UNDERSTANDING INTERNET ROUTING ANOMALIES AND BUILDING ROBUST TRANSPORT LAYER PROTOCOLS

MING ZHANG

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

SEPTEMBER 2005

Abstract

As the Internet grows and routing complexity increases, network-level instabilities are becoming more and more common. End-to-end communications are especially susceptible to service disruptions, while diagnosing and mitigating these disruptions are extremely challenging. In this dissertation, we design and build systems for diagnosing routing anomalies and improving robustness of end-to-end communications.

The first piece of this work describes PlanetSeer, a novel distributed system for diagnosing routing anomalies. PlanetSeer passively monitors traffic in wide-area services, such as Content Distribution Networks (CDNs) or Peer-to-Peer (P2P) systems, to detect anomalous behavior. It then coordinates active probes from multiple vantage points to confirm the anomaly, characterize it, and determine its scope. There are several advantages of this approach: first, we obtain more complete and finer-grained views of routing anomalies since the wide-area nodes provide geographically-diverse vantage points. Second, we incur limited additional measurement cost since most active probes are initiated when passive monitoring detects oddities. Third, we detect anomalies at a much higher rate than other researchers have reported since the wide-area services provide large volumes of traffic to sample. Through extensive experimental study in the wide-area network, we demonstrate that PlanetSeer is an effective system for both gaining a better understanding about routing anomalies and for providing optimization opportunities for the host service.

To improve the robustness of end-to-end communications during performance anomalies, we design mTCP, a novel transport layer protocol that can minimize the impact of anomalies using redundant paths. mTCP separates the congestion control for each path so that it can not only obtain higher throughput but also be more robust to path failures. mTCP can quickly react to failures, and the recovery process normally takes only several seconds. We integrate a shared congestion detection mechanism into mTCP that allows us to suppress paths with shared congestion. This helps alleviate the aggressiveness of mTCP. We also propose a heuristic to find disjoint paths between pairs of nodes. This can minimize the chance of concurrent failures and shared congestion. We implement mTCP on top of an overlay network and evaluate it using both emulations and experiments in the wide-area network.

Acknowledgments

I have been incredibly fortunate to have had three mentors during the course of my PhD study. The first one is Professor Randy Wang. I would like to thank him for his guidance, support, and help throughout the years. I consider myself very lucky to have the chance to work and learn from him. He provided the enthusiasm and encouragement that I needed to complete this work. The second one is Professor Larry Peterson. He made himself available for numerous discussions, often started by my dropping by his office unexpectedly. I always left with a deeper and clearer understanding about those research problems than I'd had when I arrived. I learned from him that research requires combination of dedication, confidence, and truly long-term thinking. I am sincerely grateful for his high standard for research, kindness, and patience. The third one is Professor Vivek Pai. He provided me invaluable guidance and frequent advice on the PlanetSeer project. His vigorous approach both to research and to life has greatly shaped and enriched my view of networking and systems research. I have to thank him for letting me steal an enormous amount of time and wisdom during the last two years of my PhD study.

I am fortunate to collaborate with Chi Zhang on lots of the work presented in this thesis. Chi is my friend, lab-mate, as well as apartment-mate. I drew immense inspiration from him both inside and outside work. He is the best collaborator one could ask for. I am also grateful to Junwen Lai. The mTCP project would not have been possible without his help on the user-level TCP implementation.

The second part of my thesis was inspired by my work at ICIR, starting in the summer of 2001. I thank Dr. Brad Karp for making my visit possible. Later, Brad gave me the chance to continue collaborating with him at Intel Research Pittsburgh in the summer of 2003. I benefited enormously from the two summers I spent working with him. While at ICIR, I thank Dr. Sally Floyd for teaching me a lot on TCP related problems. It was a

great honor to work with Professor Arvind Krishnamurthy, who provided many vigilant comments on various algorithms in my work. I am especially grateful to Professor Jennifer Rexford. She always patiently listened to my incoherent thoughts and provided me amazingly insightful and detailed feedback. I learned a tremendous amount from her on doing research as well as on writing and presentation.

I am grateful to the PlanetLab staffs for their help with deploying the PlanetSeer system. Andy Bavier answered me lots of questions on safe raw socket. Marc Fiuczynski shared with me his extensive experience in vserver. I would like to thank Scott Karlin, Mark Huang, Aaron Klingaman, Martin Makowiecki, and Steve Muir for their support and patience. I also thank KyoungSoo Park for his effort in keeping CoDeeN operational during my experiment.

I would like to thank Professor David Walker and Moses Charikar for serving as non-readers on my dissertation committee. They gave many valuable comments and suggestions on my work.

My work was supported in part by NSF grants CNS-0335214 and CNS-0435087, and DARPA contract F30602-00-2-0561.

I greatly enjoyed my life at Princeton because of the many close friends I had there. I thank Ding Liu, Chi Zhang, Yaoping Ruan, Fengzhou Zheng, Ting Liu, Wen Xu, Gang Tan, and Fengyun Cao for their support and encouragement throughput the years. I also thank my non-Princeton friends, especially Xuehua Shen and Ningning Hu. They made my life lots of fun.

This thesis is dedicated to my parents. They always gave me love, trust, and pride. They played the most important role in directing me into pursuing a research career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the Internet has experienced exponential growth in recent years, so does its complexity. The increasing complexity can potentially introduce more network-level instabilities. Today, the Internet consists of roughly 20,000 Autonomous Systems (ASes) [32], where each AS represents a single administrative entity. As shown in Figure 1.1, to go from one endpoint to another, packets have to traverse a number of ASes. Ideally, the packets should be delivered both reliably and efficiently through the network. However, in reality, the network paths may not be perfect. One pathological event occurring within a single AS could affect many ASes and a large number of network paths through those ASes. During such periods, users will perceive performance degradation. Our goal is to improve the performance and robustness of end-to-end communications on the Internet. In this dissertation, we focus on the *network performance anomalies*, which are broadly defined as any pathological events occurring in the network that cause end-to-end performance degradation.

We study performance anomalies from two perspectives. In the first part of this thesis, we aim to understand the characteristics of the anomalies. More specifically, we

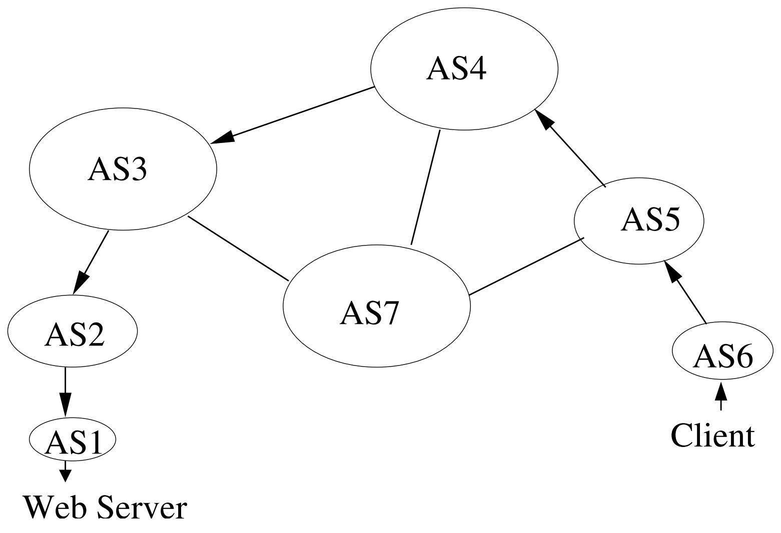

Figure 1.1: The Internet consists of many ASes

investigate how to detect and diagnose the anomalies, how to estimate their locations and scopes, and how to quantify their effects on the end-to-end performance. These types of information are very important. On the one hand, knowing where the anomalies occur will improve the accountability of the Internet. A customer may use this information to select good service providers (ISPs). Similarly, an ISP may use this information to select good peering ISPs. In addition, if two entities have service level agreements (SLAs) with each other, they may obtain compensations for violating these agreements. On the other hand, knowing why the anomalies occur will help the network operators to fix the problems quickly and to prevent the similar problems from occurring in the future.

Although understanding the characteristics and origins of performance anomalies can help us improve the long-term stability of the Internet, we are still going to encounter anomalies frequently in the foreseeable future. When an anomaly does occur, it is desirable for end users to be able to bypass the anomaly as quickly as possible. In the second part of this thesis, we describe a novel transport layer protocol that can minimize the impact of anomalies by taking advantage of redundant paths on the Internet. Today, TCP is the dominant transport layer protocol for end-to-end communications. TCP only uses a single network path between two endpoints. Should any congestion or failure occur on that path, TCP's performance will be significantly reduced. Recent work on Internet measurement and overlay networks has shown that there often exist multiple paths between pairs of hosts [78]. Using these redundant paths, we can not only aggregate the bandwidth of multiple paths in parallel but also enhance the robustness of end-to-end communications during anomalies.

In Section 1.1, we first briefly explain why anomalies occur on the Internet and how they affect end-to-end performance. Then in Section 1.2 and 1.3, we explain why it is difficult to detect, diagnose and mitigate anomalies. At the end of this chapter in Section 1.4, we provide an overview of this dissertation.

1.1 Why Do Performance Anomalies Occur on the Internet?

Although the Internet is designed to be self-healing, users often experience performance degradation. For instance, they may find certain websites are unreachable or their network speed is very slow. These problems may be caused by various pathological events that occur in the network.

Routing instability is one of the major sources of performance anomalies. Routing protocols are responsible for discovering the paths to reach any destination on the Internet. Routing protocols can be classified into interdomain and intradomain protocols. Intradomain protocols (IGP), such as OSPF[44] or IS-IS[20], are responsible for disseminating reachability information within an AS. Interdomain protocols (EGP), such as BGP[75], maintain the reachability information among all the ASes.

Routing instabilities may arise when routing protocols are adapting to topological or policy changes. Inside an AS, *link outages* often stem from maintenance, power outages, and hardware failures [53]. When an outage occurs, routing protocols may try to bypass the failure using alternate paths. This, in turn, will lead to *route changes*. Sometimes, route changes may also be caused by traffic engineering inside a network [30, 50]. At the AS-level, outages may arise due to peering link failures or eBGP session resets [91]. These outages can lead to AS-path changes. In addition, since BGP incorporates policies into route selection process, AS-level route changes may be triggered by policy changes as well [75].

Besides route changes and outages, routing instabilities can often lead to *routing loops*. When a routing instability occurs, each router needs to propagate the latest reachability information to routers within the same AS or in other ASes through routing updates. During this process, loops may evolve because different routers may have inconsistent routing states. The convergence time of the propagation process itself can be highly varied. IGPs usually converge within several hundred milliseconds [79] to several seconds [42]. In contrast, it may take tens of minutes for BGP routers in different ASes to reach a consistent view on the network topology [52].

Due to the complexity of routing protocols, routing instabilities can also be caused by misconfigurations. A recent study shows that 3 in 4 new prefix advertisements result from BGP misconfigurations [61]. In an earlier study, Labovitz, Malan and Jahanian find that 99% of BGP updates are pathological and do not reflect network topological changes[54]. These BGP misconfigurations can cause various routing problems, such as routing loops [22, 26], invalid routes [61], contract violations [27], and persistent oscillations [12, 34, 89].

Another major source of performance anomalies is congestion. Congestion arises when the packet arrival rate of a link exceeds the link capacity. It is often caused by flash crowds, distributed denial of service (DDoS) attacks, worm propagations, or sometimes even routing instabilities [86]. When a link becomes congested, it may have to delay or drop packets. This will impose negative effects on flows that are traversing that link. For instance, TCP's throughput is inversely proportional both to the round trip time (RTT) and to the square root of loss rate [70]. When the loss rate or RTT increases, the throughput of TCP will decrease. When the loss rate exceeds 30%, TCP becomes essentially unusable since it spends most time in timeouts [70].

1.2 Difficulties in Anomaly Diagnosis

Although performance anomalies occur quite frequently on the Internet, diagnosing these anomalies is nontrivial. This is because the Internet is not owned by a single administrative entity but instead consists of many autonomous systems (ASes). Each AS is operated by a network service provider (ISP) and has its own routing policy. The routing information shared between two ASes is heavily filtered and aggregated using BGP [75]. While this allows the Internet to scale to thousands of networks, it makes anomaly diagnosis extremely challenging.

As we described in the beginning of Section 1, the network path between two endpoints usually traverses multiple ASes and routers. When an anomaly arises, any intermediate component in that path can introduce the problem. Although tools like *ping* and

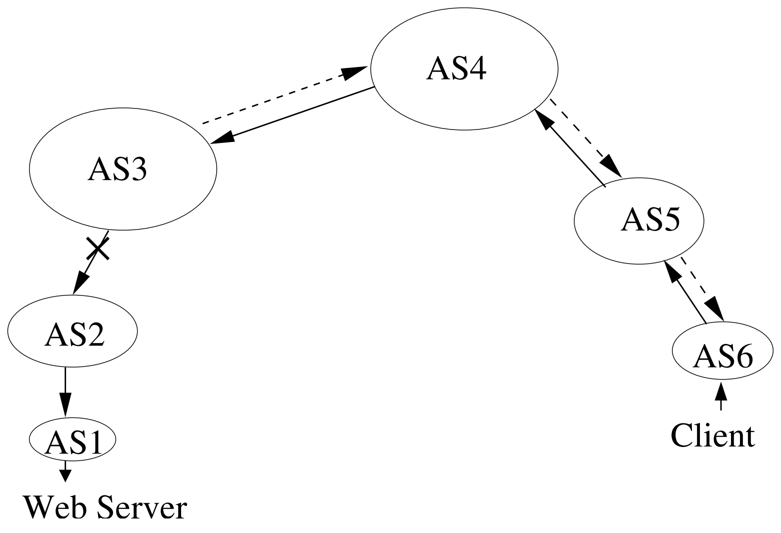

Figure 1.2: Routing anomaly is often propagated

traceroute exist for diagnosing network problems, determining the origins of anomalies is exceptionally difficult for several reasons:

Anomaly origin may differ from anomaly appearance. Routing protocols, such as BGP, OSPF, and IS-IS, may propagate reachability information to divert traffic away from failed components. When a traceroute stops at a hop, it is often the case that the router has received a routing update to withdraw that path, leaving no route to the destination. For example in Figure 1.2, the client traverses the AS path "6 5 4 3 2 1" to reach the web server. Suppose there is a link outage between AS2 and AS3 that makes the web server unreachable from AS3. This unreachability information will be propagated from AS3 to AS4, AS5, and AS6. Although the traceroute from the client will stop at AS6, AS6 is actually far away from the origin of the failure.

Anomaly information may be abstracted. The Internet consists of many ASes and each AS manages its own network independently. An AS will hide various internal information from other ASes for scalability reasons. In addition, since ASes are often competing with each other, they are unwilling to share sensitive information, such as their traffic, topology, and policy. As a result, when an AS observes an anomaly, it may not have enough detailed information to either pinpoint or troubleshoot the anomaly. For example in Figure 1.2, when AS6 loses the route to the web server, it can hardly tell whether the problem occurs in AS1, 2, 3, 4, or 5.

Anomaly durations are highly varied. Some anomalies, like routing loops, can last for days. Others may persist for less than a minute. This high variability makes it hard to diagnose anomalies and react in time.

Network paths are often asymmetric. Because BGP is a policy-based routing protocol, this may lead to asymmetric paths, which means the sequence of ASes visited by the routes for the two directions of a path differ. Paxson observed that 30% of node pairs have different forward and reverse paths which visit at least one different AS [71]. Since *traceroute* only maps the forward path, it is hard to infer whether the forward or reverse path is at fault without cooperation from the destination.

For the above reasons, to diagnose anomalies, we have to collect anomaly-related data from many locations. Historically, few sites had enough network coverage to provide such fine-grained and complete information. The advent of wide-area network testbeds like PlanetLab [74] has made it possible to diagnose anomalies from multiple geographically-diverse vantage points. In Chapter 3, we will introduce PlanetSeer, a novel diagnostic system that can take advantage of the wide coverage of PlanetLab [94]. We will describe in detail how PlanetSeer combines passive monitoring with widely-distributed probing machinery to detect and isolate routing anomalies on the Internet.

1.3 Difficulties in Anomaly Mitigation

As we have mentioned before, BGP is the *de-facto* interdomain routing protocol on the Internet today. BGP is a policy-based protocol which computes routes conforming to commercial relationships between ASes. This may lead to suboptimal routing decision for end-to-end communications [5]. For instance, Spring, Mahajan, and Anderson show that current peering policies cause the latency of over 30% of the paths to be longer than the shortest available paths [82]. In addition, because BGP has to scale to a large number of networks, it adopts various mechanisms to hide detailed information and damp routing updates. Although this reduces the chance of routing oscillations, it makes BGP less responsive to failures. Sometimes, it takes many minutes for BGP to converge to a consistent state after failures [52]. The end-to-end service disruptions could last for tens of minutes or more [65].

More recently, application-layer overlay routing has been proposed as a remedy to this problem. Overlay routing can recover from performance degradation within a shorter period of time than the wide-area routing protocols [5]. In an overlay routing system, the participating nodes periodically probe each other to monitor the performance of the paths between them. When an anomaly is detected on the direct Internet path between a pair of nodes, the system will try to bypass the anomaly by choosing a good overlay path through one or more intermediate nodes.

While overlay routing can circumvent performance degradation more quickly, its effectiveness to a large extent depends on its active probing mechanism. We use Resilient Overlay Networks (RON) [5], a representative overlay routing system, to exemplify these problems. First, when an anomaly occurs, how fast RON can recover from the anomaly is determined by its probing rate. In RON, the participating nodes probe each other every

3 seconds during the anomalous period. Correspondingly, its mean outage detection time is 19 seconds. However, the probing overhead of this approach is $O(n^2)$, where n is the total number of nodes. When n becomes large, it is difficult to maintain low measurement overhead while still achieving short recovery time.

Second, RON estimates the available bandwidth of the monitored paths using active probing. When an anomaly is detected, it chooses a good alternate path based on the estimated bandwidth. However, the state-of-the-art available bandwidth estimation tools need to inject a fair amount of probing packets to obtain reasonably accurate estimates [46, 40, 3]. For scalability reasons, RON uses a much more lightweight probing mechanism. This may lead to inaccurate bandwidth estimates under many circumstances, which in turn impairs its routing decisions.

To overcome these problems, we design mTCP, a novel transport layer protocol that can utilize multiple paths in parallel [93]. By using more than one paths, mTCP can recover from performance anomalies very quickly. Our approach incurs little measurement overhead, since mTCP can accurately estimate the available bandwidth of multiple paths by passively monitoring the traffic on those paths. We will describe more on this in Chapter 4.

1.4 Overview of the Thesis

We now give an overview of this dissertation. In Chapter 2, we describe the related work in this area and provide a background for our work. We will first introduce the network testbeds that are used for evaluating our systems. We then go through the recent work on studying performance anomaly on the Internet. Based on their methodologies, we classify them into intra- and inter-domain routing anomalies, traffic anomalies, and end-to-end measurements. At the end of Chapter 2, we will discuss the research efforts that improve the end-to-end performance using striping at the link-layer, application-layer and transport-layer.

Chapter 3 focuses on *PlanetSeer*, a large-scale distributed system for routing anomaly detection and diagnosis. We first describe the components and mechanism of PlanetSeer, including how to detect suspicious routing events by passively observing the traffic generated by wide-area services and how to coordinate multiple nodes to actively probe these events. We then analyze the anomaly data that is collected during a 3-month period in 2004. We describe our techniques for confirming the routing anomalies, classifying them, and characterizing their scopes, locations, and end-to-end effects. In the end, we quantify the effectiveness of overlay routing in bypassing path failures.

Chapter 4 presents *mTCP*, a novel transport layer protocol that is robust to performance anomaly. mTCP differs from traditional transport layer protocols in that it can use more than one paths in parallel. It has four major components: 1) new congestion control for aggregating bandwidth on multiple paths, 2) shared congestion detection and suppression for alleviating the aggressiveness of mTCP, 3) failure detection and recovery for quickly reacting to performance anomaly, and 4) path selection for minimizing the chance of concurrent failures and shared congestion. mTCP has been implemented as a user-level application running on top of overlay networks. We use experiments on both local-area and wide-area network testbeds to demonstrate its effectiveness.

Chapter 5 concludes with a summary of this dissertation and our vision for future work. We have made two main contributions in this work. First, we demonstrate that it is possible to build a distributed system for detecting and isolating routing anomalies with high accuracy. Second, we can dramatically improve the robustness of end-to-end communications using redundant paths. We are going to continue our research in several directions. First, we plan to extend our system by studying performance anomalies caused by non-routing problems. Second, We plan to investigate new ways to improve the accuracy of routing anomaly diagnosis and to reduce measurement overhead. Finally, we plan to build a network weather service that can continuously monitor the health of the Internet.

Chapter 2

Background and Related Work

In this chapter, we provide a background for our work and give an overview of the related work in this area. There have been many research efforts on studying anomalies in the Internet and designing robust network protocols. We will focus on those that are most relevant and discuss their difference from our approaches. We first briefly introduce the network testbeds used for our experiments and evaluations. We then turn to the recent studies on network anomalies, which include interdomain and intradomain routing anomalies, traffic anomalies, and end-to-end failure measurements. In the end, we discuss the research efforts that use striping techniques to improve performance and robustness. Based on the network layer where the striping techniques are applied, we classify them into link-layer, transport-layer, and application-layer striping.

2.1 Network Testbeds

We evaluate our systems with both emulations and real-world deployment. The emulations are conducted on Emulab [24], a time- and space-shared network emulator. It consists of several hundred PCs, which allows users to remotely configure and control the machines and links down to the hardware level. Users can use ns-compatible [68] scripts to build network topologies and define various parameters, such as packet loss rate, latency, bandwidth, and queue size. As a result, users are able to construct a wide range of scenarios under which the prototype systems can be evaluated. In addition, since the emulation results are repeatable, users can easily quantify the effectiveness of their design.

We use PlanetLab for our wide-area network experiments [74]. PlanetLab is a global testbed for experimenting with planetary-scale services. It currently consists of over 500 machines, hosted by more than 270 sites, spanning 25 countries. PlanetLab enables us to experiment with new systems under real-world conditions and at large scale. We can observe a realistic network substrate that experiences congestion or failures. PlanetLab also provides us with a large set of geographically-distributed machines to investigate network anomalies and study the behavior of our systems during anomalous periods.

2.2 Intradomain Routing Anomalies

As we mentioned in Section 1.1, routing anomaly is one of the major causes of performance degradation on the Internet. We first look at intradomain routing anomalies and defer the discussion about interdomain routing anomalies to the next section. Nowadays, the most commonly used intradomain routing protocols are OSPF and IS-IS. Researchers have been using routing updates collected in individual ISPs to study routing anomalies. Labovitz and Ahuja used the OSPF messages gathered in a medium-sized regional ISP, together with the data from the trouble ticket tracking system managed by the Network Operation Center (NOC) of that ISP, to characterize the origins of routing failures. They classify the failures into hardware, software, and operational problems [53]. Iannaccone *et al.* investigated the routing failures in Sprint's IP backbones using IS-IS routing updates collected from three vantage points [42]. They examined the frequency and duration of the failures inferred from routing updates and concluded that most failures are short-lived (within 10 minutes). They also studied the interval between failures.

2.3 Interdomain Routing Anomalies

There has also been extensive work on studying BGP routing instabilities. Labovitz and Ahuja [53] studied interdomain routing failures using the BGP data collected from several ISPs and 5 network exchange points. They analyzed the temporal properties of failures, such as mean time to fail, mean time to repair, and failure duration. They found that 40% of the failures were repaired within 10 minutes and 60% of them were resolved within 30 minutes. Wu *et al.* presented an online troubleshooting system that could identify significant routing disruptions in large volumes of BGP updates [91]. They applied their tool to the BGP messages collected in the AT&T backbone networks and found that many routing disruptions and traffic shifts were caused by hot-potato changes and eBGP session resets.

Mahajan, Wetherall, and Anderson [61] studied BGP misconfigurations using the BGP updates from RouteViews [69], which has 23 vantage points across different ISPs. They found BGP misconfigurations were relatively common and classified them into origin and export misconfigurations. Nearly 3 in 4 new prefix advertisements were due to misconfigurations. However, misconfigurations usually had limited impact on end-to-end performance, since only 1 in 25 misconfigurations affected end-to-end connectivity.

More recently, Feldmann *et al.* presented a methodology to locate the origin of BGP routing instabilities along three dimensions: time, prefix, and view [29]. Their basic assumption is that an AS path change is caused by some instability either on the previous best path or on the new best path. Caesar *et al.* [14] and Chang *et al.* [16] also proposed similar approaches to analyze routing changes, although their algorithms were different in details.

All of the above routing anomaly studies are based on either intra-domain (IS-IS, OSPF) or inter-domain (BGP) routing messages, from which anomalies are inferred. The first approach requires access to ISP's internal data. While it is very useful for troubleshooting anomalies inside this ISP, it cannot be easily applied to other ISPs. The second approach can be used to analyze interdomain routing anomalies but becomes less useful for debugging anomalies that are unrelated to BGP. As we will describe in Chapter 3, our work complements these two approaches by studying routing anomalies from an end-to-end perspective. We will also quantify the impact of anomalies on end-to-end performance, such as loss rate and RTT.

2.4 Traffic Anomalies

Parallel to routing anomalies, several research efforts have focused on traffic anomalies which are defined as unusual and significant changes in network traffic. These efforts examined different methodologies for extracting anomalous patterns from large volumes of noisy data. Lakhina, Crovella, and Diot applied Principle Component Analysis (PCA) to separating high-dimensional traffic data into subspaces corresponding to normal and anomalous traffic [56]. Krishnamurthy *et al.* instead relied on a variant of the sketch data structure to detect changes [51]. Both of them validated their approaches using NetFlow data [66].

Barford *et al.* used wavelet analysis to extract distinct characteristics of different types of anomalies [11]. They demonstrated their algorithm could identify outages, flash crowds, attacks, and measurement failures in SNMP [15] and IP flow data. Roughan *et al.* also used several time-series methods to detect outliers in SNMP data [76]. However, they correlated the SNMP data with BGP data to reduce the chance of false alarms.

This category of approaches for analyzing traffic anomalies require access to ISP's internal data, such as NetFlow, SNMP, or IP flows. However, these data are generally unavailable for outsiders or normal users. In Chapter 3, we will describe our technique for troubleshooting routing anomalies based on end-to-end measurements. Since our approach does not require any proprietary data, it gives end users more flexibility in anomaly diagnosis.

2.5 End-to-End Failure Measurement

There also has been much work on studying Internet anomalies through end-to-end measurement, and this work has greatly influenced our approach. Paxson [71] studied the end-to-end routing behavior by running repeated traceroutes between 37 distributed hosts. His study showed that 49% of the Internet paths were asymmetric and visited at least one different city. 91% of the paths persisted for more than several hours. He used traceroutes to identify various routing pathologies, such as loops, fluttering, path changes, and outages. However these traceroutes did not distinguish between forward and reverse failures.

Chandra *et al.* [21] studied the effect of network failures on end-to-end services using two traceroute datasets [71, 78]. They also used the HTTP traces collected from 11 proxies. They modeled the failures using their location and duration, and evaluated different techniques for masking failures. However, the HTTP and traceroute datasets were independent. In comparison, we combine passive monitoring data and active probing data, which allows us to detect failures in realtime and correlate the end-to-end effect with different types of failures. They also classified the failures into near-source, near-destination and in-middle by matching /24s IP prefixes with endhost IPs. In contrast, we study the location of failures using both IP-to-AS mapping [62] and the 5-tier AS hierarchies [85]. This allows us to quantify the failure locations more accurately and at a finer granularity.

Feamster *et al.* measured the Internet failures among 31 hosts using periodic pings combined with traceroutes [25]. They pinged the path between a pair of nodes every 30 seconds, with consecutive ping losses triggering traceroutes. They considered the location of a failure to be the last reachable hop in traceroute and used the number of hops to the closest endhost to quantify the depth of the failure. They characterized failures as inter-AS and intra-AS and used one-way ping to distinguish between forward and reverse failures. They also examined the correlation between path failures with BGP updates.

Our work is partly motivated by these approaches, but we cannot use their methodologies directly due to environmental differences. Because our system monitors the network paths to a large number of IPs, we cannot afford to ping each of them frequently. Anomaly detection and confirmation are more challenging in our case, since many destinations may not respond to pings (behind firewalls) or even are offline (such as dialup users). We infer anomalies by monitoring the status of active flows, which allows us to study anomalies on a much more diverse set of paths. We also combine traceroutes with passive monitoring to distinguish between forward and reverse anomalies, and classify forward anomalies into several categories. Since where an anomaly appears may be different from where the anomaly occurs, we quantify the scope of anomalies by correlating the traceroutes from multiple vantage points, instead of using one hop (last reachable hop) as the failure location. Finally, we study how different types of anomalies affect end-to-end performance.

2.6 Link-Layer and Application-Layer Striping

We now turn to the research efforts that use multiple paths in a network to improve performance and reliability. One area of related work is the use of striping [87] or *inverse-multiplexing* in link-layer protocols to enhance throughput by aggregating the bandwidth of different links. Adiseshu *et al* [1], Duncanson *et al* [23], and Snoeren [81] provided link-striping algorithms that addressed the issues of load-balancing over multiple paths and preserving in-order delivery of packets to the receiver. These efforts proposed transparent use of link-level striping without requiring any changes to the upper layers of the protocol stack.

Another area of related work is the use of striping at the application-layer to improve throughput by opening multiple TCP sockets concurrently [4, 35, 57, 80]. However, these multiple TCP connections utilized the same physical path. Although this approach can attain higher throughput, it may also lead to an unfair share of bandwidth at congested links and seemed to primarily benefit from increased window sizes over long-latency connections.

2.7 Transport-Layer Striping

Many researchers have proposed the use of multiple paths by the transport-layer protocols to enhance reliability [8, 67, 58, 7]. Banerjea [8] used redundant paths in his *dispersity* *routing* scheme to improve reliable packet delivery for real-time applications. Nguyen and Zakhor [67] also proposed the use of multiple paths to reduce packet losses for delay-sensitive applications. They employed UDP streams to route data whose redundancy was enhanced through forward error correction techniques.

Multiple paths can also be used for improving the throughput and robustness of end-to-end connections. SCTP [84] is a reliable transport protocol which supports multiple streams across different paths. However, it does not provide strict ordering across all the streams, and it cannot utilize the aggregate bandwidth on multiple paths. There are a few systems, such as R-MTP [59] and pTCP [38], which are able to achieve bandwidth aggregation. R-MTP provided bandwidth aggregation by striping packets across multiple paths based on bandwidth estimation. It estimated the available bandwidth by periodically probing the paths. As a result, its performance greatly relied on the accuracy of the estimation and the probing rate. It could suffer from bandwidth fluctuation as shown in [38]. pTCP used multiple paths to transmit TCP streams and provided mechanisms for striping packets across the different paths. They however assumed the existence of a separate mechanism that identified what paths to use for their pTCP connections, and they did not address the issues of recovering from path failures or obtaining an unfair share of the throughput of the congested links if the paths were not disjoint. Their study was also limited to simulations using *ns*[68].

2.8 Summary

The work in this dissertation focuses on diagnosing and characterizing routing anomalies, as well as improving the reliability of end-to-end communications using redundant paths. Previous research efforts have studied network anomalies using intradomain (OSPF, ISIS) and interdomain (BGP) routing messages, traffic statistics (NetFlow, SNMP), and end-to-end measurement (ping, traceroute). Our approach differs from previous work in that:

- It is an end-system based approach and does not require access to any privileged data. It gives both end users and ISPs much flexibility in troubleshooting routing anomalies in the wide-area networks.

- It combines passive monitoring with active probing to reduce measurement overhead and can easily scale to a large number of nodes.

- It provides a finer-grained and more complete view on routing anomaly by correlating the probing from multiple vantage points.

In the past, a series of proposals have been made to enhance network performance using striping techniques at the link-layer, transport-layer, and application-layer. We are the first to implement and evaluate a transport-layer protocol that can utilize multiple paths concurrently in real systems. We present a comprehensive design that addresses the important issues of per-path congestion control, shared congestion detection, failure recovery, and path selection.

Chapter 3

PlanetSeer: Internet Path Failure Monitoring and Characterization

As we have explained in Section 1.1, performance degradations are often caused by routing anomalies on today's Internet. Understanding routing anomalies is crucial for improving the overall stability of the Internet. In this chapter, we introduce PlanetSeer, a large-scale distributed system for routing anomaly detection and diagnosis. PlanetSeer passively watches for anomalous events in the traffic generated by wide-area services. It then actively probes the network from multiple vantage points to understand the anomalies. We will first describe how to detect suspicious routing events in network traffic and how to probe these events when they are detected. We then present our techniques for confirming routing anomalies, classifying them, and estimating their locations and scopes. Finally, we will characterize the routing anomalies based on the monitoring and probing data collected during a 3-month period in 2004.

3.1 Introduction

As the Internet grows and routing complexity increases, network-level instabilities are becoming more common. Among the problems causing end-to-end path failures are router misconfigurations [61], maintenance, power outages, and fiber cuts [53]. Inter-domain routers may take tens of minutes to reach a consistent view of the network topology after routing failures, during which time end-to-end paths may experience outages, packet losses, and delays [52]. These routing problems can affect performance and availability [5, 52], especially if they occur on commonly-used network paths. However, even determining the existence of such problems is nontrivial, since there is no central authority that monitors all Internet paths.

Previously, researchers have used routing messages, such as BGP [61], OSPF [53] and IS-IS [42] updates, to identify inter-domain and intra-domain routing anomalies. This approach usually requires collecting routing updates from multiple vantage points, which may not be easily accessible for normal users. Other researchers have relied on some form of distributed active probing, such as pings and traceroutes [5, 25, 71], to detect routing anomalies from end hosts. This approach monitors the paths between pairs of hosts by having them repeatedly probe each other. Because this approach requires cooperation from both source and destination hosts, it only measures paths among a limited set of participating nodes.

We observe that there exist several *wide-area services* employing multiple geographically-distributed nodes to serve a large and dispersed client population. Examples of such services include Content Distribution Networks (CDNs), where the clients are distinct from the nodes providing the service, and Peer-to-Peer (P2P) systems, where the clients also participate in providing the service. In these kinds of systems, the large number of clients use a variety of network paths to communicate with the service, and are therefore likely to see any path instabilities that occur between them and the service nodes.

This scenario of geographically-distributed clients accessing a wide-area service can itself be used as a monitoring infrastructure, since the natural traffic generated by the service can reveal information about the network paths being used. By observing this traffic, we can passively detect odd behavior and then actively probe it to understand it in more detail. This approach produces less overhead than a purely active-probing based approach.

This monitoring can also provide direct benefit to the wide-area service hosting the measurement infrastructure. By characterizing failures, the wide-area service can mitigate their impact. For example, if the outbound path between a service node and a client suddenly fails, it may be possible to mask the failure by sending outbound traffic indirectly through an unaffected service node, using techniques such as overlay routing [5]. More flexible services may adapt their routing decisions, and have clients use service nodes that avoid the failure entirely. Finally, a history of failure may motivate placement decisions—a service may opt to place a service node within an ISP if intra-ISP paths are more reliable than paths between it and other ISPs.

This chapter describes a monitoring system, PlanetSeer, that has been running on PlanetLab since February 2004. It passively monitors traffic between PlanetLab and thousands of clients to detect anomalous behavior, and then coordinates active probes from many PlanetLab sites to confirm the anomaly, characterize it, and determine its scope. This approach has several advantages: (1) because the clients are distributed at various geographic locations, we obtain a diverse set of network paths that we can monitor for anomalies; (2) PlanetLab nodes span a large number of autonomous systems (ASes), providing reasonable network coverage to initiate probing; and (3) active probing can be launched as soon as problems are visible in the passively-monitored traffic, making it possible to catch even short-term anomalies that last only a few minutes.

We are able to confirm roughly 90,000 anomalies per month using this approach, which exceeds the rate of previous active-probing measurements by more than two orders of magnitude [25]. Furthermore, since we can monitor traffic initiated by clients outside PlanetLab, we are also able to detect anomalies beyond those seen by a purely active-probing based approach.

In describing PlanetSeer, we make three contributions. First, we describe the design of the passive monitoring and active probing techniques we employ, and presents the algorithms we use to analyze the failure information we collect. Second, we report the results of running PlanetSeer over a three month period of time, including a characterization of the failures we see. Third, we discuss opportunities to exploit PlanetSeer diagnostics to improve the level of service received by end users.

While our focus is the techniques for efficiently identifying and characterizing routing anomalies, we must give some attention to the possibility of our host platform affecting our results. In particular, it has been recently observed that intra-PlanetLab paths may not be representative of the Internet [10], since these nodes are often hosted on research-oriented networks. Fortunately, by monitoring services with large client populations, we conveniently bypass this issue since most of the paths being monitored terminate outside of PlanetLab. By using geographically-dispersed clients connecting to a large number of PlanetLab nodes, we observe more than just intra-PlanetLab connections.

3.2 PlanetSeer Operation

This section describes our environment and our approach, including how we monitor traffic, identify potential routing anomalies, and actively probe them.

3.2.1 Components

We currently use the CoDeeN Content Distribution Network [90] as our host wide-area service, since it attracts a reasonably large number of clients (7K-12K daily) and generates a reasonable traffic volume (100-200 GB/day, 5-7 million reqs/day). CoDeeN currently runs on 120 PlanetLab nodes in North America (out of 350 worldwide), but it attracts clients from around the world. CoDeeN acts as a large, cooperative cache, and it forwards requests between nodes. When it does not have a document cached, it gets the document from the content provider (also known as the origin server). As a result, in addition to the paths between CoDeeN and the clients, we also see intra-CoDeeN paths, and paths between CoDeeN and the origin servers.

PlanetSeer consists of a set of passive monitoring daemons (MonD) and active probing daemons (ProbeD). The MonDs run on all CoDeeN nodes, and watch for anomalous behavior in TCP traffic. The ProbeDs run on all PlanetLab nodes, including the CoDeeN nodes, and wait to be activated. When a MonD detects a possible anomaly, it sends a request to its local ProbeD. The local ProbeD then contacts ProbeDs on the other nodes to begin a coordinated planet-wide probe. The ProbeDs are organized into groups so that not all ProbeDs are involved in every distributed probe.

Currently, some aspects of PlanetSeer are manually configured, including the selection of nodes and the organization of ProbeD groups. Given the level of trust necessary to monitor traffic, we have not invested any effort to make the system self-organizing or open to untrusted nodes. While we believe that both goals may be possible, these are not the current focus of our research.

Note that none of our infrastructure is CoDeeN-specific or PlanetLab-specific, and we could easily monitor other services on other platforms. For PlanetSeer, the appeal of CoDeeN (and hence, PlanetLab) is its large and active client population. The only requirement we have is the ability to view packet headers for TCP traffic, and the ability to launch traceroutes. On PlanetLab, the use of *safe raw sockets* [13] mitigates some privacy issues – the PlanetSeer service only sees those packets that its hosting service (CoDeeN) generates. In other environments, we believe the use of superuser-configured in-kernel packet filters can achieve a similar effect.

In terms of resources, neither ProbeD nor MonD require much memory or CPU to run. The non-glibc portion of ProbeD has a 1MB memory footprint. The MonD processes have a memory consumption tied to the level of traffic activity, and is used to store flow tables, statistics, etc. In practice, we find that it requires roughly 1KB per simultaneous flow, but we have made no effort to optimize this consumption. The CPU usage of monitoring and probing is low, with only analysis requiring much CPU. Currently, analysis is done offline in a centralized location, but only so we can reliably archive the data. We could perform the analysis on-line if desired – currently, each anomaly requires a 20 second history to detect, one minute to issue and collect the probes, and less than 10ms of CPU time to analyze.

3.2.2 MonD Mechanics

MonD runs on all CoDeeN nodes and observes all incoming/outgoing TCP packets on each node using PlanetLab's tcpdump utility. It uses this information to generate path-level and flow-level statistics, which are then used for identifying possible anomalies in real-time.

Although flow-level information regarding TCP timeouts, retransmissions, and round-trip times (RTTs) already exists inside the kernel, this information is not easily exported by most operating systems. Since MonD runs as a user-level process, it instead derives this information by observing packet-level activity from tcpdump. It instead infers flow-level information—e.g., timeouts, retransmissions, and round trip times (RTTs)—from the sniffed packets, and aggregates information from flows on the same path to infer anomalies on that path.

MonD maintains path-level and flow-level information, with paths identified by their source and destination IP addresses, and flows identified by both port numbers in addition to the addresses. Flow-level information includes information such as sequence numbers, timeouts, retransmissions, and round-trip times. Path-level information aggregates some flow-level information, such as loss rates and RTTs.

MonD adds new entries when it sees new paths/flows. On packet arrival, MonD updates a timestamp for the flow entry. Inactive flows, which have not received any traffic in *FlowLifeTime* (15 minutes in the current system), are pruned from the path entry, and any empty paths are removed from the table.

3.2.3 MonD Behavior

MonD uses two indicators to identify possible anomalies, which are then forwarded to ProbeD for confirmation. The first indicator is a change in a flow's Time-To-Live (TTL) field. The TTL field in an IP packet is initialized by a remote host and gets decremented by one at each hop along the traversed path. If the path between a source and destination is static, the TTL value of all packets that reach the destination should be the same. If the TTL changes in the middle of the stream, it usually means a routing change has occurred.

The second indicator, multiple consecutive timeouts, signals a possible path anomaly since such timeouts should be relatively rare. A TCP flow can time out several times from a single unacknowledged data packet, and each consecutive timeout causes the retransmission timeout period to double [88]. The minimum initial retransmission timeout in TCP ranges from 200ms (in Linux) to 1 second (in RFC 2988 [88]). Thus, n consecutive timeouts means either the data packets or the corresponding acknowledgment packets (ACKs) have not been received during the last $2^n - 1$ periods (seconds or 200ms ticks).

Our current threshold is four consecutive timeouts, which corresponds to 3.2–16 seconds. Since most congestion periods on today's Internet are short-lived (95% are less than 220ms [96]), these consecutive timeouts are likely due to path anomalies. We can further subdivide this case based on whether MonD is on the sender or receiver side of the flow. If MonD is on the receiver side, then no ACKs are reaching the sender, and we can infer the problem is on the path from the CoDeeN node to the client/server, which we call **forward path**. If MonD is on the sender side, then we cannot determine whether outbound packets are being lost or whether ACKs are being lost on the way to MonD.

3.2.4 MonD Flow/Path Statistics

We now describe how MonD infers path anomalies after grouping packets into flows. We examine how to measure the per-flow RTTs, timeouts and retransmissions.

To detect timeouts when MonD is on the sender side, we maintain two variables, *SendSeqNo* and *SendRtxCount* for each flow. *SendSeqNo* is the sequence number (seqno) of the most recently sent new packet, while *SendRtxCount* is a count of how many times the packet has been retransmitted. If we use *CurrSeqNo* to represent the seqno of the packet currently being sent, we see three cases: If *CurrSeqNo* $>$ *SendSeqNo*, the flow is making progress, so we clear *SendRtxCount* and set *SendSeqNo* to *CurrSeqNo*. If *CurrSeqNo* $<$ *SendSeqNo*, the packet is a fast retransmit, and we again set *SendSeqNo* to *CurrSeqNo*. If *CurrSeqNo* $=$ *SendSeqNo*, we conclude a timeout has occurred and we increment *SendRtxCount*. If *SendRtxCount* exceeds our threshold, MonD notifies ProbeD that a possible path anomaly has occurred.

A similar mechanism is used when MonD observes the receiver side of a TCP connection. It keeps track of the largest seqno received per flow, and if the current packet has the same seqno, a counter is incremented. Doing this determines how many times a packet has been retransmitted due to consecutive timeouts at the sender. When this counter hits our threshold, MonD notifies ProbeD that this sender is not seeing our ACKs. Since we are seeing the sender's packets, we know that this direction is working correctly. Note that this method assumes that duplicate packets are mostly due to retransmissions at the sender. This assumption is safe because previous work has shown that packets are rarely duplicated by the network [72].

Detecting TTL change is trivial: MonD records the TTL for the first packet received along each path. For each packet received from any flow on the same path, we compare its TTL to our recorded value. If MonD detects any change, it notifies ProbeD that a possible anomaly has occurred. Note that this case can aggregate information from all flows along the path.

3.2.5 ProbeD Operation

ProbeD is responsible for the active probing portion of PlanetSeer, and generally operates after being notified of possible anomalies by MonD. For the purpose of the following discussion, when an anomaly occurs, we call the CoDeeN node where the anomaly is detected the *local node*, and the corresponding remote client/server the *destination*. The active probing is performed using traceroute, a standard network diagnostic tool. ProbeD supports three probing operations:

Baseline Probes: When a new IP address is added to MonD's path table, the ProbeD on the local node performs a "baseline probe" to that destination. It is expected that the results of this traceroute reflect the default network path used to communicate with that destination under normal conditions. For actively-used communication paths, a baseline probe is launched once every 30 minutes. When PlanetSeer is run on CoDeeN nodes, these baseline probes are generated whenever a new client connects to a node, or when a node has to contact a new origin server.

Forward Probes: When a possible anomaly is detected by the local MonD and reported to ProbeD, it invokes multiple traceroutes from a set of geographically distributed nodes (including itself) to the destination; we call the traceroute from the local node the *local traceroute* or *local path*. This process is performed twice, generally within one minute, in order to identify what we term *ultrashort* anomalies. On particularly problematic paths, MonD might report possible anomalies very frequently, especially if the path is very unstable. To avoid generating too much probing traffic, ProbeD rate-limits the forward probes so that it does not probe the same destination within 10 minutes.

Reprobes: If the forward probes confirm an anomaly along a path to a destination, the local ProbeD that initiated the forward probes periodically reprobes that path to determine the duration and effects of the anomaly. We currently reprobe four times, at 0.5, 1.5, 3.5, and 7.5 hours after the anomaly detection time. These reprobes

Category	Grps	Sites	Descriptions
US (edu)	11	70	US Universities
US (non-edu)	5	13	Intel, HP, NEC, etc.
Canada	2	11	Eastern & Western Canada
Europe	7	31	UK, France, Germany, etc.
Asia & MidE	4	14	China, Korea, Israel, etc.
Others	1	6	Australia, Brazil, etc.
Total	30	145	

Table 3.1: Groups of the probing sites

can compare their traceroute results with the original baseline probe as well as the forward probes.

3.2.6 ProbeD Mechanics

When ProbeD performs the forward probes, it launches them from geographically distributed nodes on PlanetLab. Compared with only doing traceroute from the local node, using multiple vantage points gives us a more complete view of the anomaly, such as its location, pattern, and scope. Our ProbeDs are running on 353 nodes across 145 sites on PlanetLab, more than the number of nodes running CoDeeN. They are distributed across North/South America, Europe, Asia and elsewhere.

Since the set of ProbeDs must communicate with each other, they keep some membership information to track liveness. We note that an unavailable ProbeD only results in a degradation of information quality, rather than complete inoperability, so we do not need to aggressively track ProbeD health. Each ProbeD queries the others when it first starts. Thereafter, dead ProbeDs are checked every 8 hours to reduce unneeded communication. In the course of operation, any ProbeD that is unresponsive to a query is considered dead.

We divide the ProbeD nodes into 30 groups based on geographic diversity, as shown in Table 3.1, mainly to reduce the number of probes launched per anomaly. Probing every anomaly from every ProbeD location would yield too much traffic (especially to sites with conservative intrusion detection systems), and the extra traffic may not yield much insight if many nodes share the same paths to the anomaly. We also divide North America into educational and non-educational groups, because the educational (.edu) sites are mostly connected to Internet2, while non-educational sites are mostly connected by commercial ISPs.

When a ProbeD receives a request from its local MonD, it forwards it to a ProbeD in each of the other groups. The ProbeDs perform the forward probes, and send the results back to the requester. All results are collected by the originator, and logged with other details, such as remote IP address and the current time.

3.2.7 Path Diversity

We have been running PlanetSeer since February 2004. In three months, we have seen 887,521 unique clients IPs, coming from 9232 Autonomous Systems (ASes) (according to previous IP-to-AS mappings techniques [62]). Our probes have traversed 10090 ASes, well over half of all ASes on the Internet. We use a hierarchical AS classification scheme that has five tiers, based on AS relationships and connectivity [85]. The highest layer (tier 1) represents the *dense core* of the Internet, and consists of 22 ASes of the largest ISPs. Tier 2 typically includes ASes of other large national ISPs. Tier 3 includes ASes of regional ISPs. Finally, tiers 4 and 5 include ASes of small regional ISPs and customer ASes respectively. As shown in Table 3.2, we have very good coverage of the top 4 AS tiers, with complete coverage of tier 1 and nearly-complete coverage of tier 2.

Tier #	Covered	Tier Size	Coverage
Tier 1	22	22	100%
Tier 2	207	215	96%
Tier 3	1119	1392	80%
Tier 4	1209	1420	85%
Tier 5	5906	13872	43%
Unmapped	1627		

Table 3.2: Path diversity

3.3 Confirming Anomalies

Having collected the passive data from MonD and the traceroutes from ProbeD, the next step is processing the probes to confirm the existence of the anomaly. This section describes how we use this data to classify anomalies and quantify their scope. It also reports how different types of anomalies influence end-to-end performance.

3.3.1 Massaging Traceroute Data

Some of the data we receive from the traceroutes is incomplete or unusable, but we can often perform some simple processing on it to salvage it. The unusable hops in a traceroute are those that do not identify the routers or that identify the routers by special-use IP addresses [41]. The missing data is generally the absence of a hop, and can be interpolated from other traceroutes.

Identifying and pruning unusable hops in traceroute is simple: the unusable hops are identified by asterisks in place of names, and other than showing the existence of these hops, these data items provide no useful information. We simply remove them from the traceroute but keep the relative hop count difference between the existing hops.

Missing hops in a traceroute are slightly harder to detect, but we can use overlapping portions of multiple traceroutes to infer where they occur. We use a simple heuristic to identify the missing hops: we compare traceroutes that share the same destination, and if the hops leading to the destination differ by an intermediate hop that is present in one and missing in the other, we replace the missing hop with the address in the other trace. Put more formally, given two traceroutes from two sources to the same destination, suppose there are two subsequences of these two traceroutes, $X(X_1, X_2, ..., X_m)$ and $Y(Y_1, Y_2, ..., Y_n)$ ($m > 2$ and $n > 2$) such that $X_1 = Y_1$ and $X_m = Y_n$. We define $hop(X_i)$ to be the hop count of X_i. Note that the number of hops between X_1 and X_m, $hop(X_m) - hop(X_1)$, can be greater than $m - 1$, because we have removed "*" and special-use IPs from the traceroutes. If $hop(X_m) - hop(X_1) = hop(Y_n) - hop(Y_1)$, it is very likely that all the hops in X and Y are the same since they merge at $X_1(Y_1)$ and follow the same number of hops to $X_m(Y_n)$. If there exists X_i such that the hop corresponds to $hop(Y_1) + hop(X_i) - hop(X_1)$ in Y does not exist because it has been removed as "*", we consider X_i a missing hop in Y and add this hop into Y.

Our approach to inserting missing hops helps us filter out the "noise" in traceroutes so that it does not confuse our anomaly confirmation using route change as described in Section 3.3.2. However, it may mask certain hop differences. For example, we sometimes see two paths X and Y merge at $X_1(Y_1)$, and diverge at some later hops before merging at $X_m(Y_n)$ again. This usually occurs between tightly-coupled routers for load balancing reasons, where a router selects the next hop from several parallel links based on the packet IP addresses and/or traffic load [71]. In this case, inserting missing hops may eliminate the different hops between the two traceroutes.

For our purposes, we do not treat such "fluttering" [71] as anomalies because it occurs as a normal practice. We detect fluttering by looking for hops X_i and Y_j such that

TTL Change	Timeout	Fwd Timeout
994485 (44%)	754434 (33%)	510669 (23%)

Table 3.3: Breakdown of anomalies reported by MonD

$hop(Y_j) - hop(Y_1) = hop(X_i) - hop(X_1)$ but $X_i \neq Y_j$, and we merge them into the same hop in all the traceroutes. Note that this could also possibly eliminate the hop difference caused by path change and lead to underestimating the number of anomalies.

3.3.2 Final Confirmation

After we have processed the traceroutes, we are ready to decide whether an event reported by MonD is actually an anomaly. We consider an anomaly "confirmed" if *any* of the following conditions is met:

Loops: There is a loop in the local traceroute from the local node to the destination, which means the anomaly is triggered by routing loops.

Route change: The local traceroute disagrees with the baseline traceroute. Note that the baseline traceroute is no more than 30 minutes old. Given that 91% of the Internet paths remains stable for more than several hours [71], the anomaly is most likely caused by path change or path outage.

Partial unreachability: The local traceroute stops before reaching the destination, but there exist traceroutes from other nodes that reach the destination. This could be caused by path outages.

Forwarding failures: The local traceroute returns an ICMP destination unreachable message, with code of net unreachable, host unreachable, net unknown, or host un-

Non-Anomaly	Anomaly	Undecided
1484518 (66%)	271898 (12%)	503172 (22%)

Table 3.4: Breakdown of reported anomalies using the four confirmation conditions

known. This usually indicates that a router does not know how to reach the destination because of routing failures [45].

Our confirmation process is very conservative—it is possible that some of the reported anomalies are real, but do not meet any of the above conditions. However, our goal is to obtain enough samples of anomalies for our analysis and we do not want our results to be tainted by false positives. Hence, we choose stricter conditions for confirming anomalies. Similarly, we confirm a reported anomaly as *non-anomaly* if the local traceroute does not contain any loop, agrees with the baseline traceroute, and reaches the destination. For a confirmed non-anomaly, we do not perform traceroutes at remote ProbeDs, in order to reduce measurement traffic.

In three months, we have seen a total of 2,259,588 possible anomalies reported, of which we confirmed 271,898. Table 3.3 shows the number of reported anomalies of each type. As we can see, TTL change is the most common type of reported anomaly, accounting for 44% of the reported anomalies. For the remaining anomalies triggered by timeouts, passive monitoring suggests that 23% are most likely caused by forward path problems.

Table 3.4 shows the breakdown of anomalies using the 4 confirmation conditions. The non-anomalies account for 2/3 of the reported anomalies. Among the possible reasons for the occurrence of non-anomalies are: ultrashort anomalies, path-based TTL, and aggressive consecutive timeout levels. Some anomalies, which we term ultrashort, are so short that our system is unable to respond in time. Since they often are in the process of being resolved when the forward probes are taking place, the traceroute results may be inconsistent. Many false alarms from path-based TTL changes are due to NAT boxes. When clients with different initial TTL values share a NAT box, their interleaved packets appear to show TTL change. Using flow-based TTL change would reduce these false alarms, but may miss real TTL changes that occur between flows since any path history would be lost. Finally, our consecutive timeout conditions may be aggressive for hosts with short timeout periods. Excluding the non-anomalies, we confirm 271898 (35%) of the remaining anomalies and probe them from multiple vantage points. We use these anomalies for our analysis in the remainder of this section.

3.4 Loop-Based Anomalies

This section focuses on analyzing routing loops, which can occur due to inconsistencies in routing state, misconfiguration, and other causes. We are interested in their frequency, duration, size, location, and effect on end-to-end performance.

We detect routing loops by observing the same sequence of routers appearing several times in a traceroute. Since loops in traceroute may reflect upstream routing changes rather than true routing loops [71], we choose to take a conservative approach and require that the same sequence appear in a traceroute **at least 3 times** before we confirm it as a routing loop.

Using this metric, we identify a total number of 21565 routing loops in our data. If we relax the loop condition to allow loops that have the same sequence only twice, this count increases to 119,936, almost six times as many. Using this approach, our overall confirmed anomaly count would increase 36% to over 370K.

Loops are separated into persistent and temporary loops [71] based on whether the traceroute was ultimately able to exit the loop. If the traceroute stays within the loop until the maximum number of hops (32 in our case), we classify it as persistent, while if the loop is resolved before the traceroute reaches the maximum hop, it is temporary. Temporary loops can occur due to the time necessary to propagate updated routing information to the different parts of the network, while persistent loops can be caused by various reasons, including router misconfiguration [61]. Persistent loops tend to last longer and may require human intervention to be resolved, so they are considered more harmful. About 83% of the observed loops are persistent, as shown in Table 3.5. Since temporary loops only exist for a short period, it may be harder to catch them.

We use the reprobes to determine duration of the persistent loops. The reprobes for some persistent loops are missing, often because the local PlanetLab node failed or rebooted before all reprobes completed. For those loops, we do not know when the loops were resolved. We only include the loops having all 4 reprobes in our analysis. Therefore, we show the percentage of loops in each duration instead of the exact numbers in Table 3.5. We can see many persistent loops are either resolved quickly (54% terminate within half an hour) or last for a long time (23% stay for more than 7.5 hours).

Previous research has noted that routing loops are likely to be correlated [71], because nearby routers usually share routing information very quickly. If some routers have inconsistent information, such information is likely to be propagated to other nearby routers and cause those routers to form loops. We observe a similar phenomenon, which is quantified in Table 3.5. We count the number of distinct loops in traceroutes from other ProbeDs during the same period. We find that 16% of the temporary loops are accompanied by at least one disjoint loop while only 6% of the persistent loops see them. We suspect the reason is temporary loops are more likely to stem from inconsistent routing

	Temporary	Persistent
Total	3565 (17%)	18000 (83%)
$\leq$ 30min	N/A	54%
$\leq$ 1.5 hrs	N/A	11%
$\leq$ 3.5 hrs	N/A	6%
$\leq$ 7.5 hrs	N/A	6%
$>$ 7.5 hrs	N/A	23%
Single Loop	3008 (84%)	17007(94%)
Multiple Loops	557 (16%)	993 (6%)
1 AS	3021, (85%)	17895, (99%)
2 ASes	416, (12%)	101, (1%)
3 ASes	106, (3%)	4, (0%)
$\geq$4 ASes	22, (0%)	0, (0%)
Tier-1 AS	510 (15%)	244 (2%)
Tier-2 AS	859 (25%)	789 (6%)
Tier-3 AS	1378(40%)	6263 (46%)
Tier-4 AS	197 (5%)	3899 (29%)
Tier-5 AS	538 (15%)	2401 (17%)
Total	3482	13596

Table 3.5: Summarized breakdown of 21565 loop anomalies. Some counts less than 100% because some ASes are not in the AS hierarchy mapping.

state while persistent loops are more likely to be caused by other factors which may not be related to routing inconsistency.

3.4.1 Scope

Besides their frequency, one of the important factors that determines the effect of routing loops is their *scope*, including how many routers/ASes are involved in the loop, and where they are located. We use the term *loop length* to refer to the number of routers involved, and we show a breakdown of this metric in Table 3.6. The most noticeable feature is that temporary loops have much longer lengths than persistent loops. 97% of the persistent loops consist of only 2 routers, while the ratio is only 50% for temporary loops.

	2	3	4	5	6+
Persistent/All	97%	2%	1%	0%	0%
Persistent/Core	94%	4%	1%	1%	0%
Persistent/Edge	97%	2%	1%	0%	0%
Temporary/All	51%	29%	11%	7%	2%
Temporary/Core	45%	31%	13%	8%	3%
Temporary/Edge	53%	27%	12%	6%	2%

Table 3.6: Number of hops in loops, as % of loops

Intuitively, the more routers are involved in a loop, the less stable it is. Therefore, most persistent loops exist between only two routers, while temporary loops span additional routers as the inconsistent state propagates.

We next examine the number of ASes that are involved in the loops. The breakdown is shown in Table 3.5, which shows that persistent loops overwhelmingly occur within a single AS, while 15% of temporary loops span multiple ASes. Ideally, BGP prevents any inter-AS loops by prohibiting a router from accepting an AS path with its own AS number in that path. However, BGP allows transient inconsistency, which can arise during route withdrawals and announcements [37], and this is why we see more temporary loops spanning multiple ASes. In contrast, persistent loops can occur due to static routing [71] or router misconfigurations [61]. Given how few persistent loops span multiple ASes, it appears that BGP's loop suppression mechanism is effective.

In Table 3.6, we compare the loops in the core network (tiers 1, 2, 3) or the edge network (tiers 4, 5). As the table shows, both temporary and persistent loops are likely to involve more hops if occurring in the core network.

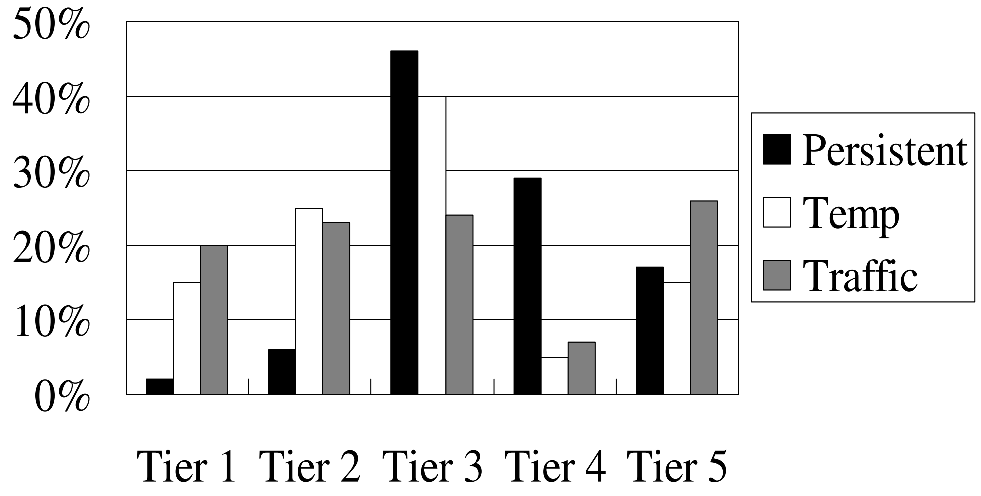

Figure 3.1: Percentage of loops and traffic in each tier

3.4.2 Distribution

Using the loop data, we can gain some insight into the distribution of the loops. We want to understand whether loops are more likely to arise in the core of the network or near the edge. In theory, we could do this by calculating the depth of the loops [25], which would tell us the minimum number of hops from the routers to the network edge. However, since we cannot launch probes from the clients, this depth metric could be misleading. If the loop occurs in an AS that does not lie near any of our ProbeD locations, our probes have to travel through the network core to reach it, and we would believe it to be very far from the edge. If we could launch probes from the clients, network depth would be meaningful.

To avoid this problem, we classify the loops according to the hierarchy of ASes involved. We map loops into tiers by using the tier(s) of the AS(es) involved. A loop can be mapped to multiple tiers if it involves multiple ASes. Note that we have to also consider the total amount of traffic of each tier when quantifying the loop distribution. Apparently, we have a higher chance to observe loops in a tier that is traversed more frequently.

For this reason, we compute the traffic of a tier as the total number of times that tier is traversed. Figure 3.1 shows the percentage of loops and traffic occurring in each tier. Compared with their relative traffic share, tier-1 ASes have very few persistent and temporary loops, possibly because they are better provisioned than smaller ASes. In contrast, a large number of loops occur in tier-3 (outer core) ASes, which suggests that the paths in those regional ISPs are less stable. Another interesting thing in Figure 3.1 is tier-5 ASes seem to have few loops although they are small ASes. One possible explanation is loops can evolve when routers adapt to failures by exploring alternate paths. Since there is less path diversity in small networks, loops are less likely to evolve.

We also examine the relative distribution of AS quality by measuring how evenly-distributed the persistent loops are. Since these loops are largely single-AS, they are very unlikely to arise from external factors, and may provide some insight into the monitoring/management quality of the AS operators. We use a metric, which we call *skew*, to provide some insight into the distribution. We calculate skew as the percentage of per-tier loops seen by the "worst" 10% of the ASes in that tier. A skew value of 10% indicates all ASes in the tier are likely to be uniform in the quality, while larger numbers indicate a wider disparity between the best ASes and the worst.

In tier 1, the top 2 ASes (10% of 22) account for 35% of the loops, while in tier 2, the top 21 ASes (10% of 215) account for 97% of the loops. This skew may translate into different reliabilities for the customers they serve. The disparity in traffic must also be considered when judging how important these skew numbers are. With respect to the traffic that we observe, we find that these ASes account for 20% of tier 1 traffic and 63% of tier 2 traffic. The disparity between the loop rates and the traffic for these ASes would indicate that these ASes appear to be much more problematic than others in their tier.

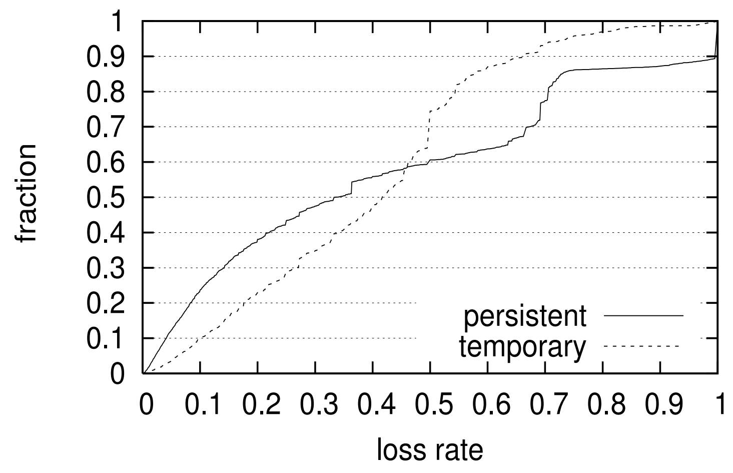

Figure 3.2: CDF of loss rates preceding the loop anomalies

3.4.3 End-to-End Effects

Loops can degrade end-to-end performance in two ways: by overloading routers due to processing the same packet multiple times [37] (for temporary loops), or by leading to loss of connectivity between pairs of hosts (for permanent loops). Since MonD monitors all flows between the node and remote hosts, we can use the network statistics it keeps to understand end-to-end effects.

When MonD suspects an anomaly, it logs the retransmission rate and RTT *leading up to that point*. Retransmission rates are calculated for the last 5 minutes. RTTs are calculated using an Exponentially Weighted Moving Average (EWMA) with the most recent value given a weight of 1/8, similar to that used in TCP. Figure 3.2 shows the CDF of the retransmission rate, and we see that 65% of the temporary loops and 55% of the persistent loops are preceded by loss rates exceeding 30%. Since the typical Internet loss rate is less than 5% [72], this higher loss rate will significantly reduce end-user TCP performance prior to the start of the anomaly.

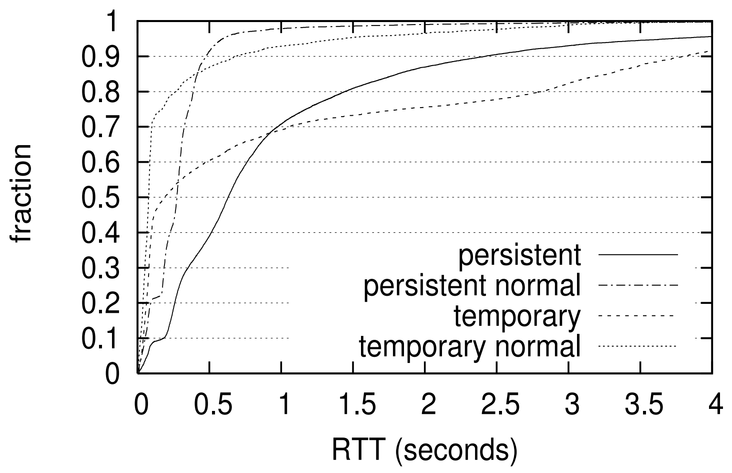

Figure 3.3: CDF of RTTs preceding the loop anomalies *vs.* under normal conditions

In addition to the high loss rates, loop anomalies are also preceded by high latency, as shown in Figure 3.3. High latency can be caused by queuing due to congestion or packets repeatedly traversing the same sequence of routers. We compare the RTT right before loops occur with the RTT measured in the baseline traceroute on the same path. It is evident that loops can significantly degrade RTTs.

3.5 Building a Reference Path

While loop-based anomalies are relatively simple to identify, they represent a relatively small fraction of the total confirmed anomalies. Classifying the other anomalies, however, requires more effort. This section discusses the steps taken to classify the non-loop anomalies, and the complications involved. The main problem is how to determine that the anomaly occurs on the forward path from the local node to the destination. Additionally, we want to characterize the anomaly's features, such as its pattern, location and affected routers.

To deal with these two problems, we need a *reference path* from the local node to the destination, which represents the path before the anomaly occurs. Then we can compare it against the local traceroute during the anomaly. This comparison serves three purposes: First, it can help us distinguish between forward-path and reverse-path anomalies. If the local path during the anomaly is different from the reference path, the route change usually indicates that the anomaly is on the forward path. Second, it can be used to quantify the scope of the anomaly. By examining which routers overlap between the local path and the reference path, we can estimate which routers are potentially affected by the anomaly. It can also be used to determine the location of the anomaly, in terms of the number of the router hops between the anomaly and the end hosts. Third, it is used to classify the anomalies. Based on whether the local traceroute reaches the last hop of the reference path, we can classify it as either path change or path outage.

Ideally, we could simply use the baseline traceroute as the reference path, if it successfully reaches the destination. If the local traceroute during an anomaly stops at some intermediate hop, we know it is an outage. If the local traceroute is different, but reaches the destination, we know it is a routing change. However, the baseline traceroute may not reach the destination for a variety reasons. Some of these include:

- The destination is behind a firewall which filters traceroutes. In this case, we still want to use it as the reference path, which can be compared with local traceroute to analyze anomalies.

- Some intermediate routers filter traceroutes. In this case, we do not have enough information about the hops after the last known hop on the forward path. When an outage occurs, we cannot quantify where it occurs since the anomaly may occur after the last known hop.

- The baseline traceroute is also affected by the anomaly and fails to reach the destination. In this case, we cannot use it as a reference because it usually does not provide more useful information than the local traceroute.

The rest of this section focuses on deciding whether the baseline traceroute can be used as the reference path when it does not reach the destination. If a baseline path S stops at hop S_x, we try to guess if S_x is a firewall using some heuristics. S_x must meet the following four requirements before we consider it a firewall:

1. From MonD's passive data, we know the client is able to send and receive TCP packets with the local node. Therefore, the path is working when the baseline traceroute is being calculated.

2. S does not return an ICMP destination unreachable message, which usually indicates that the traceroute encounters routing problems at S_x [45].

3. S_x and the destination are in the same AS. We assume that a firewall and its protected clients should belong to the same organization.

4. S_x is within n hops (close) to the destination.

The first three requirements are easy to verify, so we focus on the last requirement. Let $RevHop(h)$ be the number of hops from hop h to the local node on the reverse path. We first want to check if $0 < RevHop(dst) - RevHop(S_x) \leq n$. From MonD, we know $RevTTL(dst)$, the TTL of a packet when it arrives at the local node from the destination. If the TTL is initialized to $InitTTL(dst)$ by the destination, we have $InitTTL(dst) - RevTTL(dst) = RevHop(dst)$ because the TTL is decremented at each hop along the reverse path. The issue is how to determine $InitTTL(dst)$. The initial TTL values differ

Type	Number	Percentage
Path Change	120,283	48%
Forward Outage	23,921	10%
Other Outage	62,107	24%
Temporary	44,022	18%
Total	250,333	100%

Table 3.7: Non-loop anomalies breakdown

by OS, but are generally one of the following: 32 (Win 95/98/ME), 64 (Linux, Tru64), 128 (Win NT/2000/XP), or 255 (Solaris). Because most Internet paths have less than 32 hops, we can simply try these 4 possible initial TTL values and see which one, when subtracted by $RevTTL(dst)$, gives a $RevHop(dst)$ that is less than 32 hops. We will use that as $InitTTL(dst)$ to calculate $RevHop(dst)$. Similarly, from the traceroute, we can also calculate $RevHop(S_x)$ using $RevTTL(S_x)$.

Although inter-AS routing can be asymmetric, intra-AS paths are usually symmetric [71]. Since S_x and the destination are in the same AS, their forward hop count difference should be the same as their reverse hop count difference, which we are able compute as described above.

Choosing an appropriate n for all settings is difficult, as there may be one or more hops between a firewall and its protected hosts. We conservatively choose $n = 1$, which means we consider S as a valid reference path only when S_x is one hop away from the destination. This will minimize the possibility that a real path outage is interpreted as a traceroute being blocked by a firewall. However, it leads to bias against large organizations, where end hosts are multiple hops behind a firewall. In such cases, we cannot determine if the anomalies are due to path outage or blocking at a firewall. Therefore, we do not further analyze these anomalies.

3.6 Classifying Non-loop Anomalies

In this section, we discuss classifying anomalies by comparing the reference path R with the local path L. There are three possibilities when we compare L and R:

1. L reaches the last hop of R. In this case, L must differ from R in some intermediate hops, or else we would not have confirmed it as an anomaly. This case corresponds to a *path change*, which will be discussed in Section 3.6.1.

2. If L stops at some intermediate hop of R, it could be due to path outage on the forward path or reverse path failure. We will describe how to distinguish between them in Section 3.6.2.

3. If L diverges from R after some hops and stops before merging into R, we consider it as a path outage although it is accompanied by a route change. We will also discuss this case in Section 3.6.2.

We observe a total of 250333 non-loop anomalies, with their breakdown shown in Table 3.7. About half of them are path changes, and 10% are forward path outages. For the 24% that are classified as other outages, we cannot infer whether they are on the forward or reverse paths. The remaining 18% are temporary anomalies. In these cases, the first local traceroute does not match the reference path, but the second local traceroute matches. In these cases, the recovery has taken place before we can gather the results of the remote probes, making characterization impossible. While it is possible that some remote probes may see the anomaly, the rapidly-changing state is sure to cause inconsistencies if we were to analyze them. To be conservative, we do not perform any further analysis of these anomalies, and focus only on path changes and forward outages. These temporary anomalies are different from the ultrashort anomalies in that the ultrashort anomalies

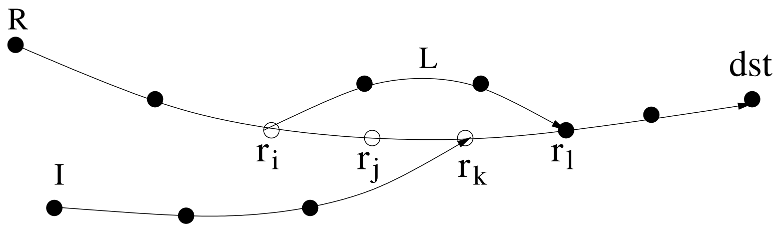

Figure 3.4: Narrowing the scope of path change

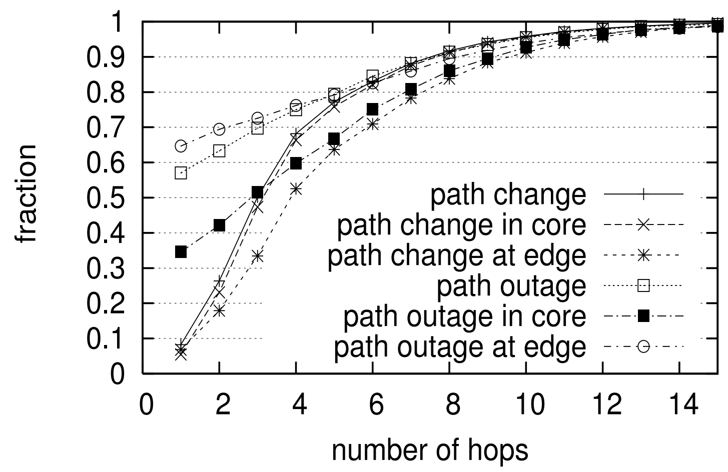

Figure 3.5: Scope of path changes and forward outages in number of hops

were already in the repair process during the first probe. So, while we choose not to analyze temporary anomalies further, we can at least inarguably confirm their existence, which is not the case with the ultrashort anomalies.

3.6.1 Path Changes

We first consider path changes, in which the local path L diverges from the reference path R after some hops, then merges back into R and successfully terminates at the last hop of R. This kind of anomaly is shown in Figure 3.4.

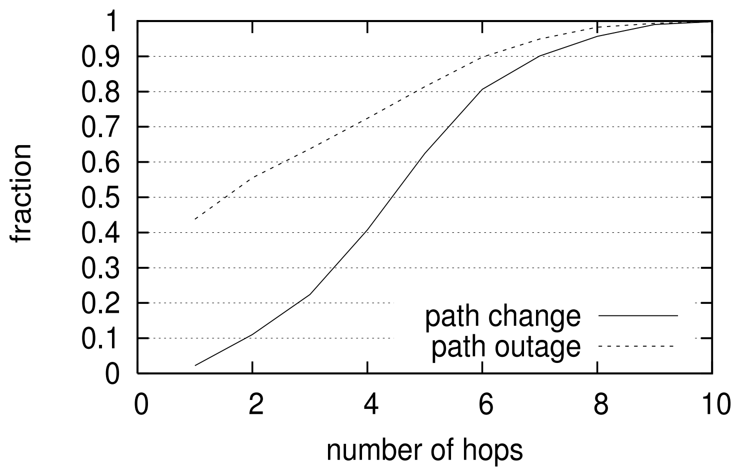

Figure 3.6: Distance of path changes and forward outages to the end hosts in number of hops

Scope and End-to-End Effects

As discussed in Section 1.2, it is usually very difficult to locate the origin of path anomalies purely from end-to-end measurement [28]. However, even if the precise origin cannot be determined, we may be able to narrow the *scope* of the anomaly. We define the scope of a path change as the number of hops on R which possibly change their next hop value. Flows through these routers may all have their paths changed. In Figure 3.4, L diverges from R at r_i and merges into R at r_l. All the hops before r_i or after r_l (including r_l) follow the same next hop towards the destination. So the hops which are possibly influenced by the path change and have different next hops are r_i, r_j and r_k.

In some cases, we may be able to use remote traceroutes to narrow the scope even further. For example, in Figure 3.4, if I, a traceroute from another ProbeD merges into R at r_k, a hop that is before r_l, we can eliminate r_k from the scope of the path change anomaly, since we know r_k has the same next hop value as it did in the reference path. We call I the *intercept path*. This method may still overestimate the scope of path change: it is possible, for example, that r_j's next hop value is unaffected, but we cannot know this unless some traceroute merges into R at r_j.

Performing traceroute from multiple geographically diverse vantage points increases our chances of finding an intercept path. Our ability of narrowing the scope is affected by the location of the anomaly. If it is closer to the destination, we have a better chance of obtaining intercept paths by launching many forward traceroutes, and thus successfully reducing the scope of the anomaly. In contrast, if the anomaly is nearer our source, the chance of another traceroute merging into the reference path early is low.

Figure 3.5 shows the CDF of path change scope, measured in hop count. We can confine the scope of 68% of the path changes to within 4 hops. We do not count fluttering as path changes, since these would appear as a large number of path change anomalies, each with a very small scope. We also examine how many ASes the path change scope spans, shown in Table 3.8. Again, we can confine the scope of 57% of the path changes to within two ASes and 82% of them to within three ASes.

To gain some insight into the location of the anomalies, we also study whether the path change occurs near end hosts or in the middle of the path [25]. We measure the *distance* of a path change to the end host by averaging the distances of all the routers within the path change scope. The *distance* of a router is defined as the minimum number of hops to either the source or the destination. Figure 3.6 plots the CDF of path change distances. As we can see, 60% of the path changes occur within 5 hops to the end hosts.

Similar to Section 3.4.2, we use AS tiers to characterize distribution of the anomalies. We map the routers within anomaly scopes to ASes and AS tiers. The breakdown of possibly affected routers by their AS tiers is shown in Figure 3.7. Compared with their relative traffic share, the ASes in Tier-3 are most likely to be affected by path changes.

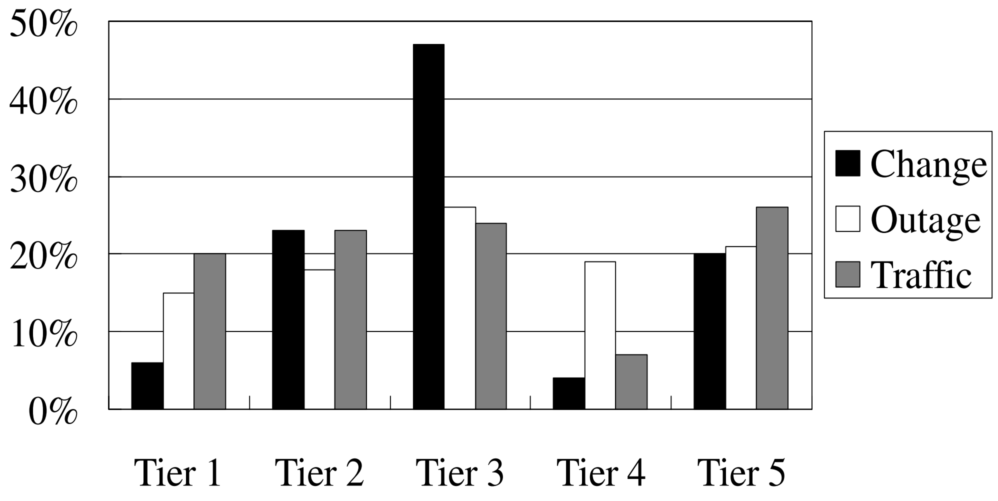

Figure 3.7: Percentage of forward anomalies and traffic in each tier

They account for nearly half of the total. By comparison, the ASes in tier-1 are rather stable.

In Figure 3.5, we see that path changes in the core network have narrower scope than those in the edge. This is probably because the paths in the core network are likely to be traversed by traceroutes from many vantage points to reach the destination. In contrast, if a route change occurs near a local node, we have less chance of finding an intercept path that happens to merge into the reference path early. As a result, the anomaly scope in these cases is more loosely confined.

Since path change is a dynamic process and anomaly scopes may evolve over time, a measured anomaly scope should be viewed as a snapshot of which routers are affected when the traceroutes reach them. In Table 3.8, we show how many path changes have changed scope between the first and second sets of forward probes. We find that only 4% of them have changed during that period (mostly within one minute). In addition, 66% of the scope changes are due to local path changes instead of intercept path changes.

	Change	Fwd Outage
Total	120283	12740
No Ref Path	N/A	11181
1 AS	24418 (20%)	6534 (51%)
2 ASes	43909 (37%)	3413 (27%)
3 ASes	29426 (25%)	1321 (10%)
4 ASes	12603 (10%)	856 (7%)
5 ASes	6322 (5%)	411 (3%)
6 ASes	3605 (3%)	205 (2%)
Guessed Last Hop	N/A	1055
Scope Changed	4292 (4%)	1225 (10%)
Tier-1 AS	12374 (6%)	2746 (15%)
Tier-2 AS	43104 (23%)	3255 (18%)
Tier-3 AS	88959 (47%)	4638 (26%)
Tier-4 AS	8015 (4%)	3501 (19%)
Tier-5 AS	38313 (20%)	3838 (21%)
Total	190765	17978

Table 3.8: Summary of path change and forward outage. Some counts exceed 100% due to multiple classification.

We now examine the effect of path changes on end-to-end performance. The effect of path changes on RTTs is relatively mild, as can be seen in Figure 3.10. The RTTs measured during path changes are only slightly worse than the RTTs measured in baseline traceroutes. But the loss rates during path changes can be very high. Nearly 45% of the path changes cause loss rates higher than 30%, which can significantly degrade TCP throughput.

3.6.2 Path Outage

We now focus on path outages and describe how to distinguish between forward and reverse path outages. In Figure 3.8, the local path L stops at r_i, which is an intermediate hop on the reference path R. At first glance, one might conclude that a failure has oc-

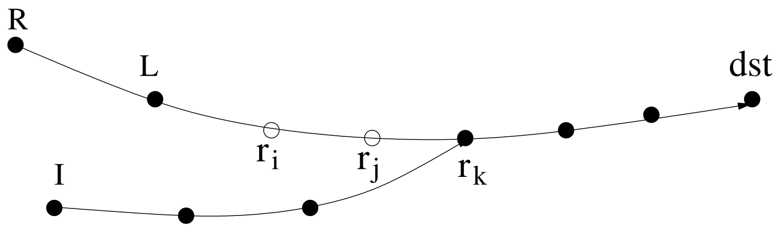

Figure 3.8: Narrowing the scope of forward outage

curred after r_i on the forward path, which prevents the packets from going through; but other possibilities also exist—because Internet paths could be asymmetric, a failure on the reverse path may produce the same results. For example, if a shared link on the reverse paths from all the hops beyond r_i to the source has failed, none of the ICMP packets from those hops can return. Consequently, we will not see the hops after r_i.

If we have control of the destination, we can simply distinguish between forward and reverse path outages using ping [25]. However, since our clients are outside of PlanetLab and not under our control, we cannot perform pings in both directions, and must use other information to disambiguate forward path outages from reverse path failures. Specifically, we can infer that the outage is on the forward path using the following rules:

- There is a route change on the forward path in addition to the outage.
- The local traceroute returns an ICMP destination unreachable message.
- The anomaly is reported as *timeouts on the forward path.* As described in Section 3.2.4, MonD will report this type of anomaly when it infers ACK losses on the forward path from the local node to the client.

Table 3.9 shows the number of forward path outages inferred from each rule. As we can see, all three rules are useful in identifying forward outages. More than half of the outages are accompanied by route changes, as the failure information is propagated and

Route change	Unreachable	Fwd Timeout
12822 (54%)	2751 (11%)	8348 (35%)

Table 3.9: Breakdown of reasons for inferring forward outage

some routers try to bypass the failure using other routes. Forward timeouts help us infer one-third of the forward outages. This demonstrates the benefit of combining passive monitoring with active probing, since we would not have been able to disambiguate them otherwise.

Scope

To characterize the scope of path outages, we use a technique similar to the one we used to characterize the scope of path change. We define a path outage scope as the number of hops in a path that cannot forward packets to their next hop towards the destination. In Figure 3.8, R is the reference path and L is the local path. L stops at r_i, which is an intermediate hop of R. Hence, all the hops after r_i (including r_i) are possibly influenced by the outage and may not be able to forward packets to the next hops towards the destination. However, when we can find another intercept path, we can narrow the scope. For example, if I merges into R at r_k and reaches the destination, then only r_i and r_j can possibly be influenced by the outage. Again, this method might overestimate the scope of a path outage, for the same reasons described earlier on estimating a path change scope.

Note that unlike previous work [21, 25], we use a set of routers to quantify the effect of path outages instead of just using the last hop where the traceroute stops. Since outage information is usually propagated to many routers, using only one router does not give us a sense of how many routers may have been affected by the outage.

In some cases, we may not have a complete baseline path which reaches the destination or the penultimate router. In these cases, we can not estimate the scope of the

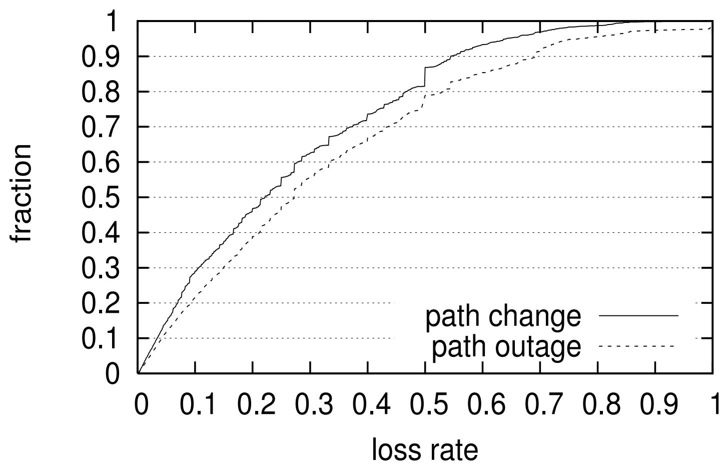

Figure 3.9: CDF of loss rates preceding path changes and forward outages

forward outage because we do not know the exact number of hops between the last hop of the baseline path and the destination. We only know that the anomaly occurs somewhere on the forward path. Among all the outages, about 47% have no complete reference path. In the following, we use only those with complete reference paths in the scope analysis.

In Figure 3.5, we plot the CDF of the number of hops in the forward outage scope. Compared with path change, we can confine the outage scope more tightly. Nearly 60% of the outages can be confined to within 1 hop and 75% of them can be confined to 4 hops.

We suspect that such tight confinement is due to last hop failures. In Figure 3.6, we plot the distances of forward outages to the end hosts. The distance of an outage is defined as the average distance of the routers within the outage scope to the end hosts, similar to the definition used for path change in Section 3.6.1. As we can see, 44% of the outages do occur at the last hop, allowing us to confine their scopes to 1 hop. This observation explains why the outages in the edge network are confined more tightly than those in core networks, as shown in Figure 3.5.

Excluding last hop failures, we can only confine 14% of the outages to one hop, a result that is slightly better than that for path changes. In general, the scopes of path outages tend to be smaller than those of path changes. Compared with path changes in Figure 3.6, path outages tend to occur much closer to the end hosts. More than 70% of the outages occur within 4 hops to the end hosts.

Table 3.8 gives the number of ASes that the outages span. Compared with path changes, we can confine a much higher percentage of outages (78%) within two ASes. If we examine the AS tiers where the affected routers are located, outages are spread out more evenly across tiers than path changes are. Paths in tier-1 ASes are the most stable and those in tier-3 ASes are most unstable. If we look at both Table 3.5 and Table 3.8, we note that paths in tier-3 ASes are most likely to be affected by all types of anomalies. They account for 40% of temporary loops, 46% of persistent loops, 47% of path changes and 26% of forward outages. In contrast, paths in tier-1 ASes are most stable.

Finally, in Table 3.8, we find that the scopes of 10% of the forward outages might have changed between the first and second set of forward probes, mostly due to local path changes. Another 8% of the forward outages have reference paths that do not terminate at the destinations. These last hops are considered firewalls based on the heuristic described in Section 3.5.

End-to-End Effect

We also study how path outages influence end-to-end performance. Not surprisingly, forward outages can be preceded by very high loss rates, which are slightly worse than those generated by path changes. The comparisons are shown in Figure 3.9. Similarly, outages tend to be preceded by much worse RTTs than path changes, as shown in Figure 3.10: 23% of the outages experience RTTs that are over one second, while only 7%

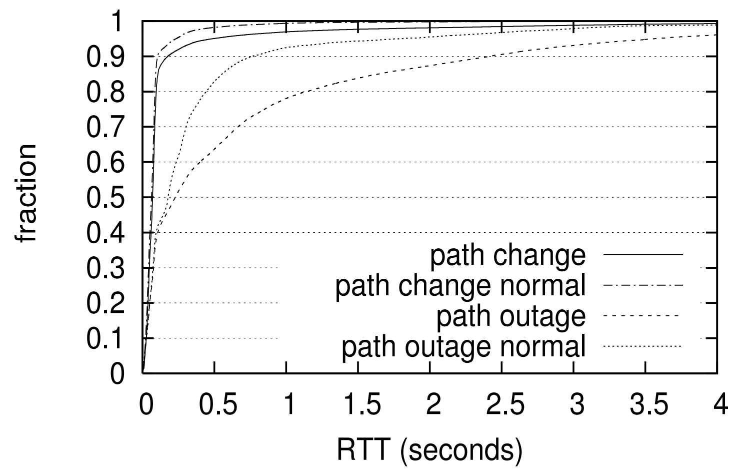

Figure 3.10: CDF of RTTs preceding path changes and forward outages *vs.* under normal conditions

do when there is no outage. The RTT variances can also be very high: 17% of them exceed 0.5 seconds.

3.7 Discussion

3.7.1 Bypassing Anomalies

In addition to characterizing anomalies, one of the goals of PlanetSeer is to provide benefits to the hosts running it. One possible approach is using the wide-area service nodes as an overlay, to bypass path failures. Existing systems, such as RON [5], bypass path failures by indirectly routing through intermediate nodes before reaching the destinations. Their published results show that around 50% of the failures on a 31-node testbed can be bypassed [25]. PlanetSeer differs in size and scope, since we are interested in serving thousands of clients that are not participants in the overlay, and we have a much higher AS-level coverage.

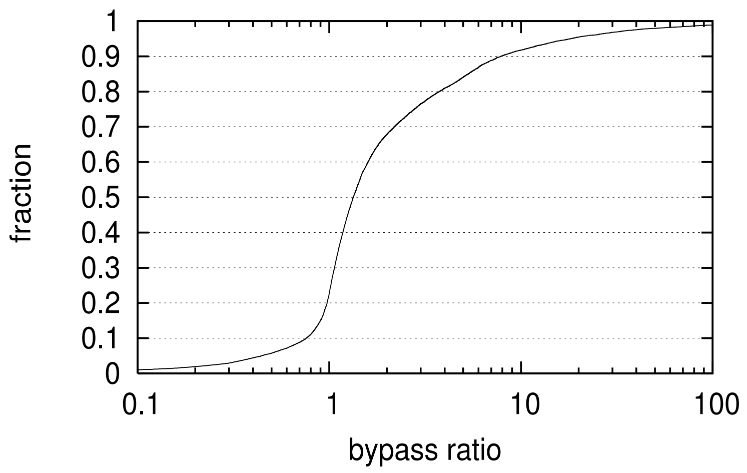

Figure 3.11: CDF of latency ratio of overlay paths to direct paths

Determining how many failures can be bypassed in our model is more complicated, since we have no control over the clients. Clients that are behind firewalls and filter pings and traceroutes may be reachable from other overlay nodes, but we may not be able to confirm this scenario. As a result, we focus only on those destinations that are reachable in the baseline probes, since we can confirm their reachability during normal operation.

For this group of clients, we have a total of **62815** reachability failures, due to anomalies like path outages or loops. Of these failures, we find that some nodes in PlanetSeer are able to reach the destinations in **27263** cases, indicating that one-hop indirection is effective in finding a bypass path for **43%** of the failures.

In addition to improving the reachability of clients using overlay paths, the other issue is their relative performance during failures. We calculate a *bypass ratio* as the ratio between the minimum RTT of any of the bypass paths and the RTT of the baseline path. These results are shown in Figure 3.11, and we see that the results are moderately promising – 68% of the bypass paths suffer less than a factor of two in increased latency. In fact, 23% of the new paths actually see a latency *improvement*, suggesting that the overlay could be used for improving route performance in addition to failure resiliency. However, some paths see much worse latency degradation, with the worst 5% seeing more than a factor of 18 worse latency. While these paths may bypass the anomaly, the performance degradation will be very noticeable, perhaps to the point of unusability.

3.7.2 Reducing Measurement Overhead

While PlanetSeer's combination of passive monitoring and distributed active probing is very effective at finding anomalies, particularly the short-lived ones, the probing traffic can be aggressive, and can come as a surprise to low-traffic sites that suddenly see a burst of traceroutes coming from around the world. Therefore, we are interested in reducing the measurement overhead while not losing the accuracy and flexibility of our approach. For example, we can use a single traceroute to confirm loops, and then decide if we want distributed traceroutes to test for the presence of correlated loops. Similarly, for path changes and outages, we can reduce the number of distributed traceroutes if we are willing to tolerate some inaccuracy in characterizing their scope. In Figure 3.12, we plot the CDF of the number of the probes from other vantages points we have to examine before we find the intercept traceroutes that can successfully narrow the scopes of the anomalies. Using only 15 vantage points, we achieve the same results as when using all 30 vantage points in 80% of the cases. We are interested in studying this issue further, so that we can determine which vantage points we need to achieve good results.

3.8 Summary

In this chapter, we have introduced what we believe to be an important new type of diagnostic tool, one that passively monitors network communication watching for anomalies,

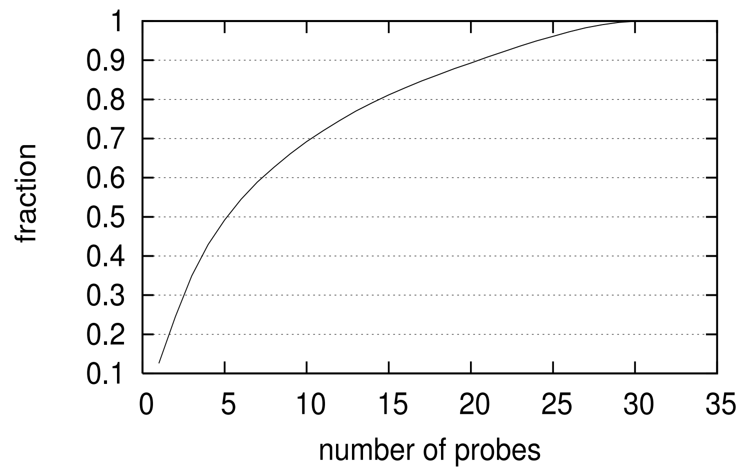

Figure 3.12: CDF of number of path examined before finding the intercept path

and then engages widely-distributed probing machinery when suspicious events occur. Although much work can still be done to improve the tool—e.g., reducing the active probes required, possibly by integrating static topology information and BGP updates— the observations we have been able to make in a short time are dramatic.

- Passive monitoring allows us to detect more anomalies in less time: we have confirmed nearly 272,000 anomalies in three months. This is roughly 3,000 a day, and is 10 to 100 times more than reported previously. We also see a qualitative change, such as large numbers of ultrashort and temporary anomalies that last less than one minute.

- Due to our wide coverage, we see new failure distribution and location properties. Failures are heavily skewed, rather than pervasively distributed: Tier 3 seems to be the most problematic, accounting for almost half of the loops, path changes, and path outages that we see. Tier 1 ASes are generally the most stable.

- We provide some new measurements about routing loop behavior. Temporary loops have much longer lengths than persistent loops. 97% of the persistent loops consist of only 2 routers, but only 50% of temporary loops do. Many temporary loops span 4 routers. This makes sense since the more routers are involved in a loop, the less stable it is. Persistent loops are either resolved in a relative short time (54% last less than 30 minute) or continue for an extended period of time (23% last more than 7.5 hours). Our results confirm Paxson's findings that routing loops are correlated.

- Path changes exhibit different characteristics than outages. Outages appear closer to the edge of the network: 63% of outages occur within 3 hops to end hosts while the figure is 20% for path changes. Path changes tend to have wider impact: 57% of path changes can be confined to two ASes and 50% of them can be confined to within three hops, while the respective figures are 78% and 70% for outages. Path changes have a much milder effect on RTTs than outages while they both can incur high loss rates.

- Our measurements suggest less opportunity for indirection-based resiliency than previous studies: alternative routes are available only 43% of the time, and a significant fraction of them suffer from high latency inflation. These results stem from most outages occurring nearer the edge of the network than the core; redundancy is less available, and less practical when it is available.

We have shown that PlanetSeer provides an effective means to detect large numbers of anomalies with broad coverage, especially in the case of wide-area services that cannot rely on cooperation from one endpoint. In addition to the detection rate, the short delay between emergence and detection allows us to capture anomaly behavior more effectively, and our distributed framework provides improved characterization.

Chapter 4

mTCP: Robust Transport Layer Protocol Using Redundant Paths

In the previous chapter, we have introduced PlanetSeer, a large-scale distributed system for routing anomaly detection and diagnosis. Although PlanetSeer can help us gain a better understanding of routing anomalies and improve the long-term stability of the Internet, it has little immediate impact on increasing the reliability of an individual end-to-end connection. In this chapter, we describe mTCP, a novel transport-layer protocol that can quickly react to performance degradations and minimize their negative effect on end users

4.1 Introduction

Recent work on Internet measurement and overlay networks has shown that redundant paths are common between pairs of hosts [78]. One can often achieve better end-to-end performance by adaptively choosing an alternate path other than the direct Internet path [5]. At the same time, stub networks are increasingly resorting to multihoming to improve the reliability of their network connectivity [2]. The reliability is usually achieved by having sufficiently disjoint paths to the destinations of interest. Moreover, with the rapid growth of wireless coverage, mobile users can often have access to multiple communication channels simultaneously [39, 59]. All of the above indicates redundant paths are quite common between pairs of hosts.

Our goal is to design a transport layer protocol, called mTCP, which can enhance the robustness of end-to-end communications during performance anomalies by taking advantage of the redundant paths. Compared with conventional single-path TCP flows, mTCP stripes a flow's packets across several paths. It can be viewed as a group of single-path subflows, with each subflow going through a separate path. mTCP can aggregate the bandwidth of several paths in parallel. As a result, even if one or more paths encounter performance problems, mTCP can still utilize the bandwidth on other good paths. A number of challenges arise when one attempts to develop such a transport layer protocol:

First, the traditional congestion control mechanism in TCP needs to be modified in order to fully exploit the aggregate bandwidth of the multiple paths. If one were to use the traditional congestion control mechanism on the whole flow (wherein the congestion window for the entire flow is halved for every packet loss), the mTCP connection would remain under-utilized in the following scenario. Assume that one of the sub-flows is using an anomalous path, i.e. a heavily congested path. Severe packet losses on that path will result in repeated shrinkage of the window, resulting in sub-optimal use of the other normal subflows. mTCP therefore needs to perform congestion control on each subflow independently so as to minimize the negative impact of anomalous subflows.

Second, paths may fail during data transmission. mTCP should not stall as long as there exists one operational path. It should be able to quickly detect failed paths and continue sending or retransmitting packets on other live paths.

Third, when subflows of a mTCP flow share congested links, the whole flow can obtain an unfairly larger share of bandwidth than other single-path TCP flows since each subflow behaves as a single-path TCP flow. To alleviate this aggressiveness problem, we integrate a shared congestion detection mechanism into our system so as to identify and suppress subflows that traverse the same set of congested links.

Finally, there might exist many alternate paths between a pair of source and destination nodes. We want to select a small number of candidate paths for mTCP flows since it is impractical to use all the paths simultaneously. We use a heuristic to identify and select disjoint paths using traceroute. This can minimize the possibility of shared congestion and concurrent path failures.

We note that mTCP provides performance benefits only when there are multiple paths that do not share congested physical links. Although in today's Internet, many congested or bottleneck links may lie at the edge of the network and limit the performance benefits of mTCP, this is likely to change with the growing popularity of high speed Internet access. Akella, Seshan and Shaikh [3] measured a diverse set of paths traversing ISPs in Tiers 1 to 4. They discovered that approximately 50% of the paths have bottleneck links that are located within ISPs or between neighboring ISPs. The available capacity of those bottleneck links are less than 50Mbps, well below the 100Mbps Ethernet speed that could be attained within local area networks. In such situations, mTCP is likely to find multiple paths that are not constrained by the bandwidths attainable at the edge of the network. Other scenarios where mTCP could find an useful set of disjoint paths to transmit data include multi-homed clients [2].

We also note that our focus is on improving the performance and robustness for large data transfers. Although most flows on the Internet are small, most of the traffic on the Internet is contributed by a small percentage of large flows [95, 28]. Therefore, improving the performance of such large flows is important. Additionally, while small flows might not necessarily benefit from throughput aggregation, they might still benefit from mTCP's robustness that would enable them to quickly detect and recover from path failures.

To the best of our knowledge, we are the first to implement and evaluate a transport layer protocol that can utilize redundant paths concurrently in real systems. We try to provide a comprehensive design that addresses the inter-related issues of sub-flow congestion control, unfair use of congested links, path selection, and recovery from path failures. We believe that it is beneficial to tackle all of these issues in a single tightly-coupled system. For instance, suboptimal decisions from the path selection mechanism could be corrected by a mechanism that detects the use of shared congested links. Alternately, shared congestion could be detected easily by monitoring TCP events (such as fast retransmits) without requiring separate probe messages. Furthermore, the system could quickly recover from performance anomalies by maintaining and transmitting along multiple paths. Finally, an mTCP flow can passively monitor the performance of several paths in parallel and estimate their available bandwidths. The bandwidth estimates are typically more accurate than the estimates provided by the underlying overlay routing mechanisms. This, in turn, can help select better paths.

The rest of this chapter is organized as follows: Section 4.2 will discuss the specific design problems of mTCP in detail. Section 4.3 briefly describes the implementation of our system. Section 4.4 demonstrates the results from experiments conducted on PlanetLab [73] and Emulab [24]. Finally, Section 4.5 concludes.

4.2 Design

The design of our system seeks to satisfy three goals. First, given several paths, mTCP should be able to make full use of the available bandwidth on those paths. Second, when mTCP uses paths with shared congested links, it should be able to alleviate the aggressiveness problem by suppressing some of the paths. Third, when some paths fail, mTCP should quickly detect and recover from the failures.

4.2.1 Transport Layer Protocol

mTCP provides the same semantics to applications as TCP. It preserves properties such as reliability and congestion control. Because mTCP uses several paths in parallel, it has to decide how to stripe packets across the paths and how to manage congestion control for each subflow.

Congestion Control

In mTCP, all subflows share the same send/receive buffer. Packets are assigned sequence numbers in the same way as in TCP. But it does congestion control independently on each subflow. Each subflow maintains a congestion window as in TCP. The congestion window changes independently as the subflow adapts to the network state. When there are no packet losses in the subflow, its congestion window linearly increases. Upon detecting packet losses, its congestion window is halved. When timeout occurs, the congestion window is reset to one and the subflow enters slow-start.

mTCP strives to keep all subflows independent from each other. Suppose we had used only one global congestion window for the entire flow that limits the total amount of outstanding data across all subflows. The packet losses on any one of the paths will cause the global congestion window to be halved. If one subflow happens to traverse a heavily congested path, it can keep the global congestion window small, and the other subflows will not be able to utilize the available bandwidth on other good paths. In fact, this can sometimes decrease the throughput of the whole flow to be even lower than that of a single-path TCP flow on a single good path. This phenomenon was also studied in [38].

Estimating Outstanding Packets

TCP uses *(sndnxt - snduna)* to estimate the number of outstanding packets in the network. (For convenience, we assume packets are of the same size and use packets instead of bytes for discussion.) Here *sndnxt* is the next packet to be sent and *snduna* is the next packet for which an ACK is expected. The difference should be no more than the congestion window (*cwnd*). In mTCP, since packets are striped across different paths, we need to keep track of how many outstanding packets are in each path to ensure that the number does not exceed the *cwnd* of that path.

Our mTCP is based on TCP SACK [63], which is an extension of TCP Reno. In Reno, the receiver only reports the greatest packet number that arrives in-order. But in mTCP, different paths have different latencies. Many packets can arrive at the receiver out of order. We want to accurately know which packets have been received, irrespective of whether or not they arrived in-order. This helps us to compute the number of outstanding packets on each path, which is essential for performing congestion control separately on each subflow. In SACK, the sender maintains a scoreboard data structure to keep track of which packets have or have not been received. An acknowledgment (ACK) packet may carry several SACK blocks, where each SACK block reports a non-contiguous set of packets that has been received. The first SACK block reports the most recently received packet and additional SACK blocks repeat the most recently reported SACK blocks. The SACK blocks allows the sender to identify what packets have been newly received irrespective of whether or not the data packets arrive in-order.

We augment the scoreboard data structure so that it records the path over which each packet is transmitted or retransmitted. For each $path_i$, we maintain a $pipe_i$ to represent the number of outstanding packets on $path_i$. $pipe_i$ is incremented by 1 when the sender either sends or retransmits a packet over $path_i$. It is decremented when an incoming ACK indicates that a packet previously sent on $path_i$ has been received. New packets are allowed to be sent over $path_i$ only when $pipe_i < cwnd_i$. Retransmitted packets require special handling. Suppose the original packet is sent over $path_i$ and the retransmitted packet is sent over $path_j$. When the retransmitted packet is ACKed, the sender decrements both $pipe_i$ and $pipe_j$ by 1, because it represents two packets having left the network: the original one on $path_i$, which is assumed to be lost, and the retransmitted one on $path_j$, which has been received. We want to emphasize that the original and retransmitted packets do not have to be sent over the same path. We will discuss this in more detail in Section 4.2.1. Finally, if $path_i$ times-out, $pipe_i$ will be reset to 0.

Fast Retransmit

Since mTCP sends packets along several paths with different latencies, packets can arrive at the receiver out-of-order. In traditional TCP, duplicate acknowledgments (*dupack*) are associated with packet losses, and *dupacks* typically trigger fast retransmits. In mTCP, since some of the *dupacks* are caused by packet reorderings, the system should be careful in using fast retransmits.

Although packets sent through different paths may be received out-of-order, packets within each subflow will still mostly arrive in-order. Each $path_i$ therefore maintains the following path-specific state: $snduna_i$, the next packet requiring an ACK over the given path, and $dupack_i$, the number of ***dupacks*** received for that path. If an incoming ACK indicates the receipt of a packet sent through $path_i$ and if that packet is $snduna_i$, this packet is considered to be in-order within that subflow. If that packet is greater than $snduna_i$, $dupack_i$ is incremented by 1. When $dupack_i$ reaches $dupthresh = 3$, $path_i$ will enter fast retransmit and fast recovery.

Sending Packets

mTCP separates the decisions of when to send a packet, which packet to send, and which path to use to send the packet. The sender is allowed to send a new packet when there exists at least one $path_i$ satisfying $pipe_i < cwnd_i$. The packet to send is usually determined by $sndnxt$, which represents the next packet to send as in TCP. But if there is a $path_i$ with packets to retransmit, *i.e.* $path_i$ is in fast recovery, the sender has to retransmit those packets inferred to be lost before sending any new data packets. Once again the scoreboard is consulted to determine whether there are any such packets that need to be retransmitted. Otherwise, a new data packet determined by $sndnxt$ will be sent.

Next, the sender needs to decide the path over which the packet will be sent. There may be several candidate paths. We associate a score of $pipe_i/cwnd_i$ with each $path_i$. We choose the path with the minimum score. This form of proportional scheduling results in a fair striping of packets and avoids sending a burst of packets on one path.

Because mTCP separates the decisions about when to send, which to send and which path to use for sending, it has more flexibility in striping packets. By postponing the decision about which path to use until just before sending out the packet, it can quickly adapt to dynamic variations in path characteristics. If a path encounters congestion or fails, its $cwnd$ will be reduced. mTCP does not have to wait for the re-opening of the

$cwnd$ on that path to retransmit the outstanding packets. It can instead retransmit the packets on other paths. We want to emphasize that, unlike the re-striping scheme used in pTCP [38], our scheme will not retransmit packets that have already been received, because we can precisely infer missing packets from the scoreboard data structure.

Single Reverse Path

In our design, despite the fact that data packets are striped over several paths, all ACKs return over the same path. There are two advantages of using a single path for conveying ACKs. First, it preserves the ACK ordering for all the subflows. If ACKs are conveyed through different paths, this may introduce ACK reorderings even within one subflow, which will increase the burstiness of the sender. Second, striping ACKs across different paths makes our system more complicated. The receiver has to maintain additional state about which ACKs are sent through which paths. We instead keep the receiver side as simple as possible, following the design principle of TCP.

The advantage of using multiple reverse paths is that it is more robust to congestion or failure. Note however that ACKs are typically small and are less likely to cause congestion than data packets. In addition to this, we try to avoid congestion by selecting the best path among all the candidate paths as the reverse path for sending ACKs. This will be discussed in Section 4.2.3. Recovering from failures of the single reverse path is a little more complicated. If we stripe ACKs across multiple paths, although the scheme is more robust to failures, we have to constantly suffer from ACK reorderings. Since failures generally will not occur very often, by using a single reverse path, we trade-off a performance degradation during failure recovery for better performance during normal operation. In Section 4.2.5, we show that the failure recovery time of mTCP is reasonably small even though it uses a single reverse path.

Comparison with Multiple TCP sockets

We could have avoided much of mTCP's complexity by opening separate TCP sockets for each path and then striping packets at the application layer [80]. But this approach has limitations that can prevent us from efficiently aggregating the bandwidth of multiple paths. To fully utilize the bandwidth of multiple paths, we must ensure that the amount of data striped over a path is proportional to the bandwidth of that path. This can be very difficult to achieve in Internet-like settings where the path conditions could vary. Modifying TCP directly gives us more flexibility on striping data streams over multiple paths. We can decide, for each packet, an appropriate path the packet should traverse, and this decision is made just before the packet is sent out. This is especially useful for migrating the outstanding packets on bad paths to good paths when the quality of paths changes dynamically or during path failures. Striping at the application layer cannot adapt to changes in path quality responsively. The pTCP study [38] has shown in simulations that such a scheme cannot fully utilize multiple paths when the number of paths exceeds two.

4.2.2 Shared Congestion Detection

When mTCP uses paths that are not completely disjoint and if some of the shared physical links are congested, the whole mTCP flow will obtain more bandwidth than other single-path TCP flows along those congested links, since each of the subflows behaves as a TCP flow. mTCP tries to alleviate the aggressiveness problem by detecting shared congestion among its subflows and suppressing some of them. Previous work [77, 36, 49, 92] on shared congestion detection is based on the observation that if two single-path flows share congestion, packets from two flows traversing a congested link at about the same time are likely to be either dropped or delayed. Rubenstein *et al.* [77] actively inject probing packets through the two paths to compute the correlation of packet losses or packet delays and thereby identify shared congestions.

We could certainly use one of the approaches mentioned above in our system since shared congestion detection is independent from other parts of the system. We however take a simpler approach based on the following observations. mTCP transmits a steady stream of packets through different paths. In this setting, there is no need to send probing packets. Instead, one can passively monitor the subflows by studying the behavior of the data packets.Furthermore, since individual packet drops will result in fast retransmits along the corresponding subflows, the sender can detect shared congestions by examining the correlations between the fast retransmit times of the subflows. Since data packets also double as probe packets and since there are a large number of data packets transmitted through a subflow, our passive monitoring strategy requires little overhead and generates a continuous stream of information resulting in fast detection of shared congestion.

Detecting Shared Congestion using Fast Retransmits

Let us focus on detecting shared congestion between a pair of subflows. For more than two subflows, we need to detect shared congestion between every pair of them. In the following discussion, we say that two subflows are *correlated* if the corresponding paths share congestion; otherwise, they are *independent*. We first assume that two paths have the same latency so that we do not have to worry about the time synchronization problem between them. Later, we will extend our algorithm so that it can deal with paths with different latencies.

Each time that a subflow enters fast retransmit, the sender records a timestamp in the subflow's list of fast retransmit events. After some time, we have two lists of timestamps,

S and T, from two flows: $(s_1, s_2, ..., s_m)$ and $(t_1, t_2, ..., t_n)$. Each timestamp represents a fast retransmit event. Then we try to match a timestamp s_i in S with t_j in T. If $|s_i - t_j| < interval$, we call (s_i, t_j) a match. Intuitively, a match means the two subflows enter fast retransmit around the same time. This also means packets from the two flows are dropped at about the same time, so it is likely they share the same congested link. We define $match(S, T)$ to be the maximum number of pairs (s_i, t_j), such that s_i matches t_j. Please note that each s_i cannot be matched with multiple t_j. Finally, two subflows are considered to be correlated if:

$$ratio = \frac{Match(S, T)}{min(m, n)} > \delta$$

$ratio$ is intended to identify what fraction of fast retransmits occur at about the same time in the two subflows. Since some of the fast retransmits are due to congestion on disjoint links, $ratio$ reflects the level of shared congestion. We consider two subflows to be correlated when $ratio$ is greater than some threshold δ.

Our method uses fast retransmits instead of individual packet losses to infer shared congestion. This is because when a data flow encounters congestion, there normally will be a burst of packet losses. All these losses are caused by one congestion period at some link. Therefore, a fast retransmit corresponds more directly to a congestion period, much more so than any individual packet loss. We would like to declare (s_i, t_j) to be a match only when packets from two subflows are dropped at one link during the same congestion period. So $interval$ cannot be too small, otherwise even if s_i and t_j occur in the same congestion period, the system will not detect the match. On the other hand, $interval$ cannot be too large, otherwise the system would consider (s_i, t_j) to be a match even when they are not due to shared congestion. Although the shared congestion detection may not work well under active queue management schemes, most routers on today's Internet use drop-tail queues, which lead to periods of bursty losses during congestion. In [96], the authors find that 95% of the duration of bursty losses are less than 220ms. So $interval$ should be on that time scale. We will study how to choose $interval$ and δ in more detail in Section 4.4.4.

Estimating Convergence Time

We need to emphasize that our goal is to suppress correlated subflows in order to alleviate the aggressiveness problem. We need to detect shared congestion as quickly as possible. Other efforts focus more on the accuracy of shared congestion detection, and they may take several hundred seconds to reach a decision. This does not work well for our purpose, because a mTCP flow could have ended before shared congestion is detected.

Our algorithm works as follows. After some number of fast retransmit events have been observed, we will check for shared congestion between the two subflows. If there is shared congestion, we can suppress one of them. Otherwise, we will wait until the occurrence of the next fast retransmit to check for shared congestion again. The question we now address is determining the number of fast retransmit events that we need to observe before we start checking for shared congestion.

We use a heuristic to estimate the probability of two fast retransmit events from two independent flows accidentally occurring within a small period of time. Suppose the fast retransmit events of two subflows, S and T, are completely independent when two subflows are independent, we compute the average interval of two consecutive fast retransmit events in S: $interval_s = \frac{now}{m}$, where now is the current time when shared congestion detection is invoked. $interval_t$ is computed in a similar way. Then we define $p = \frac{2 \times interval}{min(interval_s, interval_t)}$. Suppose $n \geq m$, we have $interval_t \leq interval_s$, and

$interval = \frac{p}{2} \times interval_t$. For each s_i, if there exists a match t_j, s_i must be in the $(t_j - interval, t_j + interval)$. Because we assume s_i and t_j are independent events, the probability that s_i matches some t_j is roughly p. So the total expected number of matches is roughly $E(Match(S,T)) = pm$. Because $min(m, n) = m$, we will misinterpret S and T to share congestion if $Match(S,T) > \delta m$. According to the Chernoff bound [17]:

$$\zeta = Prob[(Match(S,T) > \delta m] < e^{-mD(\delta||p)},$$

where $D(\delta||p) = \delta \ln \frac{\delta}{p} + (1-\delta) \ln \frac{1-\delta}{1-p}$. So we need to wait for $m = -\frac{\ln \zeta}{D(\delta||p)}$ fast retransmit events to ensure that the probability of a false positive is less than ζ. We will see in Section 4.4.4 the convergence time is mostly within 15 seconds in our Emulab and PlanetLab experiments. We want to emphasize that even if a false positive does occur, it will only degrade a mTCP flow into a single-path flow.

This heuristic might encounter problems when $min(interval_s, interval_t) \leq 2 \times interval$. Because $interval$ is small (200ms in our experiments), this can only occur when a path is so heavily congested that fast retransmit happens almost every 400ms. The mTCP flows will try to suppress such paths, because using them will not bring much benefit. This is discussed in Section 4.2.4.

Finally, when two paths have different latencies, there is a time-lag, L, between them. We estimate L by shifting one sequence, say T, by dt in time and calculating $Match_{dt}(S,T)$ on sequences (s_1, s_2, ..., s_m) and ($t_1 + dt$, $t_2 + dt$, ..., $t_n + dt$) as described before. Because the L between two paths can be at most one RTT (where RTT is the larger round trip time of the two paths), we go through all possible value dt in $(-RTT, RTT)$ incrementally using some fundamental step x, then choose dt that maximizes $Match_{dt}(S,T)$ as L. This is similar to calculating the correlation between two signals.

4.2.3 Path Selection

In the previous sections, we assumed that flows have a number of candidate paths. Now we describe how to obtain such information. We use Resilient Overlay Networks (RON) [5] as our underlying routing layer. RON is an application-layer overlay.When mTCP starts, it queries RON to obtain multiple paths between a source-destination pair. For each pair, RON provides the direct Internet path and alternate single-hop indirect paths through other RON nodes. With a RON of n nodes, there are $m = n - 1$ paths between each pair. RON uses a score to represent the quality of each path based on latency, loss rate or throughput. RON can effectively bypass performance failure or path faults by using an alternate path with higher score. In the following, we only use the throughput score.

Since m can be large (greater than 10 in our experiments), mTCP will only select at most k (5 in our experiments) paths from them. A single-path flow will normally select the path with the best score, which we call the *RON path*. mTCP could select the k best paths. But this simple strategy may select paths with many overlapping physical links. This leads to two disadvantages: First, paths are more likely to fail simultaneously, which is bad for the robustness. Second, paths are more likely to share congestion, which is bad for performance. To avoid these problems, we want to select sufficiently disjoint paths.

We use a heuristic based on traceroute to estimate the disjointness of paths. Using traceroute, we can obtain the IPs of the routers along a path and the latency of each physical link. Due to IP aliases, the same router might have different IPs in different paths. We use “Ally”, a tool from Rocketfuel [83], to resolve IP aliases and assign a unique IP to each router. Although some routers may not respond to traceroute probes and the alias resolution may not be completely accurate, we only use the traceroute information as a hint to estimate path disjointness and eliminate many of the significantly overlapping paths. We also rely on the techniques described in Section 4.2.2 to further detect shared congestion.

After alias resolution, suppose we have the IPs of two paths $X = (x_0, x_1, ..., x_m)$ and $Y = (y_0, y_1, ..., y_n)$. Let L be the set of overlapping links of X and Y, we define the overlapping between X and Y as: $Overlapping(X, Y) = \sum_{l \in L} latency(l)$. An alternative is to use the size of L to quantify the degree of overlap. We use latency instead because we hope to distinguish among different types of link. Most nodes on PlanetLab are connected through ethernet links to backbones. Those ethernet links usually have smaller latency than backbone links. Because the sharing of the local ethernet links are almost unavoidable, we focus on finding disjoint paths that traverse different backbone links. By using link latencies, $Overlapping(X, Y)$ will be mostly determined by the shared backbone links instead of ethernet links. This argument might not be true if nodes are connected through modem or wireless links that have high latency. Using traceroute to find disjoint paths is only suitable for small-scale overlay networks. As the number of nodes increases, we need a more scalable way to discover disjoint paths. In [64], Nakao, Peterson and Bavier propose to use BGP information to find disjoint Autonomous System (AS) paths, which incur little cost. Although disjoint AS paths are not as fine-grained as disjoint router-level paths, it would greatly simplify disjoint path search by providing a small set of promising candidate paths which we can further verify using traceroute.

Finally, we estimate the disjointness of X and Y by:

$$Disjoint(X, Y) = 1 - \frac{Overlapping(X, Y)}{Min(Latency(X), Latency(Y))}$$

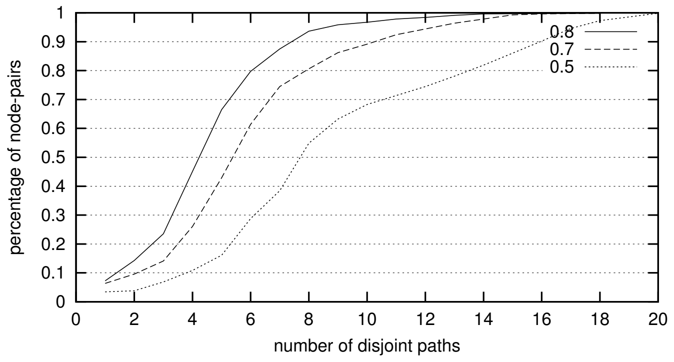

Figure 4.1: CDF of number of disjoint paths between node-pairs

We say that X and Y are disjoint if $Disjoint(X, Y) > \beta$. Using the disjointness metric between each pair of paths, we select at most k paths from m paths using a greedy algorithm as follows: (1) Initialize the set of selected paths to be empty. (2) Pick the path with the highest score from the set of m paths and check if it is disjoint from all the previously selected paths. (3) If so, select this path, otherwise pick the path with the next highest score and repeat step (2) until we find k paths or we have tried all m paths. The first selected forward path and the reverse path will always be the *RON path*, which is optimized for throughput in RON.

Figure 4.1 plots the cumulative distribution function (CDF) of the number of disjoint paths between 630 node pairs based on traceroute among 36 PlanetLab nodes that are used in our experiments. When β decreases, the number of disjoint paths between node-pairs increases. We want a β such that there are sufficient number of disjoint paths which we can choose from while eliminating most significantly overlapping paths. When $\beta = 0.5$ (the value used in our experiments), 90% of node-pairs have more than 4 disjoint paths but less than 16 disjoint paths. If we use a larger β, many node pairs will not have enough candidate disjoint paths.

4.2.4 Path Management

Path Suppression

In mTCP, a subflow f_i on $path_i$ may be suppressed because of one of three reasons: First, f_i shares congestion with another subflow f_j and its throughput $T(f_i)$ is lower than $T(f_j)$. This is because we do not want mTCP to be too aggressive to other single-path TCP flows. Second, suppose f_j has the highest throughput among all the subflows and if $T(f_i) < \frac{T(f_j)}{\omega}$ (for some constant $\omega > 1$), then f_i may be suppressed. This is because $path_i$ is too poor and using it does not bring much benefit. Third, $path_i$ fails.

We define a family of mTCP flows, called MP_d flows. An MP_d flow will try to use at least d ($d \geq 1$) paths, which means we will not suppress any path because of shared congestion when the number of paths being used is less than or equal to d. The value of d is a tradeoff between performance/robustness and friendliness. With a larger d, mTCP can obtain more bandwidth because it uses more paths. And it is more reliable because the probability that d paths fail simultaneously gets smaller as d increases. But it can be more aggressive to single-path flows under shared congestion. The aggressiveness problem can be alleviated by suppressing some subflows. But when there are only d subflows, no subflow would be suppressed. The actual value of d should be decided by the application. Applications that want higher performance and more reliability should choose a larger d. Applications that care more about friendliness should choose a smaller d. In our experiments, we choose $d = 1$ to demonstrate how much performance improvement mTCP can obtain without being too aggressive to other TCP flows.

Path Addition

An MP_d flow can dynamically add new paths because of two reasons: First, some paths that are not being used become better than those paths that are being used. Second, it is using less than d paths because some paths were suppressed. mTCP will periodically update the information about all the paths by querying RON. If an unused path has a much higher score than a path being used, it can start using the new path. Then it runs the path suppression algorithm on all the paths to suppress any paths with shared congestion. By doing this, mTCP can gradually replace bad paths with good ones. This is especially useful for long-lived flows.

4.2.5 Path Failure Detection and Recovery

Failure Detection

mTCP may encounter path failures during transmission. If all the paths fail simultaneously, we call it a *fatal path failure*, otherwise we call it a *partial path failure*. We will focus on partial failures in this section. To recover from fatal failures, mTCP relies on the routing layer to establish new paths just like single-path flows.

When a path fails, the data packets sent over it will no longer be acknowledged (ACKed) because the packets have been dropped. We maintain one failure detection timer, $timer_i$, for each $path_i$. When a data packet sent over $path_i$ is ACKed, $timer_i$ is reset. $path_i$ is considered to have failed when $timer_i$ expires.

We need to decide a timeout value I_i for $timer_i$. On one hand, we want a small I_i so that failures can be detected quickly. On the other hand, I_i cannot be too small, otherwise it may misinterpret a good path to have failed. The retransmission timeout (RTO_i) provides a good base for computing I_i. First, during timeout, the sender will go into idling and no packets will be ACKed in that period. So I_i should be at least greater than RTO_i. Second, several consecutive timeouts means either the path has failed or it is heavily congested. In either case, we would like to abandon $path_i$. So we choose $I_i = \chi RTO_i$. Here χ reflects how many consecutive retransmission timeouts mTCP is willing to tolerate before it consider a path to have failed. In our experiments, we choose $\chi = 2$, because we have observed that consecutive retransmission timeouts rarely occur on good paths. We should emphasize that even if a good path is misinterpreted as a failed one, it will only degrade the performance of mTCP to that of a single-path flow in the worst case. The path addition technique described in Section 4.2.4 allows us to reclaim a path if it had been previously misinterpreted to be a failed path.

Failure Recovery

We now describe how to recover from failure after $timer_i$ expires. Since all ACKs return over the same path, we call that path a *primary* path. The other paths are *auxiliary* paths. We need to distinguish between primary and auxiliary path failures. When an auxiliary $path_i$ fails, the sender will mark $path_i$ as failed and retransmit the outstanding packets of $path_i$ over other paths. When a primary path fails, the situation is more complicated. Because all the ACKs are lost, it may appear to the sender that all paths have failed. To deal with this problem, sender records the time π_i when $path_i$ is detected to have failed. Suppose at time now, the primary $path_p$ is also detected to have failed and let the timeout of $timer_p$ be I_p. We know that $path_p$ must have failed at some point between $now - I_p$ and now. For an auxiliary $path_i$, if $now - I_p \leq \pi_i$, its failure is possibly due to the failure of $path_p$. In this case, we will change the status of $path_i$ to be active and the status of $path_p$ to be failed. After doing this for all the paths, the sender starts to send packets over all active paths. During this period, these data packets serve as "probing" packets that solicit ACKs from the receiver. All timers are stopped to prevent any auxiliary path from being misinterpreted as failed due to the lack of an active primary path during this period. The receiver will also detect the primary path failure because it no longer receives any data packets over that path. Then it elects a new primary path and sends ACKs along that path in response to those “probing” packets from sender. It chooses the best path (based on the path score in RON) among all the active paths to be the new primary path. Later, when the sender receives the ACKs and knows that a new primary path has been elected, it restarts all the timers and proceeds as normal.

Typically, RTO_i is one second, therefore I_i is two seconds. The total detection and recovery time will be between two and three seconds in most cases. The interruption due to partial path failures will be fairly short. Furthermore, partial path failure does not cause mTCP to stall since packets will continue to be transmitted through active paths. Since mTCP uses several paths concurrently and since it typically employs disjoint paths, the probability of fatal path failures is much lower than that of single-path failure. mTCP is therefore more robust than single-path flows.

4.3 Implementation

Our system is implemented at the user-level and is composed of a Portable User-Level TCP/IP stack (PULTI) and an overlay router/forwarder modified from RON. RON is an application-layer overlay on top of the Internet. PULTI and RON run in two separate processes. We modified RON so that it can communicate with PULTI using UDP sockets and export the multiple paths between a source-destination pair. The whole system does not require any root privilege, and hence it can be easily deployed on shared distributed platforms such as PlanetLab. Currently, it runs on Linux, NetBSD and FreeBSD. Note that although our system in build on top of overlay networks, it will work in a non-overlay setting as well. The only requirement of our system is that there exists an underlying routing system that can provide multiple paths between a source-destination pair and let us select paths for sending packets. RON is just one of such systems. Our system will also work in other Internet settings, such as multi-homed hosts.

PULTI is a full user-level TCP/IP stack based on FreeBSD 4.6.2. We extract the network-related code from the kernel source and wrap it with some basic kernel environment support, such as timing, synchronization and memory allocation. We do not modify any network-related code. Because FreeBSD 4.6 does not support SACK, we also add SACK-related code in PULTI which is required by our system. OS dependent information is hidden by device drivers. With different device drivers, PULTI can send or receive through a UDP socket, IP_QUEUE in Linux or a divert socket in FreeBSD. PULTI provides standard socket interface and supports multiple applications through multithreading. It can query RON to learn about multiple paths between a source-destination pair. The mTCP code only affects a few files in PULTI. It can be easily moved into FreeBSD kernel.

4.4 Evaluation

4.4.1 Methodology

In this section, we validate our protocol in both emulation and real-world deployment. The emulations are run on Emulab [24], which is a time- and space-shared network emulator. Emulab consists of several hundred PCs, which can be configured to emulate different network scenarios. Users can specify parameters such as packet loss rate, latency, and bandwidth. While an experiment is running, the experiment gets exclusive use

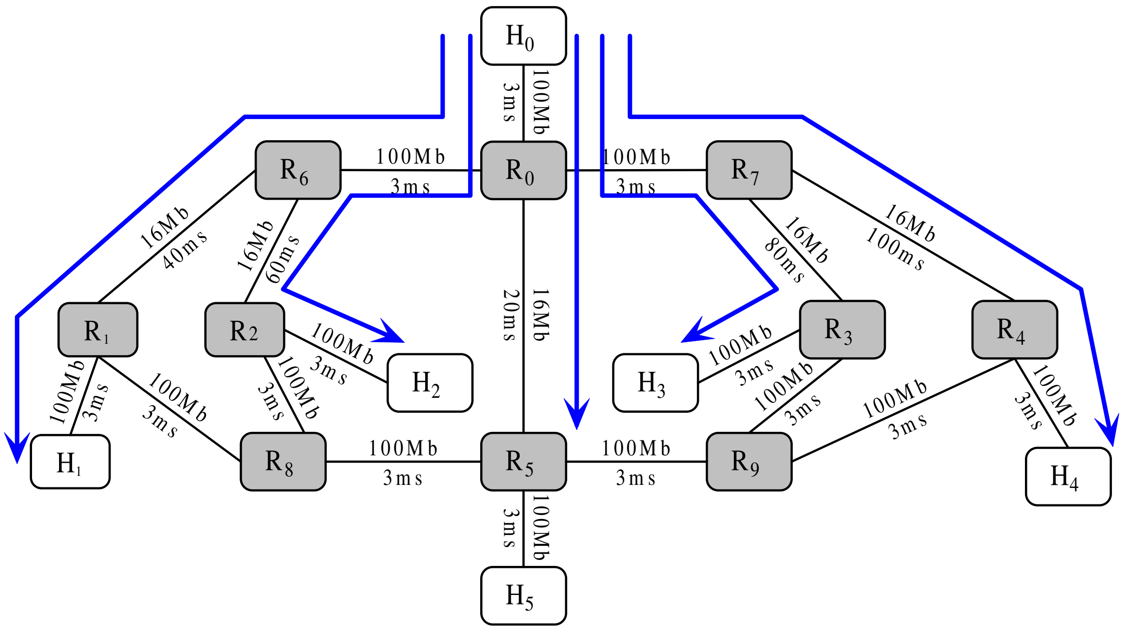

Figure 4.2: Topology of multiple independent paths on Emulab

of the assigned machines. While Emulab provides a controlled environment for our experiments, we further conduct experiments on PlanetLab, a wide-area distributed testbed for running large-scale network services [73]. The experiments on the PlanetLab allow us to study our protocol for Internet settings, where latency, bandwidth and background traffic are more realistic and unpredictable.

4.4.2 Utilizing Multiple Independent Paths

In this experiment, we study whether mTCP can obtain the total available bandwidth over multiple independent paths. We use the topology in Figure 4.2 on Emulab. Because each PC in Emulab has four Ethernet cards, each node can have at most four links. There are six endhosts (H_i) and ten routers (R_j). RON is running on the six endhosts to construct an overlay network. All routers have drop-tail queues. The source and

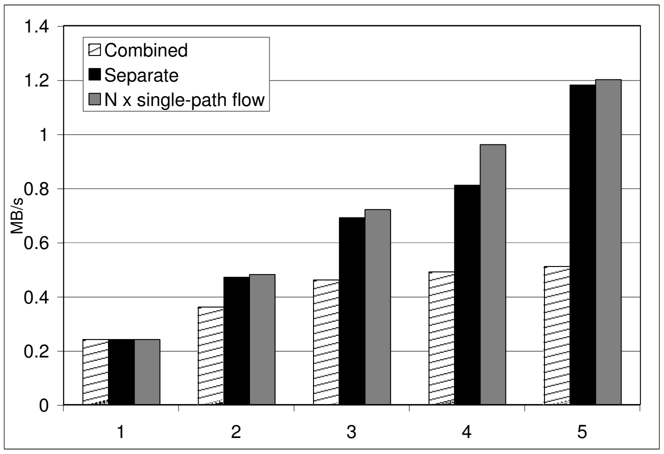

Figure 4.3: Throughput of mTCP flows with combined or separate congestion control as number of paths increases from 1 to 5

destination nodes are H_0 and H_5 respectively. Each of the remaining endhosts provides an alternate path. For example, we can use H_1 to construct an alternate path $(H_0, R_0, R_6, R_1, H_1, R_1, R_8, R_5, H_5)$. So the topology contains five independent paths, which include one direct path and four alternate paths. We use the direct path as the reverse path for ACKs. The capacity of all the paths is 16Mbps and their RTTs vary from 52-147ms. The figure annotates each link with its corresponding bandwidth and latency. The arrows represent background flows. We use Iperf [43] to generate 25 TCP and 25 1Mbps UDP flows as background traffic, with 5 TCP and 5 UDP flows on each path. Each experiment runs for 40 seconds and the results are obtained by averaging three runs.

Figure 4.3 shows the results when the number of paths used by mTCP increases from 1 to 5. In this figure, "combined" represents mTCP flows with congestion control performed on the entire flow, "separate" represents regular mTCP flows with congestion

Path	Intermediate node	RTT(ms)
0	direct path	80.165
1	planetlab1.nbgisp.com	112.503
2	planet2.berkeley.intel-research.net	71.639
3	planet2.pittsburgh.intel-research.net	96.641
4	planet2.seattle.intel-research.net	90.305

Table 4.1: Independent paths between Princeton and Berkeley nodes on PlanetLab.

control performed separately on each subflow, and “NxSingle-path flow” is the throughput of a single-path flow on one path multiplied by the number of paths. Because each path has the same available bandwidth, “NxSingle-path” throughput represents the ideal throughput of a mTCP flow. The results verify that mTCP can effectively aggregate the available bandwidth on multiple independent paths. The results also show that higher throughput can be achieved only when congestion control is performed for each subflow separately.

We conduct similar experiments on PlanetLab. We use one node in Princeton and one node in Berkeley as source and destination nodes. As shown in Table 4.1, the four Intel nodes serve as intermediate nodes for the alternate paths. We only use the four alternate paths in this experiment, because they do not share any congestion links. To verify this, we examined the traceroute data to find that any pair of the alternate paths only share the initial and final hops, which are unavoidable. The capacity of these two links are 100Mbps, which is far greater than the total throughput of the single-path TCP flows on these four paths. Therefore, we conclude that the initial and final hops are not congested and the four alternate paths are independent.

Each experiment measures the throughput of flows lasting for 60 seconds. The average throughput of three runs is reported. For convenience, we use $T(i)$ to denote the throughput of a single-path flow on $path_i$. Similarly, $T(i, j)$ denotes the through-

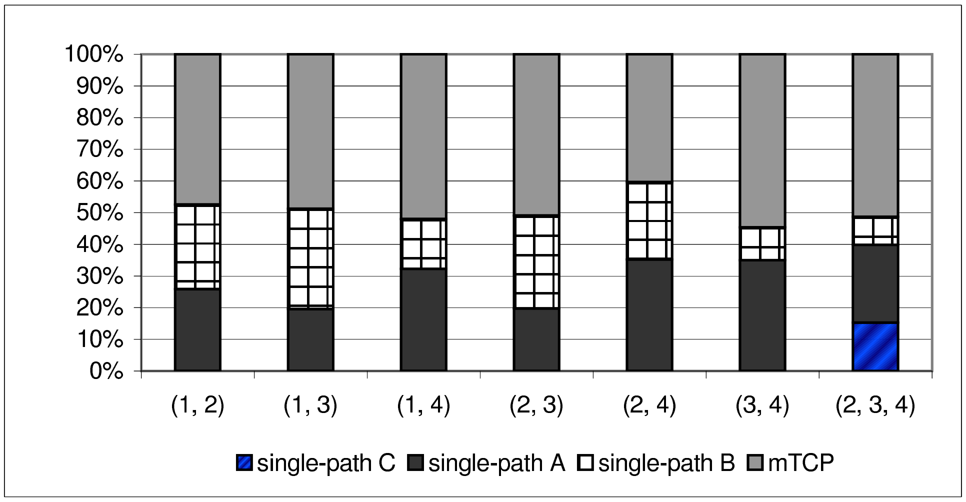

Figure 4.4: Throughput percentage of individual flows

put of a mTCP flow using $path_i$ and $path_j$. In Figure 4.4, (i,j) on the x-axis means $path_i$ and $path_j$ are used in that experiment. We first run single-path flows on $path_i$ and $path_j$ respectively, then run a mTCP flow on both paths simultaneously. The corresponding column compares the percentage that the throughput of an individual flow, $T(i)$, $T(j)$ or $T(i,j)$, contributes to the total throughput of these flows. Ideally, we expect $T(i,j) = T(i) + T(j)$, so the percentage of $T(i,j)$ should be around 50%. With the exception of the experiment involving $path_2$ and $path_4$, which suffered from unexpected bandwidth variations, the rest of the experiments indeed provide the expected throughputs. The last column in Figure 4.4 shows the result of the experiment using $path_2$, $path_3$, and $path_4$. Again, the net throughput of $T(2,3,4)$ is close to the sum of $T(2)$, $T(3)$ and $T(4)$. We have conducted experiments between different source-destination pairs on PlanetLab. The results are similar. We omit them due to space constraints.

Path	Intermediate node	RTT(ms)
0	direct path	80.165
1	planetlab02.cs.washington.edu	102.890

Table 4.2: Paths used in the failure recovery experiment.

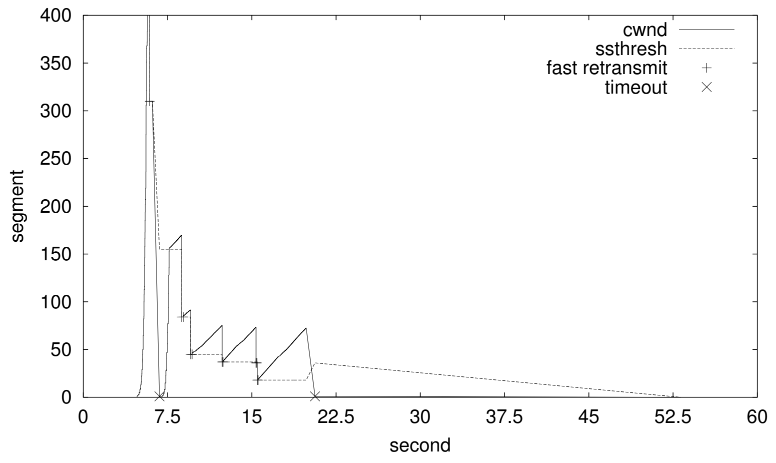

Figure 4.5: $cwnd$ of primary path, primary path fails

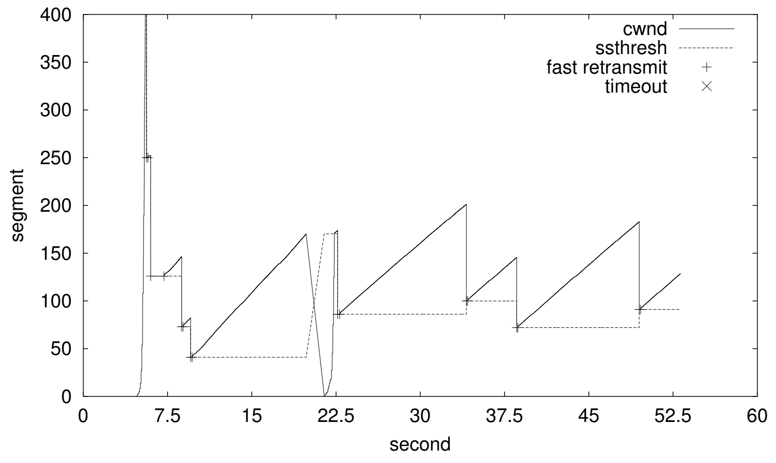

Figure 4.6: $cwnd$ of auxiliary path, primary path fails

4.4.3 Recovering from Partial Path Failures

Now we will study whether mTCP can quickly recover from partial path failures using experiments on PlanetLab. Because path failures on the Internet are unpredictable, we intentionally introduce failures by killing the appropriate RON agent. The source and destination nodes are still the Princeton and Berkeley nodes. The paths are shown in Table 4.2.

The two graphs in Figure 4.6 show how the congestion window ($cwnd$) of the primary and auxiliary paths changes over time. As shown in the first graph in Figure 4.6, the primary path fails at about 20s. It is quickly detected so that the $cwnd$ of the subflow on this path is reduced to 0. At the same time, the $cwnd$ of the subflow on the auxiliary path also decreases to 0, because the auxiliary path was misinterpreted to have failed (as explained in Section 4.2.5). But a few seconds later, the subflow on the auxiliary path recovers from this false decision by restoring its $cwnd$ to the previous value with slow start. Finally the auxiliary path becomes the new primary path and the whole flow proceeds using only one path. The behavior of mTCP during auxiliary path failures is similar, and we omit the corresponding results.

The total recovery time of mTCP during partial path failures is only about 3s, which is negligible for most applications. In contrast, a TCP flow will completely stall when its path fails, and it typically takes about 18s for RON to establish a new path. RON is optimized for quickly recovering from path failures. On wide area network that uses BGP to detect failures, recovery could take several minutes. Hence, mTCP is more responsive and robust than single-path flows.

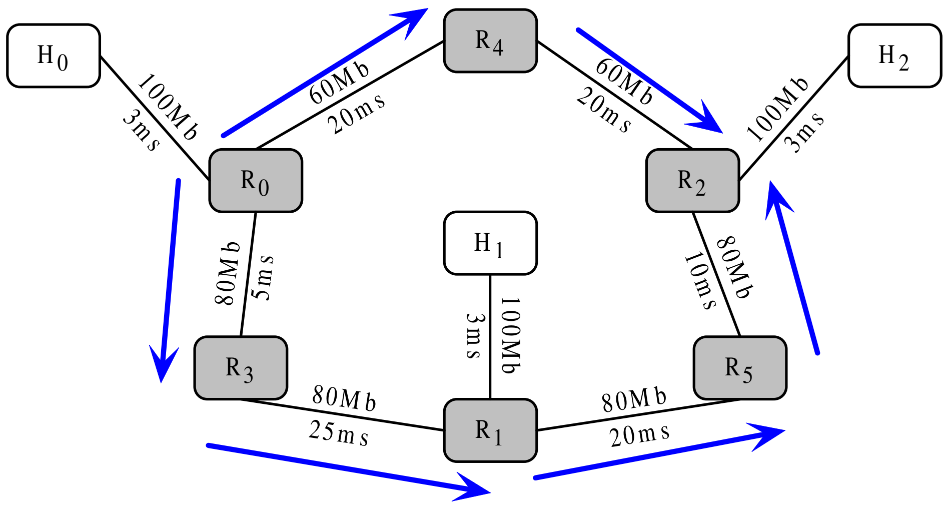

Figure 4.7: Two independent paths used in shared congestion detection

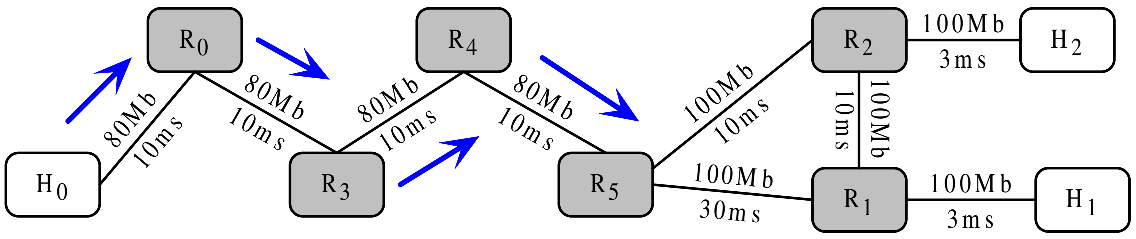

Figure 4.8: Two paths that completely share congestion

4.4.4 Detecting Shared Congestion

In this section, we will evaluate shared congestion detection. We first use experiments on Emulab to study the behavior of our algorithm with different parameters in a controlled environment. Then we further validate it using experiments on PlanetLab. The topologies for the Emulab experiments are shown in Figures 4.7 and 4.8. Between the source node H_0 and the destination node H_2, there is one direct path and one alternate path through the intermediate node H_1.

In Figure 4.7, The two paths only share the initial and final hops with link capacities of 100Mbps. We generate 12 TCP flows and 18 1Mbps UDP flows as background traffic, with 2 TCP flows and 3 UDP flows on each link between each pair of neighboring routers. With this scheme, we ensure that congestion only occurs on the links between pairs of routers and not on the links between endhosts and routers. As a result, the two paths (H_0, H_2) and (H_0, H_1, H_2) are independent.

In Figure 4.8, The two paths share the four links between H_0 and R_5. We generate 8 TCP flows and 8 1Mbps UDP flows as background traffic, with 2 TCP and 2 UDP flows on each of the four shared links. By doing this, we ensure that congestion only occurs on the four shared links. As a result, paths (H_0, H_2) and (H_0, H_1, H_2) share congested links.

We run mTCP flows for 300s using the two paths in Figure 4.8. The results in Figure 4.9 compare the estimated $ratio$ of shared congestion with different $interval$ values of 5ms, 10ms, 25ms, 50ms, 100ms, 200ms, and 400ms. Each data point represents the average of five runs. As $interval$ increases from 5 to 100ms, $ratio$ increases quickly from 0.4 to 0.8 as expected. When $interval$ increases beyond 100ms, $ratio$ only increases slightly. When $interval$ is 400ms, $ratio$ reaches 0.96. The ideal $ratio$ is 1 because the two paths share all the congestion.

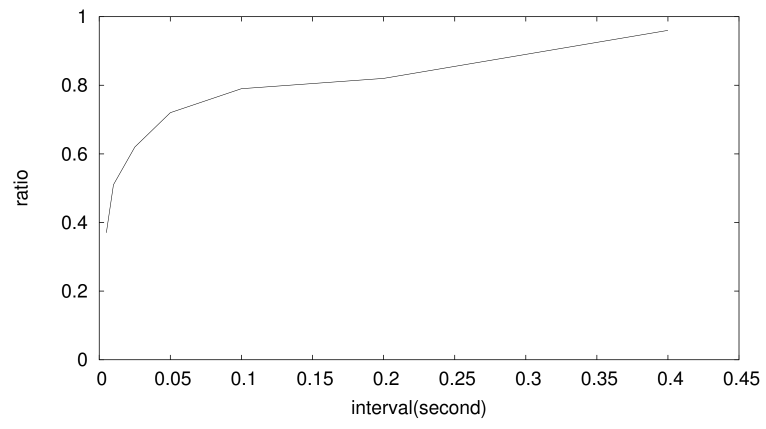

Figure 4.9: On two paths with shared congestion, $ratio$ increases as $interval$ increases

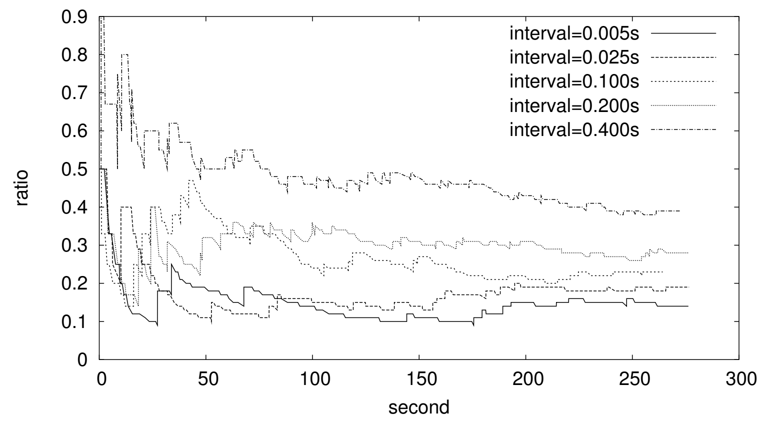

Figure 4.10: On two independent paths, $ratio$ decreases faster when $interval$ is smaller

We next run mTCP flows for 300s on the two paths in Figure 4.7. The results are shown in Figure 4.10, which plots $ratio$ over time for different $interval$ values of 5ms, 25ms, 100ms, 200ms and 400ms. As explained in Section 4.2.2, a smaller $interval$ will lead to smaller estimated values of $ratio$. At the end of the experiments, $ratio$ drops quickly from 0.39 to 0.28 when $interval$ decreases from 400 to 200ms. When $interval$ deceases further, $ratio$ drops more slowly until it reaches 0.14 when $interval = 5ms$. The ideal $ratio$ is 0 because the 2 paths are independent. We also notice that the $ratio$ curve for a smaller $interval$ value decreases faster than that for a larger $interval$.

According to the above experiment results, an $interval$ value between 100 and 200ms seems to balance the goal of minimizing both false negatives and false positives. Consequently, the $ratio$ threshold δ should fall between 0.3 and 0.8. If it is less than 0.3, it is very likely to cause false positives when the $interval$ is 200ms. If it is greater than 0.8, it can easily cause false negatives when the $interval$ is 100ms. By setting $interval = 200ms$ and $\delta = 0.5$, we successfully detect shared congestion between the two paths in all five runs for the topology in Figure 4.8. For the topology in Figure 4.7, no shared congestion is detected and the two paths are determined to be independent as expected.

Next, we go on to evaluate the shared congestion detection on PlanetLab. As explained in Section 4.2.2, by setting $interval$ to be no less than the congestion period during which bursty losses occur, we can avoid false negatives. In [96], the authors find that 95% of the duration of bursty losses on the Internet are very short-lived (less than 220ms). By choosing an $interval$ around that value, we should be able to avoid most false negatives. At the same time, the average time between consecutive fast retransmits is mostly on the order of several seconds or more, much greater than 220ms. (Otherwise, mTCP will suppress such path because the path is too lossy.) Therefore, this $interval$

Path	Run 1	Run 2	Run 3
1 2	No	No	No
1 3	No	No	No
1 4	No	No	No
2 3	No	No	No
2 4	N/A	Yes	No
3 4	No	No	No

Table 4.3: Shared congestion detection for independent paths.

value will also allow us to avoid most false positives, as long as we wait for enough number of fast retransmits. In the following experiments, we report the results using $interval = 200ms$ and $\delta = 0.5$.

We first need to choose paths such that we can be reasonably sure as to whether they share congestion or not. Then, we can compare the measured results with the expected results. We conduct two sets of experiments. The mTCP flow is running on a pair of paths for 60 seconds in each experiment. As explained in Section 4.2.2, the probability of false positive decreases very fast as the number of fast retransmit increases. We find that a 60 second period is long enough for our algorithm to converge. Each experiment is repeated three times. We use the Princeton and Berkeley nodes as source and destination in all experiments, but we choose different pairs of paths in different sets of experiments.

In the first set of experiments, we use the four alternate paths in Table 4.1, where we know that all these paths are independent. The results are in Table 4.3. The first column shows the pairs of paths used by the mTCP flows. The remaining three columns show the results. A *No* means two paths are independent, a *Yes* means they share congestion, and *N/A* means one of the subflows is suppressed because its throughput is much lower than the other subflow before the end of the experiment. All the results in Table 4.1 conform to our expectation except the one false positive for using $path_2$ and $path_4$. As explained

Path	Intermediate node	RTT(ms)
0	direct path	80.165
1	planetlab2.cs.duke.edu	96.138
2	planetlab2.cs.cornell.edu	100.382
3	vn2.cs.wustl.edu	92.267

Table 4.4: Paths with shared congestion on PlanetLab.

Path	Run 1	Run 2	Run 3	Average
0 1	7.000	9.975	4.266	7.080
0 2	4.276	3.223	6.011	4.503
0 3	6.847	3.263	14.214	8.108
1 2	12.184	8.906	16.804	12.631
1 3	4.478	10.101	13.131	9.237
2 3	12.380	9.873	17.845	13.366

Table 4.5: Shared congestion detection for correlated flows.

before, a false positive will only degrade the performance of the mTCP flow to that of a single-path flow.

The second set of experiments use the paths in Table 4.4. From traceroute, we know the underlying physical links of any pair of these paths are mostly overlapping, so they should share congested links. The results are shown in Table 4.5. The first column gives the pairs of paths used in the experiments. The following three columns give the time in seconds when shared congestion is detected in each run. The last column gives the average detection time. Shared congestion is correctly detected in all cases.

Unlike other shared congestion detection algorithms, our algorithm seeks to minimize the detection time while maintaining a low false positive rate. In the second set of experiments, shared congestion is correctly detected mostly within 15 seconds. At the same time, such early decisions do not cause too many false positives in the first set of experiments.

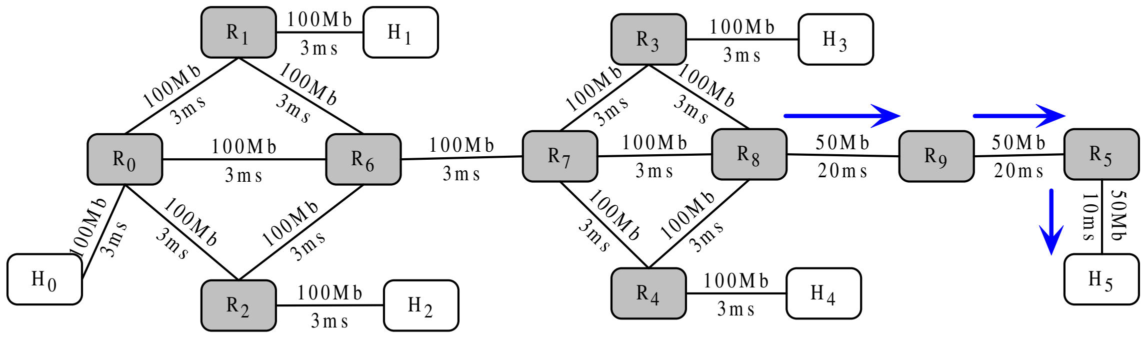

Figure 4.11: All paths share congestion in this topology

4.4.5 Alleviating Aggressiveness with Path Suppression

In this section, we demonstrate mTCP can be more friendly to other single-path flows by suppressing its subflows that share congestion. We construct the topology of Figure 4.11 on Emulab. The source and destination nodes are H_0 and H_5. There are one direct path and four alternate paths provided by the remaining four endhosts. Their RTTs are from 124ms to 133ms and they share the three links between R_8 and H_5. We generate 12 1Mbps UDP flows as background traffic, with 4 UDP flows on each of the three shared links. By doing this, we ensure that all five paths share congestion. Each experiment runs for 300 seconds and the results are obtained by averaging three runs.

In Figure 4.12, the first five columns give the throughput of the flows when the number of paths being used increases from one to five. The first column is the throughput of single-path TCP flows. Under shared congestion, the mTCP flows become more aggressive as they use more paths. The sixth column shows the throughput of the mTCP flow with path suppression. Although it uses five paths in the beginning, it quickly detects shared congestion and suppresses all but one path. So its throughput is very close to that of a single-path flow and less aggressive than the flow using all five paths without suppression.

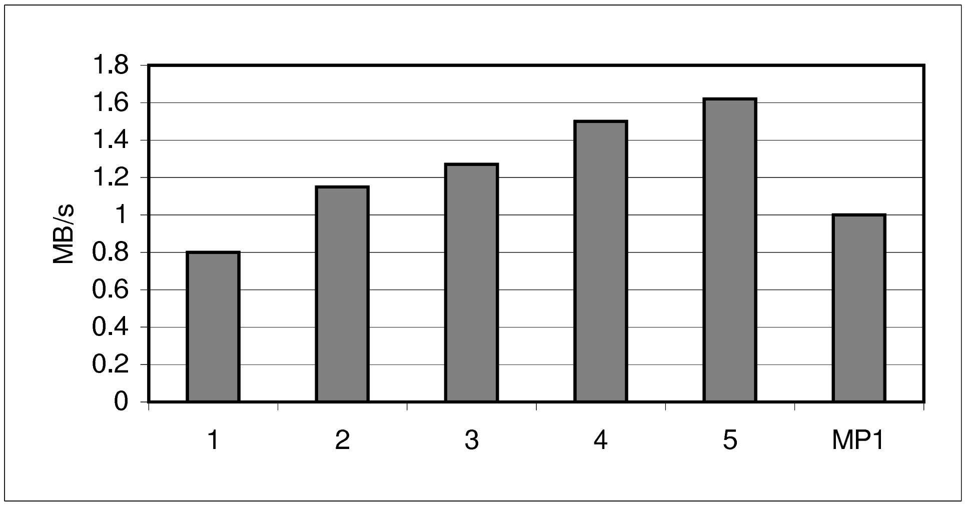

Figure 4.12: MP_1 flows are less aggressive than other mTCP flows

4.4.6 Suppressing Bad Paths

In this experiment, we demonstrate that mTCP can effectively aggregate the bandwidth of multiple paths with sufficiently differing characteristic, and path suppression can help avoid the penalty from using bad paths. We use the same topology as in Figure 4.2. The bandwidth of direct path is still 16Mbps. But the bandwidth of four alternate paths is 1/2, 1/4, 1/8 and 1/1000 of the bandwidth of the direct path. In Figure 4.13, $(1, 1/n)$ on the x-axis means the direct path and alternate path with 1/n bandwidth are used in that experiment. We first run a single-path flow on each path respectively, then run a mTCP flow on both paths. The corresponding column compares the percentage that the throughput of an individual flow contributes to the total throughput of these flows. Ideally, the throughput percentage of mTCP flows should be 50%. Figure 4.13 shows mTCP can efficiently utilize the aggregate bandwidth of two paths even when one path has only 1/8 the bandwidth of the other path. The mTCP flows in the last 2 columns use the direct path and the alternate path with 1/1000 bandwidth. Such a scenario could occur when

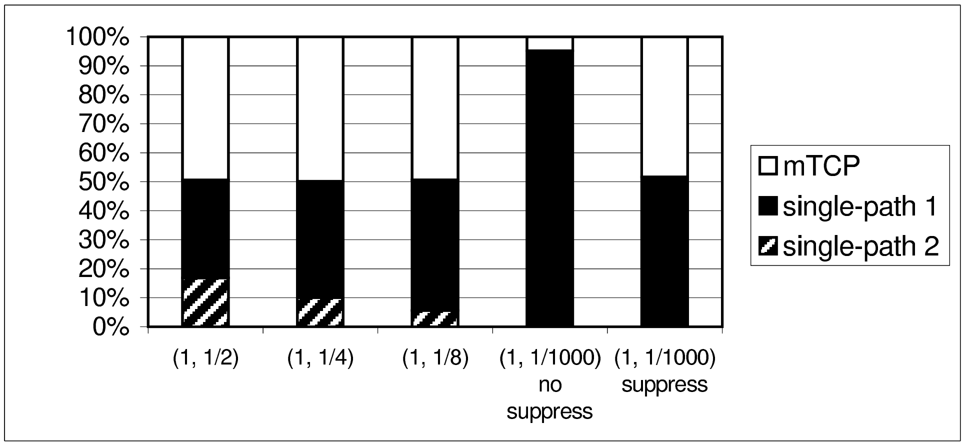

Figure 4.13: Path suppression helps avoid using bad paths.

a path becomes heavily congested or even temporarily fails. Using such bad paths can bring no benefit but impair the performance of the whole flow. Because most packets are lost along that path, it persistently causes timeouts. While packets can still be sent over the other path for some time, the flow will finally stall when the send/receive buffer is exhausted. As explained in Section 4.2.4, mTCP will suppress the paths with too low a throughput to avoid such penalty. (We choose $\omega = 10$ in our experiments.) This is confirmed by the last two columns which represent the throughput of mTCP flows with and without suppression.

4.4.7 Comparing with Single-Path Flows

We are going to compare three types of flows: single-path flows using direct Internet path (INET), single-path flows using *RON path* optimized for bandwidth (RON) and MP_1 flows. MP_1 flows will use multiple paths when there is no shared congestion. We use MP_1 flows to demonstrate how much performance improvement mTCP can obtain without being too aggressive to other TCP flows. Table 4.6 shows the 10 nodes that serve as

Host Name	Host Name
planetlab2.millennium.berkeley.edu	planetlab2.postel.org
planetlab02.cs.washington.edu	planetlab2.lcs.mit.edu
planetlab-2.cs.princeton.edu	planetlab2.cs.ucla.edu
planetlab2.cs.uchicago.edu	planet.cc.gt.atl.ga.us
planetlab2.cs.duke.edu	pl2.cs.utk.edu

Table 4.6: The 10 endhosts used in the experiments that compare mTCP with single-path flows.

endhosts in an overlay network for this experiment. (We actually use a total of 24 nodes to form the overlay network, with the remaining 14 nodes only serving the role of packet forwarders.) For each source-destination pair, we transfer data for 40 seconds using each of the three types of flows. Each experiment is repeated three times and we report the average throughput.

The available bandwidth of the paths between the pairs of endhosts can be very high, because nine of them are connected to Internet2. We bypass those pairs with very high available bandwidth on the corresponding direct paths because: First, these paths are between pairs of nodes that exhibit shared congestion/bottleneck at the initial and/or final hops. Second, the bandwidth-delay products of the paths between such pairs of nodes are very large. The maximum send/receive buffer size of our user-level TCP implementation is 1MB and is not large enough to utilize the bandwidth on other alternate paths besides the direct path. We estimate the available bandwidth of a direct path between a source-destination pair by running a TCP flow for 10 seconds. If the measured throughput is less than 12 Mbps, we will use that pair for our experiments. Among the 90 pairs, we got 15 pairs that satisfy the above condition. We want to emphasize that we are not trying to study the popularity of independent paths with distinct points of congestion between node-pairs on the Internet; such topic has been studied by others [3].Instead, we focus on

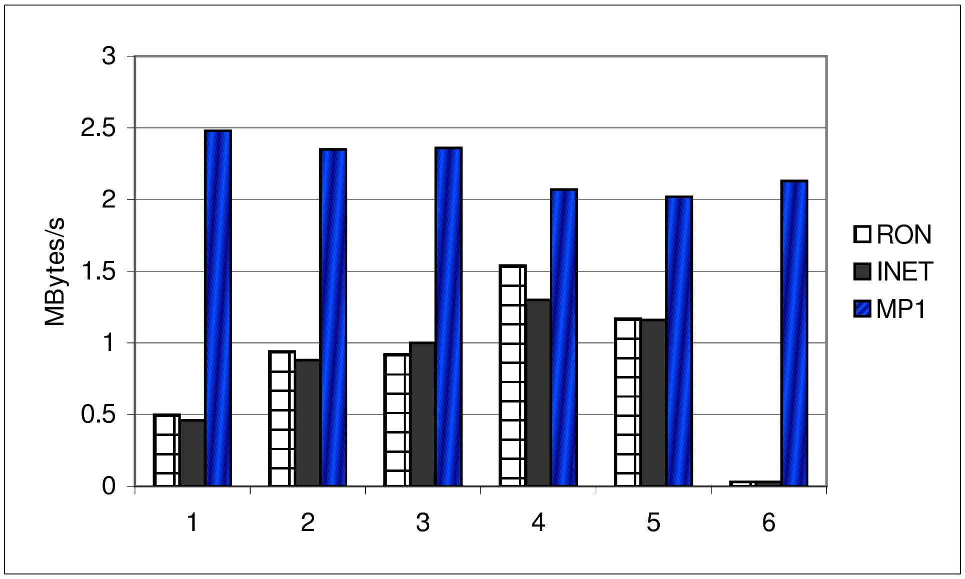

Figure 4.14: mTCP flows achieve better throughput than single-path flows

demonstrating that mTCP can achieve better performance by taking advantage of such redundant paths.

Among the 15 pairs, MP_1 flows achieve significantly higher throughput in 6 pairs, as shown in Figure 4.14. They achieve 33% to more than a factor of 60 better performance than single-path flows. We have to mention that MP_1 flows only try to aggregate the available bandwidth on multiple paths when there is no shared congestion. Other MP_d ($d \geq 2$) flows would obtain better performance, but they are potentially more aggressive.

The performance improvement of mTCP does not solely come from bandwidth aggregation on multiple paths, it is also because mTCP can help select better paths than those provided by the routing layer, such as the direct path or the *RON path* optimized for throughput. RON estimates the available bandwidth of a path using $score = \frac{\sqrt{1.5}}{rtt\sqrt{p}}$. Here p is the packet loss rate and rtt is the round trip time, both of which are obtained by active probing. Although it can help RON distinguish paths with significant perfor-

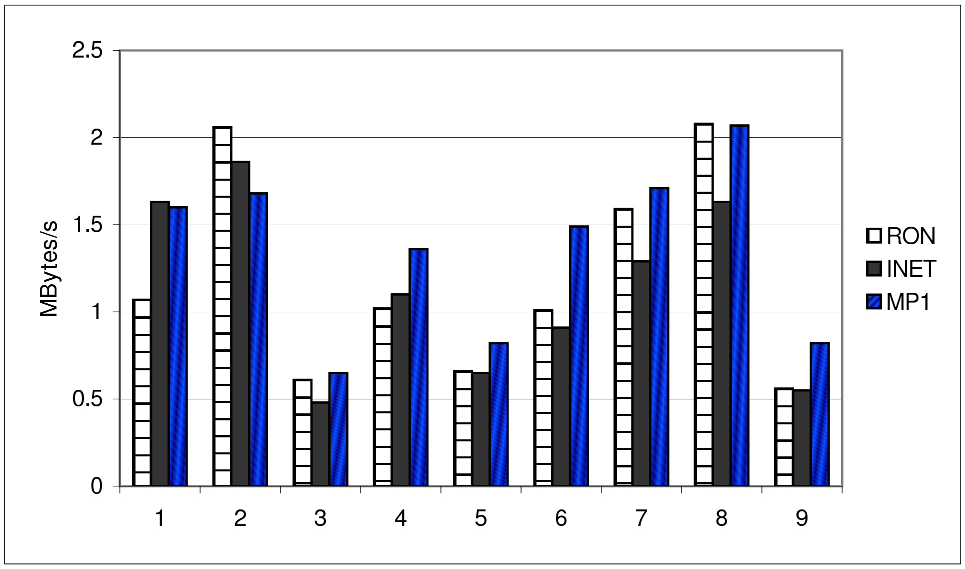

Figure 4.15: Throughput of mTCP and single-path flows is comparable

mance difference and select better alternate path, this estimate may not be accurate; a path with high score may actually have low available bandwidth [5]. In mTCP, the sender can monitor the performance of several paths in parallel. The throughput of each subflow provides a fairly good estimate of the available bandwidth on that path. This does not require any active probing because the data packets serve as probing packets. This can help mTCP discover and utilize better paths than the suboptimal *RON path* or direct path. We examined the paths in those 6 pairs and found that MP_1 flows do take paths different from either the direct paths or the *RON paths*. The achieved throughput of on those paths are higher than that of direct path or *RON path*.

Figure 4.15 shows the results of the remaining 9 pairs. By examining the paths, we find that all MP_1 flows degrade to single-path flows because of shared congestion, and the RON/INET/MP_1 flows all take the same single path for the whole transfer. Hence, the throughput of MP_1 flows should be comparable to that of INET/RON flows, as shown in Figure 4.15. In three pairs, MP_1 flows obtain slightly lower performance than RON/INET flows, this is because different types of flows are run sequentially and there is minor fluctuations in the available bandwidth of a path over time.

4.5 Summary

In this chapter, we present mTCP, a transport layer protocol, for improving end-to-end robustness and performance. mTCP can efficiently aggregate the available bandwidth on several paths in parallel, even if some paths encounter performance anomalies. mTCP flows are more robust to path failures than TCP flows, because they will not stall even when some paths fail. The failure recovery time is within a few seconds. To address the aggressiveness of mTCP during shared congestion, we integrate a shared congestion detection mechanism into mTCP so that correlated subflows can be suppressed. We also propose a heuristic to find disjoint paths based on traceroute. It can help mTCP to minimize the chance of concurrent failure or shared congestion. We have implemented our system on top of overlay networks and evaluated it on PlanetLab and Emulab.

Chapter 5

Conclusion and Future Work

5.1 Summary of the Dissertation

The Internet has experienced an exponential growth in recent years, so has its complexity. The increasing complexity can potentially lead to more network-layer instabilities. When anomalies do occur, users often experience severe performance degradations. In addition, these performance problems are extremely difficult to debug because of the heterogeneity and decentralization of the Internet. What we urgently need today is a better understanding of these anomalies and more robust network protocols that perform well during anomalous periods.

This dissertation has addressed two important issues in this area. The first is to build systems and tools that help understand the Internet's behavior under routing anomalies. This is a crucial step for us to be able to diagnose routing anomalies quickly and prevent similar problems from occurring in the future. The second is to design end-to-end network protocols that are resilient to misbehavior. So when performance problems occur, their negative impact on end users will be minimized.

5.1.1 Internet Path Failure Monitoring and Characterization

Detecting network failures generally requires examining large volumes of traffic data to find misbehavior. We observe that wide-area services, such as peer-to-peer systems and content distribution networks, exhibit large traffic volumes, spread over large numbers of geographically-dispersed endpoints. This makes them ideal candidates for observing wide-area network behavior. We have built a fault diagnosis system, called *PlanetSeer*, that passively monitors traffic generated by wide-area services. When detecting anomalous network behavior, it actively probes the network from multiple nodes to quantify and characterize the failure. This approach provides several advantages over other techniques: (1) we obtain more complete and finer-grained views of routing anomalies since the wide-area nodes already provide geographically-diverse vantage points; (2) we incur limited additional measurement cost since most active probing is initiated when passive monitoring detects oddities; and (3) we detect anomalies at a much higher rate than other researchers have reported since the services provide large volumes of traffic to sample.

We have deployed PlanetSeer on PlanetLab, a wide-area network testbed. During a 3-month period in 2004, we have monitored network paths that traversed over 10,000 autonomous systems (ASes), more than half of the total number of ASes on the Internet. We have confirmed nearly 272,000 routing anomalies, a rate that is 10 to 100 times higher than reported in previous studies. We have also made several interesting observations from this study. First, the failure distribution is highly skewed. The core of the Internet is much more stable than the edges. A small number of service providers (ISPs) contribute to a disproportionally large number of persistent routing loops. Second, persistent and temporary loops show different behaviors. Persistent loops span far fewer routers and ISPs, and they can persist for an extended period of time. Third, path changes and outages exhibit different characteristics. Outages appear closer to the edges of the network and have narrower scope. Many outages occur at the last hop. Finally, while overlay routing can circumvent failures between two endpoints by sending data indirectly on an alternate path through a third computer, the alternate paths often suffer from high latency inflation.

5.1.2 Robust Transport Layer Protocol Using Redundant Paths

TCP is the most commonly used end-to-end protocol on today's Internet. TCP uses only one path between two endpoints. When a performance anomaly, such as outage or congestion, occurs on that path, TCP's performance will be severely degraded or even become completely unusable. Recent work on Internet measurement and overlay networks has shown that redundant paths are common between pairs of hosts. Based on this observation, we have designed a transport-layer protocol, called *mTCP*, that can enhance the performance and robustness of end-to-end communications when experiencing performance anomalies. mTCP stripes one flow's packets across several paths so that it efficiently aggregates the bandwidth of several paths in parallel. When some paths encounter anomalies, mTCP can still utilize the bandwidth on other good paths.

In mTCP, we provide a comprehensive design that addresses the inter-related issues of congestion control on multiple paths, unfair use of congested links, path selection, and recovery from path failures. (1) mTCP separates congestion control of different paths so that it does not suffer from the penalty of using bad paths. (2) mTCP reacts to path failures quickly; the recovery process takes just a few seconds, much faster than relying on wide-area routing protocols. (3) Because multipath flows can unfairly obtain a larger share of bandwidth compared to single-path flows during congestion, mTCP integrates a shared congestion detection mechanism to ensure fairness to other single-path flows. (4) mTCP uses a heuristic to discover disjoint paths so as to minimize the chance of concurrent failures and shared congestion. We have evaluated mTCP using extensive experiments in both local-area and wide-area networks and have observed significant improvement opportunities.

5.2 Future Work

The overall goal of my research is to improve the performance and resilience of the Internet infrastructure by building self-diagnosing and self-repairing network protocols and services. My thesis work on failure monitoring and robust end-to-end protocols is the first step toward this goal. I intend to pursue my future research in at least three directions.

5.2.1 Debugging Routing Anomalies

Despite the recent advances in networking research, we still lack the ability to completely understand the routing behavior of the Internet. Users often experience service disruption without any insight about why it occurs or how to circumvent it. Debugging these routing problems is exceptionally hard. This is because the Internet is such a complex system that the malfunction of any of its components may affect a large portion of the whole system. Equipment failures, software bugs, and misconfigurations occur more often than we would like. The failure of an individual component can be propagated to other parts of the network, which may in turn lead to instabilities in a wider area. Such service disruptions may take hours or even days to resolve. Complicating matters even further, the Internet is not owned by a single administrative entity but instead by many service providers (ISPs); no single entity has a global view of the Internet. Troubleshooting these routing problems becomes extremely challenging without collaboration among different ISPs.

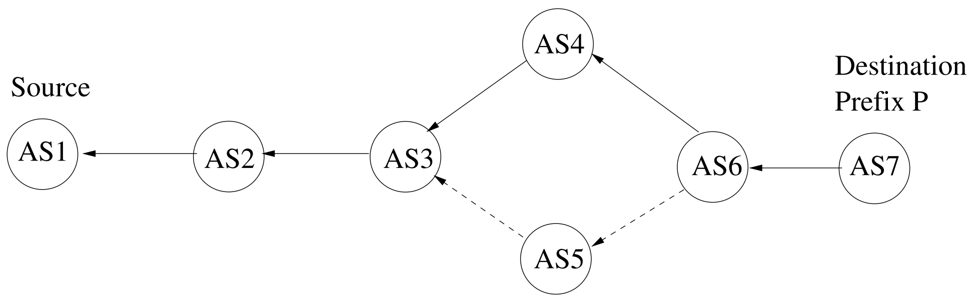

Figure 5.1: Locating the origin of AS-path change

We plan to develop algorithms and techniques for pinpointing routing problems, since routing instability is one of the major sources of performance problems. Several research efforts have tried to use interdomain routing (BGP) messages to isolate routing instability [29, 14, 16, 55]. In this approach, the basic assumption is that an AS path change is caused by some instability either on the previous best path or the new best path. By correlating the BGP updates along time, views, and prefixes, they are able to pinpoint the instability origin to a set of ASes and their peerings. However, their studies show BGP information itself is insufficient to reveal the location of instabilities under many circumstances, because BGP messages only convey AS-level information. For instance in Figure 5.1, suppose a source node in AS 1 wants to reach a destination prefix P in AS 7, and the AS path changes from "1 2 3 4 6 7" to "1 2 3 5 6 7". At the first glance, one might deduce the route change is caused by some instability in AS 3, 4, 5, or 6. However, this may not be true. An instability in AS 1, 2, or 7, *e.g.* a hot-potato routing change [86], can lead to the same phenomenon. Therefore, we cannot exclude AS 1, 2, or 7 from the set of possible instability origins.

To be able to pinpoint the origins of routing problems, we need to correlate data from multiple sources. Examining the problems from multiple vantage points will provide us a more complete view on them, such as which parts of the network are affected or what kind of anomalous events are perceived. Correlating data of different types, *e.g.* routing messages, traffic data, and end-to-end probing, will increase the chance of detecting problems and reduce the chance of raising false alarms. Using a simple example, let us show how we can improve the accuracy of anomaly diagnosis by combining end-to-end probing with BGP messages. Again in Figure 5.1, suppose we are able to launch traceroutes from AS 1 to destination prefix P, this will help us narrow the set of possible instability origins. For instance, if the traceroutes reveal that there is a hot-potato routing change in AS 2 that causes the IP-level path to exit AS 2 at a different egress point, we may infer the instability origin is very likely to be in AS 2. On the contrary, if there is no IP-level route change in AS 1, 2, or 7, or any of their peerings, it is reasonably safe to infer that the instability origin is in AS 3, 4, 5, or 6. We plan to develop the right mechanisms for aggregating the observations across different locations and sources.

5.2.2 Debugging Non-Routing Anomalies

So far, we have been focusing on routing anomaly. However, as we have explained in Section 1.1, routing anomaly is *not* the only source of performance degradation on the Internet. Many performance problems are caused by non-routing anomaly, such as congestion. To gain a better understanding of network performance problems, we need to study non-routing anomaly as well.

As described in Section 3.2.3, PlanetSeer detects possible routing anomalies by watching for consecutive timeout and TTL change in TCP traffic. Besides these two types of events, it also monitors end-to-end performance metrics, including RTT, RTTVAR, and loss rate. During the 3-month monitoring period, we often observe end-to-end performance degradations, such as high loss rate or high latency, without seeing consecutive timeout or TTL change. These performance degradations could be caused by non-routing anomalies.

We can use this type of performance degradations as an indication of non-routing anomalies and invoke active probing to diagnose them, just like what we have done with routing anomalies. We are interested in answering the following questions:

- Where does performance problem occur? Can we locate the link/router that incurs most losses and delays?
- Why does performance problem occur? Can we identify the cause of losses and delays? What are the loss/delay patterns exhibited by different types of problems?
- Since routing anomalies can cause heavy losses, can we still assume that most losses are due to congestion? What is its impact on TCP if this assumption is not true?

There has been much work on characterizing losses, delays, and congestion on the Internet [72, 60, 6, 40, 3]. Since this approach has to continuously probe a path under study, it can only monitor a limited set of network paths which may not be representative of the Internet. In our approach, since the active probing is triggered by real performance degradation, we can avoid probing the network needlessly during most time when the network is under normal condition. As a result, we will be able to study a much broader set of paths and gain a more complete view of performance problems on the Internet.

5.2.3 Internet Weather Service

Internet measurement has provided techniques to infer network properties from external observations. Researchers have measured the Internet from many different perspectives, *e.g.* mapping network topology [48, 33, 83], inferring routing policies [82], and estimating path bottlenecks [40, 3]. These isolated efforts have yet to be combined to form a complete picture of the Internet. We plan to build an *Internet Weather Service* that can not only track the *current* status of the Internet but also *predict* how the Internet will evolve based on *past* observations. To some sense, it is like that today's weather service can alert us of bad future weather conditions, the Internet Weather Service may be able to predict which parts of the network are going to experience what types of performance degradation.

This type of service is exciting because it has great potential for generating impact both on the society and on the research community. On the one hand, any user may query the service to understand why she cannot connect to certain websites or why her downloading speed is slow. On the other hand, network researchers can use this service to gain historical, real-time, or future information about network behavior. The historical information can help researchers construct realistic models of the Internet and evaluate the robustness of their system designs under failures. The real-time or future information can benefit various systems, such as application-layer multicast [19, 18, 31, 47, 9], overlay networks [5], and multi-homing [2]. These systems may use the information to select better paths or build more robust topologies.

A number of challenges arise when we attempt to build such an open and shared infrastructure. To obtain a reasonably accurate network status map, we have to decide where to place the *monitoring sensors* and what types of information need to be collected. Since we want to have a good coverage of the Internet, the number of sensors can be very large and they may produce large volumes of data. We have to develop the right kind of data abstraction and aggregation mechanism to minimize the overhead on the network.

We are also interested in exploring algorithms for predicting network behaviors based on past observations.

Bibliography

[1] H. Adiseshu, G. M. Parulkar, and G. Varghese. A reliable and scalable striping protocol. In *Proceedings of ACM SIGCOMM*, 1996.

[2] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A measurement-based analysis of multihoming. *ACM SIGCOMM*, Aug. 2003.

[3] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area Internet bottlenecks. In *Proceedings of ACM Internet measurement conference*, Oct. 2003.

[4] M. Allman, H. Kruse, and S. Ostermann. An application-level solution to TCP's satellite inefficiencies. In *Proceedings of WOSBIS*, Nov. 1996.

[5] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *ACM SOSP*, Oct. 2001.

[6] D. Andersen, A. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. *ACM IMC*, Oct. 2003.

[7] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On multiple description streaming with content delivery networks. In *Proceedings of IEEE INFOCOM*, 2002.

[8] A. Banerjea. Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels. *Proceedings of ACM SIGCOMM*, Aug. 1996.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *ACM SIGCOMM*, Aug. 2002.

[10] S. Banerjee, T. G. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Passive and Active Measurement Workshop*, April 2004.

[11] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. *ACM IMW*, Nov. 2002.

[12] A. Basu, C. Ong, A. Rasala, F. Shepherd, and G. Wilfong. Route oscillations in I-BGP with route reflection. *ACM SIGCOMM*, Aug. 2002.

[13] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Mawrzoniak. Operating system support for planetary-scale network services. In *USENIX/ACM NSDI*, Mar. 2004.

[14] M. Caesar, L. Subramanian, and R. H. Katz. Route cause analysis of Internet routing dynamics. In *Tech Report UCB/CSD-04-1302*, 2003.

[15] J. Case, M. Fedor, M. Schoffstall, and J. Davin. Simple network management protocol (SNMP). RFC 1157.

[16] D.-F. Chang, R. Govindan, and J. Heidemann. The temporal and toplogical characteristics of BGP path changes. In *IEEE ICNP*, Nov. 2003.

[17] B. Chazelle. The discrepancy method: randomness and complexity. *Cambridge University Press*, 2000.

[18] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. *ACM SIGCOMM*, Aug. 2001.

[19] Y.-H. Chu, S. Rao, and H. Zhang. A case for end system multicast. *ACM SIGMETRICS*, June 2000.

[20] D. Oran. OSI IS-IS intra-domain routing protocol. RFC 1142.

[21] M. Dahlin, B. Chandra, L. Gao, and A. Nayate. End-to-end WAN service availability. *ACM/IEEE Trans. Netw.*, Apr 2003.

[22] R. Dube. A comparison of scaling techniques for BGP. *ACM CCR*, July 1999.

[23] J. Duncanson. Inverse multiplexing. In *IEEE Communications Magazine*, volume 32, pages 34–41, 1994.

[24] Emulab. http://www.emulab.net.

[25] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of Internet path faults on reactive routing. In *ACM SIGMETRICS*, Jun 2003.

[26] N. Feamster and H. Balakrishnan. Towards a logic for wide-area Internet routing. *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Aug. 2003.

[27] N. Feamster, Z. Mao, and J. Rexford. BorderGuard: Detecting cold potatoes from peers. *ACM IMW*, Oct. 2004.

[28] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. *ACM SIGCOMM*, Aug. 2000.

[29] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *ACM SIGCOMM*, Aug 2004.

[30] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE INFOCOM*, June 2002.

[31] P. Francis. Yoid: extending the multicast Internet architecture, 1999. http://www.icir.org/yoid.

[32] G. Huston. BGP reports. http://bgp.pataroo.net.

[33] R. Govindan and H. Tangmunarunkit. Heuristic for Internet map discovery. *IEEE INFOCOM*, 2000.

[34] T. Griffin and G. Wilfong. An analysis of BGP convergence. *ACM SIGCOMM*, Sept. 1999.

[35] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Proc. of IPDPS*, 2002.

[36] K. Harfoush, A. Bestavros, and J. Byers. Robust identification of shared losses using end-to-end unicast probes. *Proceedings of IEEE ICNP*, Oct. 2000.

[37] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Detection and analysis of routing loops in packet traces. In *ACM IMW*, 2002.

[38] H. Hsieh and R. Sivakumar. ptcp: An end-to-end transport layer protocol for striped connections. In *Proceedings of IEEE ICNP*, 2002.

[39] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of ACM MOBICOM*, 2002.

[40] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. *ACM SIGCOMM*, Aug. 2004.

[41] IANA. Special-use IPv4 addresses. RFC 3330.

[42] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *ACM IMW*, Nov 2002.

[43] Iperf. http://dast.nlanr.net/Projects/Iperf/.

[44] J. Moy. OSPF version 2. RFC 2328.

[45] J. Postel. Internet control message protocol. RFC 792.

[46] M. Jain and C. Dovrolis. End-to-End available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. *ACM SIGCOMM*, Aug. 2002.

[47] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: reliable multicasting with an overlay network. *USENIX OSDI*, Oct. 2000.

[48] k claffy, T. Monk, and D. McRobb. Internet tomography. *Nature*, Jan. 1999.

[49] D. Katabi, I. Bazzi, and X. Yang. An information theoretic approach for shared bottleneck inference based on end-to-end measurements. In *Class Project, MIT Laboratory for Computer Science*, 1999.

[50] A. Khanna and J. Zinky. The revised ARPANET routing metric. *ACM SIGCOMM CCR*, Sept. 1989.

[51] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. *ACM IMW*, Oct. 2003.

[52] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. In *ACM SIGCOMM*, Sep 2000.

[53] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and wide-area backbone failures. Technical Report CSE-TR-382-98, University of Michigan, 1998.

[54] C. Labovitz, G. Malan, and F. Jahanian. Origins of Internet routing instabilities. In *IEEE INFOCOM*, 1999.

[55] M. Lad, A. Nanavati, D. Massey, and L. Zhang. An algorithmic approach to identifying link failures. In *ACM PRDC*, 2004.

[56] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM*, Aug. 2004.

[57] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area networks. In *Proceedings of CHEP*, Sept. 2001.

[58] Y. Liang, E.G.Steinbach, and B. Girod. Real-time voice communication over the Internet using packet path diversity. In *Proceedings of ACM Multimedia*, 2001.

[59] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Proceedings of ICNP*, Nov. 2001.

[60] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level path diagnosis. *ACM SOSP*, Oct. 2003.

[61] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *ACM SIGCOMM*, 2002.

[62] Z. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an accurate AS-level traceroute tool. In *ACM SIGCOMM*, 2003.

[63] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. *RFC 2018*, Oct. 1996.

[64] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of ACM SIGCOMM*, Aug. 2003.

[65] NANOG. http://www.nanog.org.

[66] NetFlow. http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml.

[67] T. Nguyen and A. Zakhor. Path diversity with forward error correction (PDF) system for packet switched networks. In *Proceedings of IEEE INFOCOM*, 2003.

[68] NS. http://www.isi.edu/nsnam/ns.

[69] U. of Oregon RouteViews Project. http://www.routeviews.org.

[70] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM*, 1998.

[71] V. Paxson. End-to-end routing behavior in the Internet. In *ACM SIGCOMM*, Aug 1996.

[72] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Trans. Netw.*, 7(3), 1999.

[73] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. *Proceedings of ACM HOTNETS*, Oct. 2002.

[74] PlanetLab. http://www.planet-lab.org.

[75] Y. Rekhter and T. Li. A border gateway protocol. RFC 1771.

[76] M. Roughan, T. Griffin, Z. Mao, A. Greenberg, and B. Freeman. IP forwarding anomalies and improving their detection using multiple data sources. *ACM SIGCOMM Network Troubleshooting Workshop*, Aug. 2004.

[77] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. *Proceedings of ACM SIGMETRICS*, June 2000.

[78] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *ACM SIGCOMM*, Aug 1999.

[79] A. Shaikh and A. Greenberg. Experience in black-box OSPF measurement. *ACM IMW*, Nov. 2001.

[80] H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing*, 2000.

[81] A. Snoeren. Adaptive inverse multiplexing for wide area wireless networks. In *Proc. of IEEE Conference on Global Communications*, 1999.

[82] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. *ACM SIGCOMM*, Aug. 2003.

[83] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM*, Aug. 2002.

[84] 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. In *RFC 2960*, Oct. 2000.

[85] L. Subrmanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. *IEEE INFOCOM*, June 2002.

[86] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of hot-potato routing in IP networks. In *ACM SIGMETRICS*, Jun 2004.

[87] B. Traw and J. Smith. Striping within the network subsystem. In *IEEE Network*, volume 9, pages 22–32, 1995.

[88] V. Paxson and M. Allman. Computing TCP's retransmission timer. RFC 2988.

[89] K. Varadhan, R. Govindan, and D. Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 2000.

[90] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *OSDI*, Dec. 2002.

[91] J. Wu, Z. Mao, J. Rexford, and J. Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. *USENIX NSDI*, May 2005.

[92] O. Younis and S. Fahmy. On efficient on-line grouping of flows with shared bottlenecks at loaded servers. In *Proceedings of IEEE ICNP*, Nov. 2002.

[93] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. *USENIX Annual Technical Conference*, June 2004.

[94] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. *USENIX OSDI*, Dec. 2004.

[95] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. *Proceedings of ACM SIGCOMM*, Aug. 2002.

[96] Y. Zhang, N. Duffield, V. Paxson, and S. Shenkar. On the constancy of Internet path properties. *ACM IMW*, Nov. 2001.