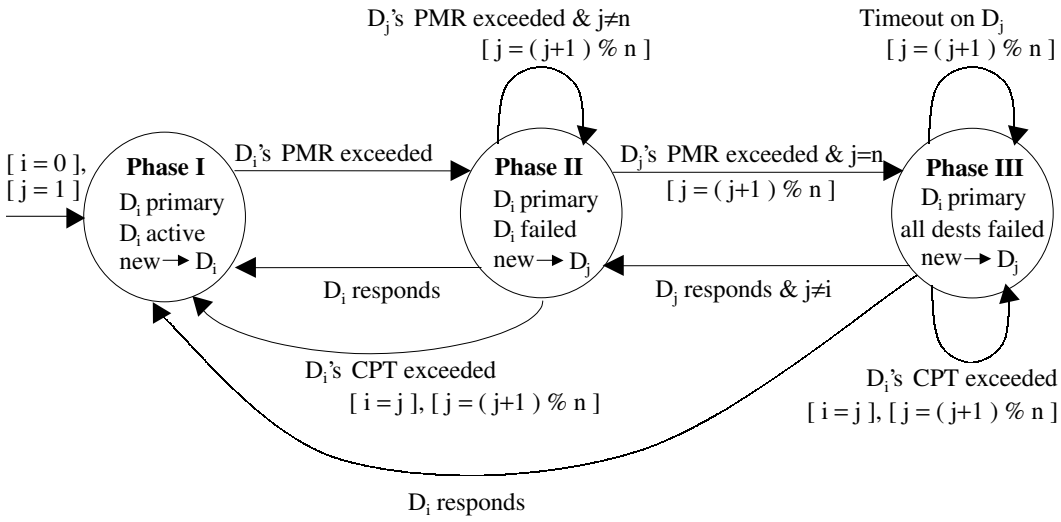

Figure 3.9: FSM for permanent failovers

of the primary path. For example, when the primary path loss rate is 5%, reducing CPT from 5 to 0 improves performance by as much as 88% and degrades performance by at most 9%.

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

3.4.2 Asymmetric Path Delays

Figure 3.10 shows that lowering CPT improves performance when path delays are symmetric, but what happens when path delays are asymmetric? Figure 3.11 plots the average 80MB file transfer for $\{3, 5, 8, 10\}$ primary path loss, a 90ms primary path

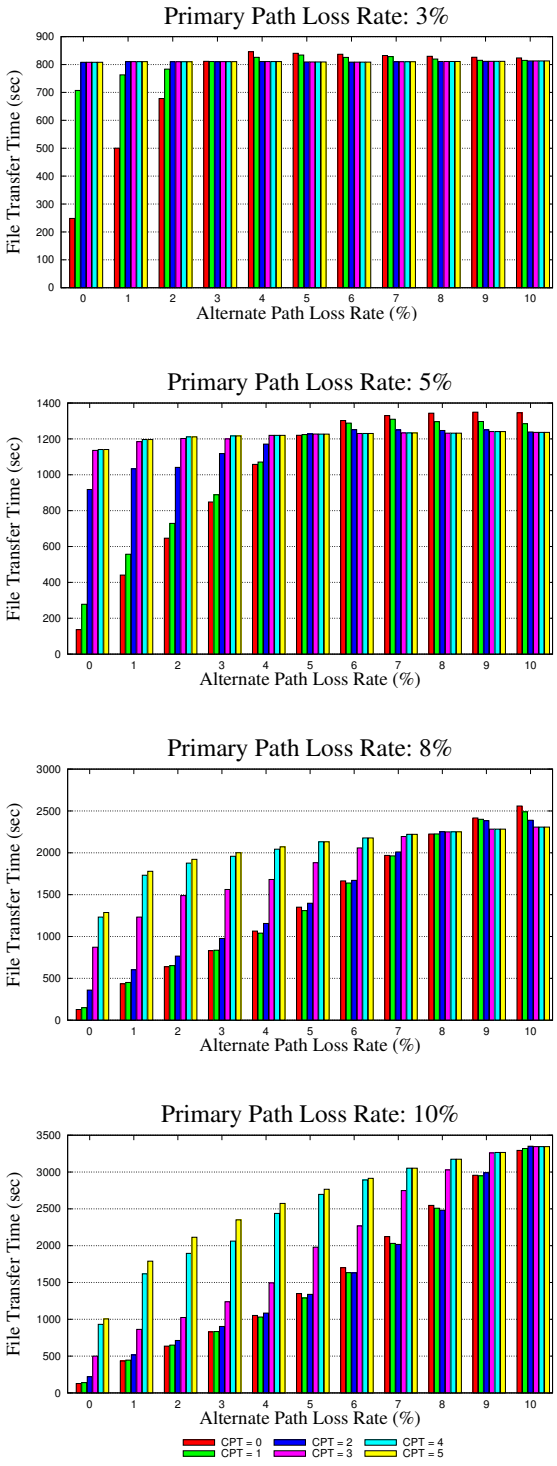

Figure 3.10: CPT evaluation: PMR = 0, 90ms primary path RTT, and 90ms alternate path RTT

RTT, and a 210ms alternate path RTT. These results show that lower CPT settings *may* improve performance, but only when the alternate path's loss rate is *much* lower than the primary path loss rate. Otherwise, aggressive permanent failover degrades performance significantly. For example, when the primary path loss rate is 5%, reducing CPT from 5 to 0 improves performance 76% and 23% for 0% and 1% alternate path loss rates, respectively. On the other hand, the performance suffers (by as much as 54%) for all other alternate path loss rates. Thus, to benefit from a change primary, the difference in path delays requires an alternate path loss rate low enough to offset the alternate path's relatively large delay.

Note that worst performance for aggressive permanent failover occurs when both paths' loss rates are similar. As expected, aggressive permanent failover's performance improves as the alternate path's loss rate decreases relative to the primary's. Surprisingly, however, aggressive permanent failover's performance also improves as the alternate path's loss rate increases relative to the primary's. As explained in Section 3.4.1, lower CPT settings allows an association to reduce the time spent on the higher loss rate path. Therefore, as the alternate path's loss rate increases, the association will spend less time on the alternate path, thereby reducing the negative effects of its longer RTT.

Recognize that the results in Figure 3.11 present only one perspective with respect to asymmetric path delays. We show in Figure 3.12 that if the association begins with the longer delay path as the primary, lower CPT settings are advantageous regardless of the paths' loss rates. When starting on the longer delay path, the sender has only to gain with more aggressive permanent failovers. If the alternate path's loss rate is lower, the association will spend more time on the shorter delay path. Otherwise, the association will spend more time on the longer delay path, which it would have anyway with higher CPT settings.

These results seem to demonstrate that failovers should be permanent only when

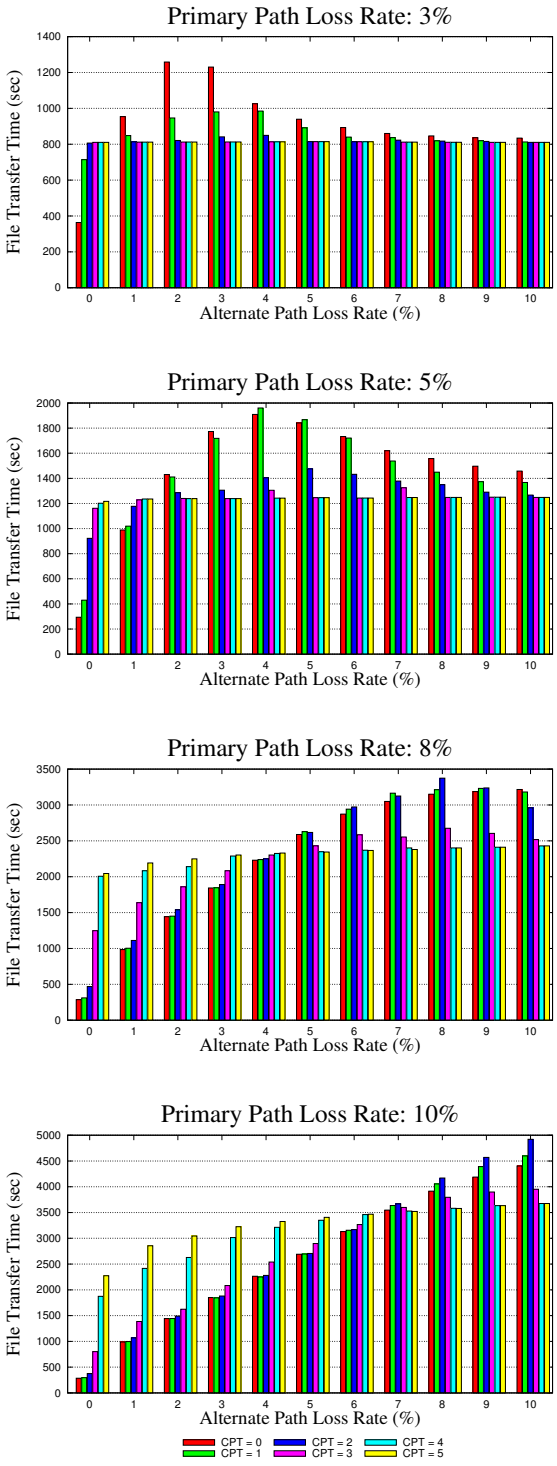

Figure 3.11: CPT evaluation: PMR = 0, 90ms primary path RTT, and 210ms alternate path RTT

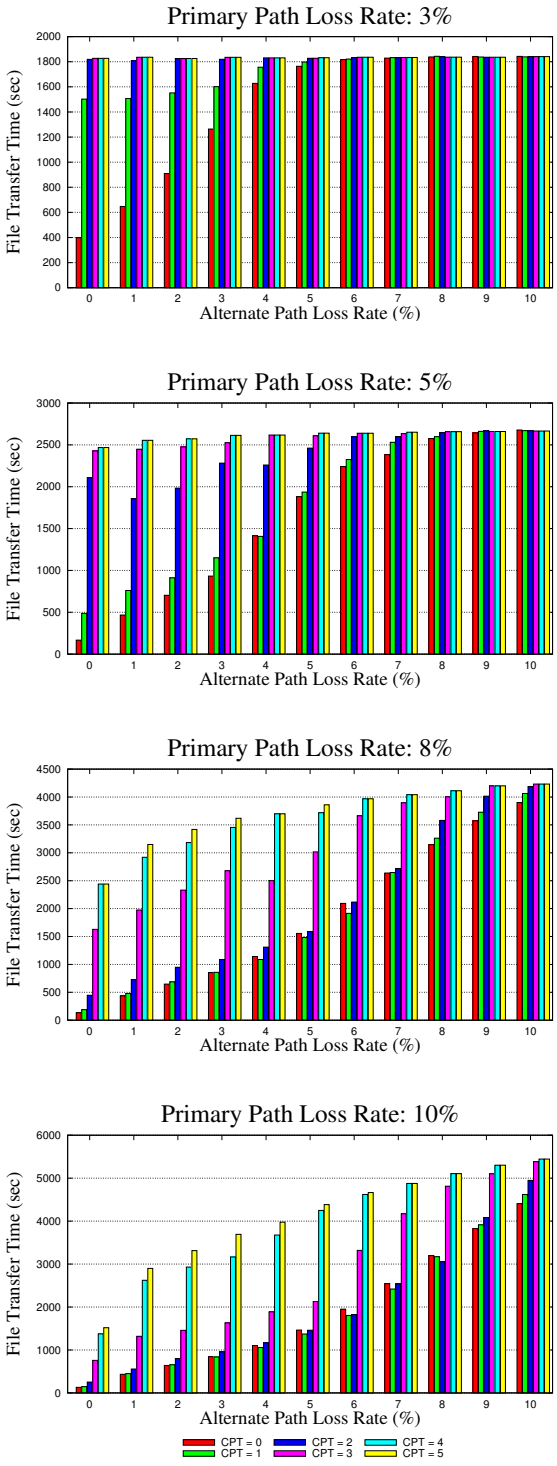

Figure 3.12: CPT evaluation: PMR = 0, 210ms primary path RTT, and 90ms alternate path RTT

the alternate path has a shorter RTT, but both RTT and loss conditions need to be considered in the decision process. A path with shorter RTT and higher loss rate may provide lower throughput than a path with longer RTT and lower loss rate. With an estimated RTT and loss rate (p) for each path, a sender can apply Padhye's simplified throughput model, $\frac{1}{RTT}\sqrt{\frac{3}{4p}}$, from [90] to compare paths and determine if a permanent failover would be advantageous. Future work is to develop a mechanism to measure the loss rate of an idle alternate path without introducing unnecessary overhead.

3.5 Conclusion

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

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

We realize that aggressive failover thresholds may draw concern. First, traditional thinking is that frequent traffic redirection is counter-productive, but that intuition comes from research in congestion-based routing algorithms. Migrating traffic back-and-forth on an end-to-end basis does not suffer the side-effects (e.g., reordering, inaccurate RTT estimates, etc.) that are introduced, for example, when an intermediate router "flip-flops" traffic between routes. These side-effects are avoided because each time a flow moves to a

new path, it begins from slow start as if it were a new flow. Furthermore, SCTP maintains path information (e.g., RTT, cwnd, ssthresh, etc.) per destination.

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

However, we argue that global failover synchronization is no worse than the existing well-known phenomenon of global TCP congestion control synchronization [21]. In both cases, synchronized timeouts cause synchronized slow starts and cwnd evolution, but in the case of failover, the cwnd evolution may occur on alternate paths that do not share bottlenecks. If so, a single flow's traffic migration appears no different than a new end-to-end flow, because each time a flow migrates to a new path, the flow begins from slow start with a cwnd of one MTU. In fact, since new flows may begin with a cwnd as large as four MTUs [10], a single flow's traffic migration is more conservative than a new flow.

On the other hand, if multiple flows do migrate to alternate paths that share a bottleneck, these flows will not disturb the network any more than a synchronized TCP timeout would. In both cases, multiple flows begin from slow start with cwnd = one MTU, and simultaneously grow their cwnd. The only difference being that in the case of failover, the cwnd evolution happens to be on a different path than where the synchronized timeout occurred. In any case, AQM techniques eliminate global synchronization [21], which also includes global failover synchronization.

Chapter 4

EFFECTS OF REACTIVE ROUTING

4.1 Introduction

In Chapters 2 and 3, we presented results that argued for a new retransmission policy and failover mechanism for multihomed transport protocols, such as SCTP. Those results and conclusions assumed wired network topologies, where proactive routing is generally used, and the amount of network layer routing traffic is independent of the transport layer traffic. As shown in Figure 4.1, hosts may be multihomed with multiple wireless interfaces using different access technologies (e.g., 802.11, GPRS, CDMA, etc.). Furthermore, wireless multihomed hosts may be connected in a mobile ad-hoc network (MANET) instead of a traditional fixed infrastructure. If a MANET uses a reactive routing protocol, such as the Dynamic Source Routing (DSR) [67] and Ad-hoc On-Demand Distance Vector (AODV) [92], then transmitting on an idle path will incur additional network layer routing overhead not found in fixed infrastructure. Therefore, this chapter reevaluates our conclusions about transport layer multihoming in the context of MANETs with reactive routing protocols, where transport layer behavior does affect the amount of routing traffic. We investigate the effects of reactive routing on retransmission policies and failover thresholds.

We show that our conclusions for proactive routing hold for reactive routing under certain circumstances also, such as when route cache lifetimes are longer than two consecutive timeouts. However, we find that the extra overhead experienced with short route cache lifetimes (necessary in highly mobile ad-hoc networks) and long route discovery delays causes AllRtxSame to outperform FrSameRtoAlt. We also find that when

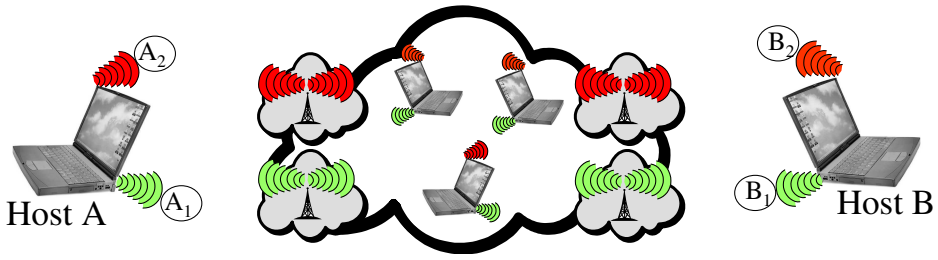

Figure 4.1: Example multihoming topology in a wireless environment

AllRtxSame is preferable, a *slightly* less aggressive failover threshold, PMR = 1 (i.e., two consecutive timeouts), should be used. We conclude with a decision tree that summarizes the results in this chapter by suggesting a retransmission policy and failover threshold based on expected network conditions. This tree can assist a network engineer fine tune a multihomed transport protocol for a particular network environment.

Section 4.2 details the methodology used in our investigation. Section 4.3 and 4.4 present the effects of reactive routing overheads on the retransmission policies and failover thresholds, respectively. Section 4.5 concludes this chapter.

4.2 Analysis Methodology

We simulate a wired network to focus on the effects of reactive routing overheads without also dealing with the side-effects and issues that wireless links and mobility entail. To simulate the added delay overhead introduced by reactive routing, this author modified the ns-2 SCTP module to track idle time of each path. Whenever an SCTP endpoint transmits a TPDU on a path that has been idle longer than the *route cache lifetime*, the sending function introduces an extra delay (i.e., the *route discovery delay*) before doing a transmission. Hence, the simulation models the delay overhead to the transport layer, but does not model the actual routing traffic that would occur. Such detailed network layer and link layer modeling is beyond the scope of this dissertation.

Perkins and Royer [92] use a fixed 3000ms cache lifetime in their AODV simulations, which we use as guidance. We simulate route cache lifetimes of 1200ms, 3200ms, and 7200ms to allow a route to be cached slightly longer than one, two, and three consecutive timeouts at the transport layer, respectively. We also simulate a 500ms route cache lifetime. Since the minimum RTO at the transport layer is 1 second, a 500ms route cache effectively forces a route discovery overhead for every single timeout. We simulate route discovery delays of 100ms, 250ms, and 500ms, which represent a reasonable range for MANETs up to 1000 nodes according to [92]. We also include 1000ms route discovery delays in our simulations as a worst case scenario. Since networks with reactive routing use the same path for forward and reverse traffic, and reverse path information is normally maintained while forward paths are setup [92], we simulate route discovery delays only in the forward path; that is, transport layer ack traffic never experiences route discovery delays.

Figure 4.2 illustrates the network topology. We simulate RTTs of 90ms, 210ms, and 1040ms, which are reasonably low delays for a lightly loaded MANET of 50-100 nodes [44].

To observe long term averages, we simulate 80MB file transfers and introduce uniform loss (0-10% each way) at the core links of each path. Seven input parameters for each simulation are:

1. primary path's loss rate
2. alternate path's loss rate
3. alternate path's core link delay
4. retransmission policy
5. PMR setting
6. route cache lifetime

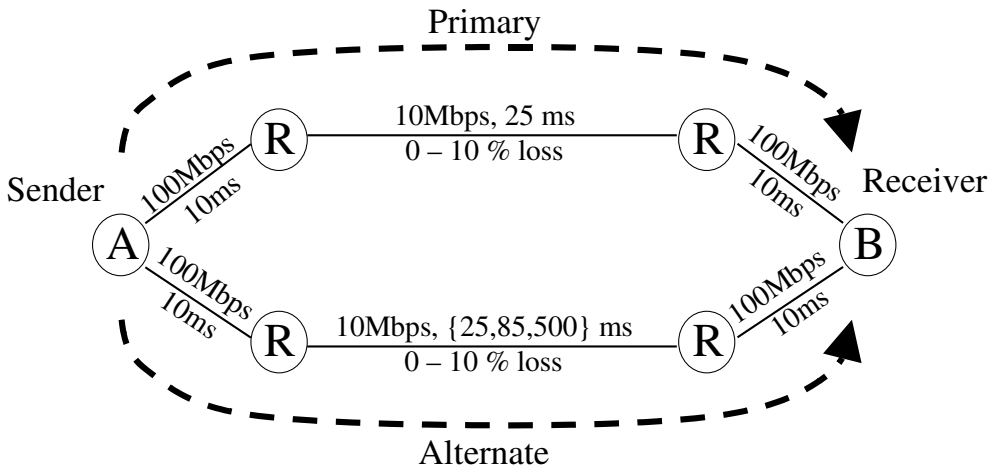

Figure 4.2: Simulation network topology with random loss, 90ms primary path RTT, and $\{90, 210, 1040\}$ms alternate path RTT

7. route discovery delay

4.3 Retransmission Policies

We have shown in Chapter 2 that sending all retransmissions to an alternate destination (AllRtxAlt), as specified in RFC2960, offers the benefit of some successful data transmission during failure scenarios (i.e., when the primary path becomes unreachable), but drastically suffers when retransmissions are lost in non-failure scenarios. Due to the idle nature of alternate paths, their round-trip time (RTT) measurements are infrequent, and thus the sender's retransmission timeout (RTO) values used for alternate paths are generally conservative (i.e., large). Sending all retransmissions to the primary destination (AllRtxSame) alleviates this problem, but suffers when the primary destination becomes unreachable, or when the primary path's loss conditions are significantly worse than the alternate's. Chapter 2 shows that over proactive network layer routing, where the network overhead is the same for all paths, a hybrid policy (FrSameRtoAlt) can achieve better performance. Since timeouts tend to occur during high loss or failure conditions,

FrSameRtoAlt is able to achieve the benefits of AllRtxAlt under such situations. Meanwhile, AllRtxSame and FrSameRtoAlt perform equally well under non-failure scenarios with low primary path loss rates.

In this section, we evaluate whether using an alternate path for timeout retransmissions continues to be advantageous when delays introduced by network layer reactive routing protocols are taken into consideration. The extra route discovery delay introduced when using an idle alternate path discourages using alternate paths for retransmissions.

Since we have already established in Chapter 2 that AllRtxAlt is worse than FrSameRtoAlt for too much use of the alternate path, and AllRtxAlt will incur that much more overhead in a reactive routing environment, we exclude AllRtxAlt from our evaluation here. We only compare AllRtxSame versus FrSameRtoAlt in this section. All our simulations use our Multiple Fast Retransmit algorithm (see Section 2.4.3) to help reduce the number of timeouts.

Before presenting the results, we model in Figure 4.3 generic timelines (with key points a, b, c, d) for AllRtxSame and FrSameRtoAlt when a timeout occurs. We assume that a timeout occurs on the primary path (from A_1 to B_1) at some arbitrary point in the association. When the retransmission timeout (RTO) expires, n TPDUs are outstanding and must be retransmitted. The retransmission policy used affects the performance in the following way.

- **AllRtxSame** The n outstanding TPDUs are retransmitted on the primary path. If the primary path's idle time from point a to point b is greater than the route cache lifetime, then the first TPDU retransmitted experiences a route discovery delay. Once all retransmissions have been sent (point c), the sender resumes sending new data on the primary path. Future timeout events follow the same sequence of events.

- **FrSameRtoAlt** The n outstanding TPDUs are retransmitted on the alternate path (from A_2 to B_2). We assume that these retransmissions are the first TPDUs sent on

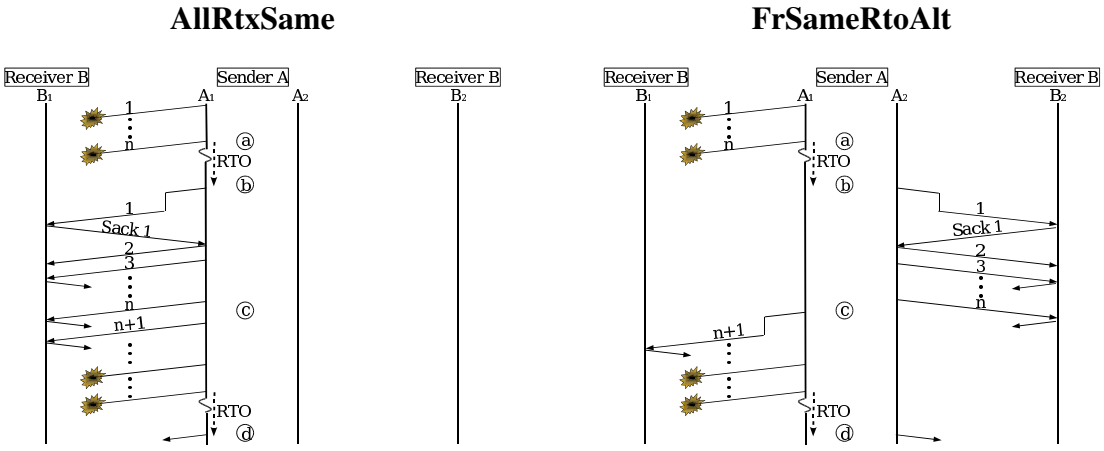

Figure 4.3: AllRtxSame vs FrSameRtoAlt in a timeout scenario with reactive routing

the alternate path, and thus, the first TPDU retransmitted experiences a route discovery delay. Once all retransmissions have been sent (point c), the sender resumes sending new data on the primary path.[1] If the primary path's idle time from point a to point c is greater than the route cache lifetime, then another route discovery delay occurs. The next timeout that occurs on the primary (point d) follows the same sequence of events. However, the route discovery delay on the alternate path may be avoided if the alternate path's idle time from point c to point d is less than the route cache lifetime.

Consequently, AllRtxSame experiences at most one route discovery delay per timeout, while FrSameRtoAlt experiences up to two route discovery delays per timeout. Furthermore, AllRtxSame never experiences more route discovery delays than FrSameRtoAlt. We therefore expect AllRtxSame to outperform FrSameRtoAlt, but the performance gap should get smaller as the route cache lifetime increases.

[1] RFC2960's ambiguity allows two interpretations of when a sender may begin sending new data in loss recovery scenarios. One interpretation is that a sender must transmit all pending retransmissions before sending any new data. The second interpretation is that new data is only blocked by pending retransmissions on the same path. Both allow some level of parallelism on both paths, but we evaluate the stricter interpretation. The latter interpretation would favor FrSameRtoAlt.

4.3.1 Symmetric Path Delays

Of the route cache lifetimes we simulated, we find (as expected) 1200ms to be the worst setting for FrSameRtoAlt with respect to its relative performance against AllRtxSame. Since the primary path's RTO generally stays fairly close to the recommended minimum of 1 second, the time elapsed in Figure 4.3 from point a to b is generally shorter than 1200ms. Hence, most timeouts with AllRtxSame do not incur a route discovery delay. On the other hand, FrSameRtoAlt often experiences two route discovery delays per timeout, because we expect the idle times in Figure 4.3 from point a to point c and from point c to point d are likely to be more than 1200ms.

In Figures 4.4 and 4.5, we plot the average file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss rates, a 90ms primary path RTT, a 90ms alternate path RTT, and a 1200ms route cache lifetime. Figure 4.4 presents results for 100ms and 250ms route discovery delays, while Figure 4.5 presents results for 500ms and 1000ms route discovery delays. We find that with short route discovery delays (less than 250ms), AllRtxSame and FrSameRtoAlt perform relatively similar to that reported for proactive routing environments in Section 2.5.2. That is, both policies perform similarly, with FrSameRtoAlt performing slightly better due to its parallelism of new transmissions on the primary path and retransmissions on an alternate path. These route discovery delays are short enough that even with frequent timeouts (as experienced at high loss rates), FrSameRtoAlt's benefits over AllRtxSame remain. For larger route discovery delays (500-1000ms), the results favor FrSameRtoAlt for low primary path loss rates ($\leq 3\%$) and AllRtxSame for high loss rates ($> 3\%$). Timeouts are infrequent at low loss rates, so route discovery delays incurred on the alternate path with FrSameRtoAlt are irrelevant. High loss rates, on the other hand, experience timeouts frequently and are thus negatively affected by large route discovery delays incurred on the alternate path by FrSameRtoAlt.

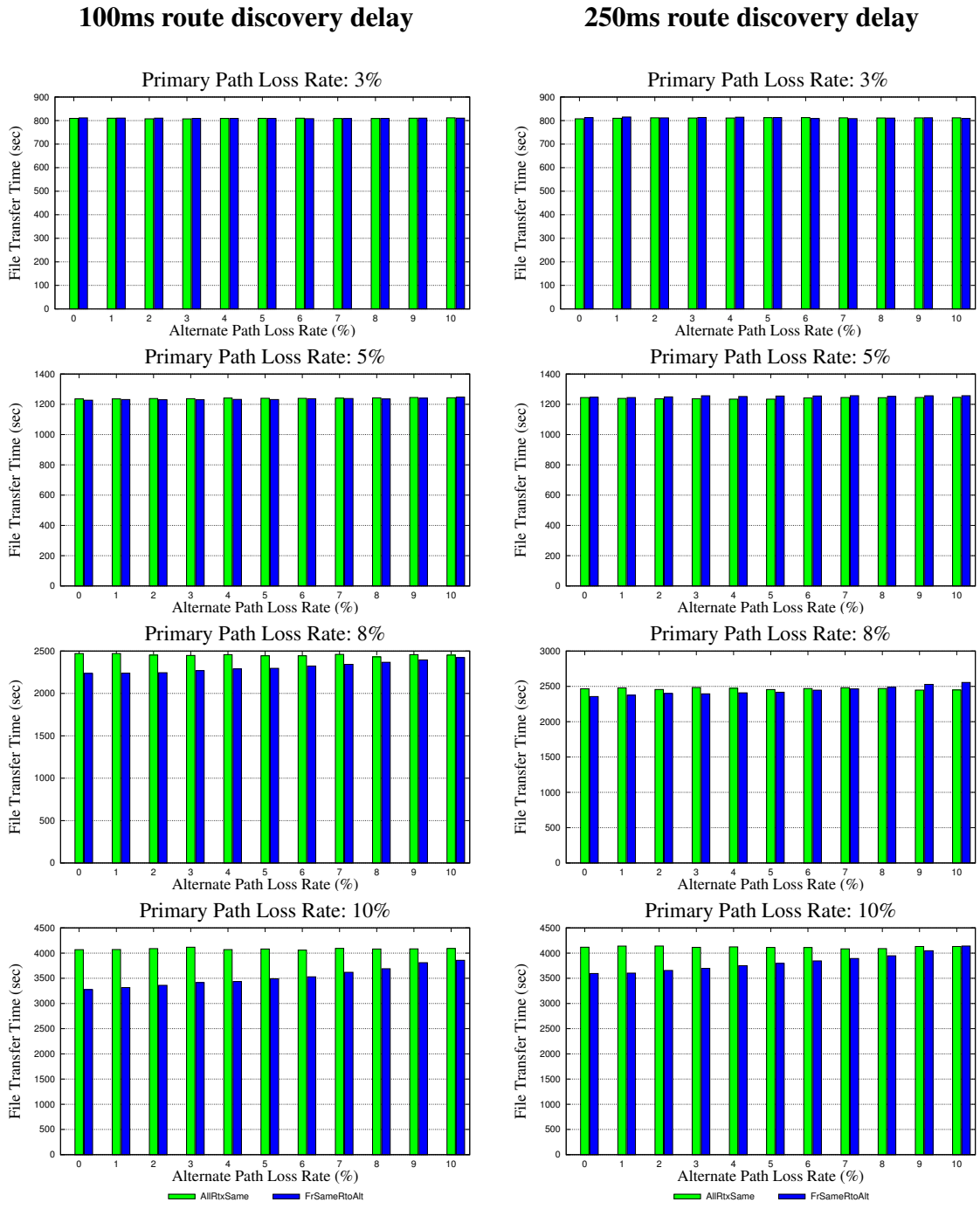

Figure 4.4: AllRtxSame and FrSameRtoAlt at $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 90ms alternate path RTT, 1200ms route cache lifetime, and $\{100, 250\}$ms route discovery delay

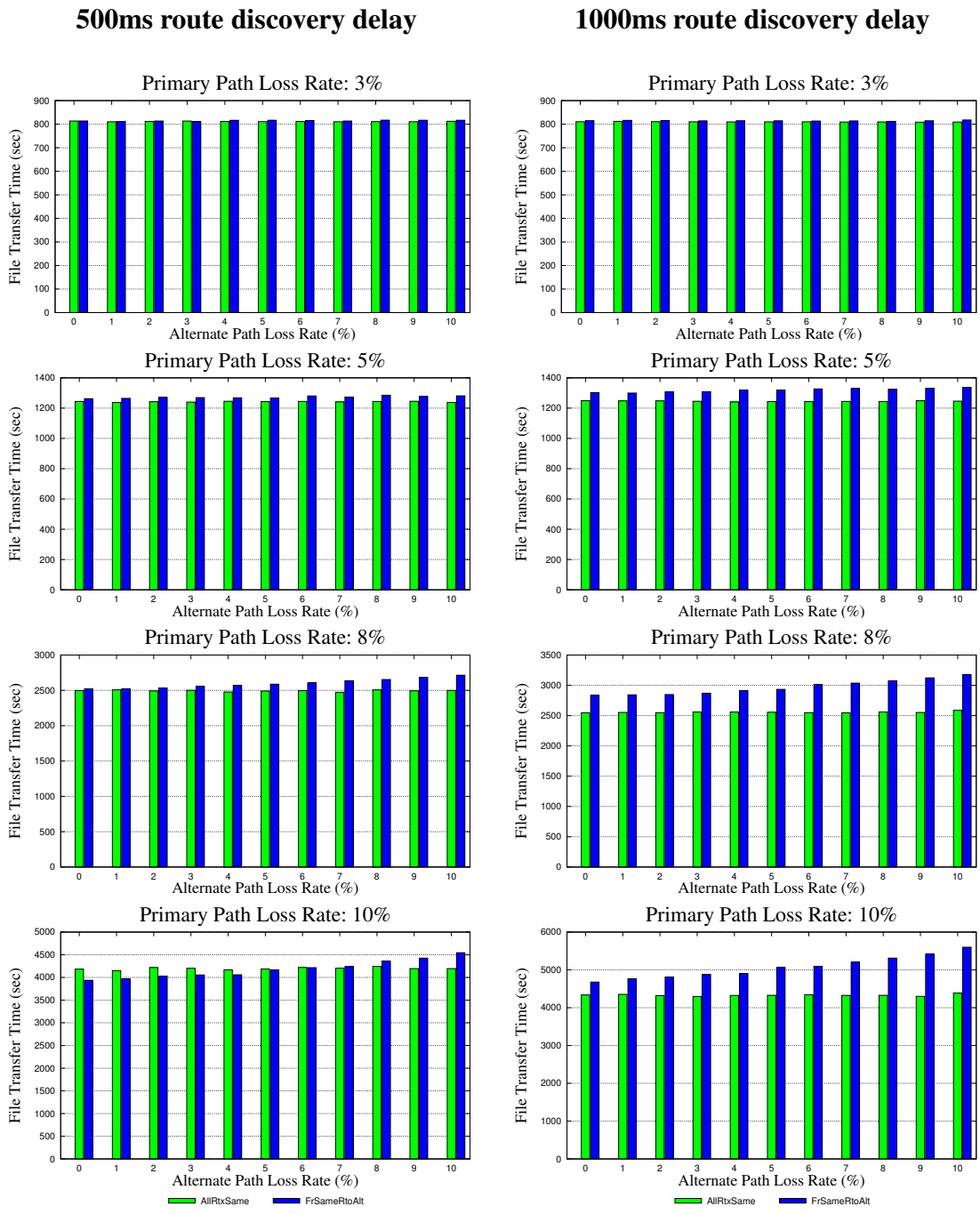

Figure 4.5: AllRtxSame and FrSameRtoAlt at $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 90ms alternate path RTT, 1200ms route cache lifetime, and $\{500, 1000\}$ms route discovery delay

Figure 4.6 shows that reducing the route cache lifetime to 500ms reduces the performance gap between FrSameRtoAlt and AllRtxSame. It is not that FrSameRtoAlt benefits from a shorter cache lifetime, but instead, AllRtxSame suffers. Since 500ms is less than the minimum RTO of 1 second (i.e., point a to point b in Figure 4.3), AllRtxSame experiences a route discovery delay with every timeout retransmission. Thus, FrSameRtoAlt only incurs one more route discovery delay per timeout than AllRtxSame.

Increasing the route cache lifetime beyond 1200ms affects FrSameRtoAlt's performance more than AllRtxSame's. Figure 4.7 shows that with a 3200ms route cache lifetime, both policies' performance improve, but FrSameRtoAlt improvement is more significant. As a result, FrSameRtoAlt performs comparable to AllRtxSame with route discoverys up to 500ms. With a 1000ms route discovery delay, FrSameRtoAlt performs only slightly worse. That is, FrSameRtoAlt performs at most 6% worse than AllRtxSame when both paths have the same loss rate. A 7200ms route cache lifetime (results not shown) improves FrSameRtoAlt's performance even further.

To summarize, as route cache lifetimes increase, the performance of both policies improve, but the performance improvements are unequal. Route cache lifetime increases that are less than the minimum RTO are more beneficial to AllRtxSame than FrSameRtoAlt, because the frequency of cache misses at point b in Figure 4.3 reduces for AllRtxSame, but not for FrSameRtoAlt. On the other hand, route cache lifetime increases that are above the minimum RTO are more beneficial to FrSameRtoAlt than AllRtxSame. Once the route cache lifetime increases beyond the minimum RTO, the frequency of cache misses at points b, c, d in Figure 4.3 reduces for FrSameRtoAlt, but remain unchanged for AllRtxSame. AllRtxSame only benefits from a route cache lifetime larger than the minimum RTO during consecutive timeout events.

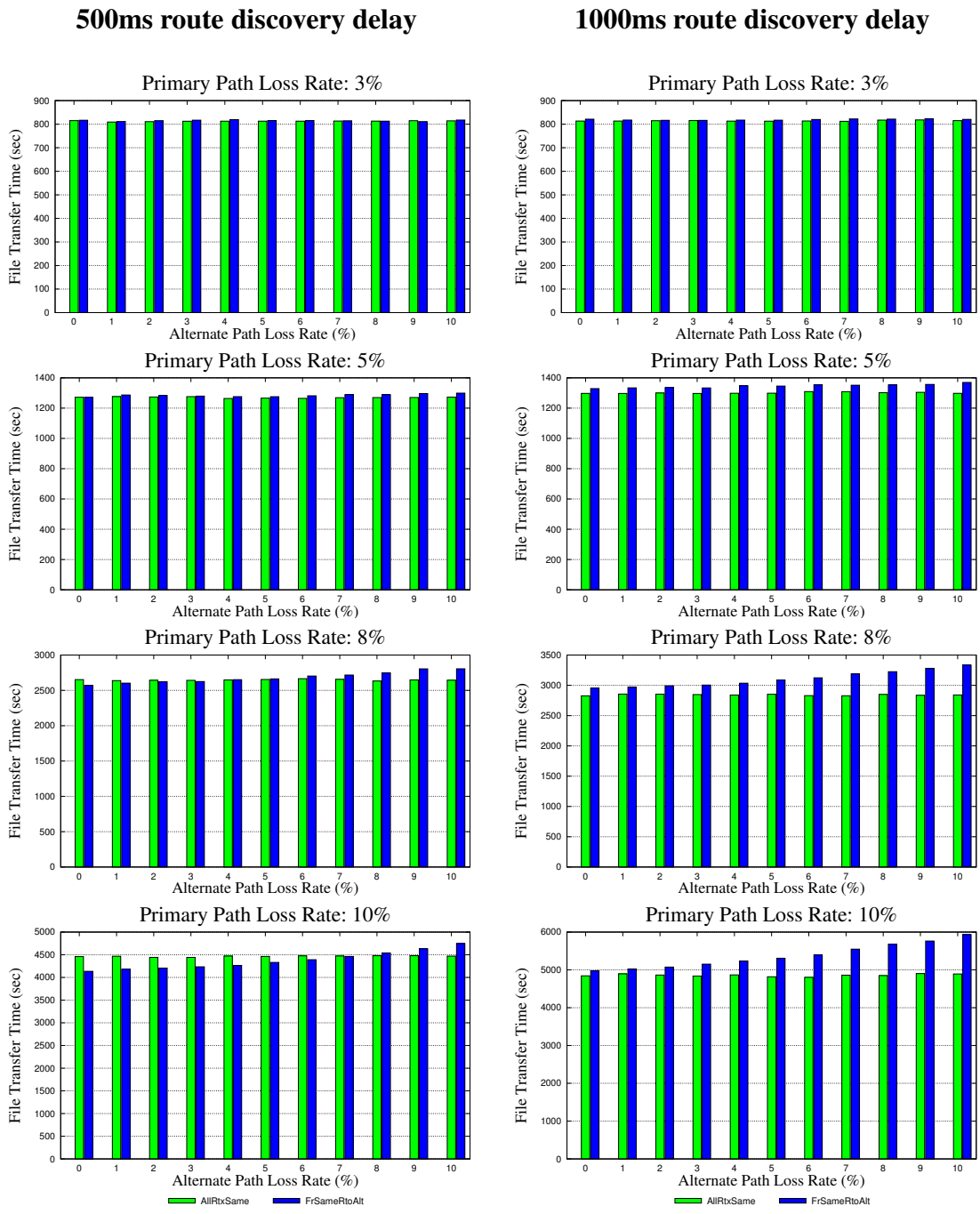

Figure 4.6: AllRtxSame and FrSameRtoAlt at $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 90ms alternate path RTT, 500ms route cache lifetime, and $\{500, 1000\}$ms route discovery delay

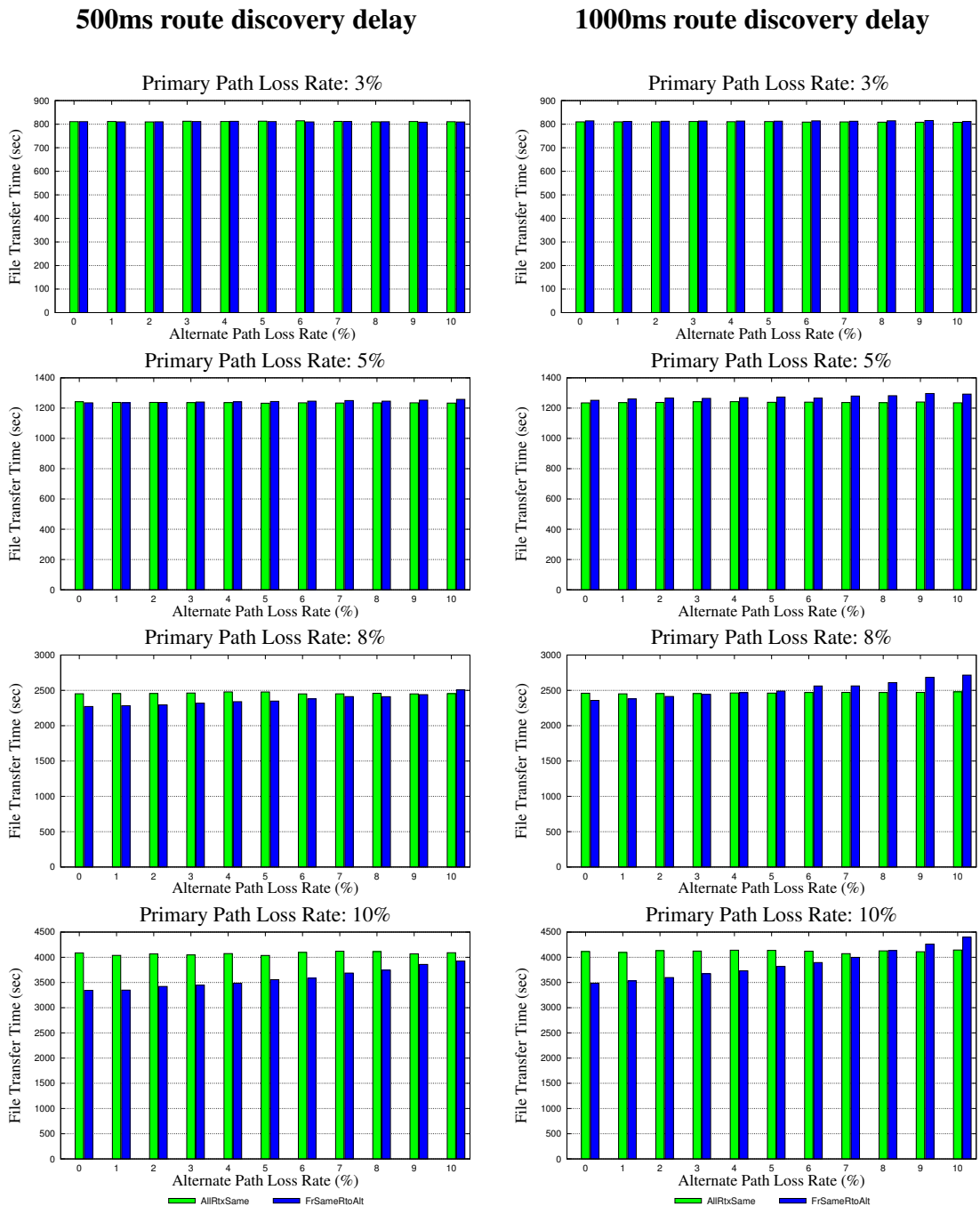

Figure 4.7: AllRtxSame and FrSameRtoAlt at $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 90ms alternate path RTT, 3200ms route cache lifetime, and $\{500, 1000\}$ms route discovery delay

4.3.2 Asymmetric Path Delays

The trends of Section 4.3.1 remain the same for asymmetric path delays (results not shown). As explained in Section 2.5, FrSameRtoAlt's performance benefits do diminish as path delay asymmetry increases, but the rate of diminishing returns is small. We find that with reactive routing, increasing the alternate path's RTT from 90ms to 210ms (more than double) degrades FrSameRtoAlt's performance relative to AllRtxSame by only 3-7%. Since Section 2.5 shows that highly asymmetric paths do not favor FrSameRtoAlt even with proactive routing, we do not evaluate FrSameRtoAlt's performance a 1040ms alternate path RTT in a reactive routing environment. Surely, the added overhead of route discovery delays on idle paths would make FrSameRtoAlt less favorable.

We conclude this section on retransmission policies for reactive routing with a tree representing our results (see Figure 4.8). This decision tree provides some suggestions to a network engineer for choosing a retransmission policy based on the expected conditions of the network environment. We recognize that we are using simplistic simulation models and discrete network parameters to suggest exact values for real, complex network conditions, so our results/suggestions are only meant to provide intuition and should be used with caution. Furthermore, no recommendation can be made for values not covered in the tree (e.g., $2*$primaryRTT $<$ alternateRTT $<$ $10*$primaryRTT).

4.4 Failover Thresholds

We have shown in Chapter 3 that reducing the failover threshold, Path.Max.Retrans (PMR), improves performance in both failure and non-failure scenarios. More aggressive failover (i.e., lower threshold) decreases failure detection time at the cost of increased spurious failovers, but we found that spurious failovers do not degrade overall performance and often improve goodput. Our conclusion was that PMR $= 0$ (i.e., failover after a single timeout) provides the highest throughput for bulk transfer. These results and supporting arguments assumed the FrSameRtoAlt policy was used since Chapter 2 established it as the best performing retransmission policy for the Internet. However, as shown

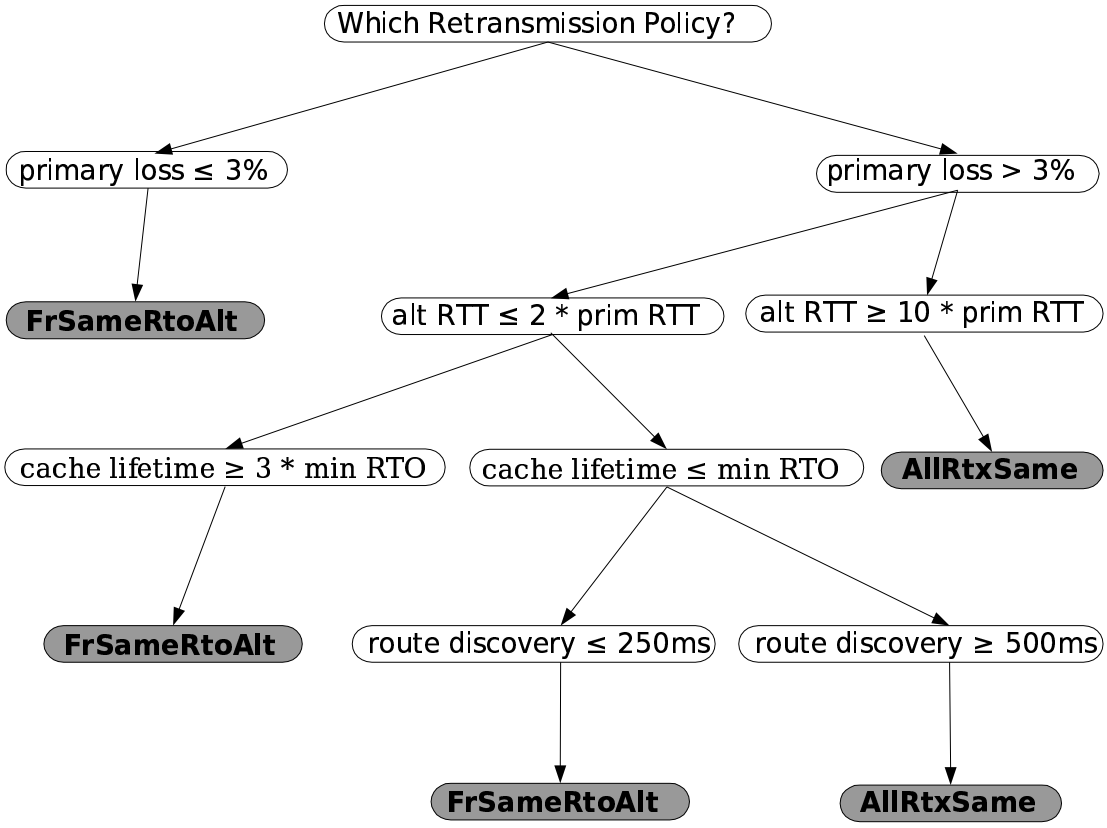

Figure 4.8: Retransmission policy suggestions

in Section 4.3, both AllRtxSame and FrSameRtoAlt have merits in a reactive routing environment, depending on the conditions. Thus, with reactive routing, PMR settings need to be evaluated for both retransmission policies.

Section 3.3.7 explained why the most aggressive failover threshold (PMR $= 0$) provides the best performance if FrSameRtoAlt is used. The underlying reason is that if an alternate path is used for timeout retransmissions, then a sender benefits by remaining on the alternate path for new data transmissions until the primary responds. We expected this argument to also hold for reactive routing environments. To confirm our intuition, we evaluated PMR settings with FrSameRtoAlt with our reactive routing model. Our results (not shown) confirm that even with reactive routing, if FrSameRtoAlt is used, failing over on a single timeout provides the best performance.

The remainder of this section evaluates failover thresholds when AllRtxSame is used in a reactive routing environment.

4.4.1 Symmetric Path Delays

Figure 4.9 presents the average file transfer time for $\{3, 5, 8, 10\}\%$ primary path loss rates, a 90ms primary path RTT, a 90ms alternate path RTT, a 1200ms route cache lifetime, and a 1000ms route discovery delay. We find that even with AllRtxSame, PMR $= 0$ provides the best performance. This aggressive failover setting does not always improve performance, but at worst, it degrades performance by 5%. However, as shown by the plots for 8% and 10% primary path loss rates, PMR $= 0$ can potentially improve performance significantly . For example, PMR $= 0$ improves performance by as much as 68% when the primary path's loss rate is 10%.

We find these results interesting because the detailed analysis provided in Section 3.3.7 is specific to the FrSameRtoAlt policy, and thus should not apply here. Our argument with FrSameRtoAlt was that if the sender is using the alternate path for timeout retransmissions anyway, the sender might as well also use the alternate path for new data until the primary destination responds. On the other hand, AllRtxSame only uses the

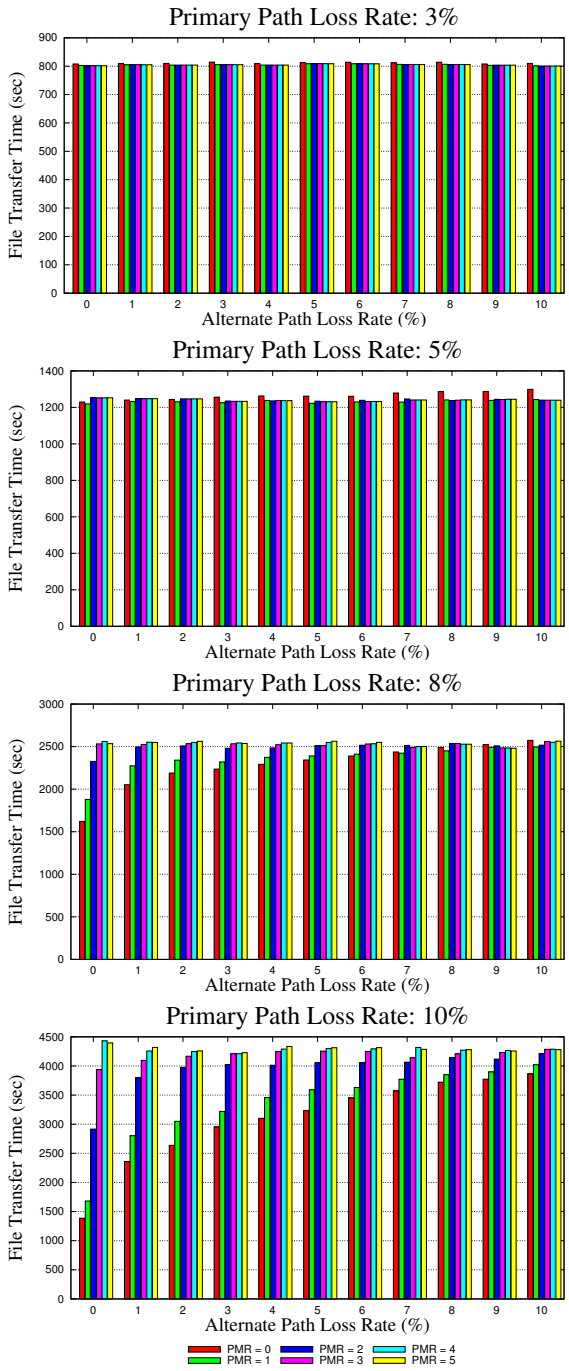

Figure 4.9: PMR settings with $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 90ms alternate path RTT, 1200ms route cache lifetime, and 1000ms route discovery delay

alternate path when a failover occurs. So in this case, a spurious failover poses a higher risk for hurting performance. If the alternate path has a longer delay (which it does in this scenario due to route discovery delays) and/or higher loss rate, spurious failover may degrade performance.

So then why does PMR $= 0$ continue to provide better performance when AllRtxSame is used? Although the specifics of the scenarios in Section 3.3.7 do not exactly apply when AllRtxSame is used, we find that the general argument still applies – the primary path's quality influences performance more than the alternate path's quality. In other words, spuriously failing over to an alternate path may momentarily degrade performance, but the failover is cancelled as soon as the primary destination responds. However, when traffic remains on the primary path after a timeout, subsequent losses completely stall the transfer. Moreover, at low primary path loss rates, the alternate path's quality becomes somewhat irrelevant as spurious failovers are too infrequent to significantly impact performance.

4.4.2 Asymmetric Path Delays

When path delays are asymmetric, the potential penalty for spurious failovers with AllRtxSame increases. We find that increasing the alternate path's RTT to slightly more than double (210ms) *does* affect goodput enough (degradation of more than 5%) to discourage using PMR $= 0$ with AllRtxSame and long route discovery delays. Our results (not shown) reveal that with such path delay asymmetry, PMR $= 1$ is the best failover threshold for AllRtxSame.

We also find (unexpectedly) that increasing the alternate path's RTT to 1040ms does not change this result (see Figure 4.10). With a 1200ms route cache lifetime, the extreme path delay asymmetry (including the route discovery delays) leaves no advantage for failover on a single timeout. Not only does PMR $= 0$ not provide any benefit, it degrades performance by as much as 49%. Interestingly, however, PMR $= 1$ now provides the best performance. This PMR setting causes negligible goodput degradation ($\leq 2\%$) at

worst, and fails over after only two consecutive timeouts (i.e., 3 seconds with RFC2960's recommend settings).

At first, we thought that the 1200ms route cache lifetime was causing PMR $= 1$ to perform best. After all, a route discovery is inevitable after the second consecutive timeout, regardless of which path is used at that point. The reason is that the second timeout duration is 2 seconds in the best case, which means that both paths' routes would have been flushed from the cache by then. With this being the case, the route discovery delays are no longer a factor when comparing performance of PMR settings greater than 0. Therefore, the goodput results for PMR settings greater than 0 are similar.

Following this logic, we would expect PMR $= \{0, 1\}$ to perform poorly, and PMR $= 2$ to be the best setting when the route cache lifetime is set to 3200ms. However, the results in Figure 4.10 show that this expectation does not explain the results for a 1200ms cache lifetime, because with a 3200ms cache lifetime, PMR $= 1$'s performance remains unchanged and continues to perform best. We realize that PMR $= 1$ performs best not because of the route cache lifetime, but because reaching that failover threshold is so rare that it does not significantly affect performance. PMR $= 0$'s performance is improved because a longer cache lifetime causes fewer heartbeat probes on the primary path to be delayed due to route discoverys. As a result, spurious failovers are cancelled more quickly, and an association spends less time on the longer delay alternate path.

4.5 Conclusion

We have investigated the effects of reactive routing on retransmission policies and failover thresholds. We surprisingly found that our conclusions for proactive routing hold in many circumstances for reactive routing. Only when route cache lifetimes are short ($<$ minimum RTO) did we find scenarios where proactive routing conclusions do not hold. We found with short route cache lifetimes and long route discovery delays, AllRtxSame provides better performance than FrSameRtoAlt. We also found that when AllRtxSame

Figure 4.10: PMR settings with $\{3, 5, 8, 10\}\%$ primary path loss, 90ms primary path RTT, 1040ms alternate path RTT, $\{1200, 3200\}$ms route cache lifetime, and 1000ms route discovery delay

is recommended, a *slightly* less aggressive failover threshold, PMR $= 1$ (i.e., two consecutive timeouts), should be used.

The decision tree in Figure 4.11 summarizes all the results for this chapter by suggesting a retransmission policy and failover threshold based on expected network conditions. Left branches require stricter criteria from network expectations, but are able to suggest aggressive use of alternate paths. Right branches suggest less aggressive use of alternate paths due to their looser network requirements. For example, if we expect a given network to have less than a 3% loss rate, then we can use aggressive settings: FrSameRtoAlt with PMR $= 0$. On the other hand, if we have no idea what to expect from a network, following the rightmost path down the tree suggests that AllRtxSame and PMR $= 1$ be used.

Again, we recognize that we are using simplistic simulation models and discrete network parameters to suggest exact values for real, complex network conditions, so our results/suggestions are only meant to provide intuition and should be used with caution. Furthermore, our evaluation only focuses on the performance from the application's perspective. Triggering route discoverys due to aggressive use of alternate paths may place a significant traffic burden on the network layer which we do not simulate. Therefore, we recommend as future work that this work be reevaluated with a model that accounts for the extra network layer traffic introduced by reactive routing protocols.

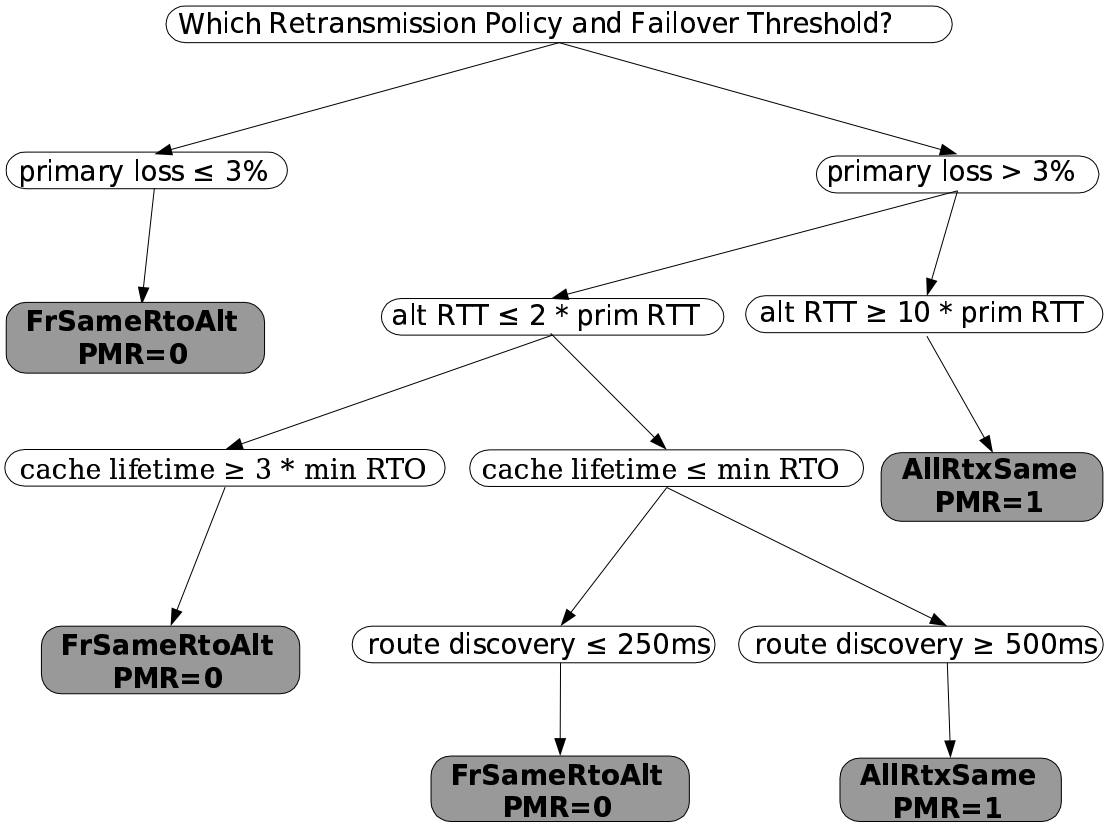

Figure 4.11: Reactive routing summary

Chapter 5

SCTP NS-2 SIMULATION MODULE

5.1 Introduction

This chapter describes the SCTP module developed for the network simulator, ns-2 [19]. Ns-2 is a discrete event simulator developed for networking research, and its strength lies in areas of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Janardhan Iyengar and I extended ns-2 to support SCTP and its Partial Reliability Extension (PR-SCTP) [102]. We worked together on the initial design and development, but after the first few months of development, I took primary responsibility of code maintenance and development.

5.2 Implementation

Ns-2 is implemented in C++ with a modular framework for extensibility. *Agents* are the basic building blocks for building a protocol stack. They represent endpoints where network-layer packets are constructed or consumed, and can be extended, using C++ derived classes, to build additional functionality. Analogous to ns-2's TCP implementation, we implemented SCTP as a derived Agent class. We further derived classes from the SCTP Agent class to implement experimental extensions of SCTP.

The SCTP agents are two-way agents, which means they implement both sender and receiver functionality. The SCTP agents currently supported are:

- `Agent/SCTP`
- `Agent/SCTP/HbAfterRto`

- `Agent/SCTP/MultipleFastRtx`
- `Agent/SCTP/Timestamp`
- `Agent/SCTP/MfrHbAfterRto`
- `Agent/SCTP/MfrTimestamp`

`Agent/SCTP` is the base SCTP agent that supports the core functionality specified in RFC2960 and the SCTP Implementer's Guide [101]. It also supports the PR-SCTP extension [102][1]. `Agent/SCTP/HbAfterRto`, `Agent/SCTP/MultipleFastRtx`, and `Agent/SCTP/Timestamp` implement the Heartbeat After RTO, Multiple Fast Retransmit, and Timestamp extensions, respectively (descriptions in Section 2.4). The Multiple Fast Retransmit extension is combined with the Heartbeat after RTO extension in `Agent/SCTP/MfrHbAfterRto` and with the Timestamp extension in `Agent/SCTP/MfrTimestamp`.

Since we are dealing with a simulation environment, some protocol details can be abstracted away. For example, much of the specification's error cases: malformed packets, unexpected events, etc., are not simulated. We also simplify other procedures such as association establishment and association shutdown. The SCTP agent establishes an association using a four-way handshake, but the handshake is simplified and does not strictly conform to RFC2960. For example, the handshake does not exchange tags, and the INIT and COOKIE-ECHO chunks are not used to update the RTT. Instead, RTT estimation begins with the first DATA chunk. In addition, the SCTP agent currently does not perform graceful shutdown. The association is abruptly terminated when the simulated connection ends.

Since the underlying infrastructure of ns-2 does not support multiple interfaces for a single node, including SCTP's multihoming feature was non-trivial. Prior to my

[1] Our PR-SCTP implementation is not fully compliant with RFC3758, because the PR-SCTP code was last updated when PR-SCTP was still an IETF draft.

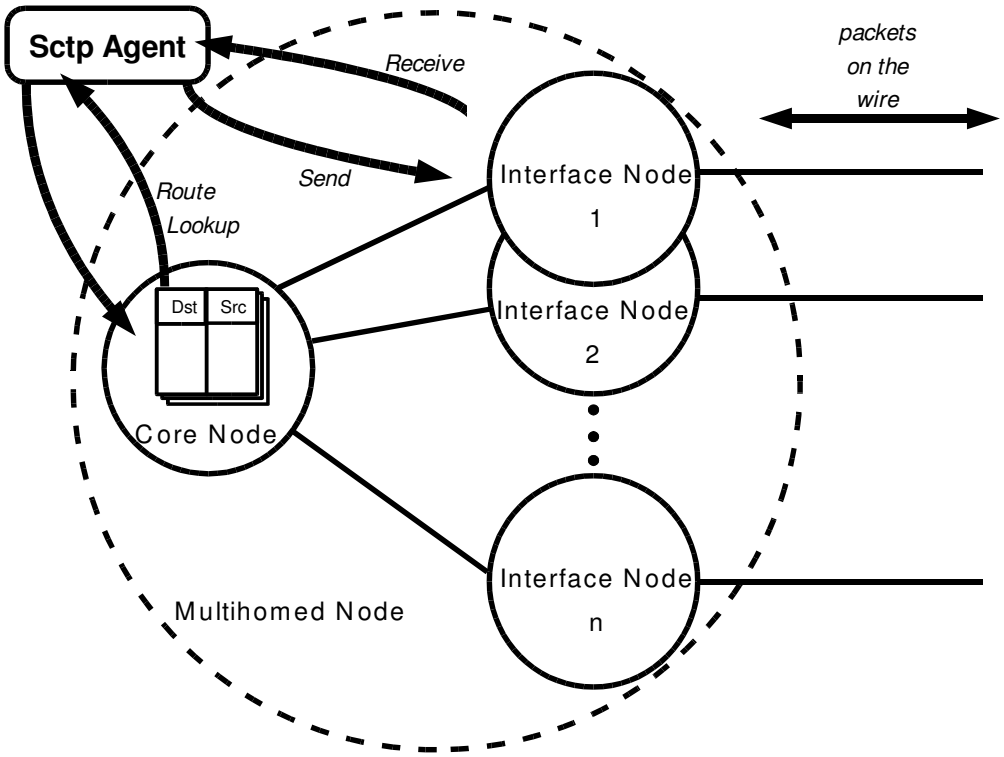

Figure 5.1: Logical representation of a multihomed node

work, an ns-2 node could have one and only one address. To get around this limitation, our approach allows the general support for logically multihoming nodes that have a multihomed transport layer, such as SCTP. Each multihomed node is actually made up of more than one node. As shown in Figure 5.1, a logically multihomed node is made up of a single ”core node” and multiple ”interface nodes”, one for each simulated interface. The core node is connected to each interface node via a uni-directional link towards the interface node, but traffic never traverses these links. These links are only in place for the core node to make routing decisions. An SCTP agent simultaneously resides on all these nodes (i.e., the core and interface nodes), but actual traffic only goes to/from the interface nodes. Whenever the SCTP agent needs to send data to a destination and does

not know which outgoing interface to use, the agent firsts consults with the core node for a route lookup. Then, the SCTP agent performs the send from the appropriate interface node. Incoming data is received at one of the interface nodes directly and passed up to the SCTP agent. This solution is applicable to any transport protocol that requires multihoming functionality in ns-2.

Further details about implementation specifics and user instructions are available in the ns-2 manual [19].

5.3 Popularity

Source code for the SCTP ns-2 module has been freely available since the first public release in July 2001. Since then, there have been nine additional releases announced roughly every few months. The fifth release (April 2002) was the first version of the SCTP module to supported multihoming and failover. However, these features were unstable until the sixth release (July 2002). The tenth release is pending and will be released in May 2005. Originally, UD's SCTP module was only available as a patch for ns-2, but it has been merged with the main ns-2 code base and available by default with the main ns-2 package since ns-2.27 (released January 2004).

Since the first release of the SCTP module, over 500 unique downloads of our patch have been recorded.[2] Now that the majority of users obtain the SCTP module with the latest ns-2 package, we no longer track the number of downloads of our patch. Currently, over 50 researchers are on our mailing list. These individuals actively use our module, submit bug reports, and occasionally submit bug fixes. To date, 14 people have contributed bug fixes and/or feature enhancements.

Several publications (including theses, papers, magazine articles, and IETF drafts) by independent research groups have been published using our SCTP module to obtain their results [9, 14–18, 36, 38–40, 45, 48, 50, 51, 56, 58, 61, 69, 70, 76, 77, 79–83, 85, 88, 89,

[2] This statistic excludes known web crawlers, web caches, and duplicate domains. Duplicate domains are filtered based on their /16 IP address or their DNS name's suffix.

112]. In addition, our SCTP module is being used for course projects and independent studies. For example, SCTP ns-2 simulations are part of one of the ten recommended course projects in the *Computer Communications and Networking* course taught by Prof. Reddy in Texas A&M University's EE Department [93]. Lecturers Johan Garcia of Karlstads University in Sweden and Mohammad Ghavami of University of London in UK have suggested doing SCTP ns-2 simulation studies as part of an independent study or MS project [52, 53]. Two student projects used our module, one in the *Advanced Computer Networks* course by Prof. Gerla in UCLA's CS Department [17] and the other in the *Advanced Computer Networking and Communications* course by Prof. Xiao in Michigan State University's CSE Department [87].

Overall, the ns-2 SCTP module has provided a significant tool to the networking research community.

Chapter 6

RELATED WORK, CONCLUSIONS AND FUTURE WORK

We conclude this dissertation with a summary of related work (Section 6.1), a summary of our key results (Section 6.2), and suggestions for future study (Section 6.3).

6.1 Related Work

This section presents related research that addresses end-to-end network fault tolerance. Section 6.1.1 presents other projects that exploit transport layer multihoming, while Section 6.1.2 presents work that aim to improve path failure resilience without using transport layer multihoming.

6.1.1 Other Transport Layer Multihoming Projects

We present in this section the Concurrent Multipath Transfer and TCP-to-SCTP translation shim projects, which use transport layer multihoming to further improve end-to-end fault tolerance.

6.1.1.1 Concurrent Multipath Transfer

The results in this dissertation argue for aggressively exploiting all available paths as much as possible, but in our investigation we always maintain the restriction of sending new data to only one path at a time. This restriction reflects the current SCTP specification which states that additional research is needed before sending data concurrently over multiple paths can be permitted. Concurrent Multipath Transfer (CMT) relaxes this restriction

and uses all available paths concurrently for improving end-to-end throughput [62]. Although CMT aims to improve throughput, it also improves end-to-end fault tolerance. Sending data on all paths concurrently has the advantage of providing a sender with continuous path information for all paths. If a path fails, the sender can simply stop sending data on that path until it recovers, and the end-to-end throughput is effectively reduced temporarily. The black box perspective of network paths, however, requires additional complexity at the sender for dealing with scheduling transmissions, packet reordering introduced at the source, retransmission policies, and proper congestion control (just to name a few). These issues are discussed further in [62].

6.1.1.2 TCP-to-SCTP Translation Shim

Even though SCTP is an IETF-standardized protocol that supports transport layer multihoming, it lacks general deployment on the Internet. One of the hurdles that slows SCTP's deployment is actually a "chicken and egg" problem. Today, most existing applications use TCP, but could easily be modified to use SCTP. However, few application developers bother to do so due to lack of user demand. Users do not demand SCTP support, because few hosts/applications support it. Essentially, a gradual migration path is needed for SCTP deployment.

A transparent TCP-to-SCTP translation shim layer can provide a gradual migration path [20]. The idea is that a shim layer intercepts system calls to TCP and translates them to equivalent calls to SCTP, thereby using an SCTP association in place of a TCP connection. Such a shim allows legacy TCP applications to use SCTP and benefit from its multihoming features, without requiring any recompilation. If the peer endpoint does not support SCTP, the shim simply falls back to using TCP. As end hosts upgrade to newer operating systems that support both SCTP and this shim layer, traditional TCP applications will use SCTP more and more. Eventually, application developers may decide to support SCTP natively due to a large user base.

6.1.2 Other End-to-End Fault Tolerance Approaches

In this section, we present other approaches to improving end-to-end fault tolerance, such as dynamic host routing, site multihoming, server replication, overlay routing networks, and interrupted connection re-establishment.

6.1.2.1 Dynamic Host Routing

Dynamic host routing is an approach used by Touch and Faber to address network fault tolerance at the end hosts using the routing layer [108]. Their approach strives towards minimal changes, including no operating system modifications and no changes to the application or transport layers. Their solution combines the use of multihomed hosts, virtual network interfaces, and the RIP [57] routing protocol running on end hosts. Each multihomed host is configured with a virtual network interface that connects to the host's real network interfaces through virtual links. All connections to and from the multihomed host bind to the virtual interface IP address, and RIP is used to dynamically route traffic through the appropriate real interface. This approach allows new and existing transport layer connections to be resilient to network failures. Since RIP needs to run on all hosts that wish to communicate resiliently, dynamic host routing is intended for small scale networks and does not scale well for the Internet.

Transport layer multihoming offers better scalability, because a host only maintains multipath reachability information for peers to which it is connected. Another advantage offered by transport layer multihoming is that actual data transfers serve as implicit probes; hence, the transport layer can detect failure more quickly than a routing protocol (assuming that the application sends data more frequently than a routing protocol sends probes).

6.1.2.2 Site Multihoming

Site multihoming is when an entire site has more than one connection to the Internet. Those connections can be either through the same ISP or different ISPs. Today,

multihomed sites obtain a dedicated block of address space, and advertise its prefix route through each of its ISP connections. Often the route for the prefix propagates to all routers connected to the fault-free zone, which increasingly burdens BGP's convergence time. Routing stability is threatened as this type of site multihoming increases.

The IETF's *multi6* working group seeks alternative approaches that offer more scalability, but their work focuses on IPv6 and requires BGP support [8]. Some commercial products [2–6] offer site multihoming capabilities without BGP support. In any case, site multihoming protects against a single access link failure, but cannot avoid the long convergence times of BGP within the Internet.

6.1.2.3 Server Replication

Server replication aims to avoid failed network paths and overloaded servers by redirecting client requests to a redundant server. Content Distribution Networks (CDNs), such as Akamai [1] and Coral [49], cache content at intermediate nodes scattered around the Internet. Client requests are then redirected using DNS or server redirects. CDNs are effective at maintaining availability during flash crowds, but they focus on replicating content instead of improving host availability. Thus, CDNs require additional reliability mechanisms to protect uncached content from failures.

Server pools avoid this problem by replicating complete servers in clusters. Several projects exist which address different aspects of the goal. The IETF's *rserpool* working group is developing an architecture and protocols for the management and operation of server pools [109]. The TCP Connection Migration [99] and Migratory TCP (M-TCP) [107] projects provide mechanisms for migrating and resuming a service session from an overloaded server to a redundant backup server. A server is considered overloaded if its system resources or network resources are fully utilized, but these projects do not address the issues in determining when a server is overloaded.

SPAND maintains a shared repository of passive statistics of Internet transfers that can be used for future transfers to choose a good server from a pool [97]. Smart

Clients use downloaded portable code to handle server selection at the client side for better flexibility and performance [114]. Crovella et al. show that latency-based server selection outperforms random selection and techniques based on geographic location or network hop count [43]. Dykes et al. independently also conclude in their study of server selection techniques that a latency-based approach is best [46]. They also conclude that picking the first server to respond to a TCP SYN packet is more effective than using statistical records of previous transfers.

Research and actual deployment of server replication techniques have shown them to provide extra resilience against path failures. Unfortunately, this approach is expensive and only high-end Web sites can afford to add server redundancy. Moreover, this approach does not improve host availability for hosts that act as clients and do not have content to serve.

6.1.2.4 Overlay Routing Networks

A Resilient Overlay Network (RON) is an overlay architecture that allows end-to-end communication over the Internet to detect and recover from path outages [13]. A RON consists of a group of nodes which together form an application layer overlay on top of the existing Internet routing infrastructure. The RON nodes monitor the Internet paths among themselves to decide between routing packets directly over the Internet or via another RON node. These decisions are optimized for the individual needs of applications running on the RON nodes.

NATRON [113] extends RON by using a Network Address Translator (NAT) [47] to reach external hosts, similar to a NAT-based proposal by the Detour project [42]. Packets that are routed through the NATRON infrastructure are tunneled to an intermediate NATRON node. The intermediate nodes use NAT to replace the source address with its own, and forwards packets to their target destination. The masqueraded IP address cause reply packets to be sent to the same intermediate node, giving NATRON the ability to control routing in both directions.

SOSR (Scalable One-hop Source Routing) is another NAT-based solution, but the decision making is done by the source node for scalability [54]. When a source detects a failure, it selects one or more intermediate nodes to route through. The source simply configures a SOSR node as its IP gateway, an its packets are automatically routed through the overlay to the destination node. Intermediate nodes forwards messages it receives to the destination, acting as a NAT proxy for the source. SOSR is transparent to the destination.

Multihomed Overlay Networks (MONET) builds on the RON concept by combining an overlay network with multihoming to address path failures both at access links and within the Internet [12]. Each MONET node is a proxy that serves as a conduit for client connections to a server. MONETs offer more scalability than RONs, because communication is not confined to a small group. MONETs are able to increase availability of existing unmodified servers.

Overlay approaches work well to route around failures, but they have some limitations. They require extra infrastructure that is not scalable due to per flow state within the infrastructure. The RON-based approaches monitor path quality to select the best available path, but this required background traffic is also not scalable beyond a small set of nodes. NAT-based solutions inherit all the issues with NAT (which are many). Furthermore, ISPs may attempt to block overlay approaches, because they do not respect BGP routing policies.

6.1.2.5 Interrupted Connection Re-establishment

Sometimes failures are unavoidable and transport layer connections are interrupted. Several projects, such as Reliable Sockets (Rocks) [115], MSOCKS [84], and TCP Migrate [98, 100], focus on providing session continuity after a disconnection or change of network attachment (i.e., mobility). However, these techniques do not attempt to maintain end-to-end connectivity in the presence of path failures.

The Mobile SCTP [96] project exploits the Dynamic Address Reconfiguration extension [103] for SCTP to provide yet another mechanism for transport layer mobility. This technique differs in that it aims to actually maintain end-to-end connectivity instead of disconnection restoration.

6.2 Conclusions

Transport layer multihoming aims to provide added fault tolerance needed for persistent sessions, i.e., maintain end-to-end connectivity in the presence of failures. Although connectivity may be maintained, a transfer may suffer from periods of inactivity due to conservative use of alternate paths. In this dissertation, we have investigated retransmission policies and failover mechanisms that attempt to minimize stall time during network congestion and failure events. Our investigation includes network topologies using proactive (for fixed infrastructure networks) and reactive routing (for mobile ad-hoc networks) protocols.

The decision tree in Figure 4.11 summarizes all the results in this dissertation by suggesting a retransmission policy and failover threshold based on expected network conditions. Recognize that we are using simplistic simulation models and discrete network parameters to suggest exact values for real, complex network conditions, so our results/suggestions are only meant to provide intuition. Left branches require stricter criteria from network expectations, but are able to suggest aggressive use of alternate paths. Right branches suggest less aggressive use of alternate paths due to their looser network requirements. For example, a MANET with highly mobile nodes would probably use a short route cache lifetime and may require long discovery delays when routes are unavailable. Thus, according to the decision tree, this environment should use alternate paths conservatively (i.e., AllRtxSame with PMR $= 1$).

On the other hand, the Internet provides an environment where an endpoint can be more aggressive in using alternate paths. Since most paths on the Internet experience less than 3% loss [116], FrSameRtoAlt with PMR $= 0$ should provide the best performance.

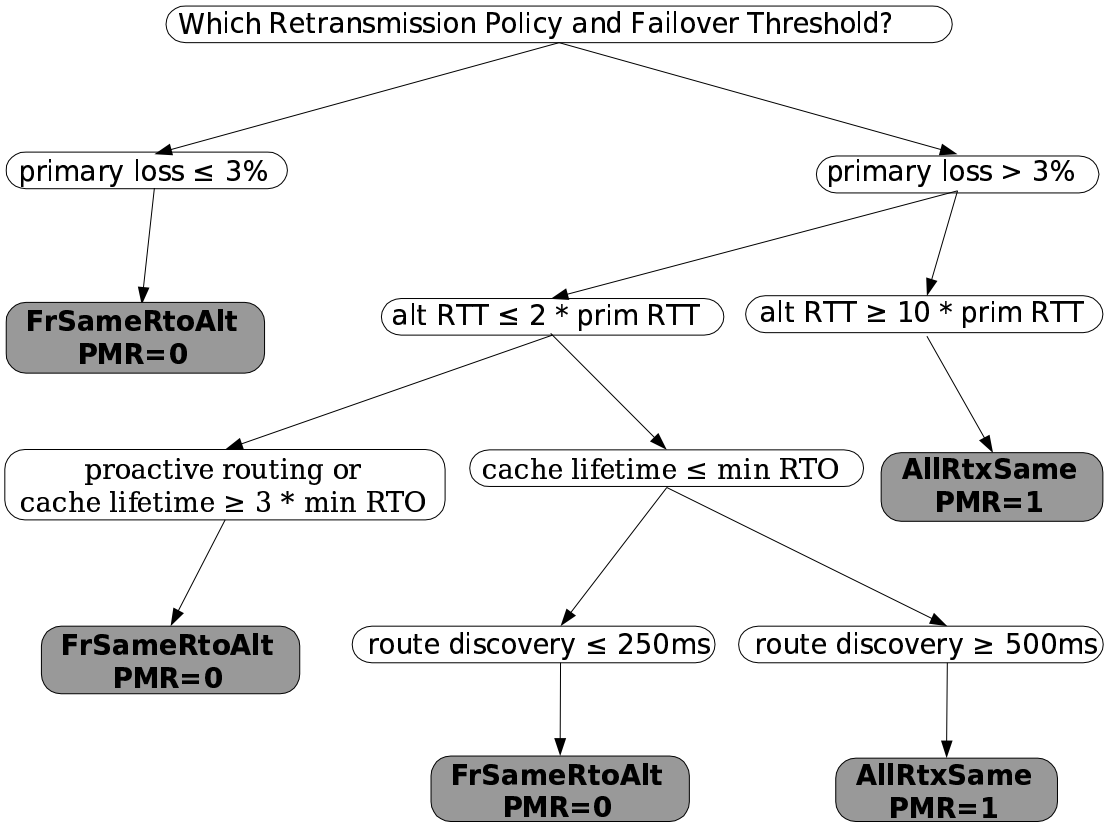

Figure 6.1: Summary of dissertation results

Even if we expect higher loss rates on the Internet, path delays generally are *roughly* symmetric (i.e., no path delay is more than double of another) and proactive routing is used. Therefore, FrSameRtoAlt with PMR $= 0$ is still a good choice for the Internet.

If absolutely nothing is known or expected of a network, then the most conservative settings should be used. Following the rightmost branches of the decision tree in Figure 4.11, these settings are AllRtxSame and PMR $= 1$. With these "conservative" settings, an end-to-end failover takes only two consecutive timeouts (i.e., 3 seconds with RFC2960's recommended minimum RTO setting). Note, however, that these settings are significantly more aggressive than those recommended currently for SCTP in RFC2960, but are conservative enough to perform well in topologies with high loss rates and highly asymmetric path delays.

Regarding our introduction and investigation of permanent failover, permanent failovers can improve performance when the sender has an accurate estimate of the RTT and loss rate of each path before making a retransmission decision. Otherwise, this dissertation shows permanent failover is not recommended; failovers should be temporary, and traffic should resume on the primary path when it recovers.

In essence, we have found that traditional conservative failover techniques used in routing do not apply when path redundancy begins at the end hosts and is handled by the transport layer. Since failovers at the routing layer are transparent to the transport layer, the failover thresholds must be conservative to avoid oscillations that could cause the transport layer to maintain inaccurate path metrics (RTT, cwnd, ssthresh). On the other hand, a multihomed transport layer is completely aware of failover events and is able to maintain separate metrics per path. As a result, transport layer multihoming can improve performance by providing aggressive failovers that reduce stall time during network congestion and failure events. These results provide insight on design decisions for future multihomed transport protocols.

6.3 Future Work

This section presents several suggestions for future work.

6.3.1 Further Investigation of Reactive Routing: Simulating Mobility and Wireless Links

We suggest further investigation of multihomed transport protocols in MANETs with reactive routing protocols. The interaction between transport layer multihoming and reactive routing is of interest, because sending traffic on idle paths in a reactive routing environment is likely to cause a cache miss in the routing tables and trigger a route discovery. The research presented in this dissertation mainly focuses on the performance hit of route discovery delays from an end host's perspective. In other words, our analysis is mostly concerned with the delays associated with a sender transmitting on a particular path at different events. However, delay is not the only consequence of route discoveries. Reactive routing protocols use flooding mechanisms to determine routes, which place a significant traffic burden on the network. Thus, aggressive failovers *may* do additional harm by degrading the overall goodput of the network. Future work should investigate this issue.

In MANETs, wireless links are also of concern. Different link layer technologies have different characteristics (channel access methods, radio range, resilience to noise, etc.) that may affect the dynamics of transport layer multihoming and failovers. A complete study of transport layer multihoming in MANETs should investigate the dynamics introduced by different link layer models.

6.3.2 Parallel Initiation Requests

Although transport layer multihoming provides failure resilience to established associations, it does not provide a mechanism for establishing associations in the face of path failure. DNS returns a list of IP addresses for multihomed hosts, and applications

are responsible for attempting more than one address until success is obtained [22]. Current applications pass only one peer IP address to a multihomed transport layer, and the transport layer then learns the other peer addresses from the peer during association establishment. However, if the peer does not respond, the initiating transport layer has no other peer addresses to try, and can only retry the same peer address. After several failed attempts, the initiating transport layer gives up, and the application is forced to try another peer IP address.

It would be ideal if applications could simply pass the list of IP addresses to a multihomed transport protocol, and handover the responsibility of trying multiple peer addresses, if necessary, during transport layer association initiation. Part of the problem with passing a list to the transport layer is that an application cannot assume that the list of IP addresses returned by DNS refers to a single host, because DNS also returns a list for replicated servers (e.g., Akamai [1]). This ambiguity poses a problem if a transport layer relies on an application passing a list of addresses, because the transport layer may potentially try to establish an association consisting of IP addresses for physically different hosts.

We propose a mechanism for future work that allows an association to be established during path failures. This mechanism requires that multihomed transport protocols accept a list of IP addresses from applications, but not assume that the entire list can be bound to a single association. Instead, the transport layer assumes, for all practical purposes, that all the addresses refer to different physical hosts. This way, the transport layer can try multiple addresses during association initiation if necessary. Once a peer replies, the normal establishment procedure occurs, and the initiating transport layer binds the association to the address list reported by the peer (and not the one passed by the application).

We also propose an enhanced version of this mechanism that can be used for determining the path (or redundant server) with lowest latency, which can be used for

selecting the primary path. When initiating an association, a multihomed transport layer sends multiple initiation requests in parallel – one to each address in the list provided by the application. The first acked request is used to establish an association, and the others are ignored. Again, the initiating transport layer binds the association to the address list reported by the peer. This method of selecting a primary path (or redundant server) is similar to Dykes et al.'s suggestion for selecting a server from a cluster of replicas [46].

BIBLIOGRAPHY

[1] Akamai. www.akamai.com.

[2] Avaya: Adaptive Networking Software (RouteScience).
www.avaya.com/gcm/master-usa/en-us/products/offers/adaptive_networking_software.htm.

[3] Fat Pipe Networks. www.fatepipeinc.com.

[4] Internap: Network-based Route Optimization.
www.internap.com/products/networkbased.htm.

[5] Radware. www.radware.com.

[6] Stonesoft: StoneGate Multi-link Technology.
www.stonesoft.com/products/ISP_Multi-homing.

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

[8] J. Abley, B. Black, and V. Gill. Goals for IPv6 Site-Multihoming Architectures. RFC3582, IETF, August 2003.

[9] R. Alamgir, M. Atiquzzaman, and W. Ivancic. Effect of Congestion Control on the Performance of TCP and SCTP over Satellite Networks. In *NASA Earth Science Technology Conference*, Pasedena, CA, June 2002.

[10] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC3390, IETF, October 2002.

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

[12] D. Andersen. *Improving End-to-End Availability Using Overlay Networks*. Ph.D. dissertation, Dept of Electrical Engineering and Computer Science, MIT, 2005.

[13] D. Andersen, H. Balakrishnan, and R. Morris M. Kaashoek. Resilient Overlay Networks. In *18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, Banff, Canada, October 2001.

[14] A. Argyriou and V. Madisetti. Bandwidth Aggregation With SCTP. In *IEEE Globecom 2003*, San Fransisco, CA, December 2003.

[15] A. Argyriou and V. Madisetti. Performance Evaluation and Optimization of SCTP inWireless Ad-Hoc Networks. In *IEEE LCN 2003*, Bonn, Germany, October 2003.

[16] M. Atiquzzaman and W. Ivancic. Evaluation of SCTP Multistreaming over Satellite Links. In *IEEE ICCCN 2003*, Dallas, TX, October 2003.

[17] A. Balk, M. Sigler, M. Gerla, and M. Sanadidi. Investigation of MPEG-4 Video Streaming over SCTP. In *SCI 2002*, Orlando, FL, July 2002. (Originally a course project in the *Advanced Computer Networks* course by Prof. Gerla in UCLA's CS Deartment).

[18] V. Basto and V. Freitas. SCTP Extensions for Time Sensitive Traffic. In *International Network Conference (INC) 2005*, Samos Island, Greece, July 2005.

[19] UC Berkeley, LBL, USC/ISI, and Xerox Parc. ns-2 Documentation and Software. www.isi.edu/nsnam/ns.

[20] R. Bickhart. *TCP-to-SCTP Translation Shim*. MS Thesis, CIS Dept, University of Delaware, May 2005.

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

[22] R. Braden. Requirements for Internet Hosts – Application and Support. RFC1123, IETF, October 1989.

[23] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC1122, IETF, October 1989.

[24] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[25] A. Caro. The ns Manual, Chapter 33: SCTP Agents. www.isi.edu/nsnam/ns/ns-documentation.html.

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

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

[28] A. Caro, P. Amer, and R. Stewart. End-to-End Failover Thresholds for Transport Layer Multihoming. In *MILCOM 2004*, Monterey, CA, November 2004.

[29] A. Caro, P. Amer, and R. Stewart. Rethinking End-to-End Failover with Transport Layer Multihoming. Tech Report 2005-04, CIS Dept, University of Delaware, July 2004. www.armandocaro.net/papers/2004.TR2005-04.acaro.pdf (under journal submission).

[30] 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.

[31] A. Caro, P. Amer, and R. Stewart. The Effects of Reactive Routing on End-to-End Failover. Tech Report 2005-05, CIS Dept, University of Delaware, October 2004. www.armandocaro.net/papers/2004.TR2005-05.acaro.pdf (under conference submission).

[32] A. Caro, P. Amer, and R. Stewart. Retransmission Policies for Multihomed Transport Protocols. Tech Report TR2005-15, CIS Dept, University of Delaware, March 2005. www.armandocaro.net/papers/2005.TR2005-15.acaro.pdf (under journal submission).

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

[34] A. Caro, J. Iyengar, P. Amer, G. Heinz, and R. Stewart. A Two-level Threshold Recovery Mechanism for SCTP. Tech report, CIS Dept, University of Delaware, July 2002.

[35] 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.

[36] C. Casetti and W. Gaiotto. Westwood SCTP: load balancing over multipaths using bandwidth-aware source scheduling. In *VTC 2004*, Los Angeles, CA, September 2004.

[37] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-End WAN Service Availability. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco, CA, March 2001.

[38] M. Chang, M. Lee, and S. Koh. Transport Layer Mobility Support Mechanism. In *LNCS*, October 2004.

[39] M. Chang, M. Lee, and S. Koh. Transport Layer Mobility Support Utilizing Link Signal Strength Information. *IEICE Transactions on Communications*, E87-B(19):2548–2556, September 2004.

[40] M. Chang, M. Lee, H. Lee, Y. Hong, and J. Park. An Enhancement of Transport Layer Approach to Mobility Support. In *LNCS*, January 2005.

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

[42] A. Collins. *The Detour Framework for Packet Rerouting*. MS Thesis, Dept of Computer Science and Engineering, University of Washington, 1998.

[43] M. Crovella and R. Carter. Dynamic Server Selection in the Internet. In *Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS)*, 1995.

[44] S. Das, C. Perkins, and E. Royer. Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks. In *INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[45] 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.

[46] S. Dykes, K. Robbins, and C. Jeffery. An Empirical Evaluation of Client-Side Server Selection Algorithms. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[47] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC1631, IETF, May 1994.

[48] J. Fang and O. Akan. Performance of Multimedia Rate Control Protocols in Inter-PlaNetary Internet. *IEEE Communications Letters*, 2004.

[49] M. Freedman, E. Freudenthal, and David Mazieres. Democratizing Content Publication with Coral. In *First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[50] S. Fu and M. Atiquzzaman. Improving End-to-End Throughput of Mobile IP using SCTP. In *2003 Workshop on High Performance Switching and Routing*, Torino, Italy, June 2003.

[51] S. Fu, M. Atiquzzaman, and W. Ivancic. Effect of Delay Spike on SCTP, TCP Reno, and Eifel in a Wireless Mobile Environment. In *International Conference on Computer Communications and Networks*, pages 575–578, Miami, FL, October 2002.

[52] J. Garcia. Independent Study Areas. CS Dept, Karlstads University. http://www.cs.kau.se/cs/education/courses/davd08/p1/areas.php.

[53] M. Ghavami. Networks MSc Projects. Centre for Telecommunications Research, University of London, 2004-2005. http://www.ctr.kcl.ac.uk/lectures/fatin/MSCprojects03-04/networks_msc_projects_2004-05.htm.

[54] K. Gummadi, H. Madhyastha, S. Gribble, H. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-Hop Source Routing. In *6th USENIX OSDI*, San Francisco, CA, December 2004.

[55] 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.

[56] I. Haddad and D. Gordon. Using Network Simulator 2 to Simulate Case Scenarios using SCTP and TCP Protocols with FTP and HTTP Traffic. *Linux Journal*, October 2002. www.linuxjournal.com/article.php?sid=5929.

[57] C. Hedrick. Routing Information Protocol. RFC1058, IETF, June 1988.

[58] G. Heinz. *Per-stream Priorities in SCTP*. MS Thesis, CIS Dept, University of Delaware, May 2003.

[59] 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.

[60] S. Iren, P. Amer, and P. Conrad. The Transport Layer: Tutorial and Survey. *ACM Computing Surveys*, 31(4), December 1999.

[61] A. Ishtiaq, Y. Okabe, and M. Kanazawa. Improving Performance of SCTP over Broadband High Latency Networks. In *IEEE LCN 2003*, Bonn, Germany, October 2003.

[62] J. Iyengar. *End-to-end Load Balancing using Transport Layer Multihoming*. Ph.D. dissertation, CISC Dept, University of Delaware, (in progress).

[63] J. Iyengar, P. Amer, and R. Stewart. Concurrent Multipath Transfer Using Transport Layer Multihoming: Performance Under Varying Bandwidth Proportions. In *MILCOM 2004*, Monterey, CA, October 2004.

[64] J. Iyengar, P. Amer, and R. Stewart. Retransmission Policies For Concurrent Multipath Transfer Using SCTP Multihoming. In *ICON 2004*, Singapore, November 2004.

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

[66] 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.

[67] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

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

[69] S. Kashihara, K. Iida, H. Koga, Y. Kadobayashi, and S. Yamaguchi. End-to-End Seamless Handover using Multi-path Transmission Algorithm. In *IC*, Tokyo, Japan, 2003.

[70] S. Koh, M. Chang, and M. Lee. mSCTP for Soft Handover in Transport Layer. *IEEE COMMUNICATIONS LETTERS*, 8(3):189–191, March 2004.

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

[72] C. Labovitz, A. Abuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *ACM SIGCOMM 2000*, Stockholm, Sweden, August 2000.

[73] C. Labovitz, A. Abuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Backbone Failures. In *29th International Symposium on Fault-Tolerant Computing (FTCS 1999)*, Madison, WI, June 1999.

[74] 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.

[75] 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.

[76] S. Liu. *Col-SCTP: SCTP in Wirelss Access Network*. MS Thesis, Network Center, University of Science and Technology of China, May 2004.

[77] S. Liu, S. Yang, and W. Sun. Collaborative SCTP: A Collaborative Approach to Improve the Performance of SCTP over Wired-cum-Wireless Networks. In *IEEE LCN 2004*, Tampa, FL, November 2004.

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

[79] L. Ma, F. Yu, V.C.M. Leung, and T. Randhawa. A New Method to Support UMTS/WLAN Vertical Handover Using SCTP. In *IEEE VTC 2003*, Orlando, FL, October 2003.

[80] L. Ma, F. Yu, V.C.M. Leung, and T. Randhawa. A New Method to Support UMTS/WLAN Vertical Handover Using SCTP. 11(4):44–51, August 2004.

[81] V. Madisetti and A. Argyriou. A New QoS Specification Is Needed For Reliable Wireless Multimedia Services. *iApplianceWeb*, December 2002. www.iapplianceweb.com/story/OEG20021206S0042.

[82] V. Madisetti and A. Argyriou. Transport-Layer QoS Management for Wireless Messaging Services. Technical report, Soft.Networks, LLC and Georgia Tech, September 2002. www.soft-networks.com/vkm-vomo.pdf.

[83] V. Madisetti and A. Argyriou. Transport-Layer QoS Standards Improve Wireless Messaging Services. *EE Times*, December 2002. www.eetimes.com/story/OEG20021127S0034.

[84] D. Maltz and P. Bhagwat. MSOCKS: An Architecture for Transport Layer Mobility. In *IEEE INFOCOM 1998*, San Francisco, CA, March 1998.

[85] G.De Marco, A.Senatore, S.Loreto, F.Postiglione, and M.Longo. On Some Open Issues in Load Sharing in SCTP. In *CCCT 2003*, Orlando, Florida, July 2003.

[86] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *2001 USENIX Security Symposium*, Washington, D.C., August 2001.

[87] K. Nandakumar. Verification of the functionalities of Stream Control Transmission Protocol (SCTP). Course Project for *Advanced Computer Networking and Communications*, CSE Dept, Michigan State University, Fall 2002. http://www.cse.msu.edu/ñandakum/.

[88] J. Noonan, A. Kelly, P. Perry, S. Murphy, and J. Murphy. Simulation of Multimedia Traffic Over SCTP Modified for Delay-Centric Handover. In *5th World Wireless Congress (B3G)*, San Francisco, CA, May 2004.

[89] J. Noonan, P. Perry, and J. Murphy. Client Controlled Network Selection. In *IEE 3G 2004: Fifth International Conference on 3G Mobile Communications Technologies*, London, UK, October 2004.

[90] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM 1998*, pages 303–314, Vancouver, CA, 1998.

[91] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.

[92] C. Perkins and E. Royer. Ad-hoc On-Demand Distance Vector Routing. In *WMCSA 1999*, February 1999.

[93] A. Reddy. Computer Communications and Networking: Course Project Guidelines. EE Dept, Texas A&M University, Fall 2004. http://ee.tamu.edu/r̃eddy/ee602_00/projects_04.txt.

[94] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC1771, IETF, March 1995.

[95] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). draft-ietf-idr-bgp4-18.txt, Internet Draft (work in progress), IETF, October 2002.

[96] M. Riegel and M. Tuexen. Mobile SCTP. draft-riegel-tuexen-mobile-sctp-05.txt (work in progress), IETF, October 2004.

[97] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared Passive Network Performance Discovery. In *1st USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, CA, December 1997.

[98] A. Snoeren. Enabling Internet Suspend/Resume with Session Continuations. In *Student Oxygen Workshop 2002*, July 2002.

[99] A. Snoeren, D. Andersen, and H. Balakrishnan. Fine-Grained Failover Using Connection Migration. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco, CA, March 2001.

[100] A. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. In *ACM MobiCom 2000*, Boston, MA, August 2000.

[101] R. Stewart, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tuexen. Stream Control Transmission Protocol (SCTP) Implementer's Guide. draft-ietf-tsvwg-sctpimpguide-13.txt, Internet Draft (work in progress), IETF, February 2005.

[102] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol Partial Reliability Extension. RFC3758, May 2004.

[103] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. draft-ietf-tsvwg-addip-sctp-11.txt, Internet Draft (work in progress), IETF, February 2005.

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

[105] 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.

[106] T. Strayer and A. Weaver. Evaluation of Transport Protocols for Real-time Communications. Technical Report TR-88-18, CS Dep., University of Virginia, June 1988.

[107] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Highly Available Internet Services using Connection Migration. In *22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, July 2002.

[108] J. Touch and T. Faber. Dynamic Host Routing for Production Use of Developmental Networks. In *ICNP 1997*, Atlanta, GA, October 1997.

[109] M. Tuexen, Q. Xie, R. Stewart, M. Shore, L. Ong, J. Loughney, J. Loughney, and M. Stillman. Requirements for Reliable Server Pooling. RFC3237, IETF, January 2002.

[110] J. Walrand. *Communication Networks: A First Course*. Aksen Associates, 1991.

[111] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-Similarity Through High-Variability: Statical Analysis of Ethernet LAN Traffic at the Source Level. In *ACM SIGCOMM 1995*, Cambridge, MA, August 1995.

[112] G. Ye, T. Saadawi, and M. Lee. SCTP Congestion Control Performance in Wireless Multi-hop Networks. In *MILCOM 2002*, Anaheim, CA, October 2002.

[113] A. Yip. *NATRON: Overlay Routing to Oblivious Destinations*. MS Thesis, Dept of Electrical Engineering and Computer Science, MIT, 2002.

[114] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using Smart Clients to Build Scalable Services. In *USENIX Annual Technical Conference*, Anaheim, CA, January 1997.

[115] V. Zandy and B. Miller. Reliable Network Connections. In *ACM MobiCom 2002*, Atlanta, GA, September 2002.

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