Systems for Improving Internet Availability and Performance

Ethan B. Katz-Bassett

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2012

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Ethan B. Katz-Bassett

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of the Supervisory Committee:

Thomas E. Anderson

Arvind Krishnamurthy

Reading Committee:

Thomas E. Anderson

Arvind Krishnamurthy

David Wetherall

Date:

University of Washington

Abstract

Systems for Improving Internet Availability and Performance

Ethan B. Katz-Bassett

Co-Chairs of the Supervisory Committee:
Professor Thomas E. Anderson
Department of Computer Science and Engineering

Associate Professor Arvind Krishnamurthy
Department of Computer Science and Engineering

The Internet's role in our lives continues to grow, but it often fails to provide the availability and performance demanded by our increasing reliance on it. The problems largely stem from the fact that network operators at an Internet service provider (ISP) have little visibility into and even less control over the routing of other ISPs on which they depend to deliver global connectivity.

In this dissertation, I design, build, and evaluate practical distributed systems that ISPs can use today to understand availability and performance problems. I develop *reverse traceroute*, a system to measure reverse paths back to the local host from arbitrary destinations. While tools have long existed to measure the forward direction, the reverse path has been largely opaque, hindering troubleshooting efforts. I show how content providers such as Google could use reverse traceroute to troubleshoot their clients' performance problems. The rest of the dissertation focuses on long-lasting routing outages. My measurements show that they occur frequently and contribute significantly to overall unavailability. To address these long-term problems, I develop a system, *LIFEGUARD*, for automatic failure localization and remediation. First, the system builds on reverse traceroute to locate faults, even in the presence of asymmetric paths and failures. Second, I develop a technique that enables edge ISPs to steer traffic to them around failures, without requiring the involvement of the network causing the failure. Deploying *LIFEGUARD* on the live Internet, I find that it can effectively route traffic around particular ISPs without causing widespread disruption.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The first person I use in this dissertation does not capture how grateful I am for the role that others played in this work.

I feel very fortunate to have had multiple great advisors at the University of Washington. Tom Anderson sharpened this work by combining an ability to see the big picture with incredible attention to the smallest details. He helped me learn which problems to work on and which to ignore. Tom's model and advice will serve me well as a teacher, advisor, and researcher. Arvind Krishnamurthy's endless enthusiasm and ideas kept me inspired and led to a number of my favorite pieces of the dissertation. Arvind was always available and willing to talk about or work on anything. Before I started working with Tom and Arvind, David Wetherall gave me confidence and direction by asking me to work with him during my first year of graduate school and advising me through my qualification exam. David's ideas and feedback on my first Internet measurement project helped put me on the path that ended in this dissertation. I greatly appreciate the flexibility and understanding Tom, Arvind, and David displayed in helping me deal with the ups and downs of graduate school.

In addition to my advisors, I benefited from conversations with my other committee members, Eve Riskin and Radha Poovendran. Eve and Richard Ladner also advised me on projects in data compression and accessibility. They and other faculty at UW were always generous with their time and advice. Researchers from other institutions, including Nick Feamster, Yatin Chawathe, Randy Bush, kc claffy, Young Hyun, Daniel Karrenberg, Robert Kisteleki, and Matt Welsh, provided valuable advice and support on pieces of this work.

It is hard to imagine ending up here without the influence of a number of great teachers and professors, including Fran O'Donnell, Duane Bailey, and Tom Murtagh. Lee Osterweil advised me while I worked at UMass – my time there led to me applying to graduate school.

I really enjoyed working with collaborators on my graduate research. The work is much better for their participation. Harsha Madhyastha, Ítalo Cunha, Dave Choffnes, Vytautas "Valas" Valancius, and John P. John contributed greatly to parts of this dissertation, and I learned from all of them. I look forward to working with them more in the future. I received very useful feedback from others in the networks group, including Colin Dixon, Tomas Isdal, Michael Piatek, Ivan Beschastnikh, Dan Halperin, and Lisa Glendenning. Other students at the University of Washington helped my research by making sure I was not consumed by it. In particular, Dan Leventhal, Peter Hornyack, Ben Ylvisaker, and Brian Ferris joined me on many trips in search of delicious meals.

One of my favorite parts of graduate school was working with some of the smartest undergraduates around, including Justine Sherry, Colin Scott, Mary Pimenova, and Ashoat Tevosyan. They inspired me, challenged me, amused me, and impressed me.

I am extremely grateful for the staff in my department at the University of Washington. Lindsay Michimoto made sure I was on track and knew the answers to any questions I had along the way. Melody Kadenko and Julie Svendsen shielded me from administrative details. The CS Lab Support staff kept the computers I used running and were understanding and helpful when I ran disruptive network experiments. With the program, logistics, and computer systems all running smoothly, I can only blame myself for any less productive periods I experienced along the way.

My research relied on Internet testbeds. It would not have been possible without the researchers, support staff, and participating sites who made PlanetLab, BGP-Mux, RON, RIPE's Routing Information Service, and RouteViews available. I am especially indebted to the PlanetLab staff, including Larry Peterson, Sapan Bhatia, Andy Bavier, Marc Fiuczynski, Vivek Pai, and KyoungSoo Park.

I completed a dissertation about the Internet without ever having a proper Internet connection in my apartment. I appreciate the friendly atmospheres and decent connections at the cafes and bars that hosted me while I worked: the Star Life on the Oasis Cafe, Sureshot, Curio Confections, Blue Dog, Remedy Tea, Big Time Brewery, Smith, Glasslands Gallery, and Manhattan Inn.

Amy Ferris Wheeless, Becky Kerman, Chelsea Kurnick, and my parents edited sections of this document. I snuck in any remaining typos on my own, afterwards.

I greatly appreciate the funding from the National Science Foundation, Cisco, and Google that made this work possible.

I cannot imagine having completed my Ph.D. without the support of friends and family. Thanks.

Chapter 1

INTRODUCTION

The role of the Internet in our lives has changed substantially in recent years. With smart phones in our pockets and so much of our lives – emails, videos, and photos – stored in the cloud, we are always online and expect to access the Internet continuously, from anywhere. We expect fast performance from interactive web applications. We demand high bandwidth to provide streaming movies. Mobile devices also enable more of us to come online. In 2011, 2 billion people – more than a quarter of the world's population – were online [57]. The continuous streams of updates on sites like Twitter and Facebook mean that we go online more often. Each minute, humankind collectively spends 30 years on Facebook alone [36]. Important services such as modern telephony and finance depend on the Internet. We would like to use the Internet to run critical services, such as power grid management and outpatient medical monitoring. Because of these current, emerging, and future uses, content and service providers place a priority on fast, reliable paths between their clients and their services – the Internet needs to provide good performance and high availability.

Is the current Internet up to the task? No. Operational experience has uncovered numerous problems that would make anyone pause before trusting the Internet with the timely delivery of traffic that truly mattered. In fact, we have all experienced the frustration of a site so slow to load that we browse away from it. Google found that 40% of their clients' connections experience a network latency of at least 400ms [66]. This 400ms is equivalent to the traffic circling the equator twice on its path between the client and Google, despite 75% of networks having a Google server within 1000 miles of them. Such slow routes cost providers clients and money – Amazon found that every additional 100ms of delay in loading a page costs them 1% of their sales [73]. Similarly, a

Yahoo! study found that, when a page took an additional 400ms to load, the latency caused 5-9% of users to browse away from web pages, rather than letting them load to completion [123]. Taken together, these results suggest that content and service providers have a strong incentive to have low latency paths to their clients.

Even worse than slow performance, traffic may disappear into *black holes* and fail to reach a destination. Networks continue to try to send traffic along the failing path, even though alternate paths exist. These problems can occur, for example, when a router fails to detect that its memory has been corrupted. A study found that 10% of routes were available less than 95% of the time, more than half were available less than 99.9% of the time, and more than 65% of routes were available less than 99.99% of the time [70]. Further, many of the route failures lasted for extended periods, with 40% taking at least 30 minutes to repair. Surveying a range of datasets collected between 1994 and 2000, researchers found that average daily unavailability was likely between 0.5% and 2.4%, with 5% of faults lasting more than 2.75 hours and some lasting over one day [32]. My own results, presented in Sections 3.4 and 3.3.2, confirm that outages lasting hours or even days still occur quite frequently.

Problem Statement: Google and other service providers want to react quickly and effectively when performance and availability problems arise, but they are stuck with outdated tools and protocols stretched to their limits. The paths between their clients and their services traverse other ISPs. The configurations and policies of multiple devices, ISPs, and protocols interact in complicated ways and are susceptible to human error. Existing tools provide little visibility into these other ISPs, leaving operators with limited understanding of the routes on which they depend. For example, communication on the Internet is generally two-way, and paths are frequently asymmetric [97, 51], with the path from a service to its client differing from the path from the client back to the service. Yet, available tools do not give a service provider a way to measure the path back from the client, and so operators have to attempt to troubleshoot problems while seeing only half of the round-trip path. Even when network operators can locate the cause of a problem, the Internet's protocols give them limited ability to influence other ISPs or even to convey information across ISP boundaries. For example, the protocols do not give a way to explicitly instruct other ISPs to avoid an ISP that fails to forward traffic; instead, operators often resort to contacting people at other ISPs via email or other channels, slowing fixes. These and similar issues force operators at major providers to spend much of their time in triage of routing problems, and the problems cause high operational costs, damaged reputations, and frustrated customers.

One possible approach to improving Internet availability and performance would be to redesign the Internet protocols. Such an approach has appealing features. Internet protocol limitations (such as the lack of visibility) and interactions between protocols contribute to unavailability and sub-optimal performance. Further, the Internet has evolved substantially from the setting in which it was designed, with many more and different devices and new types of traffic. So, a fresh design tailored to current and emerging demands would likely better address many needs. Recent research provides promising approaches to improve Internet availability by modifying Internet protocols [68, 133, 71, 60]. Such work is valuable and provides fundamental insight into underlying protocol limitations, as well as good long-term solutions. However, this type of work faces a difficult path to improve Internet availability, because of the challenges in getting the modifications adopted. Our dependence on the Internet argues for improvements in its performance and availability, yet makes it difficult to move on from the existing protocols; all existing ISPs, devices, and services use and depend on them, and the transition costs can be significant.

1.1 Approach and Goals

In this dissertation, I take a different, complementary approach towards the goal of improving Internet performance and availability. I present work that I have done to design, build, and evaluate protocol-compliant techniques that work on today's Internet to provide richer visibility, to locate problems, and to automatically repair broken Internet routes, using only deployed devices and available hosts without modification. Because of this property, individual providers seeking to improve their availability and performance can immediately and unilaterally deploy and use my techniques. While my approaches do not apply to every Internet availability or performance problem, the techniques in this thesis represent a large step forward. I now describe the challenges I solve and the types of problems to which they apply. These challenges arise because of the Internet's current design and implementation and because of practical considerations in building a real system that can be widely used.

1.1.1 Measuring Reverse Paths

Internet traffic often experiences poor performance in the form of delays much higher than what is possible given the Internet's topology, with the inflated latency often stemming from geographically circuitous routes [4, 118, 41, 106, 66]. Research into the causes of this circuitousness found that the routes taken across an individual ISP tend to be relatively direct, but that inter-ISP paths are often inflated. Further, indirectness is not caused by intentional policy decisions, but rather by the lack of good tools to allow ISPs to find better routes [118].

More recent work from Google also found that the limited view of routing afforded by available measurement tools was the central limitation when trying to understand poorly performing paths between Google and its clients [66]. The paper presents an approach that Google uses to identify and prioritize poorly performing routes. The approach allows Google engineers to explain the underlying causes of some inflated latencies and propose possible fixes, such as changing how Google announces its address space or contacting other ISPs to update out-of-date router configurations. In essence, the methodology involves measuring the path from a Google data center to a client network and comparing it with possible alternate paths available given the Internet's topology. This methodology allowed Google, within a four month period in 2008, to nearly halve the number of South American client networks experiencing extremely inflated latencies. However, the authors acknowledged that their system was unable to explain the poor performance in many remaining cases. In particular, they cited their inability to measure the reverse path back from clients to Google as the principal limitation. Since the client incurs the combined latency of the path from Google to the client and the (likely asymmetric) path from the client back to Google, this limitation meant they were attempting to troubleshoot while only being able to measure half the route.

Because the current lack of tools that provide reverse path information hinders our ability to troubleshoot performance problems, a central goal of my dissertation is the design of a technique capable of measuring the path back from an arbitrary destination to a controlled host. To fit with my objective of approaches that providers can unilaterally deploy today, my goal is to develop techniques that work without participation of the destination, using only methods supported by current routers. Such a capability would allow providers such as Google to troubleshoot and improve geographically circuitous routing problems [66]. The ability to measure reverse paths is also essential for the parts of this thesis that deal with availability problems and could likely provide benefit in troubleshooting other performance problems, such as lossy or low capacity paths.

1.1.2 Characterizing Long-Lasting Outages on the Internet

Similar to how routes are often more circuitous than required by the Internet's topology, previous research found that, when a destination is unavailable from a particular Internet host, it is often reachable from other locations, suggesting the presence of paths around the problem [4, 47]. I focus on disruptions to connectivity due to routing problems where a working policy-compliant path exists, but ISPs instead route along a different path that fails to deliver packets. In theory this should never happen – if working paths exist, the Internet protocols are designed to find them, even in the face of failures. In practice, such outages are common. In Section 3.4, I show that even well-provisioned cloud data centers experience frequent routing problems.

It is well-known that short-term disruptions occur during failover, recovery, and routing protocol convergence [69, 53, 130]; I focus instead on outages that persist over longer timescales that are less likely to be convergence-related, as these events are less understood. Existing research provides promising approaches to dealing with the transient unavailability that occurs during protocol convergence [68, 71, 60, 69]. Long-lasting problems instead currently resolve over human timescales – that is, they likely currently require human intervention. I present data showing that they contribute significantly to end-to-end unavailability, suggesting that we need to address these problems in order to substantially improve availability.

The Internet community would benefit from a characterization of these long lasting outages, in order to guide the design of approaches to addressing the problems. To fully capture the nature of the problems, this characterization should study outages across the Internet. To provide guidance useful for improving availability, it would help to understand various properties of the problems, including: how common outages are; how many paths experience outages; whether outages are partial or complete; and whether alternate routes exist that could resolve the problems.

Previous systems addressed some aspects of this goal in different ways. A number of systems monitored reachability status in real-time, but within contexts that are narrower than the whole Internet, such as a testbed [97,4,37,47], an autonomous system [132,109], or a particular distributed system's clients [136]. Other systems, such as *iPlane* [78] and Ark [5], have broad and continuous Internet coverage but, being designed for other purposes, monitor at too infrequent a rate to provide real-time fault diagnosis. Some techniques detects certain reachability issues in real-time at the Internet scale by passively monitoring Internet route updates via Border Gateway Protocol (BGP) feeds [38, 132, 22, 70]. However, relying on BGP feeds alone is insufficient because the existence of a route does not imply reachability [37]; BGP acts as a control plane to establish routes for the data plane on which Internet traffic flows, and connectivity problems that do not present themselves as events on the monitored control plane will evade such systems.

The goal, then, is a system that monitors data plane outages across the entire Internet. A practical consideration makes this goal tough. I would like the system to monitor outages affecting ISPs across the Internet, even without control over end-hosts in all of these ISPs. The Internet includes hundreds of thousands of networks, but readily available vantage points exist in only a few hundred. While I could pursue recruiting end-hosts in networks across the Internet, such an approach generally requires either becoming an integrated component of widely-used operating systems or bundling with popular software. Both these avenues could take years to come to fruition, if they were adopted at all. Such a delay in developing the ability to characterize failures would slow my efforts to build systems to address failures, compromising my goal of developing techniques that work today. Therefore, my goal is to monitor and characterize failures across the Internet, using only available vantage points. This restriction both limits where I can make measurements from and caps the rate at which I can introduce traffic into the Internet without the measurement traffic overburdening the vantage points or causing problems of its own.

1.1.3 Accurately Locating Failures from the Edge of the Internet

Motivated by the characteristics of outages I report in Chapter 3, I propose a two part approach to improving Internet availability. First, I develop techniques capable of isolating the ISP or router causing a long-lasting partial outage. My results show that outages with these characteristics contribute substantially to unavailability, and so addressing them could have a significant impact. Because I target long-lasting problems, my techniques need not work instantaneously to have an impact, and I can take time to issue measurements. Because I focus on partial outages, I can take advantage of vantage points with working paths to issue measurements to better understand the failed paths. Second, I develop an approach that, given the location of a failure, will automatically reroute traffic around that ISP to restore connectivity. My results show that alternate policy-compliant working paths frequently exist during long-lasting partial outages, revealing the promise of this approach. Because I propose to automatically reroute traffic around the failure, my techniques to localize the failure must be accurate. My rerouting approach does not require the failure to be repaired to restore connectivity and, in fact, does not require the cooperation of operators at the ISP causing the failure, and so service providers can use it unilaterally and can deploy it automatically, speeding recovery. I now address these goals in more detail.

A number of issues complicate my goal of accurately identifying the entity (ISP, router) causing an outage. First, given my previous goal for my systems to work given only the limited number of available vantage points, I will need to locate failures in ISPs within which I do not have direct access to any computers. Techniques exist to pinpoint failures within a single ISP [109, 64, 63], but they require access to privileged measurements from within the ISP. Most Internet routes traverse multiple ISPs, leaving providers of content and services dependent on transit ISPs to maintain availability. The Internet protocols do not directly expose much information about an ISP to other ISPs. Further, partly due to business concerns, ISPs do not generally make detailed real-time operational data available to other ISPs. Network operators often attempt to overcome these hurdles by posting to mailing lists dedicated to outages [94], asking others to assess reachability from their locations. This approach is not very fast or scalable. Since I want my systems to be deployable today, I need to develop techniques that accurately locate problems, despite the limited information exposed by existing protocols and ISPs.

Second, any measurement traffic sent by a system to try to locate a failure can be affected by that failure, potentially complicating accurate location. As a simple example, to locate a routing failure between a source and destination, it would be helpful to know the intended route. However, standard tools to measure that route require sending traffic over the route, which will not work if the route is broken. I have to develop approaches that work even in the face of routing failures.

Third, most communication on the Internet is two-way, and most routes are asymmetric, with the path from the source to the destination differing from the return path. My results show that this asymmetry means that the two directions often fail independently. However, existing tools do not provide visibility into the reverse path. This lack of visibility hinders network operators when troubleshooting [122]. Existing systems either assume symmetric paths [140, 25] or only monitor paths between nodes in a testbed [4, 33], allowing for unidirectional measurements both ways but limiting their coverage. Together, these facts imply that I need to develop techniques to accurately locate outages along either the forward or reverse path to arbitrary destinations, even given asymmetric routes and failures.

1.1.4 Automatic and Effective Route Repair

By addressing the above goal of accurately locating failures, I could provide an operator with valuable information to help address outages affecting the operator's ISP. With this data, operators could likely repair problems faster than they can today. However, remediation efforts would still happen over human timescales, slowing recovery and suggesting the benefit of an automated response. Even more problematic, however, is the fact that the failure could be in some ISP outside the operator's control. Under the existing Internet protocols, the operator has very limited ability to control how other ISPs route. Therefore, the operator would have to engage the cooperation of some person at the other ISP, which requires that the operator knows who to contact and that the person is "clueful." Such an approach is unlikely to produce a quick resolution.

A technique that used accurate failure location information to automatically repair failing routes could drastically improve availability. To develop a solution that works today, I will need to find protocol-compliant ways to repair the route without requiring the intervention of anyone at the ISP containing the problem. Other research proposed automated repair, but in ways that require protocol modifications that prevent easy adoption [68, 133, 71]. Because I propose rerouting traffic in response to a failure, I need to be sure to do so in a way that does not disrupt working routes. Finally, since traffic will avoid the location of the failure after the reroute, I will need a separate means to test when the failure is resolved.

1.1.5 Summary

In summary, the above set of challenges translates into my goals for addressing Internet performance and availability problems: a technique to measure reverse paths back from arbitrary destinations; an Internet-scale characterization of long-lasting outages; an accurate technique for locating failures, given available vantage points and path asymmetry; and an effective way to automatically repair failed routes in a protocol-compliant fashion, without control of the ISP causing the failure. Achieving the first goal solves a fundamental problem hindering current efforts to improve performance, and the final three goals complement each other to provide a strategy for addressing unavailability.

1.2 Thesis and Contributions

I address these goals in support of the following thesis: *It is possible to build effective systems that can help service providers improve Internet availability and performance and that are deployable on today's Internet.*

The work presented in this dissertation makes the following contributions:

The design and implementation of reverse traceroute. I develop reverse traceroute, a distributed system capable of measuring the reverse paths back from arbitrary destinations. Traceroute is the most widely used Internet diagnostic tool today. However, it has a fundamental limitation that restricts its usefulness: it does not provide reverse path information. I address this longstanding limitation by building a reverse traceroute system that works on today's Internet using only available vantage points and existing protocols. My system uses distributed vantage points and a variety of novel measurement techniques to incrementally piece together the path from the destination back to the source.

I demonstrate that the information provided by reverse traceroute can help debug circuitous paths. I also show that it can measure backbone link latencies and can discover hard to measure portions of the Internet's topology. For these applications, I demonstrate that results improve substantially when I supplement the traditional use of traceroute by using reverse traceroute to measure reverse paths.

Measurement studies characterizing long-lasting Internet outages. I conduct three studies and demonstrate that: (1) both partial and complete outages are common, with some problems lasting for hours or even days; (2) even well-provisioned cloud data centers experience frequent routing problems; and (3) working policy-compliant routes frequently exist around these types of failures. Understanding these properties helps motivate the need for and the design of my techniques for troubleshooting and repairing routing failures.

First, I develop Hubble, a distributed system capable of monitoring long-lasting routing failures in real-time, on an Internet scale. Hubble operates continuously to find Internet reachability problems in which routes exist to a destination but packets are unable to reach the destination. Conducting a three week measurement study using Hubble, I discover over 30,000 outages affecting over 10,000 destination networks. I find that most failures are unidirectional, with the route failing in one direction between two hosts, even though the other direction between them continues to work. I also find that most outages are partial, with some vantage points able to reach a destination that is unreachable from others.

Second, I conducted a measurement study using Amazon EC2, a major cloud provider [3]. EC2 presumably has the resources, business incentive, and best practices available for combating Internet outages. I show that even EC2 data centers experience many Internet connectivity problems and that long-lasting routing problems contribute much of the measured unavailability. To the extent of my knowledge, this study is the first public one of its kind. That Internet outages affect even well-connected cloud services argues for a fresh approach to addressing these problems.

Third, I also conducted a complementary study, this time monitoring paths between hosts at academic institutions across the Internet [101]. I show that, during many outages, we can observe what appear to be working, valid alternate paths between the hosts experiencing the failure. This result suggests the potential for systems to locate a failure and encourage traffic to reroute around it.

A system for: (1) isolating the location of routing failures and (2) automatically rerouting paths around the failures. I develop a system *LIFEGUARD* that has two major components. First, *LIFEGUARD* uses a set of active measurement techniques that build on Hubble and reverse traceroute in order to locate failures much more accurately than was previously possible. It isolates wide-area Internet faults at a router-level granularity, yet only requires control over one of the end-points of the failing path. The system first determines whether the outage is on the forward and/or the reverse path. Next, *LIFEGUARD* uses a combination of historical and on-demand measurements to locate the failure. The historical information allows reasoning about what changed to cause the failure and helps work around the fact that traditional measurement tools do not work during failures.

Second, as part of *LIFEGUARD*, I develop a protocol-compliant technique to allow data centers and well-provisioned edge ISPs to repair persistent routing problems on routes to them, even if they do not control the ISP causing the outage. My approach repurposes a feature of BGP in order to essentially hint to the culpable ISP that it should not make its route towards the data center available to neighboring ISPs. With the path through the culpable ISP unavailable, other ISPs will have to explore other routes that avoid it. I propose a number of techniques to make this approach practical. I also suggest how more explicit support for these hints could be added into BGP.

1.3 Organization

In Chapter 2, I provide background on the Internet, on tools operators currently use to troubleshoot performance and availability problems, and on why these tools do not suffice to address these problems. In Chapter 3, I present results of measurement studies characterizing failures across the Internet. Chapter 4 presents my reverse traceroute system and shows how it can be used to diagnose a performance problem. Chapter 5 describes *LIFEGUARD*, my approach to locating and avoiding failures. In Chapter 6, I provide an overview of related work. Finally, in Chapter 7, I summarize my work and discuss future directions.

Chapter 2

BACKGROUND

In this chapter, I discuss Internet routing problems that cause poor performance (2.4) and unavailability (2.5). Those sections describe what is known about these problems, their frequency, their impact, their causes, and the approaches ISPs can use to address them. The sections also describe how limitations of current approaches stifle ISPs' attempts to deliver good performance and high availability. Essentially, the performance and availability experienced by Internet traffic depend on the route the traffic follows. The problems of interest in this dissertation are routing problems: Internet performance suffers when traffic follows a much longer (and hence slower) route than is necessary, and Internet availability suffers when traffic is sent along a path that fails to deliver it to the destination. Operators struggle to maintain good performance and availability because they have limited visibility into and even less control over the routing of other ISPs on which they depend. My dissertation is devoted to performing studies to fill gaps in our knowledge about these problems; developing tools to give network operators visibility they lack; and building systems to automatically address some of the problems. In order to understand the problems and my solutions, it is necessary to understand Internet routing and tools to understand and improve it. Therefore, before presenting details on the problems and my solutions, I provide background on the Internet's topology (2.1), its routing (2.2), and tools to understand routing (2.3), focusing on the widely used traceroute tool and its limitations.

2.1 Internet Topology and Terminology

The Internet consists of end-hosts and routers (collectively, hosts or nodes), interconnected by links. The essential purpose of the Internet is to enable communication between end-hosts. Barring when

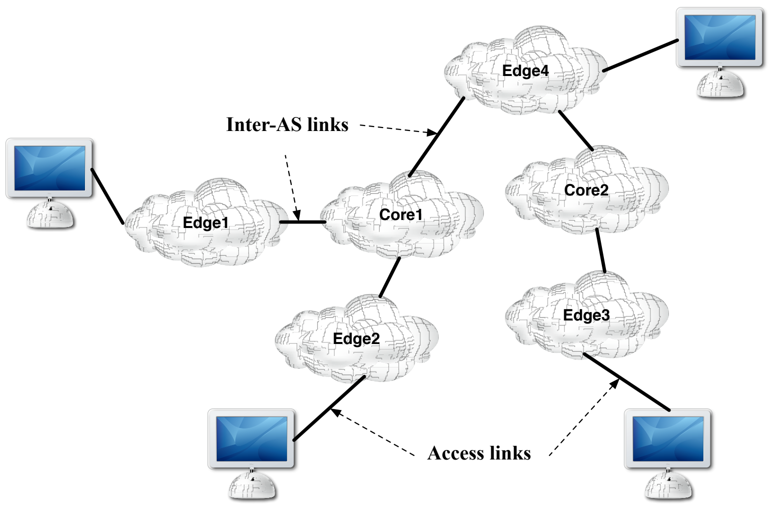


Figure 2.1: The Internet consists of a set of inter-connected autonomous systems (ASes).

policy filters certain communication to a particular end-host, the Internet should provide connectiviy between any pair of hosts.

Organizations known as autonomous systems (ASes) provide this global connectivity. An AS is a collection of routers and networks that presents a common routing view to the rest of the Internet. Only large organizations are ASes. Smaller organizations contract with one or more ASes to provide their Internet service, without becoming ASes themselves. Each AS generally has independent and complete control of its own domain, but each AS has only a limited view inside neighboring ASes. An implication is that we can often treat each AS as a single entity, as in Figure 2.1's high-level depiction of the Internet's topology. End-hosts connect to ASes at the Internet's edge, in order to have access to the rest of the Internet. The ASes at the edge are known as edge or stub ASes. The University of Washington and Comcast are example edge ASes. ASes that connect ASes to each other are called core or transit ASes. Level 3 and Sprint are examples of large transit ASes. Overall, the Internet comprises tens of thousands of ASes. For my purposes, an AS is equivalent to an Internet service provider (ISP), and I will also often use the term "network" interchangeably. I refer to ASes such as Google that offer services that clients access via the Internet as service providers or content providers. ASes inter-connect as part of business agreements. ASes without global reach generally pay provider ASes to connect them to the rest of the Internet (a customer-

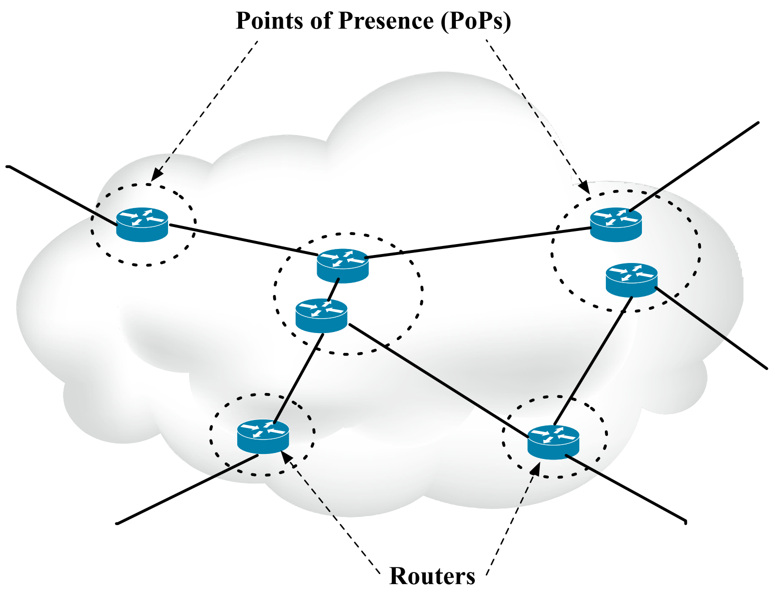


Figure 2.2: An AS consists of routers connected by links. An AS typically has routers in mulitple locations, each referred to as a Point of Presence (PoP) of the AS.

provider relationship). Many pairs of ASes also have peering (or peer-to-peer) relationships, in which they agree to freely exchange traffic.[1] The term "peering" is overloaded, as it refers both to a particular business relationship and to the connection points between adjacent ASes, regardless of relationship. The use should be clear from context. An AS with multiple providers is said to be multi-homed, whereas one with a single provider is single-homed.

The Internet Protocol (IP) is a data packet format for addressing hosts on the Internet. It facilitates communication across network boundaries. Each publicly available host on the Internet has one or more IP addresses, which allow other hosts to address packets to it. Unless otherwise specified, in this dissertation I mean IPv4 when I talk about IP. IPv4 is currently the dominant version of IP, although IPv6 is increasing in importance. An IPv4 address is a 32-bit identifier. The Domain Name System (DNS) provides a mapping from host names to IP addresses, although some IP addresses do not have host names. End-hosts use DNS to discover the IP addresses of services and websites; this process is known as DNS resolution. Major content providers like Google replicate

[1]Paid peering agreements also exist, as do so-called "sibling" relationships. For the purposes of this dissertation, the former are equivalent to unpaid peering connections, and sibling ASes can be thought of as a single entity.

their services and content at data centers distributed around the world. They then use DNS to assign clients to data centers, by resolving the hostname of the service to an IP address at the desired data center. Hosts can use IP to send traffic of various other protocols. The Internet Control Message Protocol (ICMP) is one such protocol. It defines a number of control messages, many of which relay error or diagnostic messages. The Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are protocols commonly used to deliver application traffic.

Figure 2.2 presents the structure of a typical AS. The AS is made up of routers, with links connecting the routers. An AS typically has routers at different geographic locations, and a Point of Presence (PoP) is a location in which an AS has routers. Each router has multiple interfaces, and each link connects a pair of router interfaces on different routers. A router's role is to forward a packet it receives on one of its interfaces onto the link connected to another of its interfaces, in order to send the packet to the router on the other end of the link.

We refer to the multiple interfaces of a particular router as aliases of each other or of the router. Existing work provides techniques and datasets that identify IP aliases of routers, some of which I use in this dissertation to identify cases in which two measurements revealed different addresses belonging to the same router. The process of mapping from IP addresses to routers is known as alias resolution. Most techniques for alias resolution fall into one of two categories. Some send probes to a single destination in ways that reveals multiple IP addresses for the target router [45, 115]. Others send probes to sets of IP addresses to "fingerprint" each address, then classify two addresses as aliases if they have the same fingerprint [114, 11, 62]. Some systems combine multiple techniques to improve coverage and accuracy [78].

The Internet's rich connectivity means that any two end-hosts almost always have multiple paths between them. There are often many AS-level paths between them. Two neighboring ASes can connect with each other at multiple PoPs, creating multiple PoP-level paths that represent the same AS-level path. Within an individual AS, any two PoPs typically have several paths between them [125].

2.2 Internet Routing

Routing determines which path traffic will follow from one host to another. With a few caveats I discuss below, a router generally determines the link on which to forward traffic based only on the destination, sending the traffic on to the next hop in the established route. Internet routers run two types of routing protocols to establish these next hops for each destination, inter-AS and intra-AS. Inter-AS routing is also referred to as wide-area routing, in contrast to the local routing within a single AS. The communication exchanged between routers as part of routing protocols is referred to as the control plane, in constrast to the data plane, which forwards end-host traffic via the routes established on the control plane

The Border Gateway Protocol (BGP) is the Internet's inter-domain routing protocol, used by ASes to establish routes across multiple ASes. In BGP, routes are at the AS level, for two reasons. First, since ASes are autonomous domains, often with competing business interests, the protocol allows them to avoid revealing much about their internal structure or routing. Second, this AS-level granularity improves scaling by allowing ASes to make routing decisions in isolation from the internal routing decisions of other ASes.

At a high level, BGP works by having each AS announce the route it is using for each destination to its neighbors, and every AS is free to choose amongst all of the routes it learns for a particular destination. BGP employs policy-based routing, meaning that an AS is free to apply whatever policy it prefers to select from the routes it learns from its neighbors. The selected route establishes which neighbor serves as the next hop AS towards that particular destination. These policy-based decisions are in contrast to, for example, decisions based exclusively on performance metrics, such as shortest path. ASes generally base their policies on business concerns. In particular, they tend to prefer paths through their customers (for which they can charge), then paths through their peers (which are free), and lastly paths through their providers (for which they pay) [42]. ASes often choose between equally preferred paths by using AS path length as a tiebreaker. Once an AS selects a path to a destination, it prepends itself to the AS path it learned from the chosen neighbor, then announces this modified AS path to some, none, or all of its neighbors. It selects which neighbors based on its own export policy. Typically, an AS also bases these policies on business concerns. In particular, the "valley-free" policy says that an AS will announce routes from its customers to all neighbors, but will only announce routes from peers or providers to its customers [42]. This standard policy ensures that an AS will carry traffic unless it is to/from it or one of its customers (that pay it for the traffic). Note that an AS only ever exports its most preferred route for a destination, the one it is using. It does not announce less preferred routes to its neighbors. Routes propagate in the opposite direction from the traffic they carry. The route that will carry traffic from *S* to *D* starts with *D* announcing itself to its neighbors.

Because an individual AS has redundant routes across it and adjacent ASes often connect at multiple peering point, BGP's AS paths do not completely determine how routers will forward packets from a source to a destination. The route must also be specified at the router level. To achieve this, each AS runs an Interior Gateway Protocol (IGP) to determine the route from each of its router to the next hop AS for every destination. These routing protocols set up each router's forwarding tables which contain, for every destination, the address of the next hop router. IGPs often compute these forwarding tables to minimize the latency incurred by packets traversing the AS and/or to balance the expected traffic across links in the AS's network. OSPF and IS-IS are two common IGPs (their details are not important for this dissertation).

In my description of routing thus far, I used three simplifications that I now explain. First, in the simplified description above, each AS chooses a single path to a particular destination, but, in reality, an AS can choose different AS paths from different parts of its network. Often, an AS connects to different sets of ASes at different edge routers. In some ASes, each router in the AS chooses a path to the closest edge router which learned a path to the destination from a neighboring AS. Figure 2.3 provides an example. Under a policy known as hot-potato or early-exit routing, each router in an AS that peers with the next-hop AS in multiple locations will choose to route to the nearest location.

Second, I presented routes as being per-destination. In reality, Internet routing groups IP addresses into address blocks called IP prefixes. A prefix comprises all IP addresses with p as their first n bits, typically written as p/n. Routing tables contain per-prefix routes. When a router receives

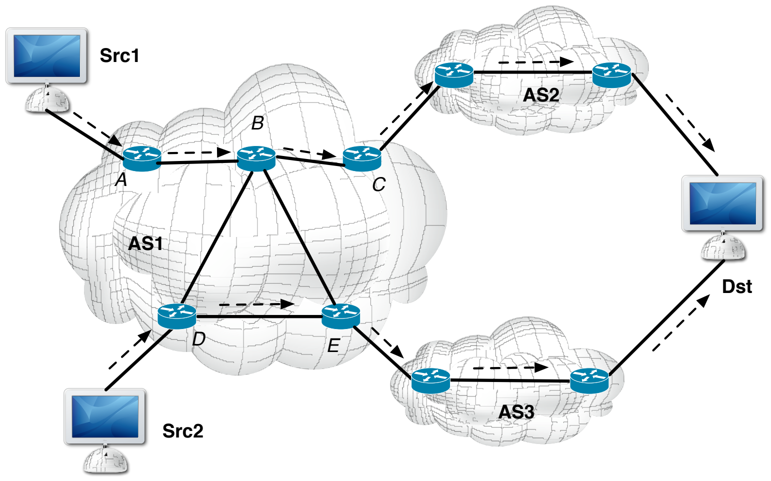


Figure 2.3: Example routes selected from *Src1* and *Src2* to *Dst*, when routers route to the nearest egress. Solid lines show links between routers, and dashed lines indicate the links followed by packets destined to *Dst*. Ingress router *A* is closer to egress router *C* than to egress router *E*, and so traffic from *Src1* to *Dst* follows the AS path *AS1-AS2*. Ingress router *D* is closer to *E*, cuasing traffic from *Src2* to follow the AS path *AS1-AS3*.

a packet destined to a particular destination, it determines the most-specific prefix that this destination belongs to, i.e., among the p/n's that the destination address matches, the one with maximum value of n. This prefix is also referred to as the longest-matching prefix. The router then forwards the packet along the route associated with this prefix. In this dissertation, I generally mean this longest-matching prefix when I refer to the prefix of an address. It should be evident from context if I mean a different prefix. IP prefixes are allocated to ASes, which then announce single-AS paths to those prefixes. The first AS on an AS path to the prefix is known as the origin AS for the prefix.

Third, while generally each router forwards a packet based on the destination prefix, some ASes do not use this destination-based routing internally. Instead, they employ tunnels internally, in which the path from the start to the end of the tunnel is predetermined, with routers in the middle forwarding along the tunnel rather than based on the destination. Multiprotocol Label Switching (MPLS) is a popular tunneling protocol. Similarly, some ASes load-balance traffic across multiple paths [6].

Routes on the Internet are frequently asymmetric, with the path between two hosts differing in the two directions [51]. They can be asymmetric at the router level and even at the AS level, as the asymmetry between forward and reverse paths stems from multiple causes. An AS is free to use arbitrary, opaque policy to choose its next hop among the alternatives, whether or not that leads to a symmetric route. Two adjacent ASes may use different peering points in the two directions due to policies such as early-exit/hot-potato routing. Even within an individual AS, traffic engineering objectives may lead to different paths.

2.2.1 Mechanisms for BGP Route Control

The origin only controls how it announces its prefixes. Once it decides how to announce the prefixes to its providers, it has no direct influence over the paths other networks select. The origin can attempt to influence the paths by modifying the parameters of the announcements, perhaps setting them differently in its announcements to different providers. I now briefly describe some specific BGP mechanisms that the destination can use to attempt to influence paths.

MEDs: An AS that connects to a neighboring network at multiple points can use Multi-Exit Discriminators (MEDs) to express to the neighbor on which peering point it prefers to receive traffic, giving the AS some control of incoming traffic. However, the MEDs only have meaning within the context of that single neighbor, so they generally only shift where that immediate neighbor delivers traffic.

Selective advertising: The origin can announce the prefix through only some providers, shifting traffic away from other providers it wants to avoid.

Advertising a more-specific prefix: By announcing a more-specific (longer) sub-prefix to one provider *P1* and only the less-specific super-prefix to a second provider *P2*, the origin can cause all traffic to the sub-prefix to arrive via *P1*. This technique is a form of selective advertising.

Prepending: Since many ASes use path length as a tiebreaker when making routing decisions, networks sometimes prepend routes they announce with multiple copies of their AS, in order to make that path longer and hence less preferred than shorter ones. For example, an AS can prepend when announcing a route to *P2* and not prepend to *P1*. This prepending may cause some networks to shift to using *P1*. Prepending can sometimes help with inbound route control. However, ASes are free to apply their own local preferences and to ignore path length as a selection criterion.

BGP communities: An AS can define a BGP community, which allows other ASes to tag that community onto routes they announce. Generally, an AS intends for it communities to provide a communication channel from its immediate customers and peers to it. The community instructs the AS on how to handle the routes. For example, Cogent defines a community that others can use to instruct Cogent to not forward the route to its peers. Because ASes are free to define communities as they wish, communities potentially provide an extremely powerful and flexible means to communicate information across network boundaries and to influence how other networks route. However, this flexibility also leads to some of the problems with communities: they are not standardized, meaning that it can be tricky to obtain consistent behavior across multiple providers; some ASes may not give enough, or even any, control over how they propagate routes; and communities are often not transitive, with many ASes not propagating community values they receive on to their neighbors [103].

2.3 Tools to Understand Routing

Researchers and network operators use a variety of tools to measure, assess, and understand routing. Given privileged access to a network, one can inspect routing tables or even information about traffic flows. However, in this dissertation I focus on wide-area routing problems, when such access is not generally available. In this section, I describe the most common techniques one can use to measure arbitrary Internet routes in various ways.

2.3.1 Tools Other than Traceroute

Route collectors and Looking Glass servers: Since BGP is the Internet's inter-domain routing protocol, a natural way to understand routing is to inspect BGP paths, routing tables, and updates. While a router's routing information is normally only accessible to administrators of the router's network, route collectors and Looking Glass servers provide two ways to access such information. Route collectors are computers that peer with BGP routers from a number of networks and publish the routes and updates they receive, for use by researchers and operators. RouteViews [87] and RIPE's Routing Information Service (RIS) [105] each make a number of route collectors available. Looking Glass servers serve a similar function; essentially, if an AS makes a Looking Glass server available, it allows a view into the routing tables of that AS's routers. These sources of data can be extremely useful. The main limitations are that they publish BGP routes, so contain AS-level paths without router-level details, and that only a small number of ASes participate, so the coverage is limited.

Ping: A tool called ping tests whether a destination is reachable and responsive [90]. The tool sends an ICMP Echo Request packet to a specified destination. If the packet reaches the destination and the destination is configured to respond to Echo Requests, the destination replies back to the source with an ICMP Echo Reply packet. The tool reports whether it received a response and also how much time elapsed from when it sent the request to when it received the response. If it receives a response, this result indicates that paths to and from the destination worked, and the elapsed time gives the round-trip time (RTT). The lack of a response does not definitively indicate a lack of working routes, as the destination could be configured not to respond to ping or to rate limit its responses. As is common, I will use the word "ping" in multiple ways: to refer to the tool ("use ping to check if . . . "), the packet sent by the tool ("a ping shows that . . . ") and the use of the tool ("the source pings the destination"); the meaning should be clear in context.

2.3.2 Traceroute

A widely used tool called traceroute measures the path from the source to a specified destination [59]. All IP packets have a Time-To-Live (TTL) field. The source initializes the TTL to an integer value, and every router is supposed to decrement the TTL by one before it forwards the packet. When a router receives a packet with a TTL value of one, it discards the packet and sends an ICMP Time Exceded message to the source. The protocol designers intended the TTL field to allow routers to discard packets that are stuck in persistent loops. Repurposing the feature, traceroute first sends a packet to the destination with $TTL = 1$. The first router discards the packet and sends an error message to the source. The error contains the IP address of the router, allowing the source to discover the start of the path. The source repeats this process, increasing the starting TTL value by one each time in order to discover the next router on the path. The traceroute stops either when it receives a response from the destination or when it receives an indication that the destination is unreachable (either an error message or no responses to a configurable number of probes in a row – typically three or five). In this manner, traceroute measures the router-level path to a destination. Like ping, traceroute times the elapsed time before receiving each response, to measure the round-trip latency for each hop. Different implementations of traceroute send different types of packets. In this dissertation, I use ICMP Echo Request (ping) packets, as they tend to elicit the most responses [74]. As with "ping," I use "traceroute" in multiple ways, but the use should be obvious from the context.

The path returned by traceroute is a feasible, but possibly inaccurate route. First, each hop comes from a response to a different probe packet, and the different probes may take different paths for reasons including contemporaneous routing changes or load balancing. The Paris traceroute customizes probe packets to provide consistent results across flow-based load balancers, as well as to systematically explore the load-balancing options [6]. For all my traceroutes, I use the Paris option that measures a single consistent path. Second, some routers on the path may not respond. For example, some routers may be configured to rate-limit responses or to not respond at all. Third, probe traffic may be treated differently than data traffic.

Despite these caveats, traceroute has proved to be extremely useful. Essential to traceroute's utility is its universality, in that it does not require anything of the destination other than an ability to respond to probe packets.

Some networks make available public traceroute servers within their network. These servers allow anyone to issue traceroutes from the machines (generally via a Web interface) and view the results. In many cases, Looking Glass servers also function as public traceroute servers. Only a few hundred ASes (of tens of thousands total) currently offer public traceroute servers.

PlanetLab is a testbed that allows researchers access to computers at hundreds of sites around the world [101]. While researchers use PlanetLab for a range of purposes, in my research, I typically use these hosts to issue ping, traceroute, and other similar measurements from sites distributed around the world.

2.3.3 Common Uses and Limitations of Traceroute

Table 2.1 shows an example traceroute. I highlight some of the information that traceroute provides, example uses of this information, and limitations with it:

Path: Traceroute provides hops along the path from the source to the destination. This information is used to discover and understand Internet routing [122,78,83,131,61,35]. However, traceroute has a fundamental limitation. Most Internet communication is two-way, and many paths are asymmetric, but (without control of both endpoints) traceroute can measure only the forward path. First, the destination resets the TTL to a normal value. Second, even if the TTL value expires, the ICMP Time Exceded responses will go to the destination.

In Section 2.4, I will explain how this limitation restricts the ability of content providers to address performance problems. Faced with this shortcoming with the traceroute tool, operators and researchers turn to various limited workarounds. Network operators often resort to posting problems on operator mailing lists asking others to issue traceroutes to help diagnosis [94, 91].

Table 2.1: Traceroute from University of Washington towards *18.0.0.1*, which is unreachable/unresponsive.

	Traceroute from zooter.cs.washington.edu to 18.0.0.1	
Hop #	DNS Name / IP Address	RTT
1	acar-atg-02-vlan77.cac.washington.edu	0ms
2	vl3855.uwcr-atg-01.infra.washington.edu	0ms
3	uwcr-atg-01-vlan1889.cac.washington.edu	1ms
4	vl1850.uwbr-chb-01.infra.washington.edu	1ms
5	ge-2-0-0–4013.iccr-sttlwa01-03.infra.pnw-gigapop.net	1ms
6	iccr-sttlwa01-02-ge-0-2-0–0.infra.pnw-gigapop.net	1ms
7	nlr-packetnet.trans.pnw-gigapop.net	2ms
8	losa-seat-49.layer3.nlr.net	28ms
9	hous-losa-87.layer3.nlr.net	59ms
10	atla-hous-70.layer3.nlr.net	84ms
11	wash-atla-64.layer3.nlr.net	97ms
12	newy-wash-98.layer3.nlr.net	103ms
13	216.24.184.102	103ms
14	*	*
15	*	*
16	*	*

It also forces many systems to assume symmetric paths [78, 131, 46, 83, 140], even though most paths are asymmetric [51] due to policy routing and traffic engineering. Public web-accessible traceroute servers hosted at various locations around the world provide some help, but their numbers are limited. Without a server in every network, one cannot know whether any of those available have a path similar to the one of interest. Further, they are not intended for the heavy load incurred by regular monitoring. A few modern systems attempt to deploy traceroute clients on end-user systems around the world [111, 26], but none of them are close to allowing an arbitrary user to trigger an on-demand traceroute towards the user from anywhere in the world. Another approach is to use loose source routing, in which the sender specifies waypoints through which the packet should be routed, to measure paths from a host without controlling it [95, 45, 23]. However, in 2000, the Mercator study found that only 8% of routers supported source routing [45], and a 2009 study found that most or all source-routed packets are blocked [100, 8]. Further, sending source-routed packets generates many complaints [45, 23].

Latency: Traceroute measures the round-trip time (RTT) to each hop. While some systems need RTT values [106, 30], one-way link latencies (for both forward and reverse path links) would provide strictly more information. Techniques for geolocation [131, 61], latency estimation [78], and ISP comparisons [83], among others, depend on link latency measurements obtained by subtracting the RTT to either endpoint, then halving the difference (possibly with a filter for obviously wrong values [78, 83]). This technique should yield fairly accurate values if the link is traversed symmetrically, but previous work found that 88-98% of paths are asymmetric [51] and that delay asymmetry often accompanies path asymmetry [96], resulting in errors in link latency estimates [122].

A few other alternatives exist for estimating link latencies but none are satisfactory. IPMP is a protocol that allows better delay measurements [75], but it requires changing routers to support it. Rocketfuel annotates links with their link weights used in routing decisions [81], which may or may not reflect latencies. The geographic locations of routers provide an estimate of link latency, but geolocation information may be missing, wrong, or outdated, and latency does not always correspond closely to geographic distance [61]. Tomographic approaches base delay estimates on shared portions of paths [28, 127, 102], but lack the reverse path measurements necessary to accurately measure to arbitrary destinations. Other techniques assume either symmetric paths [112] or require control of both endpoints [49] and clock synchronization [110, 20]. These assumptions keep the techniques from being broadly applicable.

Topology: Researchers use traceroutes to measure the Internet topology at the IP [5, 74], router [115, 119], Point of Presence (PoP) [78], and AS level [5, 52, 113]. From the traceroute in Table 2.1, for example, one can infer that NLR's Los Angeles PoP connects to its Houston PoP, and that NLR connects to PNW Gigapop. In addition to the issues that affect traceroute path measurements, problems arise using it to map the Internet. Traceroutes issued from a small number of vantage points (relative to the number of networks in the Internet) miss any links not used on forward paths from those vantage points and, in particular, cannot see most peering links [93, 52]. BGP route collectors also have this problem, since they too have vantage points in only a small fraction of the ASes lower in the AS hierarchy. Testbeds that seek to recruit participants in many networks potentially avoid this limitation [111, 26]; however, no deployments seem likely to allow arbitrary researchers to issue on-demand measurements soon.

Failures: Traceroute also determines if the destination is responsive and reachable. In Table 2.1, the trailing '*' hops indicate that the destination is unresponsive, that the source does not have a working path to the destination, or that the destination does not have a working path to the source. In Section 2.5, I discuss the use and limitations of traceroute in diagnosing availability problems.

Despite these caveats, traceroute has proved to be extremely useful. Essential to traceroute's utility is its universality, in that it does not require anything of the destination other than an ability to respond to probe packets.

2.4 Performance Problems

In this section, I discuss what is known about performance problems on the Internet. Many Internet routes are slow, following circuitous paths even though more direct routes exist. Network operators generally try to optimize performance, but available tools do not provide a full picture of wide-area routing. In particular, while traceroute gives good visibility into the forward path, it cannot measure the reverse path. In order to improve performance, operators need a way to measure the reverse paths taken to reach their ASes.

Many clients suffer from slow performance when accessing major websites. As described in Chapter 1, studies from major providers including Yahoo! [123], Google [18], and Amazon [73] establish that, when clients experience slow web performance, they use the services less. In fact, as clients continue to encounter degraded performance, they continue to use a service less and less, and this reduced usage persists even after performance returns to previous levels [18]. Coupled with the enourmous revenues these companies generate through advertisements and sales to clients, providers have a financial inducement to provide fast service. Various components contribute to a client's latency, including the speed of the client's access link, queueing delays along the path, propagation delays of the links on the path, transmission delays at routers, processing delay at the server, and the fact that a single transaction may take multiple round-trips between the client and server. Researchers found that the bulk of the latency comes from the round-trip network latency between client and server [50], and that is what I focus on in this dissertation.

Despite it being in their financial interest to serve clients over low latency paths, a Google study found that many clients suffer high latencies [66]. Measuring round-trip network latencies on connections between Google servers and clients, the study found that 40% of connections had a network latency of at least 400ms, sufficient for packets to circle the equator twice.[2] The connections incur this delay despite Google's efforts to provide fast service. To improve performance to clients spread around the world, providers like Google replicate their services at distributed data centers, then direct each client to the one with the lowest network latency to the client. In fact, Google's data centers are well-distributed enough that 75% of client prefixes have a server within 1000 miles (which would frequently yield round-trip latencies of less than 50ms [46]) and half of prefixes have a server within 500 miles. Further, the study found that the vast majority of clients were being served by geographically nearby data centers, indicating that the latency was due to indirect routes and/or queueing.

Faster routes exist, but networks need better tools to find them. The Google study and other research establish that, in many cases, indirect routes lead to long end-to-end delays, even though more direct paths exist. One study found that 10% of paths incurred at least 15ms of additional latency, as compared to a theoretical direct path between the endpoints [118]. Most other available studies found even worse inflation. Comparing to the minimum latency from a Google server to a client's region, more than 20% of client prefixes had an inflation of over 50ms [66]. The fact that other prefixes in the same region had much lower latencies to Google's servers suggests the presence of shorter routes through the network. Research explicitly found such routes [106, 41, 4]. A typical technique for identifying these routes is to measure pair-wise delays between a set of nodes and identify violations of the triangle inequality. In such violations, the delay measured between a pair

[2]The speed of light in fiber is approximately 200km per ms, and the circumference of the equator is just over 40,000km.

of nodes is more than the delay of the indirect path between them composed by detouring through a third node. Investigating five different data sets, one study found that, for 10% of paths, they could improve the latency by 50% using such a detour [106]. Other studies report similar results: one found that between 10% and 25% of paths had a detour that was half the latency or less [41], and another found that more than 10% of paths saw latency improvements of 40ms or more on detours [4]. However, while they represent working physical paths, these detour paths generally violate valley-free routing policy, so cannot be selected by BGP or used to carry the bulk of Internet traffic. The commercial offerings that exist [15] must maintain costly detour nodes and pass on the cost to clients.

Evidence suggests that better performing policy-compliant alternate paths also exist but that ASes often lack the tools and visibility necessary to select them. The length of an Internet path depends on the AS path selected, the peering link chosen between each adjacent pair of ASes in the path, and the route used to traverse each individual AS. Two possible causes could inflate any of these three components (as compared to the great circle distance between the endpoints): (1) a lack of a more direct physical path and/or (2) routing decisions that select a less direct route over a more direct one. To determine the contribution of each of these factors, researchers measured the Internet's topology and compared selected paths to available paths that were not selected [118]. They concluded that inter-domain and peering decisions contribute most of the inflation. However, the inflation did not stem from commercially-driven routing policies, such as valley-free or prefer-customer. Instead, most of the inflation resulted from using shortest AS path as the tie breaker. This metric need not correlate with performance, but routers making BGP decisions generally do not have access to actual performance measures. Further, the inflation was not caused by intended routing goals, but rather by a lack of tools to enable the seletion of better paths. They based this conclusion on two facts. First, within individual networks, in which routing engines have access to the complete network topology, 80% of routes were not inflated, suggesting that networks tend to optimize for path length. Second, even though many routes used suboptimal peering links or inter-domain paths, the study also found widespread evidence of cooperation among ASes to arrive at better paths.

To improve poorly performing paths, providers need a way to measure paths from their clients back to their servers. Recent work from Google demonstrates that, given access to information about paths, a provider can improve the performance of its clients [66]. After identifying inflated paths, as described above, the researchers attempted to assign explanations, such as inadequate peering or a misconfiguration. They then gave these explanations to Google network operators. In many cases, once directed to a problem, the operators were able to resolve it, even if it requiring contacting operators at the network at fault. For example, operators were able to reduce the latency to a set of clients in Japan from over 100ms to only a few milliseconds by changing routes. On a larger scale, in just three months, operators were able to nearly halve the number of prefixes which incurred over 50ms of inflation when served by a South American data center.

However, improving inflated paths generally requires pinpointing the cause of the inflation, and current tools do not give adequate visibility into the reverse path. A client of a service incurs the latency of both the path to the server and the path back from the server, and these paths generally differ. Operators at the provider can use traceroute to measure the path to the client, and the results can highlight when an AS selects a longer path than is necessary. Without access to the client, however, the operators do not have equivalent visibility into the other direction of the path. The Google researchers inferred inflated reverse paths in cases where the forward path did not explain the round-trip latency. In fact, even before their interventions improved some forward paths, they concluded that 30% more reverse paths were circuitous than forward paths. However, lacking tools capable of measuring the reverse path, they were unable to point operators to the source of the problems. They cited the lack of information about the path from clients back to Google as the predominant limitation faced in trying to optimize clients' network latencies.

2.5 Availability Problems

In this section, I discuss what is known about availability problems on the Internet. The Internet suffers from many outages. While most are short, a substantial number are long-lasting, and these long-lasting problems contribute a lot of unavailability. Prolonged outages are not well understood. Network operators lack tools to locate them, as standard tools like traceroute reveal only limited information during failures. The decentralized administration of the Internet means that an outage affecting an operator's AS may stem from a problem in another AS. The operator has little visibility into the routing of other ASes and even less control over it. This current state argues for the need for approaches to locating and repairing outages.

The Internet suffers from many long-lasting availability problems. The Internet's availability lags far behind that of the telephone network [67]. Various studies used active monitoring, such as pings, between pairs of nodes in testbeds and found extended data plane outages [97, 4, 37]. In seminal early work, Paxson identified routing loops that persisted for more than 10 hours, as well as other types of unavailability [97]. Researchers using the RON testbed found that 10% of outages lasted at least 15 minutes, with failures appearing at a range of locations (near the edge and in the core, at AS boundaries and within a single AS) [37]. Other studies looked at paths from testbed nodes to destinations across the Internet and found similar results [47, 136]. A study monitoring paths to popular (and presumably well-connected) web servers found over 1000 outages in a week, with the average failure lasting for over 10 minutes [47]. Monitoring paths betweens a distributed system's servers and its clients, PlanetSeer found many prolonged outages, including over a thousand loops per month that lasted over seven hours [136]. My own results, in Chapter 3, show that even paths between data centers of a major cloud provider and core Internet routers suffer from many long-lasting outages. The results show that, to substantially improve availability, we must address these protracted failures.

In Section 3.5, I present evidence that suggests that misconfigurations may cause some long-lasting outages. The Internet has sufficient redundancy to maintain connectivity after a range of failures, and routing protocols are designed to recover and find new routes after link and router failures. Under proper operation, protocols such as BGP should restore connectivity after a brief period of routing convergence [69]. However, many long-lasting outages occur with few or no accompanying routing updates [37]. With routing protocols failing to react, networks continue to send traffic along a path that fails to deliver packets. Such problems can occur, for example, when a router fails to detect an internal fault (e.g., corrupted memory on a line card causing traffic to be black-holed [135]) or when cross-layer interactions cause an MPLS tunnel to fail to deliver packets even though the underlying IP network is operational [63]. Further, routing relies on the proper configuration of multiple devices in multiple networks, running multiple protocols. These devices can be tricky to configure, and currently operators lack tools to let them express network-wide objectives, instead relying on tuning individual devices and protocols [89]. Operators make many errors in configurations [82], some of which result in outages, and such misconfigurations seem especially likely to occur on the rarely exercised backup paths that only come into use following failures.

Current approaches are not able to sufficiently explain or locate the source of these lengthy availability problems. The Internet protocols do not automatically recover from these lengthy problems in a timely manner, suggesting they currently require human intervention. Rich techniques exist to locate failures within a single network, but they require access to proprietary information only available within that particular network [33, 64, 63, 109]. Existing studies found that failures appear both near the edge and in the core [37], and so a failure between, for example, a server and its client could appear anywhere along the path. Unfortunately, the techniques designed to work within a single network do not apply to these sorts of wide-area problems. Operators in the server's and client's networks may be able to detect the failure, as it affects their traffic, but will not have access to the proprietary information of any transit network in the middle. Operators at the transit network may have access to information to localize the failure, but, lacking access to the server and client, they have no way to know what traffic should be flowing through the network.

Since more sophisticated techniques do not apply, operators routinely use traceroute to understand wide-area problems [122], but it provides only limited information. Table 2.2 provides an actual example illustrating traceroute's limitations, taken from emails to the Outages.org mailing list [94] on December 16, 2010. Operator *A* reported that "traffic attempting to pass through Level 3's network in the Washington, D.C., area is getting lost in the abyss" and provided a traceroute issued from his home. As seen on the left of the figure, the route passed through Verizon's Baltimore PoP before eventually reaching Level 3's Washington, D.C. PoP, then trailed off. The trailing "*"s indicate that traceroute did not receive any further responses, and so the operator suggested that the problem might be with Level 3's network in Washington, D.C. However, because paths

Table 2.2: Traceroutes taken from emails sent in response to an outage reported on the Outages.org mailing list on December 16, 2010. While traceroute provides some information about the outage, it is confusing and provides an incomplete view. Operator *A*'s traceroute suggests a problem forwarding from Level 3 in Washington, D.C., but operator *B*'s traceroute successfully goes through Washington, D.C., before terminating in Denver. Further, the failure could be on the reverse path, into which traceroute gives no visibility.

	Operator *A*'s traceroute		**Operator *B*'s traceroute**
Hop #	DNS Name / IP Address	Hop #	Hostname / IP Address
1	wireless_broadband_router.home	1	192.168.1.1
2	1100.**BLTMMD-vfttp-40.verizon-gni.net**	2	1100.**WASHDC-vfttp-47.verizon-gni.net**
3	g10-0-1-440.**BLTMMD-lcr-04.verizon-gni.net**	3	g4-0-1-747.**WASHDC-lcr-07.verizon-gni.net**
4	so-2-0-0-0.**PHIL-bb-rtr2.verizon-gni.net**	4	0.ae1.br1.**IAD8.alter.net**
5	0.ae2.br2.**IAD8.alter.net**	5	ae6.edge1.**WASHINGTONDC.level3.net**
6	ae7.edge1.**WASHINGTONDC4.level3.net**	6	vlan90.csw4.**WASHINGTON1.level3.net**
7	vlan80.csw3.**WASHINGTON1.level3.net**	7	ae-71-71.ebr1.**WASHINGTON1.level3.net**
8	ae-92-92.ebr2.**WASHINGTON1.level3.net**	8	ae-6-6.ebr2.**CHICAGO2.level3.net**
9	*	10	ae-1-100.ebr1.**CHICAGO2.level3.net**
10	*	11	ae-3-3.ebr2.**DENVER1.level3.net**
11	*	12	ge-9-1.hsa1.**DENVER1.level3.net**
		13	*
		14	*
		15	*

are frequently asymmetric and traceroute gives no visibility into the reverse path, traceroute cannot provide enough information to definitively support this conclusion. As my results in Section 3.3.3 will show, path failures are often unidirectional, with one direction between a pair of nodes working while the other is broken. So, *A*'s traceroute could mean that Level 3 was not successfully forwarding traffic to the destination (a broken forward path). Alternatively, the forward path could be working, but the destination and other hops past Washington, D.C., could lack working reverse paths back to *A*. Because it requires bidirectional connectivity from any hops it reports, traceroute cannot differentiate these cases. Even if the forward direction is working, traceroute cannot measure it without a working reverse path, since the replies will not reach the source. Also, from a single location, traceroute gives little sense of how widespread the problem is.

With traceroute yielding such limited information, operators often resort to asking others for help, but their traceroutes may not illuminate the problem. The right side of Table 2.2 displays the traceroute of an operator *B* who replied to *A*'s post to the mailing list. *B*'s traceroute goes through

Level 3's Washington, D.C., PoP before trailing off after Denver. This measurement does not clarify the situation. Has the initial problem shifted to Denver? Are the failures on the reverse path? Do the reverse paths even pass through Level 3? For outages like this one, traceroute, "the number one go-to tool" for operators troubleshooting problems [122], does not suffice.

Researchers have developed various approaches that attempt to locate the source of events by analyzing BGP updates [38, 132, 22, 70, 82], but the control plane may not accurately reflect data plane failures. One study showed that data plane outages often do not manifest on the control plane [37]. Because this disconnect can exist between the control plane and the data plane, and, ultimately, the Internet exists to deliver data, an approach that relies solely on BGP updates cannot provide a complete solution to the problem of locating failures across the Internet.

Existing approaches to restoring connectivity do not suffice. While evidence exists of working routes around failures, existing techniques to use those routes are incompatible with either current routing protocols or routing policies. Similar to how researchers showed the existence of lower latency paths by looking for violations of the triangle inequality, existing research established that a pair of hosts can often route traffic around a failure between them by "detouring" the traffic through a third node. RON was able to route around all failures that lasted more than 30 minutes [4]. In 56% of cases in which a host could not directly fetch documents from popular web servers, a similar approach could successfully fetch the documents by relaying traffic through an intermediary [47]. These approaches provide a great alternative when no other solutions exist, and commercial offerings exist [15]. However, it is expensive to maintain the nodes necessary for detouring, the detour paths generally violate standard routing policy (so using such paths would likely be expensive, if feasible at all), and the paths are unlikely to be capable of carrying much traffic at core Internet data rates. Approaches which overcome these limitations generally require modifications to the core Internet protocols, as with an approach that allows ASes to advertise multiple paths to, among other goals, lend the ability to route around failures [133]. This approach provides a promising direction, but modifications to core protocols face a long path to adoption, meaning that such a solution is unlikely to improve Internet availability in the near future.

Given the constraints of current policies and protocols, operators often have limited ability to address problems. Suppose a failure exists between a network hosting a server and a client trying to access the server. Operators at the server's network would like to restore connectivity. If the problem is within the operators' network, they should have a means to resolve it. However, if the problem is in some transit AS on the path, the operators have very little control. If the problem is on the forward path towards the client, a well-connected network can try reaching the client through a different provider or via a different data center. If the problem is on the reverse path, all the network can do is change how it announces its prefixes, using techniques explained in Section 2.2.1. These techniques provide only minimal, very coarse control. I discuss this limited control in detail in Section 5.1. Essentially, the techniques provide mechanisms to affect which provider and which PoP traffic enters the server's network through, but, for the most part, other ASes will still be free to select whichever path they want, often including the one that is being advertised but is not working. So, in general, with current practices, operators at the AS containing the failure need to address it. Often, operators from the server's network have to contact the transit AS, opening incident tickets or asking on mailing lists for contact information [94, 91]. The prevalence of long-lasting outages suggests that it can take awhile for transit networks to be alerted to and address the problem.

Chapter 3

CHARACTERIZING INTERNET OUTAGES

In this dissertation, my goal is to develop techniques to improve availability and performance on today's Internet. In order to build tools and systems to identify and remediate routing failures and performance problems, it is necessary to understand the problems: how frequently they occur, what their causes are, and other properties.

Fortunately, as I presented in Section 2.4, a rich literature examines performance problems. For example, researchers established that faster, more direct routes frequently exist but that networks lack tools to find them [118]. Research from Google characterized the performance problems that the company faces and highlighted types of problems they can and cannot resolve [66]. These and other studies suggest what networks would need from tools in order to improve performance, and so in Chapter 4 I develop one such tool.

While existing studies characterized performance problems on the Internet and how providers might address them, much less is known about availability problems. Studies mentioned in Section 2.5 assess various types of outages in different settings, but do not provide the complete picture. My goal in this chapter is to supplement these studies by characterizing failures on an Internet scale. To provide direction in how to improve availability, these studies should help answer: whether it is possible to monitor availability at this scale; how common outages are; how many paths experience outages; whether outages generally render a destination completely unreachable, or whether outages are partial; whether, when two hosts cannot communicate, the problem affects the routes in both directions between them; and whether valid routes exist that could restore connectivity to an unreachable destination. In this chapter, I present three measurement studies to characterize Internet outages. The studies establish that many outages occur, suggesting the need for new approaches to improving availability. The characteristics of the outages motivate my approach to improving availability and inform the design of my systems to address them, which I will present in later chapters.

Monitoring outages at Internet-scale with Hubble: To motivate the design of my failure location techniques in Chapter 5, I design and build Hubble, a system that identifies availability problems over the global Internet in real-time. The system monitors for Internet *black holes*, in which traffic silently disappears without reaching the destination. Conducting a study at scale helps insulate the conclusions against any biases introduced by monitoring only a small number of destinations.

Using Hubble, I found availability problems to be more common, widespread, and longer lasting than expected. In a given week, thousands of networks are reachable from some locations and not from others, and many of the problems last for hours or even days. This result both suggests that we need new approaches to improve availability and that these approaches can issue measurements from vantage points with working paths when trying to locate a problem. Using a novel technique to isolate the direction of failures, I find that most failures are unidirectional, with the route in one direction between a source and destination failing even though the other direction works. The existance of unidirectional failures motivates the need to measure reverse paths in order to locate failures.

In addition to presenting these measurement results, in this chapter I describe the design and evaluation of Hubble. The major challenge in building Hubble is that of scale: how can the system provide spatial and temporal coverage that scales to the global Internet, monitoring the data plane from all vantages to all destinations, without requiring a prohibitive number of measurement probes. To identify potential problems, Hubble monitors BGP feeds and continuously pings prefixes across the Internet from distributed PlanetLab sites. It then uses traceroutes and other probes from the sites to collect details about identified problems. I show that it monitors 89% of the Internet's edge prefix address space with a 15-minute time granularity and discovers 85% of the availability issues that would be identified by a heavyweight approach (in which all vantage points traceroute those same prefixes every 15 minutes), while issuing only 5.5% as many probes. I believe Hubble to be the first real-time tool to identify availability problems on this scale across the Internet. The next largest system of which I am aware, PlanetSeer, covered half as many ASes in a 3-month study and monitored paths only for the small fraction of time that they were in use by clients [136]. Hubble ran continuously for two years, starting in September, 2007. During that time, it identifed over 1.7 million black holes and availability problems.

Supplemental studies: I supplement the results from Hubble with two additional studies. One study, in Section 3.4, establishes that even well-provisioned cloud data centers experience frequent routing problems. In the study, I assess the availability of key Internet routers from Amazon's EC2 cloud infrastructure [3]. Hubble monitors from PlanetLab sites, which are mainly located at academic institutions. To complement the Hubble results, my EC2 study finds similar results monitoring from commercial data centers. The data from these studies shows that, even though most outages are short, the small number of long-lasting ones contribute much of the end-to-end unavailability. Techniques exist to address many short outages [60, 68, 71]. However, my Hubble and EC2 results suggest that, to improve availability substantially, we will also need to address long-term outages. Few approaches exist to tackle these problems. Just as with Hubble, I find that most of the outages observed from EC2 are partial, with one EC2 data center able to reach a destination unreachable from another data center. The fact that the destinations are available from some locations suggests the possibility of routing around the failures.

The final study, in Section 3.5, demonstrates that, in fact, working policy-compliant alternate routes often exist between pairs of PlanetLab hosts that are unable to communicate directly. In fact, I show that, in half the cases, an alternate path demonstrably exists that avoids the AS that seems to contain the failure. Since BGP uses AS-level policy routing, this result suggests that BGP may be able to find working alternate paths, if we can discern which AS contains the failure and trigger BGP path exploration that avoids that AS. This observation forms the basis for my technique for route repair, which I present in Chapter 5.

The rest of this chapter is organized as follows. In Section 3.1, I describe the design of Hubble. I present an evaluation of Hubble in Section 3.2 and use it to study Internet availability in Section 3.3. In Sections 3.4 and 3.5 I present the two supplemental studies. I provide a summary in Section 3.6.

3.1 Hubble Design and Architecture

3.1.1 Deriving Hubble's Methodology

In this section, I describe why I want Hubble to characterize wide-area data plane failures across the Internet, and I discuss the design and methodology implications of this goal. At the end of this dissertation, in Section 7.3.1, I discuss how future work might extend this work by overcoming some of the limitations I describe here.

Data plane focus: I desire a system that detects data reachability problems, and so I use active measurements to decide whether a destination is reachable. The Internet is intended to deliver data, and a BGP path on top of a broken data plane is useless. If BGP perfectly reflected the data plane – that is, data could reach a destination if and only if the source had a BGP path to it – then it would suffice to detect problems by monitoring BGP feeds from services such as RouteViews [87] and RIPE RIS [105]. However, other studies found that many data plane failures are not visibile in BGP [37] and that default routes allow some sources to reach destinations for which they lack BGP routes [21]. My own results in Section 3.2 also demonstrate that many data plane outages do not register as BGP updates in route feeds. The only way to absolutely discern whether the data plane is functioning and packets can reach a destination is to monitor traffic towards that destination. A system could achieve this monitoring either by passively monitoring existing traffic or by introducing probe traffic. Other studies achieve interesting results by passively monitoring existing application traffic [27, 136]. One drawback with this approach is that it ties the studies to the particular traffic patterns of the monitored applications, and few applications have enough traffic to enough destinations around the world to provide continuous global coverage. To supplement these studies, I base my results on active probing using ping and traceroute.

Wide-area routing focus: My dissertation focuses on deployable systems that edge ASes can use to repair problems affecting them and their users, and so I focus on wide-area routing problems that the ASes may be able to resolve. This focus has three components.

First, I concern myself only with problems affecting addresses that one would expect to be reachable. For an IP address to be reachable, it needs to be part of a prefix announced by some origin AS. So, I can monitor BGP advertisements and ignore any addresses that are not being announced. Also, even within an announced prefix, a particular address may not be in use, or it may not offer the particular service I am monitoring (in my case, ping). So, I only monitor addresses that I previously observed as reachable via ping and which become reachable again eventually.

Second, I focus on problems in which the destination's AS seems not to be reachable, rather than simply checking whether the particular destination is unreachable. I am primarily interested in problems in which the Internet's routing infrastructure fails to provide connectivity along advertised AS paths, rather than problems in which the traffic traverses the AS path, but the specific host or service happens to be down. As such, I focus on those destinations with reachability problems to the origin AS. Even so, I find many problems, as shown in Section 3.3. I also focus on AS reachability because my approach to problem remediation preserves AS autonomy, and so it would not work if the destination AS is reachable but does not deliver traffic. Other than being down, a destination could instead be unreachable within a reachable origin AS if the AS failed to deliver the traffic. Whereas the source may be able to avoid problems in a transit AS by routing via a path that avoids that AS, in general every path to the destination will have to traverse the destination AS. So, if the destination AS is not delivering traffic to the destination, the source is unlikely to be able to resolve the problem unilaterally. If the destination does not reply to a direct probe, I use traceroutes to reveal the path towards the destination, to discern whether the destination's origin AS is reachable.

Third, I focus on problems that affect more than a few routes, to eliminate source-specific problems. I am concerned with the reachability of destinations on an Internet-scale, rather than problems caused only by issues near one of our sources. In a study of four months of daily traceroutes from 30 PlanetLab vantage points to destinations in 110,000 BGP prefixes, I found that most of the time, all but a few of the 30 traceroutes reached the destination's origin AS. If fewer than 90% reached, though, it was likely the problems were more widespread; in half those cases, at least half of the traceroutes failed to reach. I use this value as my conservative threshold: if at least 90% of probes reach the origin AS of a destination, then I assume that any probes that did not reach may have experienced congestion or problems near the source, and I ignore the issue. By avoiding further investigation of spurious problems caused by flaky PlanetLab hosts, I can limit the measurement load required by my system. I use 90% reachability to define if a destination is experiencing a *reachability problem*. I define a *reachability event* as the period starting when a destination begins to experience a reachability problem and concluding when its reachability increases to 90% or higher.

My methodology may be better at detecting forward path failures from its vantage points to remote targets, than it is at detecting reverse failures on the paths back from the targets. My measurement study in Section 3.3.3 finds more forward failures, but the following aspect of my experiment design could partly explain the result. Because my approach excludes problems that affect fewer than 10% of vantage points and does not join across targets, it would miss a problem that disrupted connectivity between multiple targets and only a single source. I use traceroutes, which do not differentiate the direction of failure, to determine if a target's AS is reachable. Assume a failure is on the forward path towards a target. Because Internet routing is generally destination-based, all vantage points route to the target via the same prefix, with the paths eventually converging on their way to the target. If one vantage point's route experiences a failure on a particular link, presumably other vantage points that route via that link will also experience it. If enough share that link, the failure would cross the 90% threshold. However, the vantage points each belong to different institutions (and hence different prefixes), and so the target's reverse paths to them will route via different destination prefixes, even over shared links. Reverse path failures might affect converging paths from multiple targets back to a vantage point, without affecting other vantage points, but a single vantage point experiencing multiple failures will not excede the threshold on its own. In this and later chapters, I will show how we can partly account for asymmetry and unidirectional failures.

Characterization at a global-scale: In order to enrich understanding of wide-area routing problems impacting the data plane, I want to characterize these problems on a global scale. For each of these problems, I want to determine whether it affects all vantage points or only some, whether it is on the forward path to the destination or the reverse path back, and how long it lasts. I also want to determine the location of the problem and whether alternate working paths exist. I mainly leave these last two determinations to Chapter 5 and discuss the rest in this chapter.

In order to monitor the problems of interest at this scale, I need to use distributed vantage points and limit the aggregate probing rate. If all failures were complete, then identifying potential problems would be easy – a probe from any vantage point could identify it. However, as I will show in this chapter, many outages are partial, and so it might be necessary to probe from many vantage points before identifying a particular problem. As described above, I make determinations of reachability based on responses from routers along the paths to a destination. If not rate limited, the probe packets could overwhelm routers along the paths. This restraint is especially important because I intend to monitor areas of the Internet experiencing problems, and, in doing so, I do not want to exacerbate the problems. So, it is infeasible to monitor all paths to and from all ASes at a high rate.

I use three approaches to limit the probing rate. First, because I am interested in wide-area routing problems, it suffices to monitor at the granularity at which wide-area routing works. BGP is prefix-based, and so I need only monitor a single destination within each BGP-routable prefix. Second, I use a lightweight technique to identify potential failures, trigger more extensive measurements to exclude uninteresting cases (such as those that affect only a single source), and finally use further measurements to characterize the interesting ones. This approach means that the measurements and analysis have to be real-time, so that, when the system detects a potentially interesting problem, it can focus further measurements on it. Third, I focus on detecting persistent issues. I show in Section 3.4 that long problems contribute much of the unavailability, justifying their choice as a priority. Because I am not concerned with transient problems, I can probe at a slower rate. I consider only problems that persist through two rounds of quarter-hourly probes.

3.1.2 Overview of Measurement Components

As depicted in Figure 3.1, Hubble combines multiple types of measurements into four main components to identify and characterize problems: pingable address discovery to decide what to monitor (not shown in figure); active ping monitoring and passive BGP monitoring to identify potential problem prefixes as targets for reachability assessment; triggered traceroutes to assess reachability and monitor reachability problems; and spoofed probes, combined with the same triggered traceroutes, to classify the direction of failures. Unless otherwise noted, Hubble uses PlanetLab sites to perform the active measurements. I now present an overview of each of these measurements, and then elaborate on how the system uses these measurements to monitor and classify reachability problems.

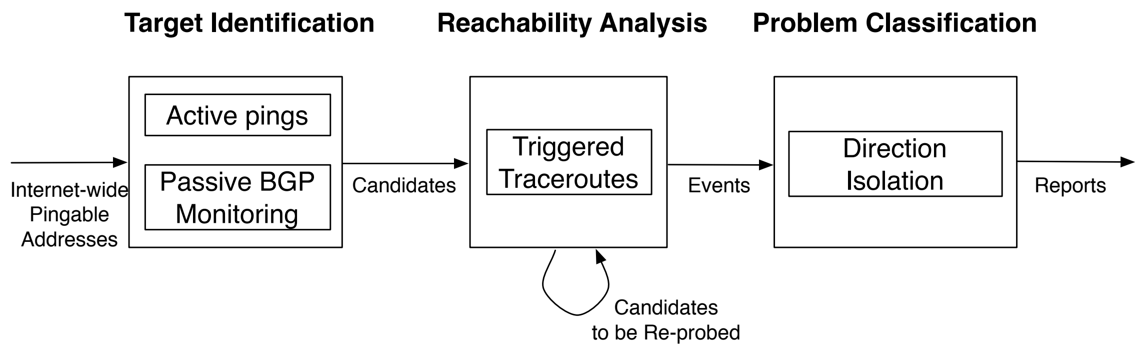


Figure 3.1: Diagram of the Hubble architecture.

Pingable address discovery: Pingable address discovery supplies the set of destinations for monitoring to the active ping monitoring system. It discovers these destinations by probing the .1 in every $/24$ prefix present in a BGP snapshot obtained from RouteViews [87] and retaining those that respond. In the wide-area Internet, $/24$ prefixes tend to be the most specific uniquely routable blocks, because announcements for $/25$ and more specific prefixes are widely filtered. Therefore, probing one target per $/24$ allows Hubble to monitor the unique routes available from its vantage points.

Active ping monitors (details in §3.1.3): Hubble issues pings from vantage points around the world to the pingable addresses. As my results in this chapter will show, many outages are partial, and so it is necessary to send pings to a destination from multiple locations to attempt to discover problems that affect only some vantage points. The system in aggregate probes each address every two minutes. When a vantage point fails to get a response from a previously responsive address, it reports the prefix as a candidate potentially experiencing more widespread reachability problems, resulting in triggered traceroutes to the prefix (§3.1.4). The system needs to send traceroutes because I want to be able to determine whether or not the destination AS is reachable. However, given the limitations of available vantage points, it is not feasible to frequently send traceroutes to an Internet-scale set of targets. iPlane, for example, only refreshes its traceroutes daily [78]. Therefore, Hubble uses lightweight pings to identify candidates for further investigation with more heavyweight traceroutes.

Passive BGP monitor (details in §3.1.3): The system observes BGP updates from multiple locations in quarter-hourly batches to maintain current AS-level routing information. This approach allows continuous monitoring in near real-time of the entire Internet from diverse viewpoints, while providing scalability by gathering information without issuing active probes. Supplementing its ping monitoring, Hubble analyzes the BGP updates and identifies as targets for triggered traceroutes those prefixes undergoing BGP changes, as they may experience related reachability problems. BGP feeds also allow Hubble to monitor prefixes in which no pingable addresses were found, which cannot be monitored by active ping monitors.

Triggered traceroute probes (details in §3.1.4): Every 15 minutes, Hubble issues traceroute probes from distributed vantage points to targets selected as potentially undergoing problems. The system selects three classes of targets: (1) previously reachable addresses that become unreachable, as identified by the active ping monitors; (2) prefixes undergoing recent BGP changes, as identified by the passive BGP monitor; and (3) prefixes found to be experiencing ongoing reachability problems in the previous set of triggered probes.

Spoofed probes (details in §3.1.4): Internet routes are often asymmetric, differing in the forward and reverse direction [51]. A failed traceroute signals that at least one direction is not functioning, but leaves it difficult or impossible to infer which direction is the problem. Hubble employs spoofed probes, in which one monitor sets the source of packets to the IP of another monitor while probing a problem prefix. This technique isolates many problems to either the forward or reverse path.

3.1.3 Identifying Potential Outages

Selective targeting allows the system to monitor the entire Internet by identifying as possible outages only prefixes suspected to be experiencing problems. Hubble uses a hybrid approach, combining active ping monitoring with passive BGP monitoring. If Hubble used only passive BGP monitoring, it would miss any reachability event that did not correlate with BGP updates. As I present later in Section 3.2, BGP is not a good predictor of most problems, but allows Hubble to identify more problems than ping monitoring alone. I now present more details on how the two monitoring subsystems work.

Active Ping Monitoring

To meet the goal of a system with global scale, Hubble employs active monitoring of the reachability of prefixes. Hubble uses traceroute probes to perform its classification of reachability problems. However, it is not feasible to constantly traceroute every prefix in order to detect all problems. On heavily loaded PlanetLab machines, it would take any given vantage point hours to issue a single traceroute to every prefix, and so problems that were only visible from a few vantage points might not be detected in a timely manner or at all.

Hubble's active ping monitoring subsystem achieves the coverage and data plane focus of active probing, while substantially reducing the measurement overhead versus a heavy-weight approach using pervasive traceroutes. If a monitor finds that a destination has become unresponsive, it reports the destination as a target for triggered traceroutes.

I design the ping monitors to discover as many reachability problems as possible, while reducing the number of spurious traceroutes sent to prefixes that are in fact reachable or are experiencing only transient problems. When a particular vantage point finds an address to be unresponsive, it reprobes two minutes later. If the address does not respond six times in a row, the vantage point identifies it as a target for reachability analysis, triggering distributed traceroutes to the prefix. I found that delaying the reprobes for a few minutes eliminates most transient problems, and I conducted a simple measurement study that found that the chance of a response on a seventh probe after none on the first six is less than 0.2%. Hubble sends traceroutes from 30 vantage points to perform its reachability analysis. The number of vantage points is fairly arbitrary but is intended to allow for probes from most geographic regions represented on PlanetLab, without sending excessive packets towards a destination experiencing problems. Hubble issues traceroutes in 15 minute rounds, and so the aggregate traceroute rate to a destination is comparable to the aggregate ping rate to a destination. Because 30 traceroutes to a destination entail around 500 total probe packets, a 0.2% chance of a response to further pings means that (in expectation) it requires fewer packets to trigger traceroutes immediately, rather than sending more pings and only sending traceroutes if the pings fail.

A central controller periodically coordinates the ping monitors, such that, on average, within any two minute period, one vantage point should send an initial ping to each destination, and up to five vantage points may send reprobes to the destination. Once a day, the system examines performance logs, replacing monitors that frequently fall behind or report improbably many or few unresponsive destinations. Hubble thus regularly monitors every destination.

Passive BGP Monitoring

Hubble uses BGP information published by RouteViews [87] to continually monitor BGP routing updates from more than 40 sources. Hubble maintains a BGP snapshot at every point in time by incorporating new updates to its current view. Furthermore, it maintains historical BGP information for use in problem detection and analysis.

Hubble uses BGP updates for a prefix as an indicator of potential reachability problems for that prefix. In some cases, reachability problems trigger BGP updates, such as when BGP explores other paths after a prefix becomes unreachable. In other cases, a misconfigured router can advertise an incorrect BGP path, resulting in BGP updates that precede a reachability problem. Hubble therefore uses BGP updates to generate further targets for traceroutes, complementing its ping tests. Specifically, it selects those prefixes for which the BGP AS path has changed or been withdrawn at multiple vantage points. Hubble uses BGP monitoring to try to discover outages that ping monitoring misses. In particular, because of limited probing capacity at PlanetLab sites and because of a need to limit how often a destination receives probe packets, each PlanetLab vantage point takes multiple hours to progress through the entire target list. Therefore, a short outage that affects only a small number of vantage points may not be discovered by ping monitoring alone.

3.1.4 Real-time Reachability Analysis

Given a list of targets identified by the ping and BGP monitors, Hubble triggers traceroutes and integrates information from up-to-date BGP tables to assess the reachability of the target prefixes.

Triggered Traceroutes

Constantly performing traceroutes to every prefix is both inefficient and impractical. However, we do not want to sacrifice the level of detail exposed by traceroutes regarding actual routing behavior in the Internet, especially since such detail can then be used to localize the problem. Hubble strikes a balance by using triggered traceroutes to target prefixes identified by either the passive BGP monitor or the active ping monitors, plus prefixes known to be experiencing ongoing reachability problems. So, as long as a routing problem visible from the PlanetLab vantage points persists, Hubble will continually reprobe the destination prefix to monitor its reachability status.

Every 15 minutes, Hubble triggers traceroutes to the destinations on the target list from 30 PlanetLab nodes distributed across the globe. I limit these measurements to only a subset of PlanetLab nodes. Traceroutes from over 200 PlanetLab hosts within a short time span might be interpreted by the target end-hosts as denial of service (DoS) attacks. Each of the 30 hosts sends traceroutes to the targets in an independent random order over the course of the 15 minute period, and Hubble then collects the results. In an early version of Hubble, with target selection based only on RouteViews updates, which are published in quarter-hourly batches, I discovered a surprising number of long-lasting problems and decided that this coarseness was still interesting for my study of unavailability. I then based my design on techniques that suffice for this granularity, with probing in quarter-hourly rounds.

Analyzing Traceroutes to Identify Problems

In this section, I describe how Hubble identifies that a prefix is experiencing reachability problems. The analysis uses the triggered traceroutes, combined with Hubble's passive routing view as obtained from RouteViews.

Since Hubble chooses as probe targets a single .1 in each of the suspect prefixes, we cannot be assured of a traceroute reaching all the way to the end-host, even in the absence of reachability problems. In some cases, a host with the chosen .1 address may no longer exist, may be offline, or may stop responding to ICMP probes. Hence, I take a conservative stance on when to flag a prefix as being unreachable from a specific source. Hubble flags a path as working if the last hop on the traceroute is in the origin AS for the prefix. It does this by mapping the last hop seen in the traceroute to its prefix and then to the AS originating that prefix in the BGP snapshot. A prefix may have multiple origins [141], in which case Hubble classifies the path as working if it includes a hop from any of the origin ASes. Since border routers often have IP addresses from the adjoining ASes, Hubble also classifies a path as working if the last IP address on the traceroute has an alias (i.e., another IP address that belongs to the same router) that belongs to the destination's origin AS.

Note that, because I define reachability based on the origin AS for a prefix in routing tables, Hubble ignores prefixes that are not present in any of the BGP table feeds it monitors. These prefixes are easily classified as unreachable just by observing BGP messages. Further, note that the system's reachability check matches against any of the BGP paths for the prefix, rather than the particular path visible from the source of the traceroute. This is because Hubble issues traceroutes from PlanetLab nodes, while it gets its BGP data from RouteViews' vantage points, and the two sets are disjoint.

Traceroutes may occasionally not reach the destination for reasons that have little to do with the overall reachability of the target prefix, such as short-lived congestion on a single path or problems near the source. As described in Section 3.1.1, Hubble flags a prefix as experiencing reachability problems worth monitoring only if less than 90% of the triggered probes to it reach the origin AS.

Isolating Failure Direction using Spoofed Probes

Consider Figure 3.2, with a traceroute from monitor *V1* reaching through provider *A*, and a traceroute from monitor *V2* terminating with its last hop in *B*. One might assume that the problem is between *B* and the origin, but it could also be a problem on the reverse path to *V2*. With just the forward path information supplied by traceroute, these cases are indistinguishable. Hubble employs source-spoofed probes to differentiate the cases and provide much more specific information about the failure. A source-spoofed probe (henceforth referred to as a spoofed probe) is one in which the

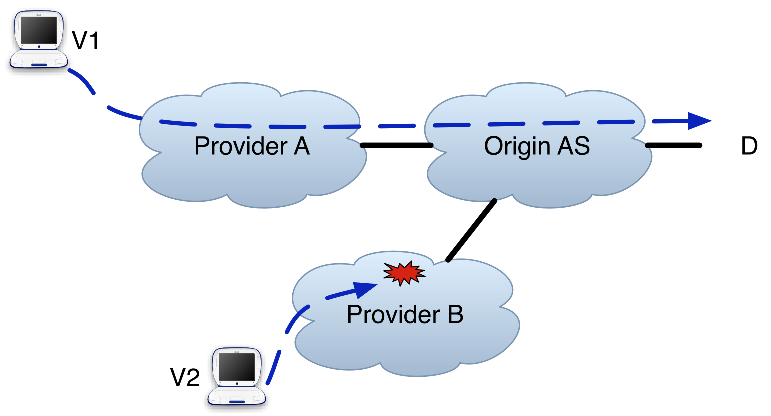
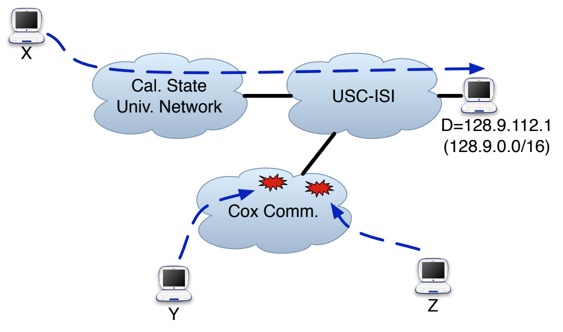


Figure 3.2: An instance in which Hubble can use spoofed probes to isolate the direction of the failure. The arrows depict traceroutes from Hubble's vantage points. While the measurements initially suggest a problem in *B*, the problem could actually be on the reverse path back to *V2* (which may not even traverse *B*). By having *V1* ping *D* spoofing as *V2*, Hubble uses *V1*'s working path to *D* to test *D*'s reverse path to *V2* in isolation. Similarly, by having *V2* spoof as *V1*, the system can test *V2*'s forward path in isolation.

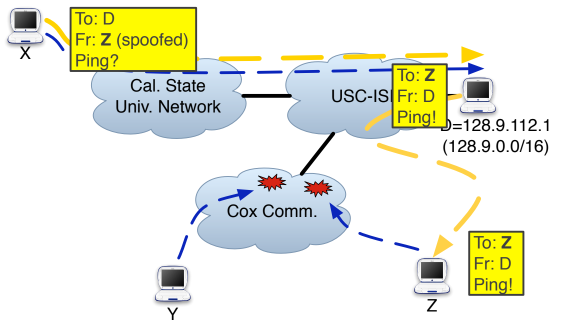
prober sets the source address in the packet to one other than its own. Note that Hubble only ever spoofs packets using the source address of one of the other vantage points.

To determine why *V2* cannot reach *D*, monitor *V1* pings *D* with the source set as *V2*. These probes reach *D* along *V1*'s forward path. If the responses to these probes reach *V2*, then the reverse path from *D* to *V2* must work, and Hubble determines that the failure is on *V2*'s forward path. Otherwise, *V2* pings *D* with the source set as *V1*. If *V2*'s forward path works, then the responses to these probes should reach *V1*, and Hubble determines that the failure is on the reverse path back to *V2*. A central controller coordinates the spoofing, assigning, for instance, *V1* to probe *D* spoofing as *V2*, then fetching the results from *V2* for analysis. Because Hubble only draws conclusions from the receipt of spoofed probes, not their absence, it does not draw false conclusions if *V2* does not receive the probe for other reasons or if the controller is unable to fetch results.

At the time of the Hubble study, the PlanetLab kernel did not allow spoofed probes to be sent, and so spoofing was not fully integrated into Hubble. In this chapter, I provide results using a parallel deployment on RON [4]. Since the time of that study, I received technical support and permission



(a) Traceroutes from vantage points with paths through Cal. State Univ. Network reached *D*, but traceroutes from other vantage points terminated with their last hops in Cox Communication. It was premature to blame Cox, because the problem could have been on asymmetric reverse paths.



(b) *X* pinged *D*, spoofing as *Z*, to cause *D* to reply to *Z*. This probe isolated the reverse path (*D* to *Z*) from the forward path (*Z* to *D*). *Z* received *D*'s response, meaning the reverse path worked. The forward path through Cox must not have been working.

Figure 3.3: These figures depict a failure on paths to ISI that Hubble found on October 8, 2007. Because *X* had a working path to *D*, Hubble used a spoofed ping sent from *X* to probe the reverse path from *D* to *Z* independent from the forward path from *Z* to *D*. *Z*'s receipt of this probe established that the reverse path worked, meaning the problem was on the forward path through Cox.

from PlanetLab to spoof from PlanetLab sites, and so measurements in later chapters use spoofed probes from PlanetLab.

3.2 Evaluation

Hubble ran continuously as an automated system for two years, starting in September 2007. I start by giving an example of one of the problems Hubble found. Figure 3.3 illustrates a simplified version of the probes sent by the system. On October 8, 2007, at 5:09 a.m. PDT, one of Hubble's ping monitors found that *128.9.112.1* was no longer responsive. At 5:13, Hubble triggered traceroutes from around the world to that destination, part of *128.9.0.0/16*, originated by AS4, the Information Sciences Institute (ISI) at the University of Southern California. Four vantage points were unable to reach the origin AS, whereas the others reached the destination. All of the failed probes stopped at one of two routers in Cox Communications, one of ISI's providers, whereas the successful probes traversed other providers, as shown in Figure 3.3(a). In parallel, six of 13 RON vantage points were unable to reach the destination, with traceroutes ending in Cox, while the other seven RON nodes successfully pinged the destination. As shown in Figure 3.3(b), Hubble launched pings from some of those seven nodes, spoofed to appear to be coming from the other six, and all six nodes received responses from *128.9.112.1*. This result revealed that the problems were all on forward paths to the destination, and Hubble determined that Cox was not successfully forwarding packets to the destination. It continued to track the problem until all probes launched at 7:13 successfully reached the destination, resolving the problem after two hours.

In this section, I assess Hubble's efficacy and coverage. In Section 3.3 I present results of a measurement study conducted using Hubble.

How much of the Internet does Hubble monitor? Hubble selects targets from prefixes seen in BGP announcements collected by RouteViews. Its active ping monitoring includes more than 110,000 prefixes discovered to have pingable addresses, distributed over 92% of the edge ASes, i.e., ASes that do not provide routing transit in any AS paths seen in RouteViews BGP tables. These target prefixes include 85% of the edge prefixes in the Internet and account for 89% of the edge prefix address space. I classify a prefix as non-edge if an address from it appears in any of our traceroutes to another prefix. Hubble pings each of its target prefixes every two minutes.

I next gauge whether Hubble is likely to discover problems Internet users confront. To help me do so, my collaborators collected a sample of BitTorrent users by crawling popular sites that aggregate BitTorrent metadata and selecting 18,370 target swarms. For a month starting December 20, 2007, they repeatedly requested membership information from the swarms. They observed 14,380,622 distinct IPs, representing more than 200 of the nearly 250 DNS country codes. I am interested in whether the routing infrastructure provides connectivity, and so Hubble monitors routers rather than end-hosts. End-hosts are more likely to go offline (and do not represent wider reachability issues reachability when they do). They are often in prefixes that do not respond to pings. Further, a router generally uses the same path to all prefixes originated by a given AS [72]. Therefore, I assess whether these representative end-users gathered from BitTorrent share origin ASes with routers monitored by Hubble. I find that 99% of these IP addresses belong to ASes containing prefixes monitored by Hubble.

How effective are Hubble's target selection strategies? To reduce measurement traffic overhead while still finding reachability events, Hubble combines passive BGP monitoring and active pings to select targets for further investigation. As a comparison, suppose a system instead simply used 30 vantage points to send traceroutes to all monitored prefixes every 15 minutes. This scheme would provide as frequent traceroute information during failures as Hubble provides. I refer to this strawman proposal as *pervasive traceroute*. The total probe traffic sent by Hubble, including pings and traceroutes, is 5.5% of that required by *pervasive traceroute*. Note that Hubble probes nearly as fast as is possible from PlanetLab sites, and so *pervasive traceroute* would not be able to monitor nearly as many destinations as Hubble in practice.

Given its much lighter measurement load, I next assess how effectively Hubble's target selection strategies discover events compared to the alternative proposed above. For this evaluation, I issued traceroutes every 15 minutes for 10 days beginning August 25, 2007, from 30 PlanetLab vantage points to 1500 prefixes, and I compare the reachability problems discovered in these traceroutes with those discovered to the same set of prefixes by Hubble's BGP- and ping-based target selection. I use the quarter-hourly traceroutes as "ground truth" reachability information. I only consider events that both begin and end within the experiment and only consider events that persist for at least one additional round of probing after they start. There were 1100 such reachability events, covering 333 of the prefixes, with the longest lasting almost 4 days. 236 of the events involved complete unreachability, and 874 were partial. Here and in later sections, I classify a reachability event as being complete if, at any point during the event, none of the traceroute vantage points is able to reach it. Otherwise, the event is partial.

Figure 3.4 shows the fraction of the events also discovered by Hubble's target selection strategies, both individually and combined. Individually, active ping monitoring discovered 881 of the problems (79%), and passive BGP monitoring discovered 420 (38%); combined, they discovered 939 (85%). For events lasting over an hour, the combined coverage increases to 95%. The average length of an event discovered by ping monitoring is 2.9 hours, whereas the average length of an event discovered by BGP monitoring and not by ping monitoring is only 0.8 hours.

This experiment yields a number of conclusions. First, BGP monitoring is not sufficient. Related work supports this result [37]. I was surprised at how low BGP-based coverage was; in fact, I had

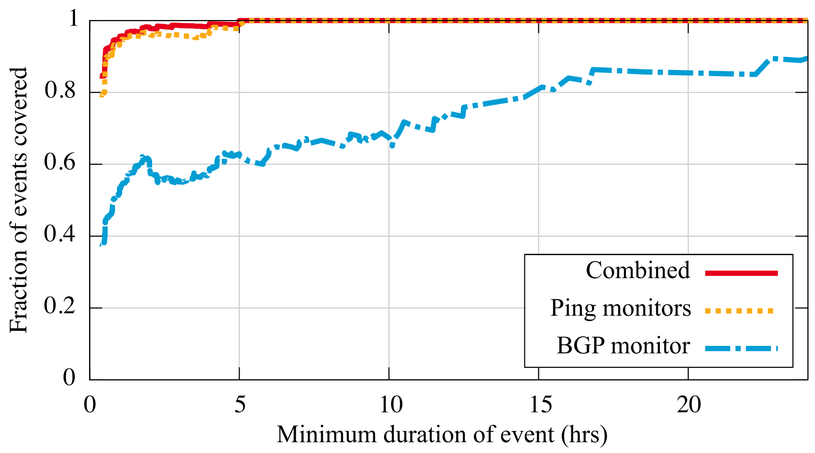


Figure 3.4: For reachability events discovered in 10 days of quarter-hourly traceroutes, fraction of events also discovered by Hubble's target selection. While BGP alone catches only a fraction of the total events, Hubble's techniques combine to be nearly as effective as *pervasive traceroute*, a heavyweight approach with the same time granularity.

originally intended to only do BGP monitoring, until I discovered that it discovered too few events. Second, BGP monitoring provides a supplement to ping monitoring, particularly with short events. Because I strive to limit the rate at which I probe destinations, an inherent tradeoff exists between the number of monitors (more yielding a broader viewpoint) and the rate at which a single monitor can progress through the list of ping targets. Short reachability problems visible from only a few vantages may not be discovered by ping monitors. BGP monitoring often helps in these cases. Third, Hubble discovers many of the outages that *pervasive traceroute* would discover, while issuing many fewer probes.

How quickly after a problem starts does Hubble discover it? Besides discovering a high percentage of all reachability events, a goal for Hubble is to identify the events in a timely fashion. For the same reachability events as in Figure 3.4, Figure 3.5 shows the delay between when the event is detected by *pervasive traceroute*'s quarter-hourly probes and when the prefix is identified as a target by Hubble's target selection. It is possible that Hubble's monitoring identifies problems before the quarter-hourly traceroutes. In these cases, for ease of readability, I give the delay as 0. I additionally plot events lasting longer than an hour separately to avoid the concern that the large number of events shorter than that might distort Hubble's performance. The ideal plot in the graph would

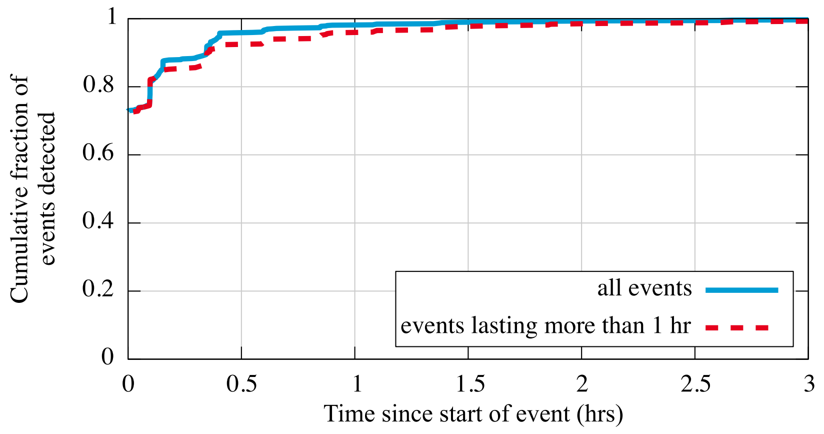


Figure 3.5: For reachability events in 10 days of quarter-hourly traceroutes, time from when the traceroutes detect the event until Hubble identifies the prefix as a target. Events lasting over an hour are also given separately. Fractions are out of only the events eventually identified, 85% overall and 95% of those longer than an hour. Hubble identifies 73% of events at least as quickly as *pervasive traceroute*.

be a vertical line at 0; Hubble achieves that for 73% of the events it identifies, discovering them at least as early as quarter-hourly probes. Of the events lasting over an hour, Hubble discovers 96% of them within an hour of the event's start. So Hubble's lightweight probing approach still allows it to discover events in a timely fashion, and we can generally trust the duration it gives for the length of an event.

3.3 Characteristics of Reachability Problems on the Internet

After assessing the effectiveness of Hubble in achieving its goals, I now present the results of a measurement study using Hubble to detect and measure reachability problems on the Internet for 3 weeks starting September 17, 2007. Hubble issued traceroutes from 35 PlanetLab sites across 15 countries (though only 30 sites at a time) and deployed ping monitors at 104 sites across 24 countries. In Section 3.1.1, I defined a reachability problem to be when a prefix is reachable from less than 90% of probing sites, and a reachability event is the period starting when I first identify that a prefix is experiencing reachability problems and concluding when its reachability increases to

90% or higher. I consider only events that began and ended during the study and persisted through at least one additional round of probing after being detected.

3.3.1 Case Studies

I present a few case studies as examples of problems Hubble detects. These examples are not meant to be exhaustive in any way.

Example of complete unreachability: For a prefix originated by an AS in Zimbabwe, probes to routers along previously successful paths to the prefix showed that the link to its primary provider seemed to have disappeared, and traffic was being routed through a backup provider. However, all probes terminated in this backup provider, either due to a misconfiguration in the secondary provider or due to the origin AS being down. In subsequent rounds of probing, packets started getting through to the destination only after the link to the primary provider came up again. This type of problem cannot be detected without active measurements, as the backup exported a valid AS path.

Example of partial reachability due to a problem in one provider: Hubble found that all probes to a particular prefix in Hong Kong that went through FLAG Telecom were dropped, whereas those that used other transit ASes reached the destination AS, Hutchinson. Of the 30 traceroutes to this destination, 11 went through FLAG and failed to reach the destination. This observation strongly suggests problems with the FLAG-Hutchinson connection.

Example of partial reachability due to a problem with some paths through a transit AS: When monitoring an AS in Vietnam, probes from 15 of Hubble's vantage points passed through the Level 3 network, with some of the probes being dropped in Level 3 while others reached the destination. Comparing the failed probes with earlier ones in which all 15 probes through Level 3 were successful, the internal route within Level 3 had changed. In the earlier successful traceroutes, packets reached a router 4.68.120.143 in the Level 3 network and were forwarded to another router 213.244.165.238 (also in Level 3), and then to the destination AS. However, in the failed probes flagged as a reachability problem, packets reached router 4.68.120.143, which then sent them to another Level 3 router 4.68.111.198, where the traceroutes terminated. This path change could have been due to load balancing or changes in IGP weights, because all the routers on the old path, including router 213.244.165.238, still responded to Hubble's probes. These responses suggest that either router 4.68.111.198 was misconfigured or that the routing information was not consistent throughout the AS.

3.3.2 Prevalence and Duration

During the three week study, Hubble identified 31,692 reachability events, involving 10,224 distinct prefixes. Of these, 21,488 were cases of partial reachability, involving 6,202 prefixes, and 4,785 prefixes experienced periods of complete unreachability. Hubble detected an additional 19,150 events that either were transient or were still ongoing at the end of the study, involving an additional 6,851 prefixes. Of the prefixes that had problems, 58% experienced only a single reachability event, but 25% experienced more than two and 193 experienced at least 20.

The cases of partial reachability are more interesting. Especially for small ASes originating a single prefix, cases of complete unreachability may indicate that the prefix has simply been taken offline. When a destination is partially reachable, a working physical path exists from at least some portions of the Internet. However, some hosts are unable to find the working routes to the destination, either due to a physical partition, due to a routing policy that restricts the export of the working path, or due to a router that is announcing a route that fails to deliver traffic.

In my experiments, however, we can eliminate physical partitions as an explanation. All Hubble vantage points maintained connectivity with the Hubble controller at the University of Washington throughout the study. So, we can piece together a working physical path from the destination to some vantage point, from that vantage point to the controller, then from the controller to the vantage point that could not reach the destination. Since the problems are not due to physical partitions, either routing policies are eliminating all working paths, or routers are advertising paths that do not work. Policy-induced unreachability and misconfigurations might help explain why BGP proved to be a

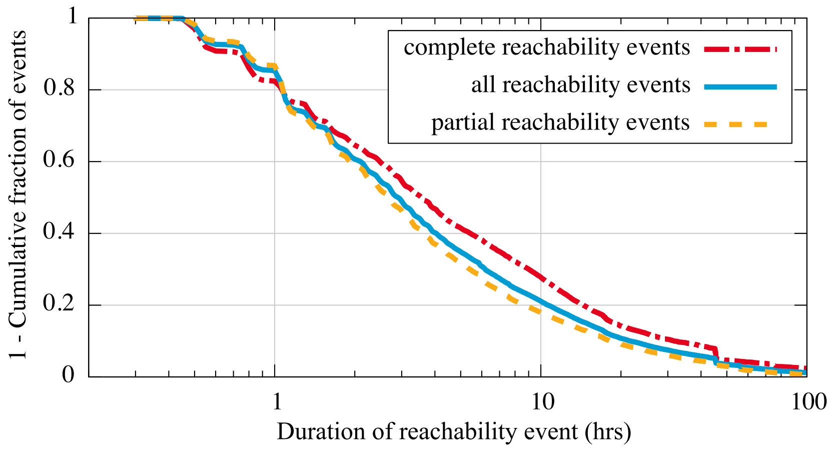


Figure 3.6: Duration of reachability events discovered by Hubble during a three week study starting September 17, 2007. The X-axis is on a log scale. Many of the events lasted for hours, and some lasted for days.

poor predictor of reachability problems, as seen in Section 3.2. By detouring through the University of Washington, the physical paths I demonstrated around the failures would generally violate routing policy. In Section 3.5, I demonstrate that we can often piece together policy-compliant paths around failures in a similar way. In Chapter 5, I present a technique to switch traffic onto alternate working policy-compliant paths, when they exist.

Figure 3.6 shows the duration of reachability events. More than 60% lasted over 2 hours. From Section 3.2, we know the system has excellent coverage of events this long, but may miss some shorter ones. Still, 19,000 events lasted longer than 2 hours, and 2,940 of the events lasted at least a day. Cases of partial reachability tend to resolve faster, with a median duration of 2.75 hours, $\frac{3}{4}$ of an hour shorter than for cases of complete unreachability. Even so, in 1,675 instances a prefix experienced partial reachability for over a day, an astounding violation of the Internet's goal of global reachability.

3.3.3 Direction Isolation

I conducted a study on the RON testbed [4] to determine how often Hubble's spoofed probes establish that a problem is unidirectional, with either the forward or reverse path failing while the other direction continues working. My study used 13 RON nodes, six of which permitted spoofing of source addresses.

I issued pings every half hour for a day from the RON nodes to destinations in all the prefixes known by Hubble to be experiencing reachability problems at that time. I then discarded destinations that were either pingable from all 13 RON nodes or unresponsive from all, as spoofed probes between RON nodes cannot isolate the direction of failures in those cases. For every partially reachable destination *D* and for each RON node *S* which did not receive a ping response from *D*, I chose a RON node *V* that received a response from *D* and could send out spoofed probes. I had *V* send a probe to *D* with the source address set to *S*. If *S* received *D*'s response, it indicated a working reverse path back from *D* to *S*. Thus, there was a problem on the forward path from *S* to *D*. Similarly, in cases when a node *S* did not receive a response from *D* and was able to send spoofed probes, I had *S* send out probes to *D* with the source address set to that of a node *V* that had received a response from *D*. If *V* received *D*'s response, it demonstrated a working forward path from *S* to *D*. Therefore, the problem was on the reverse path from *D* back to *S*. I issued redundant probes to account for congestion losses.

I evaluated 25,286 instances in which one RON node did not receive a response from a destination, but another node did receive a response from the destination. In 53% of these cases, spoofing allowed Hubble to determine that the failure was on the forward path, and in 9% it determined the failure to be on the reverse path. The fact that I could only verify a working forward path from the six nodes capable of spoofing limits these results. Looking only at the 11,355 failed paths from sources capable of spoofing, I found the problem to be on the forward path in 47% of cases and on the reverse path in 21%. The remaining 32% may have had failures both ways, or transient loss may have caught packets.

I next evaluated the same RON data to determine how often the reachability issues from multiple RON nodes to a particular destination could either be blamed entirely on forward paths to the destination or entirely on reverse paths back from the destination. In each half hour, I considered those targets to which at least one RON node had connectivity and at least three did not. I then

Table 3.1: Out of sets of pings in which at least three vantage points failed to reach the destination, the percentage of sets in which Hubble's technique using spoofed packets determined that all problems were on the forward path, all on the reverse path, or a mix of both. The table also gives the percentage of sets for which the system could not make a determination.

Class	Forward	Reverse	Mix	Unknown	Total
All nodes	49%	0%	1%	50%	3605
Spoofing nodes	42%	16%	3%	39%	2172

determined, for each target, whether forward paths were responsible for all problems; whether reverse paths were; or whether each failed path could be pinned down to one direction or the other, but it varied across sources. I then repeated the experiment, but considered only sources capable of spoofing and only destinations unreachable from at least three of these sources. The top half of Table 3.1 presents the results. I determined the failing direction in half of the sets of pings, with nearly all of them isolated to the forward direction (note that the 1% difference accounts for sets in which some of the spoofing nodes had reverse path failures while other nodes had forward path ones). When considering just the spoofing nodes, I was able to explain all failures in 61% of sets. In 95% of those sets, the problems were isolated to either reverse or forward paths only, meaning that all nodes had paths to the destination or that the destination had paths to all nodes, respectively.

3.4 Quantifying Unreachability from Amazon EC2

It is possible that the Hubble data is skewed by only monitoring from PlanetLab's mainly academic sites. To test the prevalence of outages experienced by commercial data centers, I conducted a measurement study using Amazon EC2, a major cloud provider [3]. EC2 presumably has the resources, business incentive, and best practices available for combating Internet outages, whereas previous studies, including Hubble's, focused on academic testbeds that may experience biased conditions [37, 97, 47]. I show that even EC2 data centers experience many long-term network connectivity problems.

I rented EC2 instances in each of the four available EC2 regions from July 20, 2010 until August 29, 2010. Each vantage point issued a pair of pings every 30 seconds to 250 targets, consisting of

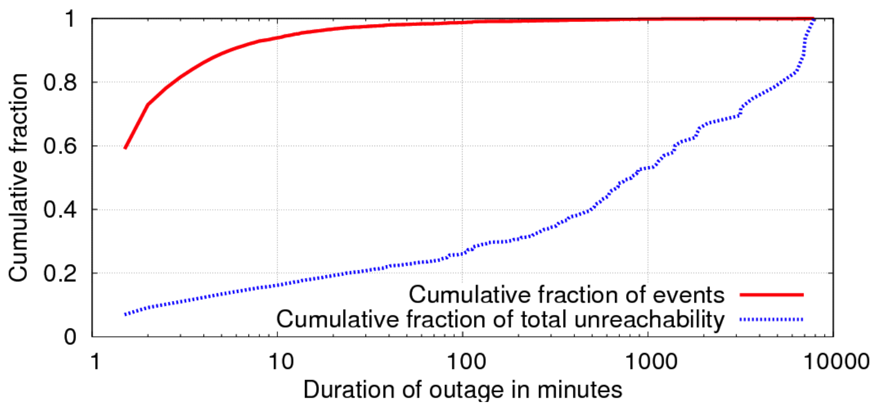


Figure 3.7: For partial outages observed from EC2, the fraction of outages of at most a given duration (solid) and their corresponding fraction of total unreachability (dotted). Note that the x-axis is on a log-scale. I found that more than 90% of outages lasted 10 minutes or less, but that 84% of the total unavailability was due to outages longer than 10 minutes.

five routers each from the 50 highest-degree ASes [128]. I selected the routers randomly from the iPlane topology [58], making sure that each router was from a distinct BGP prefix. I focus on paths to routers in major networks, as these should be even more reliable than paths to end-hosts. I define an outage as four or more consecutive dropped pairs of pings from a single vantage point to a destination. This methodology means that the minimum outage duration I consider is 90 seconds. Because EC2 does not allow spoofing, I could not isolate the direction of failures.

In 79% of the outages in the EC2 study, some vantage points had connectivity with the destination (one of the routers), while others did not. During these partial outages, the destination is up, and some paths to it work, suggesting that it may be possible to route around the problem. Figure 3.7 shows the duration of the 10,308 partial outages. By comparison, an earlier study found that 90% of outages lasted less than 15 minutes [37]. I also find that most outages are relatively short; more than 90% lasted less than 10 minutes (solid line). However, these short outages account for only 16% of the total unavailability (dotted line). The relatively small number of long-lasting problems account for much of the overall unavailability. Delayed protocol convergence does not explain long outages [69], nor do transient loss or congestion.

The Hubble and EC2 studies differ too much for the results to be directly comparable. In particular, Hubble used more vantage points and used traceroutes to test reachability to ASes, whereas EC2 probed much more frequently to a much smaller destination set and used ping to test reachability to particular routers. However, both studies found that most outages were partial and that long-lasting outages contributed substantially to unavailability.

3.5 Assessing Policy-Compliant Alternate Paths

Both the Hubble study and the EC2 study found many long-lasting partial outages, during which a destination was reachable from some vantage points and not others. During both studies, the vantage points maintained connectivity with a controller at the University of Washington. As explained in Section 3.3.2, this connectivity provides potential detours around the outages, establishing that the outages were not due to physical partition. Since the problems are not due to physical partitions, either routing policies are eliminating all working paths, or routers are advertising paths that do not work. Other work has restored routing between a pair of hosts by routing through another end-host [4, 47, 15]. By detouring through the University of Washington or another end-host, the paths around these failures violate the valley-free routing policy [42] – in not making those paths available, routers are properly enacting BGP export policy. If routing policy eliminates all working paths, then the problem will persist until a repair. However, if working policy-compliant paths also exist, it might be possible to switch traffic onto them.

The techniques I present in Chapter 5 rely on the underlying network being physically connected and on the existence of policy-compliant routes around the failure, and so I need to establish that there are long-lasting outages that are not just physical failures or the result of routing export policies. I now consider how often there is a policy-compliant path between a pair of hosts experiencing an outage. I cannot directly answer this question, since BGP policy is opaque. However, I show that alternate paths appear to exist during many failures, and, generally, the longer a problem lasted, the more likely it was that an alternative route existed.

Previous work found that many long-lasting failures occur outside of the end-hosts' networks [37], and I focus on these types of problems. Every 10 minutes for a week starting September

5, 2011, I issued traceroutes in both directions between all pairs of PlanetLab sites. This setting allows me to issue traceroutes from both ends of every path, and the probes from those end-hosts to other PlanetLab sites give a rich view of other paths that might combine to form alternate routes. I considered as outages all instances in which a pair of hosts were up and had previously been able to send traceroutes between each other, but all traceroutes in both directions failed for at least three consecutive rounds, before working again. As with Hubble, I then discarded any outages in which any of the traceroutes terminated in the destination AS. Once traffic reaches the destination AS, it is a local routing (and likely end-host) issue, and my techniques do not address those issues. I was left with nearly 15,000 outages.

I checked if the traceroutes included working policy-compliant routes around the failures. After the fact, for each outage, I inspected all rounds of failed probes except the first and last one, during which the problem may have been developing or resolving. For each such round, I considered paths from the source to other PlanetLab sites, and paths from other PlanetLab sources to the destination. From these sets of paths, I attempted to find a path from the source that intersected (at the IP-level) a path to the destination, such that the spliced path did not traverse the AS in which the failed traceroute terminated. I only considered a spliced path as valid if it would be available under observable BGP export policies. To check export policies, when splicing together a potential path, I only accepted it if the AS subpath of length three centered at the splice point appeared in at least one traceroute during the week [79]. This check suffices to encode the common valley-free export policy [42], thought to be the basic policy in place at most ASes.

My methodology may fail to identify some valid paths that exist. I only considered routes exposed by PlanetLab, a testbed that has somewhat homogenous routing. I also required that spliced paths intersect at a shared IP address. Two traceroutes might intersect at a router or a PoP without sharing an IP address, and I would not consider this intersection when trying to splice paths. So, it is possible that alternate policy paths existed in cases when my probes did not discover them.

Overall, I found that, for 49% of outages, alternate paths existed for the duration of the failure. Considering just long outages that lasted at least an hour, I found alternate routes in 83% of failures. If an alternate path existed during the first round of a failure, it was likely to persist: in less than 2% of cases was there an alternate path in some rounds and not others.

3.6 Summary

In this chapter, I presented three studies of Internet outages. For the largest study, I developed Hubble, a system to perform continuous and fine-grained probing of the Internet in order to identify reachability problems in real-time on a global scale. Hubble establishes that it is feasible to monitor outages on this scale with a 15 minute granularity. At the core of Hubble's approach is a hybrid monitoring scheme, combining passive BGP monitoring with active probing of the Internet's edge prefix space. I estimate that this approach allowed Hubble to discover and monitor 85% of reachability problems, while issuing only 5.5% of the measurement traffic required by an alternative brute force approach that probes all destinations with the same 15 minute granularity that Hubble uses once it discovers a problem. To broaden and support my conclusions, I supplemented the Hubble results with two other studies, one from EC2 and one from PlanetLab.

Results from the studies motivate the goals, requirements, and design of a new approach to addressing outages on the Internet:

The Internet suffers from many outages: In a three week study, Hubble identified persistent reachability problems affecting more than 10,000 distinct prefixes.

Long-lasting outages contribute substantially to unavailability: One in five of the outages discovered by Hubble lasted over 10 hours. Similarly, the EC2 study showed that even commercial data centers experience such problems. Long outages contribute signficantly to unavailability, and so we need to address them to substantially improve availability.

Many of the outages are not discoverable from available public BGP collectors: Only 38% of the problems discovered by Hubble showed up in BGP updates from RouteViews. This result implies that it is necessary to monitor the data plane to discover the problems, and it suggests that connectivity is failing in cases when BGP routes are available.

Many of the outages are partial: Two-thirds of the problems found by Hubble were cases of partial reachability. The EC2 study also found many cases of partial outages. These results suggest the possibility of routing around the failure. Also, as my use of spoofing to isolate the direction of failures demonstrates, it is possible to use measurements along the working paths to learn more about the failures.

Many failures are unidirectional: In most cases, when a partially reachable destination was not pingable from a particular Hubble vantage point, Hubble was able to isolate the problem to either a failed forward path to the destination or a failed reverse path back to the source, while the other direction continued to work. Since a working path in one direction will generally be policy-compliant in the other direction, this finding is further evidence that valid alternatives may exist. It also means that techniques to locate failures need to address asymmetry and unidirectional failures.

Working policy-compliant alternate paths often seem to exist: My study of failures between PlanetLab nodes found that hosts experiencing a failure may actually have a valid alternate path around the failure. Because these other paths avoid the AS where the failing paths terminate, the possibility exists to restore reachability via a BGP reroute. To do so, we need a way to determine which AS is causing the failure and make other ASes choose paths that do not use it.

Chapter 4

REVERSE TRACEROUTE

Traceroute is a simple and widely used Internet diagnostic tool. Operators use it to investigate routing failures and performance problems [122]. Researchers use it as the basis for Internet maps [5, 78, 111], path prediction [78], geolocation [131, 61], AS performance analysis [83], and anomaly detection [140, 138, 136]. My system Hubble uses it.

However, traceroute has a fundamental limitation – it provides no information about the reverse path, despite the fact that policy routing and traffic engineering mean that paths are generally asymmetric [51]. As Richard Steenbergen, CTO for nLayer Communications, presented at a recent tutorial for network operators on troubleshooting, "the number one go-to tool is traceroute," but "asymmetric paths [are] the number one plague of traceroute" because "the reverse path itself is completely invisible" [122].

This invisibility hinders operators. As described in Section 2.4, it limited Google's ability to troubleshoot slow paths to clients. As in the example traceroute in Section 2.5, it limits operator's ability to troubleshoot outages. I presented data in Section 3.3.3 showing that most failures are unidirectional, making it even more evident that we need the ability to measure both directions of a path to understand failures.

Similarly, the lack of reverse path information restricts researchers, as explained in Section 2.3.3. Traceroute's inability to measure reverse paths forces unrealistic assumptions of symmetry on systems with goals including path prediction [78], geolocation [131,61], AS performance analysis [83], and prefix hijack detection [140]. Recent work shows that measured topologies miss many of the Internet's peer-to-peer links [93,52] because mapping projects [5,78,111] lack the ability to measure paths from arbitrary destinations.

A goal of my thesis is to address this basic restriction of traceroute by building a tool to provide the same basic information as traceroute – IP-address-level hops along the path, plus round-trip delay to each – but along the reverse path from the destination back to the source. I have implemented my reverse traceroute system and made it available at `http://revtr.cs.washington.edu`. Google is reportedly also implementing its own version of the tool for its own use. While traceroute runs as a stand-alone program issuing probes on its own behalf, reverse traceroute is a distributed system comprised of a few tens to hundreds of vantage points, owing to the difficulty in measuring reverse paths. As with traceroute, reverse traceroute does not require control of the destination and hence can be used with arbitrary targets. All it requires of the destination is an ability to respond to probes, the same requirement as standard traceroute. It does not require new functionality from routers or other network components.

My system builds a reverse path incrementally, using a variety of methods to measure reverse hops and stitching them together into a path. It combines the view of multiple vantage points to gather information unavailable from any single one. It starts by measuring traceroutes from the vantage points to the source. This limited atlas of a few hundred or thousand routes to the source serves as a set of baseline paths, allowing reverse traceroute to measure the path from an arbitrary destination by building back the path from the destination until it intersects the atlas.

Reverse traceroute uses three main measurement techniques to build backwards. First, Internet routing is generally destination-based, allowing reverse traceroute to piece together the path one hop at a time. Second, reverse traceroute employs the IP timestamp and record route options to identify hops along the reverse path. Third, reverse traceroute use limited source spoofing – spoofing from one vantage point as another – to use the vantage point in the best position to make the measurement. This controlled spoofing is similar to Hubble's and allows reverse traceroute to overcome many of the limitations inherent in using IP options [116, 115, 40], while remaining safe, as the spoofed source address is one of reverse traceroute's hosts. Just as many projects use traceroute, others have used record route and spoofing for other purposes. Researchers used record route to identify aliases and generate accurate topologies [115], and others used spoofing to measure router processing delays [44]. With reverse traceroute, I am the first to show that the combination of these techniques can be used to measure arbitrary reverse paths.

Experienced users realize that, while traceroute is useful, it has numerous limitations and caveats and can be potentially misleading [122]. Similarly, my tool has limitations and caveats. Section 4.4.1 includes a thorough discussion of how the output of my tool might differ from a direct traceroute from the destination to the source, as well as how both might differ from the actual path traversed by traffic. Just as traceroute provides little visibility when routers do not send TTL-expired messages, my technique relies on routers honoring IP options. When my measurement techniques fail to discern a hop along the path, reverse traceroute falls back on assuming the hop is traversed symmetrically. My evaluation results show that, in the median (mean) case for paths between PlanetLab sites, reverse traceroute measures 95% (87%) of hops without assuming symmetry. The need to assume symmetry in cases of an unresponsive hop points to a limitation of my tool compared to traceroute. Whereas traceroute can often measure past an unresponsive hop or towards an unreachable destination, my tool must sometimes guess that it is measuring the proper path. In Chapter 5, I describe techniques that can sometimes overcome these issues to provide useful information even in the presence of failures.

I rely on routers to be “friendly” to my techniques, yet some of my techniques have the potential for abuse and can be tricky for novices to use without causing disturbances. As I ultimately want my tool widely used operationally, I have attempted to pursue my approach in a way that encourages continued router support. My system performs measurements in a way that emphasizes network friendliness, controlling the probe rate to individual routers, in aggregate across all measurements. I presented the work at NANOG [92] and RIPE [104] conferences, and so far the response from operators has been positive about supporting my methods (including source spoofing). I believe the goal of wide use is best served by a single, coordinated system that services requests from all users. In Section 7.3.2, I discuss the future work necessary to open my reverse traceroute system up to public use.

I evaluate the effectiveness of my system as deployed today, although it should improve as I add vantage points. In the median (mean) case for paths between PlanetLab sites, as a conservative estimate, my technique reveals at least 87% (83%) of the routers and 100% (94%) of the PoPs, compared to a traceroute issued from the destination. Paths between public traceroute servers and

PlanetLab show similar results. Because my technique requires software at the source, my evaluation is limited to paths back to PlanetLab nodes I control.

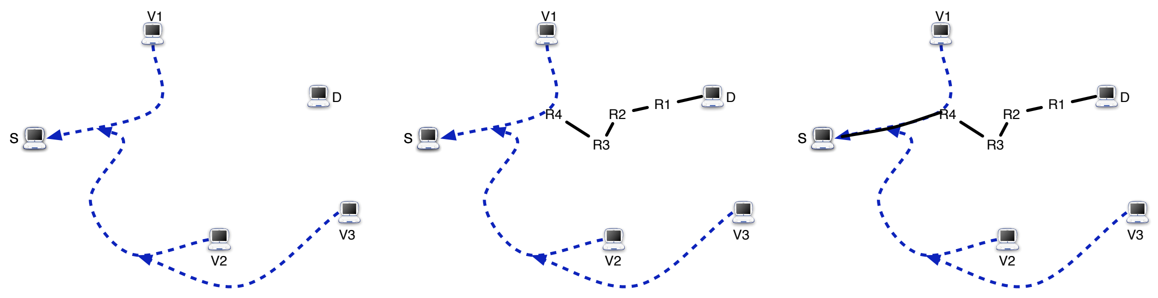
I believe my reverse traceroute system can be useful in a range of contexts, to help address performance and availability problems, as well as to help us understand the Internet. In this chapter, I provide three illustrative examples. I present a case study of how a content provider could use my tool to troubleshoot poor reverse path performance. I also uncover thousands of links at core Internet exchange points that are invisible to current topology mapping efforts. I use my reverse traceroute tool to measure link latencies in the Sprint backbone network with less than a millisecond of error, on average. In Chapter 5, I describe how reverse traceroute can be used as a key building block in a system to locate failures and restore connectivity.

4.1 Basic Reverse Traceroute Techniques

My goal is a reverse path tool equivalent to traceroute. Like traceroute, reverse traceroute should work universally without requiring control of a destination, and it should work using only the features of Internet routers and protocols as they exist today. The reverse traceroute tool should return IP addresses of routers along the reverse path from a destination back to the source, as well as the round-trip delay from those routers to the source. In this section, I describe the basic techniques I use to measure a reverse path. Then, in Section 4.2, I describe other techniques I use to increase the accuracy, coverage, and efficiency of my reverse traceroute system.

I assume the availability of a set of vantage points distributed around the Internet. I design the system to work with tens or hundreds of vantage points, a small number relative to the tens of thousands of ASes and hundreds of thousands of BGP prefixes. I use these vantage points in several ways: to precompute an atlas of Internet routes from the vantage points to destinations around the Internet, to measure a set of paths from the vantage points to the reverse traceroute source, and to issue directed probes to uncover pieces of the reverse path.

At a high level, the source requests a path from my system, which coordinates probes from the source and from a set of distributed vantage points to discover the path. First, distributed vantage



(a) Vantage points traceroute to S, creating an atlas of known paths.

(b) Vantage points measure path from D until it intersects a path in the atlas.

(c) Combine to yield complete path.

Figure 4.1: High-level overview of the reverse traceroute technique. § 4.1.1- 4.1.2 explain how to measure from D back to the atlas.

points issue traceroutes to the source, yielding an atlas of paths towards it (Fig. 4.1(a)). This atlas provides a rich, but limited in scope, view of how parts of the Internet route towards the source. Reverse traceroute uses this limited view to bootstrap measurement of the desired path. Because Internet routing is generally destination-based, the system assumes that the path to the source from any hop in the atlas is fixed (over short time periods) regardless of how any particular packet reaches that hop. Once the path from the destination to the source reaches a hop in the atlas, it uses the atlas to derive the remainder of the path.

Second, using techniques explained in Sections 4.1.1 and 4.1.2, reverse traceroute measures the path back from the destination incrementally until it intersects this atlas (Fig. 4.1(b)). The assumption that routing is destination-based also enables this hop-by-hop path stitching; once reverse traceroute knows that the path from D goes through R, it needs only determine the route at R towards S when attempting to discover the next hop. This assumption produces a valid path even in cases of packet-, flow-, and destination-based load balancing, as long as R does not use source-specific routing. Essentially, each intermediate router R reverse traceroute finds on the path can become the new destination for a reverse traceroute back to the source.

Finally, as shown in an example in Section 4.1.3, reverse traceroute merges the two components of the path – the destination-specific part from the destination until it intersects the atlas and the atlas-derived path from this intersection back to the source – to yield a complete path (Fig. 4.1(c)).

4.1.1 Identify Reverse Hops with IP Options

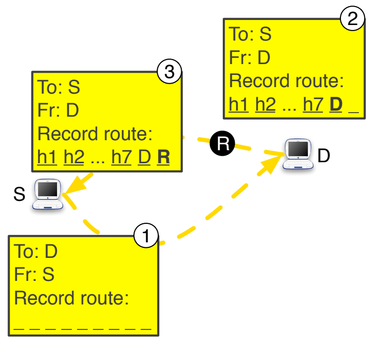
Reverse traceroute uses two basic measurement primitives, the Record Route and Timestamp IP options, to measure segment by segment until intersecting the atlas. IP options are standard options defined as part of the Internet Protocol. They can be enabled on any IP packet and instruct routers to process the packet in particular ways. Whereas TTL values are reset by the destination, restricting traceroute to measuring only on the forward path, a destination replying to a ping generally reflects any IP options in its reply, so routers along both the forward and reverse path process the options. I briefly explain how the two options I use work:

IP Record route option (*RR*): With this option set, a probe records the router interfaces it encounters. The IP standard limits the number of recorded interfaces to nine; once those fill, no more interfaces are recorded. If RR slots remain when the destination sends its response, then routers on the reverse path will record some of that route.

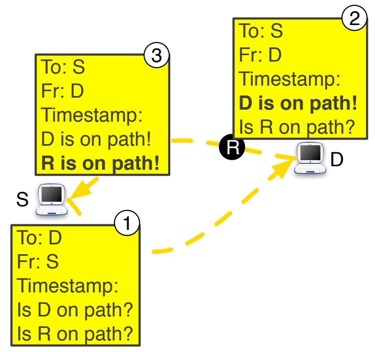
Reverse traceroute uses RR to gather reverse hops directly if the destination is within eight hops. Figure 4.2(a) depicts this probe, which I refer to as *RR-Ping*$(S \rightarrow D)$. The source S issues a ping to D with the RR option enabled. This ping allows a limited measurement of the reverse path, as long as the destination is fewer than 9 hops from the source. While it might seem as though this probe would be enough to build reverse traceroute, the average path on the Internet is approximately 15 hops [58]. So, *RR-Ping*$(S \rightarrow D)$ is just one component of the system.

IP timestamp option (*TS*): IP allows probes to query a set of specific routers for timestamps. Each probe can specify up to four IP addresses, in order; if the probe traverses the router matching the next IP address that has yet to be stamped, the router records a timestamp. The addresses are ordered, so a router will not timestamp if its IP address is in the list but is not the next one.

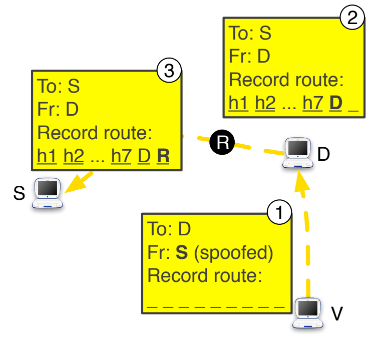
The timestamp option allows reverse traceroute to confirm whether a particular router is on the reverse path, using what I refer to as *TS-Query-Ping*$(S \rightarrow D|D, R)$. As shown in Figure 4.2(b), the source S issues an ICMP ping probe to D with the timestamp query enabled for the ordered pair of IP addresses D and R. R will record its timestamp only if it is encountered by the probe after D



(a) S sends a record route ping. The header includes slots for 9 IP addresses to be recorded (1). If the packet reaches D with slots remaining, D adds itself (2), and routers on the reverse path fill the remaining slots (3).



(b) S sends a timestamp ping, asking first for D to provide a stamp if encountered, then for R to provide one (1). If D supports the timestamp option, it fills out a timestamp (2). Because the timestamp requests are ordered, R only fills out a timestamp if encountered after D, necessarily on the reverse path (3).



(c) Vantage point V sends a record route ping to D, spoofing as S (1). D replies to S (2), allowing S to discover that R is on the reverse path (3). Reverse traceroute uses this technique when S is out of record route range of D, but V is close enough.

Figure 4.2: Three measurement techniques that allow us to establish that R is on the reverse path from D back to S. In §4.2, I give two techniques, akin to (c), that use spoofing to overcome limitations in timestamp support. Because reverse traceroute builds a reverse path incrementally, D can either be the original target destination or some intermediate router already discovered to be along the path.

has stamped the packet. In other words, if S receives a timestamp for R, then it knows R appears on the reverse path. For reverse traceroute's purposes, the value of the timestamp is meaningless; it just cares whether or not a particular router processes the packet. Thus, if reverse traceroute guesses a router on the return path, the TS option can confirm the hypothesis.

Reverse traceroute uses existing network topology information – specifically IP-level connectivity of routers from Internet topology maps [58] – to determine candidate routers for the reverse path. Routers adjacent to D in the topology are potential next hops. Reverse traceroute uses timestamp query probes to check whether any of these potential next hops is on the path from D to S.

There are some caveats to using record route and timestamp. For example, from any particular source, only a fraction of targets will be reachable within record route's limit of nine hops. In Section 4.1.2, I explain how reverse traceroute uses multiple vantage points to overcome this limitation.

Section 4.1.3 provides an example of using IP options and multiple vantage points to measure a reverse path. Other caveats exist. Some ASes filter and drop probes with IP options set. Further, some routers do not process the IP options in the prescribed manner. Section 4.2 explains how reverse traceroute can overcome these limitations in many cases by carefully orchestrating multiple vantage points and/or multiple probes.

4.1.2 Record Route with Spoofed Source Addresses

As with Hubble, reverse traceroute uses a limited form of spoofing, where it replaces the source address of a probe with the actual source of the reverse path it is measuring. This form of spoofing is an extremely powerful measurement tool. When V probes D spoofing as S, D's response will go to S; I refer to V as the spoofer and S as the receiver. This method allows the probe to traverse the path from D to S without having to traverse the path from S to D and without having a vantage point in D's prefix. I could hypothetically achieve a similar probe trajectory using loose source routing (from V to S, but source routed through D) [95]. However, a source-routed packet can be identified and filtered anywhere along the path, and such packets are widely filtered [8], too often to be useful in my application.[1] On the other hand, if a spoofed packet is not ingress filtered near the spoofer, it thereafter appears as a normal packet. I can use a source capable of spoofing to probe along any path.

This arrangement allows reverse traceroute to use the most advantageous vantage point with respect to the particular measurement it wants to perform. Hubble uses limited spoofing to check one-way reachability; reverse traceroute uses it to overcome limitations of IP options. Without spoofing, RR's 9 IP address limit restricts it to being useful only when S is near the target. However, as shown in Figure 4.2(c), if some vantage point V is close enough to reach the target within eight RR slots, then reverse traceroute can probe from V spoofing as S to measure addresses on the path back to S. Similarly, spoofing can bypass problematic ASes and machines, such as those that filter timestamp-enabled packets or those that do not correctly implement the option.

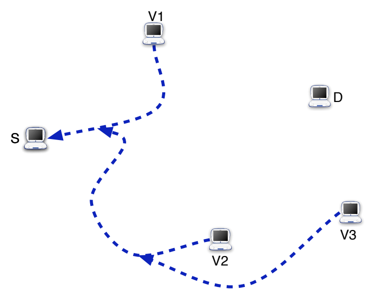
[1]Existing work [40, 115] and my own study in Section 4.4.2 show comparatively wide support for timestamp and record route options.

Although spoofing is often associated with malicious intent, I use it in a very controlled, safe fashion. Nefarious uses generally spoof as an address outside the control of the spoofer. Reverse traceroute only spoofs as a source under its control, avoiding any issues of concealment or unwanted traffic. A source requests a reverse path measurement, then receives responses to probes sent by vantage points spoofing as the source. No harm can come from the source causing itself to receive measurement packets. Since some ASes filter all spoofed packets sent by end-hosts within them [12], reverse traceroute tests from each host and only sends further spoofed probes where allowed. My systems have been issuing spoofed probes for over three years without complaint.

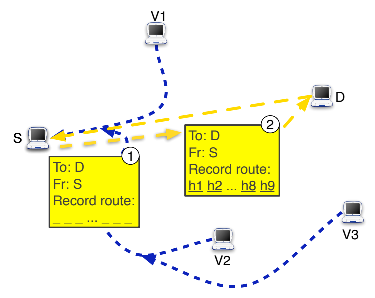
Roughly one-third of PlanetLab sites currently allow spoofing, with the number slowly increasing from around 20% over the course of this dissertation. This result is fairly representative; the Spoofer project tested 12,000 clients and found that 31% could send spoof packets [12]. Even if filtering increases, I believe, based on positive feedback from operators, that the value of my service will encourage an allowance (supported by router ACLs) for a small number of strategically-placed measurement nodes to issue spoofed probes using a restricted set of ports. Measurement Lab (M-Lab) [85] could provide part of such a deployment. M-Lab is a distributed platform for measurement tools to enhance Internet transparency. Of the 16 current M-Lab sites, 15 can spoof. It is likely that a higher percentage of M-Lab sites can spoof, as compared to PlanetLab or the Spoofer project results, because the M-Lab nodes are hosted in commercial data centers. Another possible approach is to have routers rate limit spoofed options packets (just as with UDP probes) and filter spoofed probes sent to broadcast addresses, thereby reducing the security concerns without diminishing their utility for network measurements.

4.1.3 Example of Incrementally Building a Reverse Traceroute

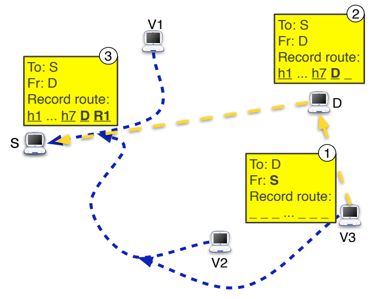
IP option-enabled probes, coupled with spoofing as S from another vantage point, give reverse traceroute the ability to measure a reverse hop from D on the path back to S. It uses these probes to stitch together the path incrementally until it reaches a router R that is on a path from some vantage point V to S. Reverse traceroute then infers the rest of D's reverse path from R onward as being the same as V's.



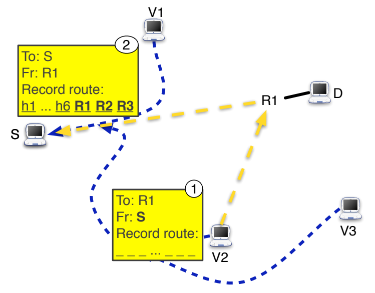
(a) Vantage points traceroute to S, creating an atlas of known paths.



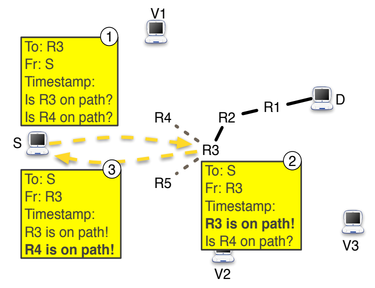
(b) S sends an RR-Ping to D (1), but all the RR slots fill along the forward path (2), so S does not learn any reverse hops in the reply.



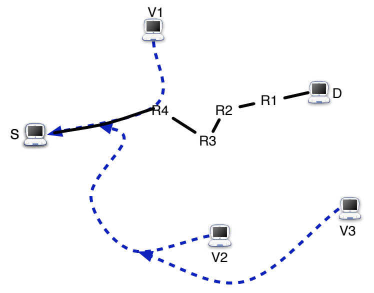
(c) $V3$ is closer to D, so sends an RR-Ping spoofing as S (1). D records its address (2), and the remaining slot fills on the reverse path, revealing $R1$ (3).



(d) A vantage point $V2$ close to $R1$ sends an RR-Ping spoofing as S (1), discovering $R2$ and $R3$ on the reverse path (2).



(e) Reverse traceroute uses an Internet map to find routers adjacent to $R3$, then send each a TS-Query-Ping to verify which is on the reverse path.



(f) When the path intersects a path in reverse traceroute's traceroute atlas, reverse traceroute assumes the rest of the path from D follows that route.

Figure 4.3: Incremental construction of a reverse path using diverse information sources.

Figure 4.3 illustrates how reverse traceroute composes the above set of techniques to determine the reverse path from D to S, when it has control over S and a set of other vantage points (V_1, V_2, V_3). As described at the beginning of Section 4.1, the system also uses a partial map of router-level connectivity, from iPlane's offline mapping [58].

Reverse traceroute begins by having the vantage points issue traceroute probes to S (Figures 4.1(a) and 4.3(a)). These probes serve as a baseline set of observed routes towards S that can be used to complete a partially inferred reverse path. Reverse traceroute then issues *RR-Ping*$(S \rightarrow D)$ to determine if the source S is within eight RR hops of the destination, i.e., whether a ping probe from S can reach D without filling up its entire quota of 9 RR hops (Figure 4.3(b)).[2] If the source

[2]This is not quite as simple as sending a TTL=8 limited probe, because of issues with record route implementa-

is within eight RR hops of D, this probe would determine at least the first hop on the reverse path, with further hops recovered in an iterative manner.

If the source is not within eight hops, reverse traceroute determines whether some vantage point is within eight RR hops of D (Section 4.2.4 describes how it does this). Let V_3 be one such vantage point. Reverse traceroute then issues a spoofed RR ping probe from V_3 to D with the source address set to S (Figure 4.3(c)). This probe traverses the path $V_3 \rightarrow D \rightarrow S$ and records IP addresses encountered. In many cases, most of these addresses will be on the forward path from V_3 to D, but, since V_3 is within eight hops, the probe will discover some hops along the reverse path back to S. In the example, the probe reveals R_1 to be on D's path to S. Reverse traceroute then iterates over this process, with the newly found reverse hop as the target of its probes. For instance, it next determines a vantage point that is within eight RR hops of R_1, which could be a different vantage point V_2. Reverse traceroute uses this new vantage point to issue a spoofed RR ping probe to determine the next hop on the reverse path (Figure 4.3(d)). In some cases, a single RR ping probe may determine multiple hops, as in the illustration with R_2 and R_3.

Now, consider the case where neither S nor any of the vantage points is within eight hops of R_3. In that case, reverse traceroute considers the potential next hops to be routers adjacent to R_3 in the known topology. Suppose R_4 and R_5 are candidates, determined from a precomputed atlas of Internet topology. Reverse traceroute issues timestamp probes from S to verify whether R_4 or R_5 respond to timestamp queries *TS-Query-Ping*$(S \rightarrow R_3|R_3, R_4)$ and *TS-Query-Ping*$(S \rightarrow R_3|R_3, R_5)$ (as shown in Figure 4.3(e)). In the example, R_4 responds, so it is adjacent to R_3 in the network topology and is on the reverse path from R_3. Thus, reverse traceroute assumes R_4 is the next hop on the reverse path. Reverse traceroute continues to perform incremental reverse hop discovery until it intersects with a known path from a vantage point to S,[3] at which point it considers that to be the rest of the reverse path (Figures 4.1(c) and 4.3(f)). Once the procedure has determined

tions [115]. Some routers on the forward path might not record their addresses, thereby freeing up more slots for the reverse path, while some other routers might record multiple addresses or might record their address but not decrement or respond to TTL-limited probes.

[3]Measurement techniques may discover different addresses on a router [116], so reverse traceroute determine intersections using alias data from topology mapping projects [62, 78, 115] and a state-of-the-art technique [11].

the hops in the reverse path, reverse traceroute issues pings from the source to each hop in order to determine the round-trip latencies.

Sometimes, reverse traceroute may be unable to measure a reverse hop using any of its techniques, but I still want it to provide the user with useful information. When reverse traceroute is unable to calculate the next hop on a path, the source issues a standard traceroute to the last known hop on the path. Reverse traceroute then assumes the last link is traversed symmetrically, and it tries to calculate the rest of the reverse path from there. In Section 4.4.1 and Section 4.4.2, I present results that show that reverse traceroute usually does not have to assume many symmetric hops and that, even with this approximation, it still achieves highly accurate paths.

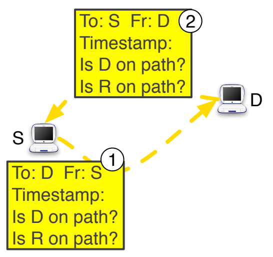
4.2 Advanced Reverse Traceroute Techniques

For a workable reverse traceroute for the Internet, I needed to add techniques to address limitations in available vantage points and variations in router implementations. The following goals drive these changes:

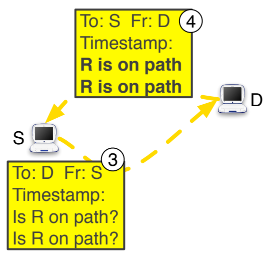
- *Accuracy:* The system should be robust to variations in how routers handle IP options.
- *Coverage:* It should work for arbitrary destinations irrespective of AS-specific configurations.
- *Efficiency:* It needs to be selective with its use of vantage points to introduce as little measurement traffic as possible.

4.2.1 Correcting for Variations in IP Options Support

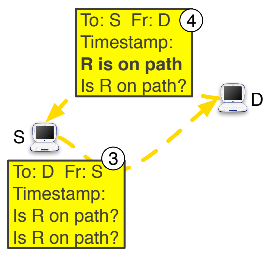
I next explain how reverse traceroute compensates for two variations in support for the timestamp option. In the first variation, as I will present in Section 4.4.2, 15.5% of addresses respond to timestamp-enabled pings, but do not record stamps in their responses. When checking if R is on the reverse path from D, reverse traceroute normally asks for both D's and R's timestamps, to force R



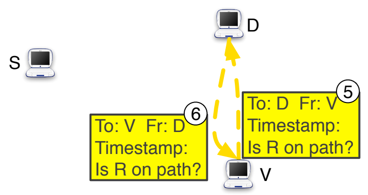
(a) S sends a timestamp ping to D (1) and receives a reply, but D has not filled out a timestamp (2).



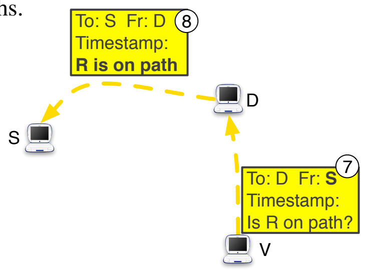
(b) S pings D asking for two timestamps from R (3). If it receives two stamps (4), R must be on both the forward and reverse paths.



(c) Suppose instead that R only timestamps once (4). It could be on the forward or reverse path. Reverse traceroute uses an extra vantage point to determine which.



(d) Reverse traceroute finds a V s.t., when V pings D asking for R's timestamp (5), it does not receive a stamp (6). This response suggests that R is not on V's path to D.



(e) V spoofs as S, pinging D (7), and S receives a timestamp for R (8). Because the system established that R is not on V's path to D, R must be on the reverse path from D to S.

Figure 4.4: Two examples of how reverse traceroute discovers a reverse hop with timestamp even though D does not stamp, as long as it replies. Because reverse traceroute builds a reverse path incrementally, D can either be the original target destination or some intermediate router already discovered to be along the path. In (b), S received two stamps from R, indicating that R is on both the forward and reverse paths. In (c), S instead received only one stamp, which could be from either the forward or reverse path. (d) and (e) depict how reverse traceroute can isolate the stamp to the reverse path.

to only stamp on the reverse path, but this approach will not work for destinations (or routers) that do not record stamps.

Figure 4.4 illustrates how reverse traceroute addresses this behavior. After identifying that D responds without stamping (Fig. 4.4(a)), S pings D asking for two timestamps from R. If S receives no stamps, reverse traceroute concludes that R is not on the path. If it receives two stamps, it concludes that R is on both the forward and reverse paths between S and D (Fig. 4.4(b)). If S receives only a single stamp, R could be on the forward or reverse path (Fig. 4.4(c)). To determine which, reverse traceroute enlists an additional vantage point. Essentially, it finds a vantage point V

which it can establish does not have R on its path to D (Fig. 4.4(d)). Then, V pings D spoofing as S, asking for R's timestamp (but not D's). If S receives a stamp for R, it proves R is on the reverse path from D (Fig. 4.4(e)). This technique relies on being able to find a vantage point that does not have R on its path. To establish that reverse traceroute vantage points have sufficiently diverse paths to many destinations, I examined iPlane traceroutes to destinations in 140,000 prefixes and found at least two adjacent hops for 55% of destinations.

Unfortunately, the absence of a timestamp response in V's non-spoofed timestamp ping to D (Fig. 4.4(d)) does not definitively establish that R is not on V's path. R could be on the path, but not timestamp a particular packet, because it was rate-limiting its stamps, because the packet was lost due to congestion, or because the packet was load-balanced onto a path that did not include R (but the next probe might be load-balanced through R). In all these cases, R could record a timestamp on subsequent probes. Normally, reverse traceroute uses timestamp to establish that a router is on a path. If a router actually on the path fails to timestamp, the system misses inferring a reverse hop – a false negative. In the technique described in the last paragraph, reverse traceroute instead uses timestamp to establish that R is not on V's path. If R was on V's forward path but failed to record a timestamp, the subsequent probe from V spoofing as S could cause reverse traceroute to falsely conclude that R was on the reverse path to S – a false positive.[4] To guard against this possibility, reverse traceroute uses multiple probes to reduce the likelihood of false positives. To test this approach, for each iPlane destination D, each PlanetLab source V, and each router R on V's traceroute to D, I sent a dozen timestamp pings from V to D asking for R's timestamp. Table 4.1 gives the likelihood of receiving a timestamp in the last $(12 - n)$ probes after not receiving one in the first n. In practice, reverse traceroute sends 3 probes to test whether R is on V's path. If none are stamped, the system considers that sufficient evidence that R is likely not on the path.

Table 4.1 suggests that this approach will lead to some false positives, but this is acceptable. Reverse traceroute selected R as a candidate because it was adjacent to D in the network topology. The timestamp from R established that R was on either S's forward or reverse path to D (Fig. 4.4(c)). A false positive will mean that reverse traceroute assumed V was the first hop on the reverse path,

[4]It would also be difficult to use traceroute to establish that R was not on the path, as an address along the traceroute could be an alias of R, and existing techniques to identify aliases do not provide complete solutions.

Table 4.1: For routers on traceroutes from PlanetLab to iPlane destinations, the chance that the router will timestamp a ping sent to the destination as a function of the number of previous pings it has not timestamped. The more probes unstamped, the less chance a router will stamp future probes.

Consecutive Probes without Timestamp	Chance of Timestamp with More Probes
1	10.1%
2	6.3%
3	4.3%
4	3.1%
5	2.4%
10	0.9%

when in fact it was likely the last hop on the forward path. In the case when it is unable to determine a hop using timestamp, reverse traceroute assumes that the last hop on the forward path is symmetric, so false positives with timestamp are equivalent to its fallback assumption.

In the second variation, some hosts record stamps in two slots, even if the request only asks for their timestamp once, and even if the second address asked for is provably not on the path [114]. All Linux machines I tested display this bug. As I will present in Section 4.4.2, 5.8% of addresses in the iPlane topology display this pathology. If reverse traceroute fails to recognize these cases, then any address it guesses as being adjacent will appear to have stamped and be declared as being on the reverse path. Reverse traceroute accounts for this extra stamping by testing for R on the reverse path by requesting *TS-Query-Ping*$(S \rightarrow D|D, R, R)$. Since I have never observed any hosts stamping two extra slots, if the third slot is stamped then R must have done it. However, many routers will only stamp once, even if asked twice, so, if reverse traceroute only receives two stamps, it needs to decide whether R made the second stamp or D recorded an extra stamp. I initially thought to evaluate these cases by checking whether the values of the first and second stamps were the same. However, I found that 5.4% of extra stamps are actually incremented from the initial stamp, a false positive under the strawman test. Further, the test would lead to false negatives for adjacent nearby routers that stamp the same time as the previous hop. By probing for timestamps to adjacent hops seen in iPlane traceroutes, I found that adjacent routers gave identical stamps in 6.3% of cases.

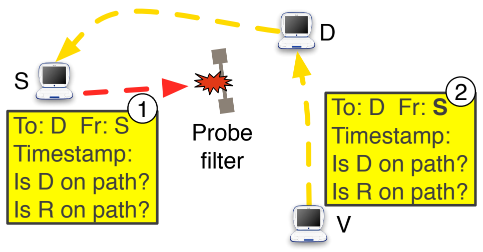


Figure 4.5: If a timestamp probe from S encounters a filter (1), reverse traceroute can often bypass it by spoofing as S from a different vantage point (2), as long as the filter is just on the forward path.

Reverse traceroute employs an additional test to exclude these false positives and false negatives. If the second timestamp is equal to or one more than the first, reverse traceroute sends a followup probe to D, *TS-Query-Ping*$(S \rightarrow D|D, X)$, where X is the address of a machine known not to be on the path (specifically, a desktop in our lab at the University of Washington). If D does not record an extra stamp for X, then reverse traceroute declares R as being on the reverse path.

Variations also exist in how routers process the record route option [115]. However, known implementations only vary in whether the router records an IP when TTL-expiring a packet and in which IP address (router alias) the router records. Reverse traceroute does not send TTL-limited packets, and so variations in TTL-expiration behavior do not affect the system. Variations in which alias the router records do not affect how reverse traceroute probes. The system does use alias datasets to decide if paths / measurements intersect, however.

4.2.2 *Avoiding Probe Filters to Improve Coverage*

I next discuss techniques to improve the coverage of reverse traceroute's measurements. Some networks may filter ICMP packets, and others filter packets with options enabled. Further, some vantage points have all their outgoing packets filtered, while others are filtered along some routes and not others. In the course of measuring a reverse path, if a source attempts a TS or RR measurement and does not receive a response, reverse traceroute retries the measurement with a vantage point (VP) spoofing as the source. As seen in Figure 4.5, if filtering occurs only along the source's forward path, and the VP's forward path does not have a filter, the original source should receive the response. Previous results suggest that most filtering occurs in edge ASes [40], and so filters along one forward path often will not occur along another.

I demonstrate the effectiveness of this approach on a small sample of 1000 IP addresses selected at random out of those in the iPlane topology known to respond to timestamp probes. The 1000 destinations include addresses in 662 ASes. I chose 10 spoofing PlanetLab vantage points I found to receive (non-spoofed) timestamp responses from the highest number of destinations. To simulate reverse traceroute sources, I chose one host at each of the 209 working non-spoofing PlanetLab sites. First, each (non-spoofing) PlanetLab source sent a series of six timestamp pings to each of the 1000 destinations. I used redundant probes to account for packet loss due to congestion, rate limiting, or other factor unrelated to permanent filters. Of the 209 sources, 103 received responses from at least 700 destinations; I dropped them from the experiment, as they do not experience significant filtering. Then, each spoofing vantage point sent timestamp pings to each destination, spoofing as each of the remaining PlanetLab hosts in turn. Of these, 63 sources failed to receive any responses to either spoofed or non-spoofed probes; they are completely stuck behind filters or were not working. For the remaining 43 hosts, Figure 4.6 shows how many destinations each host receives responses from, both without and with spoofing. The results show that some sites benefit significantly. In reverse traceroute's timestamp measurements, whenever the source does not receive a response, it retries with 5 spoofers. Since some vantage points have filter-free paths to most destinations, the system uses the 5 best overall, rather than choosing per destination. Spoofing enables many of the nodes that experience widespread filtering to still use timestamps as part of reverse traceroute.

4.2.3 Filtering of Record Route

Similarly, I would like to differentiate between cases when the destination does not respond to record route probes and cases when the probes are being filtered (but might work over a different path to the destination). As explained in the previous section, if a timestamp probe from a source fails to receive a response, reverse traceroute can have a number of vantage points probe the destination at

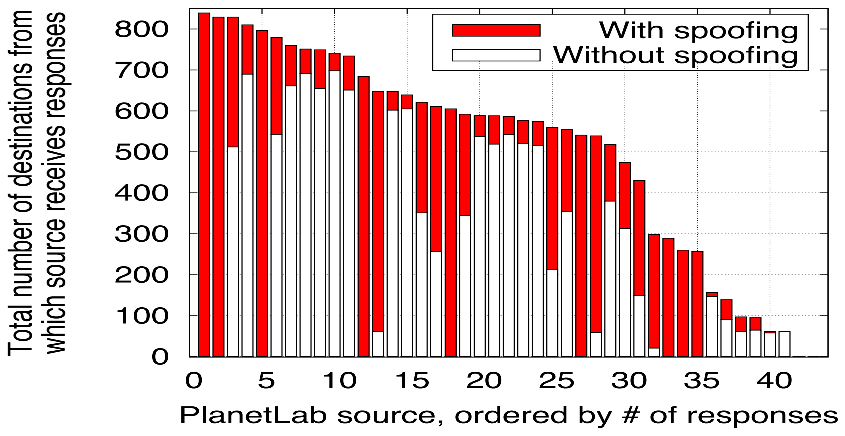


Figure 4.6: For 43 PlanetLab nodes, the number of destinations (out of 1000) from which the node receives timestamp responses. The graph shows the total number of unique destinations when sending the ping directly and then when also using 10 spoofers. The nodes are ordered by the total number of responding destinations. Other PlanetLab sites were tested but are not included in the graph: 103 did not experience significant filtering and 63 did not receive responses even with spoofing.

once, spoofing as the source. We need not worry about rate limiting – any probe that reaches the destination and receives a response is equivalent.

However, reverse traceroute cannot send record route probes to a target simultaneously from a large number of vantage points. With spoofed record route, only nearby spoofers can find reverse hops, since each packet includes only nine slots. Because many routers rate limit after only a few probes, reverse traceroute cannot send from many vantage points at once, in the hopes that one will prove close enough – the router might drop the probes from the VPs within range. Since reverse traceroute can send only a few probes at a time, I would like to limit the number of sets it has to send before deciding that a destination is simply not responsive.

To set an appropriate threshold, I probed every IP address in the DisCarte topology from every PlanetLab site, and I considered every site that received responses from at least 10,000 destinations (so ran properly and does not have a filter near the source). Figure 4.7 shows the fraction of those sites that received responses from the destinations. The graph shows that, if a destination responds to a few vantage points, it most likely responds to many.

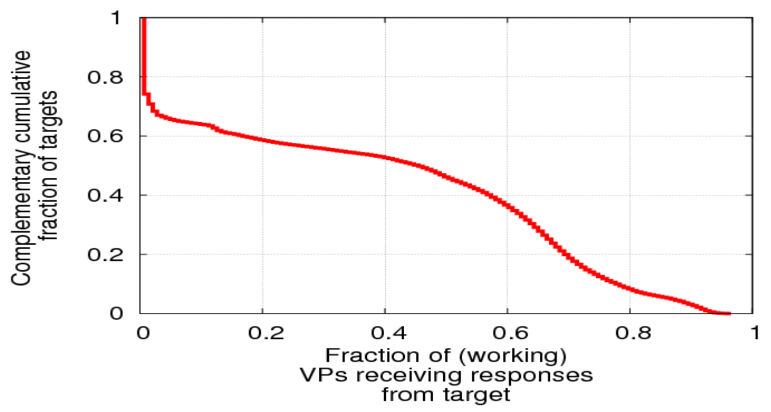


Figure 4.7: For each address in the DisCarte topology, the fraction of PlanetLab sites that receive responses when sending record route probes to the address. The graph depicts the distribution of this fraction as a CCDF over the DisCarte addresses. Most addresses replied to either very few sites or to at least half the sites. This result suggests that there is little utility in continuing to probe a target that has not responded to a set of vantage points. In practice, reverse traceroute gives up after trying 10 sites.

In measuring a reverse path, reverse traceroute first attempts a non-spoofed ping, then, if it fails, reverse traceroute attempts spoofed probes from three vantage points at a time. It gives up after three such sets if none yields a response. Making the simplifying assumption that each vantage point is equally likely to receive a response, we would expect to get a response with probability at least $0.89 = 1 - (1 - 0.2)^{10}$ for the 59% of destinations that reply to at least 20% of sites and probability at least 0.99 for the 52% of destinations that reply to at least 40% of sites. I consider these fractions to be adequate and do not worry about the destinations that reply to fewer vantage points, since exhaustive search would take too long to justify the limited benefit.

4.2.4 Selective Use of Vantage Points for Scalability

When sending spoofed record route probes, reverse traceroute determines which VPs are likely to be near a router before probing the router, then uses only the ones likely to be nearby. Because Internet routing is generally based on the longest-matching prefix for a destination, a VP close to one address in a prefix is likely close to other addresses in the prefix. Reverse traceroute precomputes the set of vantage points to use for each prefix. Each week, reverse traceroute harvests the set of router IP addresses seen in the Internet atlas gathered by iPlane and supplements the set with a recent list of pingable addresses [56]. Every VP issues a record route ping to every address in the set. For each address, the system determines the set of VPs that were near enough to discover reverse hops.

Reverse traceroute uses this information in two ways. First, if it encounters one of the probed addresses, it knows the nearest VP to use. Second, if it encounter a new address, the offline probes provide a hint: the group of VPs within range of some addresses in the same prefix as the target might be within range of the target. Given a new address in some prefix, reverse traceroute first tries the vantage point that was within record route range of the most atlas addresses from that prefix. In the case of multiple vantage points both within range of the maximal number, reverse traceroute favors the one that measured the most reverse hops overall from addresses in the prefix (was closest in aggregate to the prefix). However, since some prefixes span many addresses and include many routers, a different vantage point may ingress into the prefix in a different location and be close to a different set of addresses than the first vantage point is close to. So, reverse traceroute next tries the vantage point that is within record route range of the most addresses in the prefix that were not close to the first vantage point. The system continues selecting vantage points in this manner until it has selected at least one near to every iPlane address in the prefix. In this way, reverse traceroute generates the ordered list of vantage points to try when probing a new address in the prefix. Note that reverse traceroute's selection process is an instance of the standard greedy algorithm for the well-known set cover optimization problem – for each prefix, each vantage point defines a set consisting of the addresses near it.

For a representative day, Figure 4.8 shows the coverage achieved by given numbers of VPs per prefix. The system determines the covering VPs for all prefixes, but the graph only includes prefixes for which it probed at least 15 addresses, as it is trivial to cover small prefixes. The graph shows that, for most prefixes, reverse traceroute only needs a small number of VPs. For example, in the median case, a single VP suffices for over 95% of addresses in the prefix, and the system rarely needs more than 4 VPs to cover the entire prefix.

Because only measurements from nearby vantage points yield reverse hops, reverse traceroute sets the initial TTL to 11 for all RR and spoofed RR probes. Probes from distant vantage points will

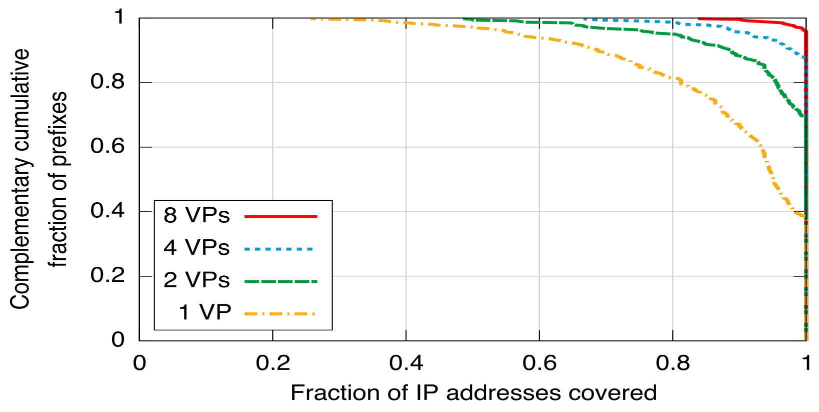


Figure 4.8: For prefixes in which iPlane observed ≥ 15 addresses within RR range of at least one vantage point, the fraction of the addresses for which reverse traceroute can find reverse hops using RR probes from a given number of vantage points per prefix. Reverse traceroute chooses the vantage points greedily on a per-prefix bases, so each additional one is within range of as many new addresses in the prefix as possible. Prefixes with few addresses are trivial to cover using a small number of vantage points, so the graph excludes them to clearly show that reverse traceroute still only needs a small number for most prefixes.

expire before reaching the destination, reducing the effects of rate limiting. I use a value of 11 to account for the fact that record route and TTL hop counts may differ from each other [115].

4.3 System Implementation

The reverse traceroute system consists of vantage points (VPs), which issue measurements, a controller, which coordinates the VPs to measure reverse paths, and sources, which request paths to them. I use a local machine at the University of Washington as a controller. When a VP starts up, it registers with the controller, which can then send it probe requests. A source runs my software to issue standard traceroutes, *RR-Pings*, and *TS-Query-Pings*, and to receive responses to probes from VPs spoofing as the source. However, it need not spoof packets itself. Currently, the reverse traceroute source software only runs on Linux and (like the ICMP traceroute option `traceroute -I`) requires superuser permission. The controller receives requests from sources and combines the measurements from VPs to report reverse path information. This centralized controller carefully controls the rate at which reverse traceroute probes any particular router. It also reuses measurements when a particular source requests multiple destinations.

Reverse traceroute uses topology maps from iPlane [58] to identify adjacent routers to test with *TS-Query-Pings* (Fig. 4.3(e)). iPlane issues daily forward traceroutes from PlanetLab sites and traceroute servers to around 140,000 prefixes. To increase the set of possible next-hop routers, I consider the topology to be the union of maps from the previous 2 weeks. Since reverse traceroute verifies the reverse hops using option-enabled pings, stale topology information makes my system less efficient but does not introduce error.

4.3.1 Current Deployment

The reverse traceroute deployment in this dissertation uses one host at each of the more than 200 active PlanetLab sites as VPs to build an atlas of traceroutes to a source (Fig. 4.3(a)). Over the course of the study below, more than 60 PlanetLab sites allowed spoofed probes at least some of the time.[5] My implementation of reverse traceroute employs one host at each of these sites as spoofing nodes. Routers upstream from the other PlanetLab sites filter spoofed probes, so I did not spoof from these vantage points. Reverse traceroute also uses one host at each of 14 Measurement Lab sites [85] that allow spoofing. Various organizations provide public web-accessible traceroute servers, and reverse traceroute employs 1200 of them [8]. These nodes issue only traceroutes and cannot set IP options, spoof, or receive spoofed probes. Like PlanetLab nodes that do not spoof, reverse traceroute uses them to map paths to sources requesting reverse path measurements.

The system maintains an atlas of these traceroute paths from all vantage points to all recent sources, with the measurements refreshing at a slow rate in the background over time. If a requested reverse traceroute intersects an overly stale atlas traceroute to the requesting source, the system refreshes that traceroute to make sure it still traverses the intersection point. Some areas of the Internet are stable for longer periods of time, and applications like real-time anomaly detection

[5]As of the writing of this dissertation, nearly twice as many allow spoofing, and so the coverage and accuracy of the system have likely improved.

require fresher paths than applications like topology discovery. Calculating appropriate adaptive staleness intervals is an area for future research. For now, the system uses a single interval. For the studies in this chapter, I issued all traceroutes in a batch at the beginning of the study (including the traceroutes to which I compare reverse traceroute measurements) and consider them up-to-date for the duration of the studies. Most paths on the Internet are stable for long periods of time [97, 137], and so I do not expect this simplification to have a large effect on my results.

I have currently tested my client software only on Linux machines. Packaging the code for widespread deployment is future work. Because I have dozens of operators asking to use the system, I am being patient to avoid a launch that does not perform up to expectations. In Section 7.3.2, I discuss these future plans.

4.4 Evaluation

To test how well the reverse traceroute system can determine reverse paths, I consider evaluation settings that allow me to compare a reverse traceroute from *D* to *S* to a direct traceroute from *D* to *S*. A complete evaluation of the accuracy of reverse traceroute would require ground truth information about the path back from the destination. Obviously, I lack ground truth for the Internet, but I use two datasets, one PlanetLab-based and one using public traceroute servers, that allow me to compare to a traceroute from *D*. For the reverse traceroute, I assume the system does not control *D* and must measure the path using the techniques described in this chapter. For the direct traceroute comparison, I simply issue a standard traceroute from *D* to *S*.

For the PlanetLab dataset, I employed as sources a host at each of 11 PlanetLab sites chosen at random from the spoofing nodes. As destinations, I used one host at each of the 200 non-spoofing PlanetLab sites that were working. Although this small number of destinations cannot be representative of the entire Internet, the destinations include hosts in 35 countries. The measured reverse paths traversed 13 of the 14 transit-free commercial ASes. Previous work observed route load balancing in many of these types of networks [6], providing a good test for the reverse traceroute techniques.

For the traceroute server dataset, I employed as sources a host at 10 of the same PlanetLab sites (one had gone down in the meantime). The 1200 traceroute servers I utilized belong to 186 different networks (many of which offer multiple traceroute servers at different locations). For each source, I chose a traceroute server at random from each of the 186 networks. I then issued a traceroute from the server to the PlanetLab source. Because in many cases I did not know the IP address of the traceroute server, I used the first hop along its path as the destination in the reverse traceroute measurements. When measuring a reverse traceroute from this destination back to the source, I excluded from reverse traceroute all traceroute servers in the same network, to avoid providing it with an unfair advantage. There are tens of thousands of ASes and hundreds of thousands of routable prefixes, so even 1200 traceroute servers do not cover the entire routing space.

One challenge when comparing reverse traceroute and traceroute is that the two techniques may find different IP addresses on the same router. As explained in Section 2.1, routers have multiple IP addresses, known as aliases. Traceroute generally finds the ingress interface, whereas record route often returns the egress or loopback address [116]. To account for this, I consider a traceroute hop and a reverse traceroute hop to be the same if they are aliases for the same router. I use alias data from topology mapping projects [62, 78, 115] and aliases I discovered using a state-of-the-art technique [11]. However, alias resolution is an active and challenging research area, and false aliases in the data could lead to falsely labeling two different routers as equivalent. Conversely, missing aliases could cause me to label as different two interfaces on the same router. Because reverse traceroute uses alias data based on measurements from PlanetLab, the alias data for the PlanetLab dataset is likely more complete than the alias data for the traceroute server dataset. By relying on existing alias resolution techniques, I intend the comparison to be conservative – I only count two IP addresses as matching if alias datasets identify them as belonging to the same router.

4.4.1 Accuracy

How similar are the hops on a reverse traceroute to a direct traceroute from the destination back to the source? For the PlanetLab dataset, the *RevTR* line in Figure 4.9 depicts the fraction of hops seen on the direct traceroute that are also seen by reverse traceroute. Figure 4.10 shows the same for the traceroute server dataset. Note that, outside of this experimental setting, I would not normally have access to the direct traceroute from the destination.

Using the available alias data, I find that the paths measured by my technique are quite similar to those seen by traceroute. In the median (mean) case, reverse traceroute measured 87% (83%) of the hops in the traceroute for the PlanetLab dataset. For the traceroute server dataset, reverse traceroute measured 75% (74%) of the hops in the direct traceroute, but 28% (29%) of the the hops discovered by reverse traceroute do not appear in the corresponding traceroute.

The figures also compare reverse traceroute to other potential ways of estimating the reverse path. All techniques used the same alias resolution. Researchers often (sometimes implicitly) assume symmetry, and operators likewise rely on forward path measurements when they need reverse ones. The *Guess Fwd* lines depict how many of the hops seen in a traceroute from *D* to *S* are also seen in a traceroute from *S* to *D*. In the median (mean) case, the forward path shared 38% (39%) of the reverse path's hops for the PlanetLab dataset and 40% (43%) for the traceroute server dataset.

Another approach would be to measure traceroutes from a set of vantage points to the source. Using iPlane's PlanetLab and traceroute server measurements, the *Intersect TR* line in Figure 4.9 shows how well this approach works, by assuming the reverse path is the same as the forward path until it intersects one of the traceroutes[6]. No system today performs this type of path intersection on-demand for users. When performing the intersection, I exclude any traceroutes from the AS hosting the destination, because in general most networks do not host traceroute servers. For the PlanetLab dataset, in the median (mean) case, this traceroute intersection shared 69% (67%) of the actual traceroute's hops. This result suggests that simply having a few hundred or thousand traceroute vantage points is not enough to reliably infer reverse paths. My system uses novel measurement techniques to build off these traceroutes and achieve much better results.

What are the causes of differences between a reverse traceroute and a directly measured traceroute? Although it is common to think of the path given by traceroute as the true path, in reality it is

[6]I omit the line from Figure 4.10 to avoid clutter.

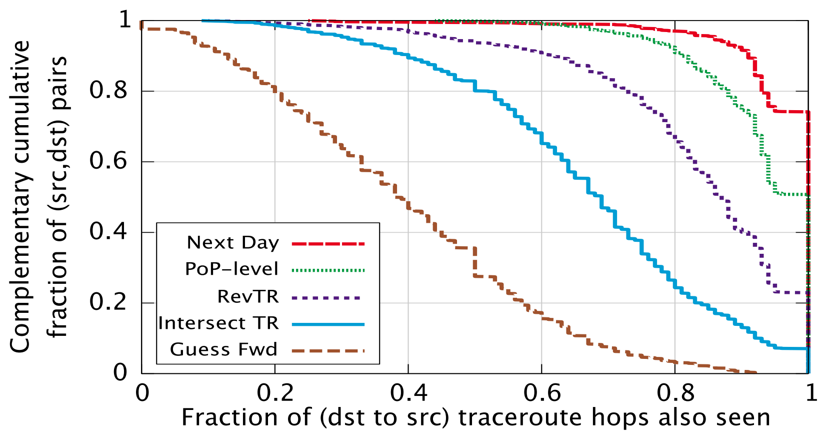


Figure 4.9: For the PlanetLab dataset, the fraction of hops on a direct traceroute from the destination to the source that reverse traceroute and other techniques also discover. Key labels are in the same top-to-bottom order as the lines.

also subject to measurement error. In this section, I discuss reasons traceroute and reverse traceroute may differ from each other and/or from the true path taken. As a starting point, I present results of a study demonstrating that it can be difficult to compare measurements made using different techniques. From each of the destinations in the PlanetLab dataset, I issued both a traceroute and an RR ping to each of the 11 used as sources, so the RR ping had the same source and destination as the traceroute (unlike with reverse traceroute's RR probes to intermediate routers). Since the RR slots could fill up before the probe reaches the destination, I only check if the traceroute matches the portion of the path that appeared in the RR. After alias resolution, the median fraction of RR hops seen in the corresponding traceroute is 0.67, with the other factors described in this section accounting for the differences. This fraction is 0.2 lower than that for reverse traceroute, even though the traceroute and RR probes are sent from the same source to the same destination, showing the difficulty in matching RR hops to traceroute hops.

Incomplete alias information: This lower fraction suggests that many of the differences between the paths found by reverse traceroute and those found by traceroute are due to missing alias information. Most alias identification relies on sending probes to the two IP addresses and comparing the IP-IDs of the responses, to see if they likely came from a shared counter [11]. For the PlanetLab dataset, of all the *missing* addresses seen in a traceroute that are not aliases of any hop in the corresponding

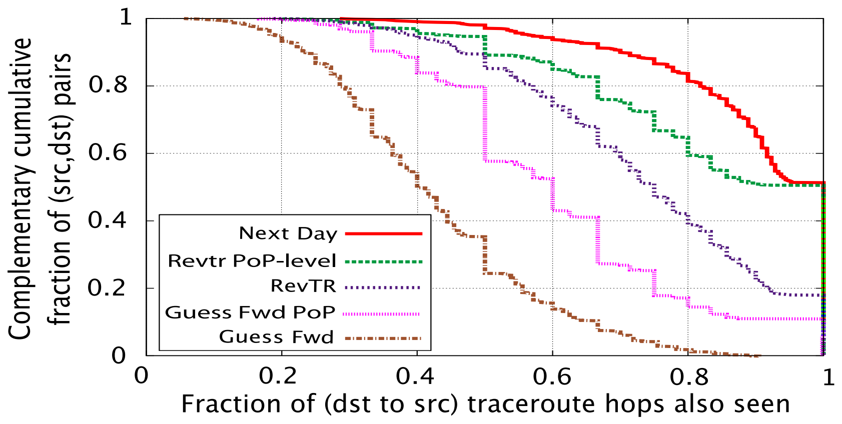


Figure 4.10: For the traceroute server dataset, the fraction of hops on a direct traceroute from the destination to the source that reverse traceroute and other techniques also discover. Key labels are in the same top-to-bottom order as the lines.

reverse traceroute, 88% do not allow for this type of alias resolution [11]. Similarly, of all *extra* addresses seen in some reverse traceroute that are not aliases of any hop in the corresponding reverse traceroute, 82% do not allow for alias resolution. For the traceroute server dataset, 75% of the missing addresses and 74% of the extra ones do not allow it. Even for addresses that do respond to alias techniques, reverse traceroute's alias sets are likely incomplete.

In these cases, it is possible or even likely that the two measurement techniques observed IP addresses that appear different but are in fact aliases of the same router. To partially examine how this lack of alias information limits the comparison, I use iPlane's Point-of-Presence (PoP) clustering, which maps IP addresses to PoPs defined by (AS,city) pairs [78]. For many applications such as the fault location system described in Chapter 5 and the diagnosis of inflated latencies [66], PoP level granularity suffices. iPlane has PoP mappings for 71% of the missing addresses in the PlanetLab dataset and 77% of the extra ones. For the traceroute server dataset, for which I have less alias information, it has mappings for 79% of the missing addresses and 86% of the extra ones. Figures 4.9 and 4.10 include *PoP-Level* lines showing the fraction of traceroute hops seen by reverse traceroute, if I consider PoP rather than router-alias-level comparison. For the PoP-level comparisons, I use the iPlane data to map the IP addresses on the paths to PoPs. If multiple adjacent IP addresses on a path map to the same PoP, I merge them. If an IP address is not in iPlane's PoP mappings, I default to using alias comparisons for that particular hop. After mapping the addresses, I calculate the fraction of PoPs (and unmapped aliases) on the direct traceroute that are also on the corresponding reverse traceroute. In the median case, the reverse traceroute included all the traceroute PoPs in both graphs (mean=94%, 84%), and, overall, most of the remaining unmatched hops are ones for which iPlane lacks PoP mappings. If reverse traceroute were measuring a different path than traceroute, then one would expect PoP-level comparisons to differ about as much as alias-level ones. The implication of the measured PoP-level similarity is that, when traceroute and reverse traceroute differ, they usually differ only in which router or interface in a PoP the path traverses. In these cases, reverse traceroute is measuring the correct path, but the alias-level accuracy metric undercounts. As a point of comparison, Figure 4.10 includes a PoP-level version of the *Guess Fwd* line. In the median (mean) case, it includes only 60% (61%) of the PoPs. The paths are quite asymmetric even at the PoP granularity.

Assumptions of symmetry: When reverse traceroute is unable to identify the next reverse hop, it resorts to assuming that hop is symmetric. Reverse traceroute will be unable to measure a reverse hop if the system's traceroute, record route, and timestamp techniques all fail. Each could fail either because of routers that are configured not to respond or because the system lacked sufficient coverage (an intersecting traceroute, a spoofing vantage point near the target, or knowledge of the correct next hop to test with timestamp). The assumption of symmetry may lead to inaccuracies. For the PlanetLab dataset, if I consider only cases when reverse traceroute measured a complete path without assuming symmetry, in the median (mean) case reverse traceroute matched 93% (90%) of the traceroute alias-level hops. Similarly, for the traceroute server dataset, in the median (mean) case reverse traceroute found 83% (81%) of the traceroute hops. I discuss how often reverse traceroute has to assume symmetry in Section 4.4.2.

Load-balancing and contemporaneous path changes: Another measurement artifact is that traceroute and reverse traceroute may uncover different, but equally valid, paths, either due to following different load-balanced options or due to route changes during measurement. To partly capture these effects, the *Next Day* lines in Figure 4.9 and 4.10 compare how many of the traceroutes' hops are also on traceroutes issued the following day. For the PlanetLab dataset, 26% of the paths exhibit some router-level variation from day to day. For the traceroute dataset, 49% of paths changed at the router level and (not shown in the graph) 15% changed at the PoP-level. Even the same measurement issued at a different time may yield a different path, and reverse traceroute and traceroute measurements were issued hours apart.

Hidden or anonymous routers: Previous work comparing traceroute to record route paths found that 16% of IP addresses appear with RR but do not appear in traceroutes [116, 115]. *Hidden* routers, such as those inside some MPLS tunnels, do not decrement TTL. *Anonymous* routers decrement TTL but do not send ICMP replies, appearing as '*' in traceroute.

Exceptional handling of options packets: Packets with IP options are generally diverted to a router's route processor and may be processed differently than on the line card. For example, previous work suggests that packets with options are load-balanced per-packet, rather than per-flow [115].

An additional source of discrepancies between the two techniques is that traceroute and reverse traceroute make different assumptions about routing. Reverse traceroute's techniques assume destination-based routing – if the path from *D* to *S* passes through *R*, from that point on it is the same as *R*'s path to *S*. An options packet reports only links it actually traversed. With traceroute, on the other hand, a different packet uncovers each hop, and it assumes that if *R1* is at hop *k* and *R2* is at hop *k+1*, then there is a link *R1–R2*. However, it does not make the same assumption about destination routing, as each probe uses *(S,D)* as the source and destination. These differing assumptions lead to two more causes of discrepancies between a traceroute and a reverse traceroute:

Traceroute inferring false links: Although I use the Paris traceroute technique for accurate traversal of flow-based load balancers, it can still infer false links in the case of packet-based load balancing [6]. These spurious links appear as discrepancies between traceroute and reverse traceroute, but in reality show a limitation of traceroute.

Exceptions to destination-based routing: With many tunnels, an option-enabled probe will see the entire tunnel as a single hop. With certain tunnels, however, reverse traceroute's assumption of destination-based routing may not hold. When probed directly, an intermediate router inside the tunnel may use a path to the destination other than the one that continues through the tunnel. I conducted a study to show that, at least on the small dataset in question, few paths show violations of destination-based routing. To do so, I adapted the methodology of an earlier technique to measure how long it takes routers to generate packets [44]. From a host at each of 46 PlanetLab and Measurement Lab sites that allowed spoofing, I sent an RR ping to a host at the University of Washington (UW). Each spoofing host V_i yielded a set of hops $H_{(i,j)}$ for $1 \leq j \leq 9$ (assuming UW was at least 9 hops from V_i). This probe is the ground truth path from V_i to UW. For each such hop $H_{(i,j)}$, V_i then sent an RR ping to $H_{(i,j)}$ spoofing as UW. This probe is similar to the direct probes to routers that reverse traceroute uses to measure reverse hops. If routers use destination-based routing, then, from V_i onward, the direct RR ping from V_i to UW should follow the same path as this probe. This study includes 163 unique hop IP addresses and 230 unique $(V_i, H_{(i,j)})$ pairs.

The assumption of destination-based routing usually holds. I compared the next hop after $H_{(i,j)}$ seen by the two measurements. If V_i uses destination-based routing, the spoofed probe should also return $H_{(i,j+1)}$ (or one of its aliases). Overall, 223 of 230 (97%) of $(V_i, H_{(i,j)})$ pairs had the same next hop. Of the seven that differed, five were within Cogent's network, and the discrepancies were due to either missing alias data or due to the paths going through parallel routers in the same PoP. Further, of the seven, only one was different for more than the first hop.

I also calculated the edit distance between each pair of probes. Overall, 179 (77%) of paths had an edit distance of 0 (were identical), and 18% had an edit distance of 1. Only nine cases had an edit distance of two, and one case had an edit distance of four. Of the ten instances with edit distance greater that one, eight involved either Cogent's JFK PoP or differences within the UW campus. By examination, missing alias data and campus load balancing explained the UW cases. The Cogent cases and the final two cases either were due to missing alias data or may have differed by a single router.

At least over this dataset, reverse traceroute's assumption of destination-based routing seems generally valid.

4.4.2 Coverage

What fraction of hops does reverse traceroute measure without assuming symmetry? In Section 4.4.1, I noted that reverse traceroute's paths are more accurate when the techniques succeed in

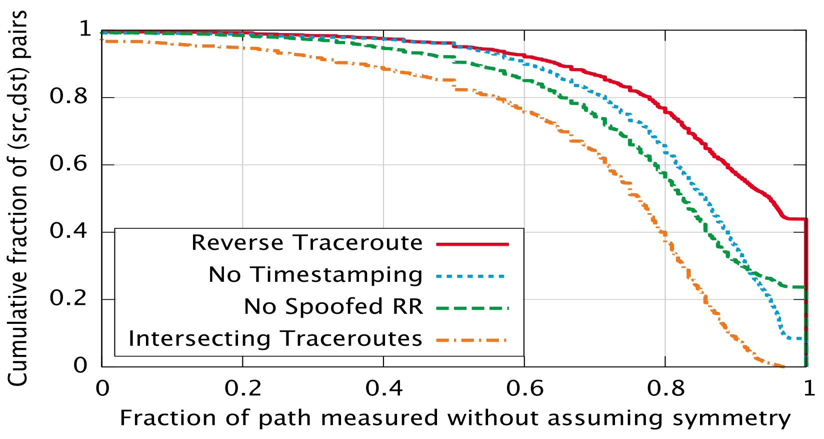


Figure 4.11: For the PlanetLab dataset, the fraction of reverse path hops measured, rather than assumed symmetric. The graph includes results with subsets of the reverse traceroute techniques.

measuring the entire path without having to fall back to assuming a link is symmetric. As seen in Figure 4.9, if forced to assume the entire path is symmetric, in the median case we would discover only 39% of the hops on a traceroute. In this section, I investigate how often reverse traceroute's techniques are able to measure reverse hops, keeping it from reverting to assumptions of symmetry. Using the PlanetLab dataset, Figure 4.11 presents the results for my complete technique, as well as for various combinations of the components of the reverse traceroute technique. The graph shows the fraction of hops in the reverse traceroute that were measured, rather than assumed symmetric.

Reverse traceroute finds most hops without assuming symmetry. In the median path in the PlanetLab dataset, reverse traceroute measured 95% of hops (mean=87%), and in 80% of cases it was able to measure at least 78% of the path without assuming any hops were symmetric. By contrast, the traceroute intersection estimation technique from Figure 4.9 assumes in the median that the last 25% of the path is symmetric (mean=32%). Although not shown in the graph, the results are similar for the traceroute server dataset – in the median case, reverse traceroute measured 95% of hops (mean=92%) without assuming symmetry.

The graph also depicts the performance of reverse traceroute if it does not use spoofed record route pings or does not use timestamping. In both cases, the performance dropped off somewhat.

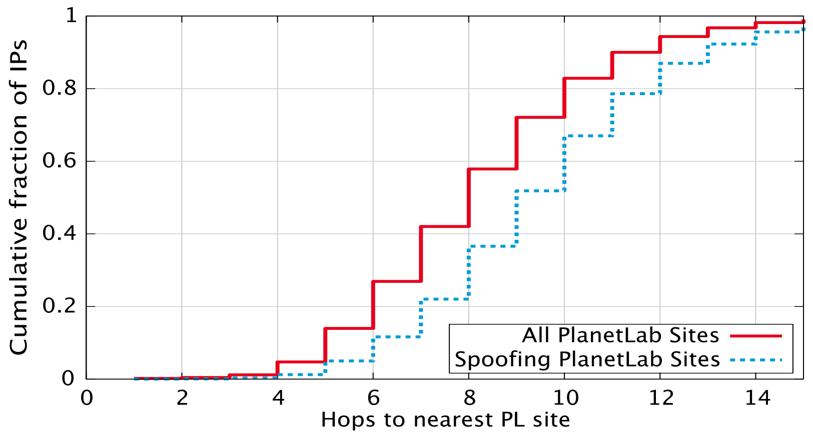


Figure 4.12: For IP addresses seen in iPlane's daily Internet map, CDF of number of hops to nearest PlanetLab site. Note that this graph does not include the Measurement Lab nodes used by reverse traceroute, as they were unavailable at the time.

What fraction of routers are within record route range of reverse traceroute vantage points? For reverse traceroute's spoofed record route technique to be effective, its spoofing vantage points must be near its targets, so that probes will reach their destinations before the RR slots fill. I now show that our set of vantage points is distributed enough to be useful for many paths.

I refer the reader to related work for a detailed evaluation of record route and the challenges in using it [116, 115]. I focus on how likely reverse traceroute is to have vantage points within eight hops of nodes it wants to probe. I consider all addresses observed in one of iPlane's daily Internet maps [78], which include paths from all PlanetLab sites and 500 traceroute servers to destinations in 140,000 prefixes. I issued traceroutes from all PlanetLab sites to all the iPlane addresses, including those observed as intermediate hops on iPlane's traceroutes to its destinations. With reverse traceroute's spoofed RR technique, the key is whether a spoofing vantage point is near the destination or intermediate router, not whether the source is, so these results help characterize the coverage of my system when asked for paths to arbitrary sources. Figure 4.12 shows the percentage of iPlane IP addresses within a given number of hops of a reverse traceroute vantage point. To discover reverse hops, either a spoofing node or the source must be within eight record route hops of the destination, and the closer the node, the more reverse hops discovered with a single probe. To estimate the vantage points' coverage, I analyzed the traceroutes to iPlane addresses and found that 37% of addresses are within eight hops of at least one spoofing node.[7] The graph also presents the traceroute distance to the nearest PlanetLab site, regardless of whether it can spoof. If all PlanetLab sites were to allow spoofing, reverse traceroute would be within eight hops of 58% of all addresses in the dataset, which could lead to a corresponding increase in accuracy. Since some routers respond to traceroute but not record route, and vice versa [115], this measure is not perfect, but it gives a reasonable estimate of PlanetLab coverage.

What fraction of routers support the IP timestamp option? Whereas previous studies assessed support of the record route option [115], little was known about the timestamp option. Although timestamps are a standard option in the IP specification, many routers do not honor them and some ASes filter them. To assess how many routers honor such requests, I performed a measurement study of all IP addresses discovered by iPlane on May 10, 2010 [78]. To generate the data set, I gathered all 351,214 IP addresses observed in iPlane traceroutes from that day, then removed all private addresses and addresses in prefixes whose operators opted out of our experiments. I then issued (options-free) pings to each remaining address and retained the 267,736 addresses that responded. Note that some of these addresses are aliases of each other. I did not attempt to correct the data for differing numbers of interfaces on each router.

For these 267,736 addresses, I sent *TS-Query-Ping*$(S \rightarrow D|D, X)$, to test whether D timestamps and whether it records extra stamps. To attempt to avoid filters, I resent the probes from multiple PlanetLab vantage points. Further, to account for packet loss, I sent each measurement five times redundantly from each vantage point. As seen in Table 4.2, I did not receive any responses from 31% of addresses. An additional 15.5% of addresses responded to the probes without recording a timestamp value. As mentioned in Section 4.2.1, 5.8% of addresses exhibited a common faulty implementation, in which the router recorded two stamps on encountering its address, even though the second requested address did not belong to it. The table lists this as *Extra Stamp* behavior. The remaining 47.7% of addresses correctly responded to these probes with a single timestamp.

[7]The results in this study should be considered a lower bound on reverse traceroute's RR coverage. It does not include Measurement Lab nodes; including them would improve the coverage, especially since they tend to be close to the core of the Internet. The number of spoofing PlanetLab sites has also increased significantly since I conducted this study.

Table 4.2: Responsiveness to timestamp probes for a set of 267,736 public addresses discovered by iPlane on May 10, 2010. PlanetLab sites sent each address D probes requesting *TS-Query-Ping*$(S \rightarrow D|D, X)$, where X is an IP address known not to be on any of the paths. The table classifies how many addresses D did not respond, how many responded without stamping, how many stamped just for D, and how many stamped for both D and (incorrectly) for X.

Classification	IP Addresses	%
Stamps Properly	127706	47.7%
Unresponsive	83002	31.0%
Responsive, Zero Stamps	41422	15.5%
Extra Stamp	15606	5.8%
Total	267736	100%

4.4.3 Overhead

I assess the overhead of reverse traceroute using the traceroute server dataset from Section 4.4.1, comparing the time and number of probes required by the system to those required by traceroute.

How long does it take to measure a reverse traceroute? The median (mean) time for one of the 10 PlanetLab sites to issue a traceroute to one of the 186 traceroute servers was 5 seconds (9.4 seconds). At the time of these experiments, using the system as described in Section 4.3, the median (mean) time to measure a reverse traceroute was 41 seconds (116.0 seconds), including the time to send an initial forward traceroute (to determine if the destination is reachable and to present a round-trip path at the end).

How many probes does it take to measure a reverse traceroute? For each reverse traceroute measurement, the system sends the initial forward traceroute and a number of options-enabled ping packets, some of which may be spoofed. In cases when it is unable to determine the next reverse hop, it sends a forward traceroute and assumes the last hop is symmetric. In addition, reverse traceroute requires traceroutes to build an atlas of paths to the source, and it uses ongoing background mapping to identify adjacencies and to determine which vantage points are within RR-range of which prefixes.

If I ignore the probing overhead of the traceroute atlas and the mapping, in the median case, the only traceroute required was the initial one (mean=1.2 traceroutes). In the median (mean) case, a reverse traceroute required 2 (2.6) record route packets, plus an additional 9 (21.2) spoofed RR packets. The median (mean) number of non-spoofed timestamp packets was 0 (5.1), and the median (mean) number of spoofed timestamp packets was also 0 (6.5). The median (mean) total number of options packets sent was 13 (35.4). As a point of comparison, traceroute uses around 45 probe packets on average, 3 for each of around 15 hops. At the end of a reverse traceroute, reverse traceroute also sends 3 pings to each hop to measure latency. So, ignoring the creation of the various atlases, reverse traceroute generally requires roughly two to three times as many packets as traceroute.

The atlases represent the majority of the reverse traceroute probe overhead. Reverse traceroute borrows the adjacency information needed for its timestamp probes from an existing mapping service [78]. To determine which spoofing vantage points are likely within record route range of a destination, reverse traceroute regularly issue probes from every spoofer to a set of addresses in each prefix. In the future, I plan to investigate if I can reduce this overhead by probing only a single address within each prefix.

In many circumstances these atlases can be reused and/or optimized for performance. For example, if the source requests reverse paths for multiple destinations within a short period of time [137], reverse traceroute can reuse the atlas. As an optimization, it may need to only issue those traceroutes that are likely to intersect [78], and it can use known techniques to reduce the number of probes to generate the atlas [34]. While the results in this section focus on the overhead of measuring a single reverse traceroute, Section 5.4.4 presents basic scalability results from a reverse traceroute deployment designed to regularly refresh the reverse paths back from a set of destinations to a set of sources. By reusing probe results across paths and by focusing on refreshing a stale path instead of starting from scratch, that deployment greatly reduces the number of probes required to maintain the traceroute atlas and measure paths. In Section 7.3.2, I propose ideas to further reduce the time and number of probes required by reverse traceroute in the future.

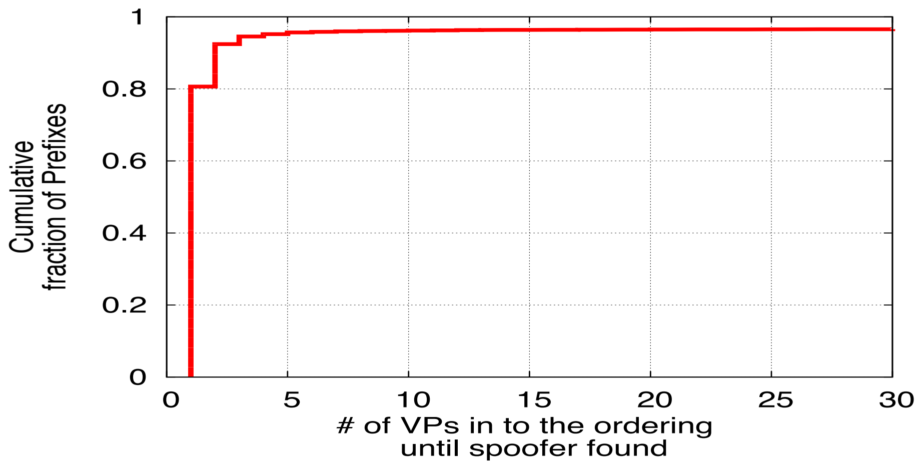


Figure 4.13: For iPlane destinations that are not in DisCarte training data, number of vantage points reverse traceroute needs to send RR pings from before discovering a reverse hop.

Does reverse traceroute efficiently find which vantage points to use? In Section 4.2.4, I described how reverse traceroute uses background training measurements to determine the order of vantage points to spoof from for a particular prefix, and I showed that it took few vantage points to cover a set of training addresses. Now, I demonstrate that the sets learned on a training set allow reverse traceroute to determine quickly which vantage points to use to probe new addresses. I first generated the per-prefix orders based on probes I issued from reverse traceroute's vantage points to addresses in the DisCarte topology. I then found the set of 7823 iPlane destinations that were not in the DisCarte topology, but were in prefixes that appeared in the DisCarte topology. For each destination, I mapped it to its prefix, then used the greedy set cover orderings for that prefix to issue one record route probe at a time until I discovered a reverse hop. As seen in Figure 4.13, for 80% of destinations I discovered a reverse hop after only a single probe, for more than 90% I needed at most 2 vantage points, and there was very little benefit from using more than 5 VPs. For any destination for which the orderings did not produce a reverse hop, I tried the vantage points which were not part of that prefix's set cover solution. I found that, for 97% of the iPlane targets within range of some vantage point, reverse traceroute's set cover orderings suggested a vantage point capable of finding a reverse hop. Reverse traceroute's technique for selecting vantage points allows it to quickly discover these hops without sending excessive probes.

Table 4.3: Traceroute giving forward path from University of Central Florida to *66.79.146.129*. The path starts in Florida, goes up to Washington, DC, then comes back down to Florida before going across the country. However, this routing does not explain the jump in latency between the last two hops, suggesting an indirect reverse path.

Hop #	IP address	AS name	Location	RTT
1	132.170.3.1	UCF	Orlando, FL	0ms
2	198.32.155.89	FloridaNet	Orlando, FL	0ms
3	198.32.132.64	FloridaNet	Jacksonville, FL	3ms
4	198.32.132.19	Cox Comm.	Atlanta, GA	9ms
5	68.1.0.221	Cox Comm.	Ashburn, VA	116ms
6	216.52.127.8	Internap	Washington, DC	35ms
7	66.79.151.129	Internap	Washington, DC	26ms
8	66.79.146.202	Internap	Washington, DC	24ms
9	66.79.146.241	Internap	Miami, FL	53ms
10	66.79.146.129	Internap	Seattle, WA	149ms

4.5 Debugging Poor Performance with Reverse Traceroute

As explained in Section 2.4, providers like Google want to optimize client performance by serving clients along low latency paths, but the lack of information about reverse paths back to Google servers from clients hinders these efforts [66].

To illustrate how reverse traceroute can solve such problems, I next describe one example of how I used reverse traceroute to diagnose an instance of path inflation. The round-trip time (RTT) on the path from the PlanetLab node at the University of Central Florida to the IP address *66.79.146.129*, which is in Seattle, was 149ms. Table 4.3 shows the forward path returned by traceroute, annotated with the locations of intermediate hops inferred from their DNS names. The path has some circuitousness going from Orlando to Washington via Ashburn and then returning to Florida. But, that does not explain the steep rise in RTT from 53ms to 149ms on the last segment of the path, because a hop from Miami to Seattle is expected to only add 70ms to the RTT[8].

To investigate reverse path inflation back from the destination, Table 4.4 gives the reverse path, as measured by reverse traceroute. The reverse path is noticeably circuitous. Starting from Seattle,

[8]Interestingly, the latency to Ashburn seems to also be inflated on the reverse path.

Table 4.4: Reverse traceroute giving reverse path from *66.79.146.129* back to the University of Central Florida. The circuitous reverse path back from Seattle explains the huge RTT between the last two hops on the forward path (Miami and Seattle). The third hop, 137.164.130.66 (internap-peer.lsanca01.transitrail.net), is a peering point between Internap and TransitRail in L.A.

Hop #	IP address	AS name	Location	RTT
1	66.79.146.129	Internap	Seattle, WA	148ms
2	66.79.146.225	Internap	Seattle, WA	141ms
3	137.164.130.66	TransitRail	Los Angeles, CA	118ms
4	137.164.129.15	TransitRail	Los Angeles, CA	118ms
5	137.164.129.34	TransitRail	Palo Alto, CA	109ms
6	137.164.129.2	TransitRail	Seattle, WA	92ms
7	137.164.129.11	TransitRail	Chicago, IL	41ms
8	137.164.131.165	TransitRail	Ashburn, VA	23ms
9	132.170.3.1	UCF	Orlando, FL	0ms
10	132.170.3.33	UCF	Orlando, FL	0ms

the path goes through Los Angeles and Palo Alto, and then returns to Seattle before reaching the destination via Chicago and Ashburn. I verified with a traceroute from a PlanetLab machine at the University of Washington that TransitRail and Internap connect in Seattle, suggesting that the inflation was due to a routing misconfiguration. Private communication with an operator at one of the networks confirmed that the detour through Los Angeles was unintentional. Without the insight into the reverse path provided by reverse traceroute, such investigations would not be possible by the organizations most affected by inflated routes.

4.6 Other Applications of Reverse Traceroute

I believe many other opportunities exist for improving systems and studies using reverse traceroute. As discussed in Section 2.3.3, traceroute has a range of uses, but its lack of reverse path visibility limits its applicability. In this section, I present two further examples of how reverse traceroute can be used in practice. I intend these sections to illustrate a few ways in which one can apply the tool to help understand the Internet; they are not complete studies of the problems.

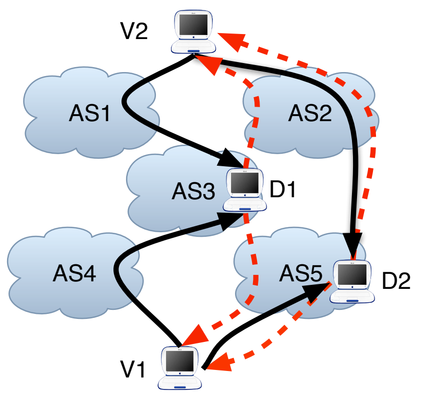


Figure 4.14: Example of reverse traceroute techniques aiding in topology discovery. With traceroutes alone, V1 and V2 can measure only the forward (solid) paths. If V2 is within 8 hops of D1, a record route ping allows it to measure the link AS3-AS2, and a record route ping spoofed as V1 allows it to measure AS3-AS5.

4.6.1 Topology Discovery

Studies of Internet topology rely on the set of available vantage points and data collection points. With a limited number available, routing policies bias what researchers measure. As an example, with traceroute alone, topology discovery is limited to measuring forward paths from a few hundred vantage points to each other and to other destinations. Reverse traceroute allows us to expose many peer-to-peer links invisible to traceroute.

Figure 4.14 illustrates one way in which my techniques can uncover links. Assume that *AS3* has a peer-to-peer business relationship with the other ASes. Because an AS does not want to provide free transit, most routes will traverse at most one peer-to-peer link; this policy is an aspect of valley-free routing. In this example, traffic will traverse one of *AS3*'s peer links only if it is sourced or destined from/to *AS3*. *V1*'s path to *AS3* goes through *AS4*, and *V2*'s path to *AS3* goes through *AS1*. Topology-gathering systems that rely on traceroute alone [78, 5, 111] will observe the links *AS1-AS3*, *AS4-AS3*, and *AS2-AS5*. But, they will never traverse *AS3-AS5*, or *AS3-AS2*, no matter what destinations they probe (even ones not depicted). *V2* can never traverse *AS1-AS3-AS5* in a forward path (assuming standard export policies), because that would traverse two peer-to-peer links. However, if *V2* is within 8 hops of *D1*, then it can issue a record route ping that will reveal *AS3-AS2*, and a spoofed record route (spoofed as *V1*) to reveal *AS3-AS5*.[9]

Furthermore, even services like RouteViews [87] and RIS [105], with BGP feeds from many ASes, offer limited views and likely miss these links. Typical export policies mean that only routers in an AS or its customers see the AS's peer-to-peer links. Since RouteViews has vantage points in only a small percentage of the ASes lower in the AS hierarchy, it does not see most peer links [93, 52].

To demonstrate how reverse traceroute can aid in topology mapping, I apply it to a recent study on mapping Internet exchange points (IXPs) [8]. That study used existing measurements, novel techniques, and thousands of traceroute servers to provide IXP peering matrices that were as complete as possible. As part of the study, the researchers published the list of ASes they found to be peering at IXPs, the IXPs at which they peered, and the IP addresses they used in those peerings.

I measured the reverse paths back from those IP addresses to all PlanetLab sites. I discovered 9096 IXP peerings (triples of the two ASes and the IXP at which they peer) that are not in the published dataset, adding an additional 16% to the 58,534 peerings in their study. As one example, reverse traceroute increased the number of peerings found at the large London LINX exchange by 19%. If we consider just the ASes observed peering and not which IXP they were seen at, reverse traceroute found an additional 5057 AS links not in the 51,832 known IXP AS links, an increase of 10%. Of these AS links, 1910 do not appear in either traceroute [78] or BGP [128] topologies – besides not being known as IXP links, reverse traceroute is discovering links not seen in some of the most complete topologies available. Further, of the links in both my data and UCLA's BGP topology, UCLA classifies 1596 as Customer-to-Provider links, whereas the fact that reverse traceroute observed them at IXPs strongly suggests they are in fact Peer-to-Peer links. Although the recent IXP study was by far the most exhaustive yet, reverse traceroute provides a way to observe even more of the topology.

[9]Note that reverse traceroute's timestamp pings are not useful for topology discovery, because the system only uses them to query for hops already known to be adjacent.

4.6.2 Measuring One-Way Link Latency

In addition to measuring a path, traceroute measures a round-trip latency for each hop. As explained in Section 2.3.3, many research efforts use traceroute to estimate link latencies, even though it cannot accurately measure these delays in the presence of asymmetric routing. Operators also want accurate ways to measure link latencies to aid in troubleshooting [122]. More generally, researchers have proposed combining topology measurements with end-to-end measurements of loss, latency, bandwidth, and other properties, in order to isolate the metrics to individual links [28, 19]. However, this approach, known as network tomography, works best only when the links are traversed symmetrically or when one knows both the forward and reverse paths. Of course, asymmetric routing means that, traditionally, this data has only been available in very limited settings.

In this section, I revisit the problem of estimating link latencies since I now have a tool that provides reverse path information to complement traceroute's forward path information. Given path asymmetry, the reverse paths from intermediate routers likely differ from the end-to-end traceroutes in both directions. Without reverse path information from the intermediate hops back to the hosts, we cannot know which links a round-trip latency includes. Measurements to endpoints and intermediate hops yield a large set of paths, which I simplify using IP address clustering [78]. I then generate a set of linear constraints: for any intermediate hop R observed from a source S, the sum of the link latencies on the path from S to R plus the sum of the link latencies on the path back from R must equal the round-trip latency measured between S and R. I then solve this set of constraints using least-squares minimization, and I also identify the bound and free variables in the solution. Bound variables are those sufficiently constrained to solve for the link latencies, and free variables are those that remain under constrained.

I evaluate this approach on the Sprint backbone network by comparing against inter-PoP latencies Sprint measures and publishes [121]. I consider only the directly connected PoPs and halve the published round-trip times to yield ground-truth link latencies, for 89 links between 42 PoPs. Forward traceroutes observed 61 of the 89 links, and reverse traceroutes found 79. I use these measurements to formulate constraints on the inter-PoP links, based on round-trip latencies measured

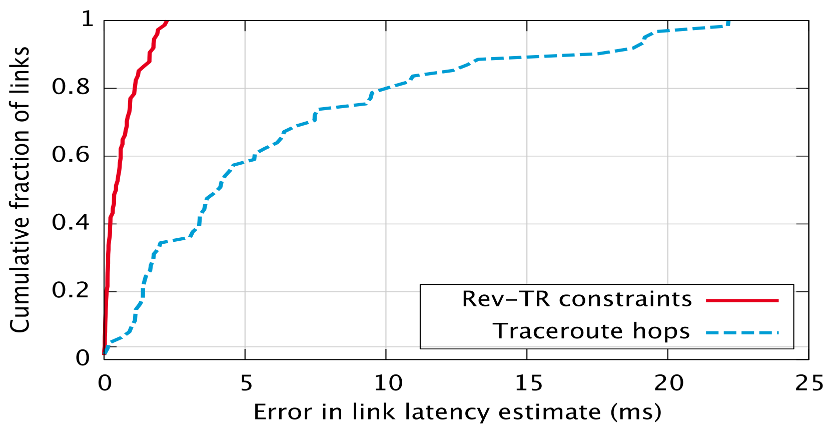


Figure 4.15: Error in estimating latencies for Sprint inter-PoP links. For each technique, I only include links for which it provided an estimate: 61 of 89 links using traceroute, and 74 of 89 using reverse traceroute. Sprint reports its ground truth latencies only to 0.5ms granularity.

from PlanetLab nodes to the PoPs using ping. This set of constraints allows me to solve for the latencies of 74 links, leaving 5 free and 10 unobserved. As a comparison point, I use a traditional method for estimating link latency from traceroutes [78]. For each link in a forward traceroute, I estimated the link latency to be half the difference between the round-trip delay to either end. Then, for each Sprint link, I used the median across all traceroute-derived estimates for that link.

Figure 4.15 shows the error in the latency estimates of the two techniques, compared to the published ground truth. My reverse traceroute-based approach infers link latencies with errors from 0ms to 2.2ms for the links, with a median of 0.4ms and a mean of 0.6ms. Sprint reports round-trip delays with millisecond granularity, so the median "error" is within the precision of the data. The estimation errors using the traditional traceroute method range from 0ms to 22.2ms, with a median of 4.1ms and a mean of 6.2ms – $10x$ reverse traceroute's worst-case, median, and mean errors. For the dataset studied, using reverse traceroute to generate and solve constraints yields values very close to the actual latencies, whereas the traditional approach does not.

4.7 Summary

Although widely-used and popular, traceroute is fundamentally limited in that it cannot measure reverse paths. This limitation leaves network operators and researchers unable to answer important questions about Internet topology and performance. To solve this problem, I developed a reverse traceroute system to measure reverse paths from arbitrary destinations back to the user. The system uses a variety of methods to incrementally build a path back from the destination hop-by-hop, until it reaches a known baseline path. I believe that my system makes a strong argument for both the IP timestamp option and source spoofing as important measurement tools, and I hope that PlanetLab and ASes will consider them valuable components of future measurement testbeds.

The reverse traceroute system is both effective – in the median case finding all of the PoPs seen by a direct traceroute along the same path – and useful. I show that the tool enables investigations impossible with existing tools, such as tracking down performance problems caused by path inflation along a reverse route. The system's probing methods have also proved useful for topology mapping. In illustrative examples, I demonstrated how the system can discover more than a thousand peer-to-peer links invisible to both BGP route collectors and to traceroute-based mapping efforts, as well as how it can be used to accurately measure the latency of backbone links.

Chapter 5

IMPROVING INTERNET AVAILABILITY WITH LIFEGUARD

In Chapter 3, I presented studies showing that the Internet suffers from many long-lasting outages. Monitoring paths from Amazon's EC2 cloud service, we found that, for outages lasting at least 90 seconds, 84% of the unavailability came from those that lasted over 10 minutes. Section 3.5 showed that policy-compliant reroutes often seem to exist but are not being found. Faults should not persist if alternate routes exist, but router configurations are keeping ASes from discovering paths that can successfully forward data.

When an outage keeps a network from sending or receiving some of its traffic, the network would like to restore connectivity. However, the failure may be caused by a problem outside the network, and available protocols and tools give operators little visibility into or control over routing outside their local networks. Operators struggle to obtain the topology and routing information necessary to locate the source of an outage, since measurement tools like traceroute and reverse traceroute require connectivity to complete their measurements.

Even knowing of the location of a failure, operators have limited means to repair or avoid the problem. Traditional techniques for route control give the operators' network direct influence only over routes between it and its immediate neighbors, which may not be enough to avoid a problem in a transit network farther away. Even if one of the techniques does enable rerouting around a particular failure, it may also force networks with working paths to reroute, as the techniques cannot be precisely targeted. Having multiple providers still may not suffice, as the operators have little control over the routes other ASes select to it.

To substantially improve Internet availability, we need a way to combat long-lived failures. I believe that Internet availability would improve if data centers and other well-provisioned edge networks were given the ability to repair persistent routing problems, regardless of which network along the path is responsible for the outage. The edge network is in a position to observe routing disruptions, when it can no longer reach parts of the Internet. If some alternate working policy-compliant path can deliver traffic, the data center or edge network should be able to cause the Internet to use it.

I propose achieving this goal by enabling an edge network to disable routes that traverse a misbehaving network, triggering route exploration to find new paths. While accomplishing this goal might seem to require a redesign of the Internet's protocols, my objective is a system that works today with existing protocols, even if it cannot address all outages. In this chapter, I present the design and implementation of a system that enables rerouting around many long-lasting failures while being deployable on today's Internet. I call my system *LIFEGUARD*, for *Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically*. *LIFEGUARD* aims to automatically repair partial outages in minutes, replacing the manual process that can take hours. Existing approaches often enable an edge AS to avoid problems on its forward paths to destinations but provide little control over the paths back to the AS. *LIFEGUARD* provides reverse path control by having the edge AS *O* insert the problem network *A* into path advertisements for *O*'s addresses, so that it appears that *A* has already been visited. When the announcements reach *A*, BGP's loop-prevention mechanisms will drop the announcement. Networks that would have routed through *A* will only learn of other paths, and will avoid *A*. Using the BGP-Mux testbed [14] to announce paths to the Internet, I show *LIFEGUARD*'s rerouting technique finds alternate paths 76% of the time.

While this BGP *poisoning* provides a means to trigger rerouting, I must address a number of challenges to provide a practical solution. *LIFEGUARD* combines this basic poisoning mechanism with a number of techniques. *LIFEGUARD* has a subsystem to locate failures, even in the presence of asymmetric routing and unidirectional failures. To accurately locate problems, *LIFEGUARD* combines active measurement probes from tools including reverse traceroute, along with historical path and router responsiveness information. My evaluation shows that, even though *LIFEGUARD* controls only one endpoint of a failing path, it can often locate problems as accurately as if it could assess the path from both ends. I validate my failure isolation approach and present experiments suggesting that, lacking *LIFEGUARD*'s novel measurements, the commonly used traceroute technique for failure location gave incorrect information 40% of the time. I address how to decide whether to poison; will routing protocols automatically resolve the problem, or do I need to trigger route exploration? I show empirically that triggering route exploration could eliminate up to 80% of the observed unavailability. When it reroutes failing paths to remediate partial outages, *LIFEGUARD* carefully crafts BGP announcements to speed route convergence and minimize the disruption to working routes. My experimental results show that 94% of working routes reconverge instantly and experience minimal ($\leq$ 2%) packet loss. After rerouting, *LIFEGUARD* maintains a *sentinel prefix* on the original path to detect when the failure has resolved, even though live traffic will be routing over an alternate path. When *LIFEGUARD*'s test traffic reaches the sentinel, *LIFEGUARD* removes the poisoned announcement.

LIFEGUARD can locate and reroute around many failures, but it is not capable of repairing all outages and addressing all the associated issues. I intend the system to serve both as an approach that networks can experiment with today and as a prototype that the community might build upon in the future. By acting as a model, it can highlight both the potential benefits of such a system and the challenges that are better addressed through protocol modifications. In this chapter and in Section 7.3.4, I propose some slight modifications that could enable more systematic approaches to avoiding long-lasting outages.

In Section 5.1, I provide an overview of current approaches and why they do not suffice. Section 5.2 describe both the high level goal in avoiding outages as well my specific approach to restore connectivity. Section 5.3 addresses practical questions, including how *LIFEGUARD* locates problems, how *LIFEGUARD* determines whether to poison a particular problem, and when to stop poisoning it. Section 5.4 presents my evaluation of *LIFEGUARD*. In Section 5.5, I discuss a case study in which *LIFEGUARD* locates a failure and then routes around it. Section 5.6 presents two discussion points. I summarize the chapter in Section 5.7.

5.1 Current Approaches to Locating and Remediating Long-Lasting Failures

Currently, operators rely on insufficient techniques to try to locate and resolve long-lasting outages, especially if the failure is in a network outside the operators' control. The lack of better options to address outages contributes to the duration of these problems. Asymmetric paths leave operators with a limited view even when paths work. Tools like traceroute require bidirectional connectivity to function properly, and so failures restrict their view further. As shown in Figure 2.2, traceroute output can be confusing during failures. Public traceroute servers and route collectors [87, 105] extend the view somewhat, but only a small percentage of networks make them available. As seen in Section 3.2, many outages do not manifest in available feeds. In fact, these challenges mean that operators frequently resort to asking others to issue traceroutes on their behalf to help confirm and isolate a problem [94].

If operators successfully identify a failure outside their own networks, they have little ability to effect repair:

Forward path failures: The source network's operators can select an alternative egress in an attempt to avoid the problem. When choosing, they can see the full BGP paths provided by their neighbors. Each of the source's providers announces its preferred path, and the source is free to choose among them. If the network's providers offer sufficiently diverse paths, the failure may be avoided. For example, I inspected BGP routes from five universities (University of Washington, University of Wisconsin, Georgia Tech, Princeton, and Clemson) [14] to prefixes in 114 ASes. If these universities were my providers, the routes are sufficiently diverse that, if the last AS link before the destination on one of the routes failed silently, I could route around it to reach the destination in 90% of cases by routing via a different provider. In Section 5.4.2, I present an equivalent experiment demonstrating that my techniques I present in this chapter would allow me to avoid 73% of these links on reverse paths back from the 114 ASes, without disturbing routes that did not use that link. [1]

Reverse path failures: Using traditional techniques, however, having multiple providers may not offer much reverse path diversity. Under BGP, the operators can only change how they announce a prefix to neighbors, perhaps announcing it differently to different neighbors. They have no other direct influence over the paths other networks select. In Section 2.2.1, I explained how operators can change announcements to attempt to influence routes towards the prefix. A major limitation of all

[1]The 114 ASes were all those that both announce prefixes visible at the universities, needed for the forward path study, and peer with a route collector [105, 87, 99, 1], needed for the reverse study.

the techniques is that they generally act on the next hop AS, rather than allowing a network to target whichever AS is causing a problem. More specifically, MEDs can only be effective if the problem is in the immediate upstream neighbor. Similarly, a destination can use selective advertising to direct traffic away from one of its providers. If the problem is not in the immediate provider, selective advertising may be deficient in two ways. First, all working routes that had previously gone through that provider will change. Second, even if all sources with failing paths had selected routes through a particular provider before selective advertising, the paths may continue to experience the failure even after selective advertising. Advertising a more-specific prefix is a form of selective advertising and suffers from similar limitations. Prepending is a very blunt mechanism, for reasons akin to those mentioned for selective advertising, with the additional caveat that ASes are free to apply their own local preferences and ignore path length as a selection criterion.

BGP communities are a promising direction for future experiments in failure avoidance but do not currently provide a complete solution. Some ASes offer communities that provide behavior that could be used to avoid routing problems. For example, SAVVIS offers communities to specify that a route should not be exported to a particular peer or that the route should be prepended when exported to the peer. Cogent, meanwhile, does not seem to allow targeting of a particular peer, but it does have communities to restrict export to peers on a given continent. However, communities are not standardized across ASes, and some ASes give limited control over how they disseminate routes.

Further, many ASes do not propagate community values they receive [103], and so communities are not a feasible way to notify arbitrary ASes of routing problems. If most ASes forwarded the communities on announcements they received, then an edge network could use communities like the ones offered by SAVVIS to control some routes, without needing to be directly connected to SAVVIS. To demonstrate that communities do not propagate widely through the Internet, I announced five prefixes with a meaningless (but syntactically valid) community set on them, one from each of five U.S. universities [14], and then inspected RouteViews to see how many ASes had the community still attached to the announcements. Of 35 RouteViews peer ASes, between 0 and 4 had the community, depending on the prefix. In general, any AS that used a Tier-1 to reach a prefix did not have the community on my announcement. Similarly, I looked up my prefixes in Level 3 and ATT Looking Glass servers and found that the community was no longer part of the announcement.

Since other ASes are free to use arbitrary policy to select paths to the prefix, changes to the announcements and to local configuration may be unable to repair outages. In such cases, operators often must resort to phone calls or e-mails asking operators at other networks for support. These slow interactions contribute to the duration of outages. Eventually, an operator may update configuration to address the problem.

However, without systematic monitoring to detect when the underlying problem resolves or automatic measures to undo it at that point, the change may persist long after it is needed. For example, researchers at Google found that a Japanese AS routed traffic to Google in California long after Google had a Japenese PoP, presumably due to legacy configuration dating from when Google was only present in California [66]. The example in Section 4.5, in which traffic went from Seattle to Los Angeles before returning to Seattle, likely reflects a similar out-of-date configuration.

I now describe my approach to enable automatic repair of failing reverse paths.

5.2 Enabling Failure Avoidance

Suppose an AS *O* wants to communicate with another AS *Q* but cannot because of some problem on the path between them. If the problem is within either *O* or *Q*, operators at that network have complete visibility into and control over their local networks, and so they can take appropriate steps to remedy the problem. Instead, consider a case in which the problem occurs somewhere outside of the edge ASes, either on the forward path to *Q* or on the reverse path back to *O*. Further suppose that *O* is able to locate the failure and to determine that an alternate route likely exists.[2]

O would like to restore connectivity regardless of where the problem is, but its ability to do so currently depends largely on where the problem is located. If the problem is on the forward path

[2]I discuss how *LIFEGUARD* does this in Section 5.3.

and *O*'s providers offer suitable path diversity, *O* can choose a path that avoids the problem. By carefully selecting where to locate its PoPs and which providers to contract with, *O* should be able to achieve decent resiliency to forward path failures. However, having a diversity of providers may not help for reverse path failures, as *O* has little control over the routes other ASes select to reach it. As explained in Section 5.1, route control mechanisms like MEDs and selective advertising only let *O* control the PoP or provider through which traffic enters *O*. However, these BGP mechanisms give *O* essentially no control over how other ASes reach the provider it selects.

O needs a way to notify ASes using the path that the path is not successfully forwarding traffic, thereby encouraging them to choose alternate routes that restore connectivity. As a hint as to which paths they should avoid, *O* would like to inform them of the failure location. AS-level failure locations are the proper granularity for these hypothetical notifications, because BGP uses AS-level topology abstractions. In particular, when one of the notified ASes chooses an alternate path, it will be selecting from AS paths announced to it by its neighbors. Therefore, *O* needs to inform other ASes of which AS or AS link to avoid, depending on whether the failure is within a single AS or at an AS boundary.

Ideally, I would like a mechanism to let the origin AS *O* of a prefix *P* specify this information explicitly with a signed announcement I will call AVOID_PROBLEM(X,P). Depending on the nature of the problem, *X* could either be a single AS (AVOID_PROBLEM(A,P)) or an AS link $A - B$ (AVOID_PROBLEM(A-B,P)). Note that AS *O* is only able to directly observe the problem with prefix *P*; it cannot determine if the issue is more widespread. Announcing this hypothetical primitive would have three effects:

- *Avoidance Property*: Any AS that knew of a route to *P* that avoided *X* would select such a route.

- *Backup Property*: Any AS that only knew of a route through *X* would be free to attempt to use it. Similarly, *A* would be able to attempt to route to *O* via its preferred path (through *B* in the case when *X* is the link *A*-*B*).

- *Notification Property*: *A* (and *B*, for link problems) would be notified of the problem, alerting its operators to fix it.

5.2.1 LIFEGUARD's Failure Remediation

Deploying AVOID_PROBLEM(X,P) might seem to require changes to every BGP router in the Internet. Instead, I use mechanisms already available in BGP to perform the notifications, in order to arrive at a solution that is usable today, even if the solution is not complete. A usable approach can improve availability today while simultaneously helping the community understand how we might improve availability further with future BGP changes. I call my approach *LIFEGUARD*, for *Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically*.

To approximate AVOID_PROBLEM(A,P) on today's Internet, *LIFEGUARD* uses BGP's built-in loop prevention to "poison" a problem AS that is announcing routes but not forwarding packets. To poison an AS *A*, the origin announces the prefix with *A* as part of the path, causing *A* to reject the path (to avoid a loop) and withdraw its path from its neighbors [29, 21]. This withdrawal causes ASes that previously routed via *A* to explore alternatives. Importantly, the poison affects only traffic to *O*'s prefix experiencing the problem. By allowing an AS to poison only prefixes it originates, my approach is consistent with the goals of ongoing work that seeks to authenticate the origin of BGP announcements [88]. Proposals to verify the entire path [9] are consistent with the future goal for my approach, in which AVOID_PROBLEM(X,P) would be a validated hint from the origin AS to the rest of the network that a particular AS is not correctly routing its traffic. By the time such proposals are deployed, it may be feasible to develop new routing primitives or standardized BGP communities to accomplish what I currently do with poisoning.

Although BGP loop prevention was not intended to give *O* control over routes in other ASes, it lets me experiment with failure avoidance. In effect, poisoning *A* implements the *Avoidance Property* of AVOID_PROBLEM(A,P), giving *O* the means to control routes to it. *A*'s border routers will receive the poisoned announcement and detect the poison, a form of the *Notification Property*.

On its own, poisoning is a blunt, disruptive instrument, a limitation that *LIFEGUARD* must overcome. Poisoning inserts *A* into all routes, so even ASes that were not routing through *A* may undergo route exploration before reconverging to their original route, leading to unnecessary packet loss [69].

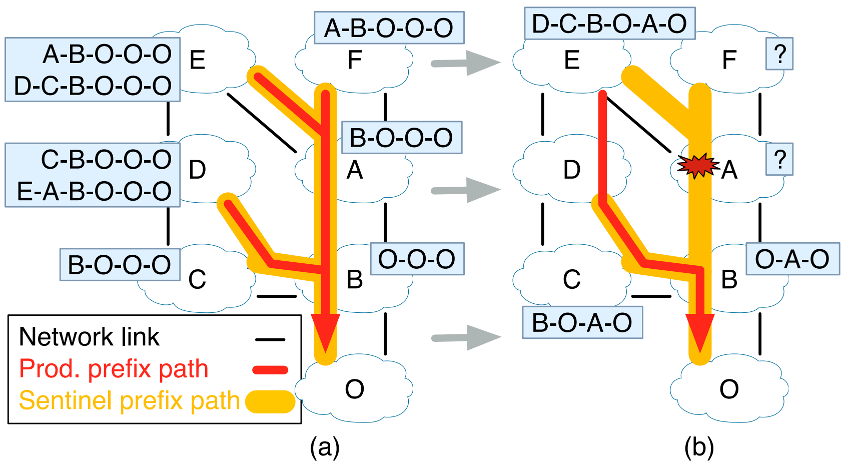


Figure 5.1: Routes and routing tables (a) before and (b) after *O* poisons *A* to avoid a problem. Each table shows *only paths to the production prefix*, with the in-use, most-preferred route at the top. Poisoning *A* for the production prefix causes it to withdraw its route from *E* and *F*, forcing *E* to use its less-preferred route through *D* and leaving *F* with only the sentinel. Routes to the sentinel prefix do not change, allowing *O* to check when the problem has resolved.

Instead of providing the *Backup Property*, poisoning cuts off ASes that lack a route around *A*. Poisoning disables all paths through *A*, even if some work.

In the following sections, I show how *LIFEGUARD* overcomes what initially seem like limitations of poisoning in order to better approximate AVOID_PROBLEM(X,P).

Minimizing Disruption of Working Routes

Inserting an AS to poison an announcement increases AS path length. Suppose that an origin AS *O* decides to poison *A* for *O*'s prefix *P*. The poisoned path cannot be *A*-*O*, because *O*'s neighbors need to route to *O* as their next hop, not to *A*. So, the path must start with *O*. It cannot be *O*-*A*, because routing registries list *O* as the origin for *P*, and so a path that shows *A* as the origin looks suspicious. Therefore, *O* announces *O*-*A*-*O*. Experiments found that BGP normally takes multiple minutes to converge when switching to longer paths, with accompanying packet loss to the prefix during this period [69]. This loss would even affect networks with working paths to the prefix.

To poison in a way that shortens and smooths this convergence period, *LIFEGUARD* crafts steady-state unpoisoned announcements in a way that “prepares” all ASes for the possibility that some AS may later be poisoned. Figure 5.1 provides an example of an origin AS *O* with a *production prefix P* which carries real traffic. Figure 5.1(a) depicts the state before the problem, and Figure 5.1(b) depicts the state following a failure, after *O* has reacted to repair routing.

LIFEGUARD speeds up convergence and reduces path exploration by prepending to the production prefix *P*’s announcement, announcing *O*-*O*-*O* as the baseline path. If *O* detects that some networks (ASes *E* and *F* in Figure 5.1) cannot reach *P* due to a problem in *A*, *O* updates the announcement to *O*-*A*-*O*.[3] These two announcements are the same length and have the same next hop, and so, under default BGP, they are equally preferred. If an AS is using a route that starts with *O*-*O*-*O* and does not go through *A*, then receives an update that changes that route to start with *O*-*A*-*O* instead, it will likely switch to using the new route without exploring other options, converging instantly. I will show in a measurement study in Section 5.4.2 that this prepending smooths convergence, helping ease concerns that an automated response to outages might introduce needless routing instability. This approach is orthogonal to efforts to reduce convergence effects [71, 68, 60], which would benefit *LIFEGUARD*.

Partially Poisoning ASes

LIFEGUARD tries to avoid cutting off an entire AS *A* and all ASes that lack routes that avoid *A*. I have three goals: (1) ASes cut off by poisoning should be able to use routes through *A* to reach *O* as soon as they work again; (2) if some paths through *A* work while others have failed, ASes using the working routes should be able to continue to if they lack alternatives; and (3) when possible, the system should steer traffic from failed to working paths within *A*.

Advertising a less-specific sentinel prefix. While *O* is poisoning *A*, ASes like *F* that are “captive” behind *A* will lack a route [17]. To ensure that *F* and *A* have a route that covers *P*, *LIFEGUARD*

[3]I only intend *O* to poison one AS at a time. If future work reveals a need to poison multiple ASes, then *O* could prepend its baseline path more times.

announces a less-specific *sentinel prefix* that contains *P* (and can also contain other production prefixes). When *P* experiences problems, the system continues to advertise the sentinel with the baseline (unpoisoned) path. As seen in Figure 5.1(b), ASes that do not learn of the poisoned path, because they are "captive" behind *A*, will receive the less specific prefix and can continue to try routing to the production prefix on it, through *A*, instead of being cut off. This effect is the *Backup Property* desired from AVOID_PROBLEM(A,P) and helps achieve goals (1) and (2).

Selectively poisoning to avoid AS links. Although most failures in a previous study were confined to a single AS, 38% occurred on an inter-AS link [37]. I use a technique I call *selective poisoning* to allow *LIFEGUARD*, under certain circumstances, to implement AVOID_PROBLEM(A-B,P). Poisoning does not provide a general solution to AS link avoidance, but, given certain topologies, selective poisoning can shift traffic within *A* onto working routes.

Specifically, under certain circumstances, *O* may be able to steer traffic away from a particular AS link without forcing it completely away from the ASes that form the link. Suppose *O* has multiple providers that connect to *A* via disjoint AS paths. Then *O* can poison *A* in advertisements to one provider, but announce an unpoisoned path through the other provider. Because the paths are disjoint, *A* will receive the poisoned path from one of its neighbors and the unpoisoned path from another, and it will only accept the unpoisoned path. So, *A* will route all traffic to *O*'s prefix to egress via the neighbor with the unpoisoned path. This *selective poisoning* shifts routes away from *A*'s link to the other neighbor, as well as possibly affecting which links and PoPs are used inside *A*.

Figure 5.2 illustrates the idea. Assume *O* discovers a problem on the link between *A* and *B2*. This failure affects *C3*, but *C2* still has a working route through *A*, and *C4* still has a working route through *B2*. *O* would like to shift traffic away from the failing link, without forcing any networks except *A* to change which neighbor they select to route through. In other words, *O* would like to announce AVOID_PROBLEM(A-B2,P). If *O* only uses selective advertising without poisoning, announcing its prefix via *D1* and not *D2*, *C4*'s route will have to change. If *O* poisons *A* via both *D1* and *D2*, *C2* and *C3* will have to find routes that avoid *A*, and *A* will lack a route entirely (except via a less-specific prefix). However, by selectively poisoning *A* via *D2* and not via *D1*, *O* can shift *A* and *C3*'s routes away from the failing link, while allowing *C3* to still route along working paths

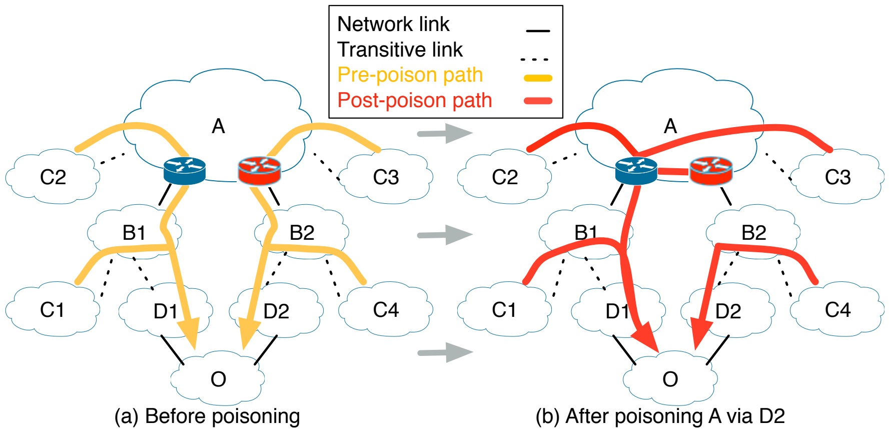


Figure 5.2: A case in which *LIFEGUARD* can use *selective poisoning*. By selectively poisoning *A* on announcements to *D2* and not on announcements to *D1*, *O* can influence which routes are selected within *A* without disturbing routes outside *A*. In particular, *O* can cause traffic to avoid the link from *A* to *B2*, without disrupting how *C3* routes to *A* or how *C[1,2,4]* route to *O*.

in *A* and without disturbing any other routes. Selective poisoning functions like targeted prepending – prepending requires that *A* use path length to make routing decisions and potentially causes other ASes to shift from using routes through *D2*, whereas selective poisoning forces only *A* to change. In Section 5.4.2 I find that selective poisoning lets *LIFEGUARD* avoid 73% of the links I test.

5.3 Applying Failure Avoidance

In the previous section, I described how *LIFEGUARD* uses BGP poisoning to approximate AVOID_PROBLEM(X,P). Poisoning allows me to experiment with failure avoidance today, and it gives ASes a way to deploy failure avoidance unilaterally. In this section, I describe how *LIFEGUARD* decides when to poison and which AS to poison, as well as how it decides when to stop poisoning an AS.

5.3.1 Locating a Failure

An important step towards fixing a reachability problem is to identify the network or router responsible for it. To be widely applicable and effective, I require my fault isolation technique to: (1) rely only on deployed protocols and available vantage points; and (2) be accurate even in the face of asymmetric paths and unidirectional failures.

This setting means that some routers may not respond to probes and that I can only assume control of one direction of any path I am monitoring. Further, the probing overhead cannot be more than the available vantage points can support, and the probing rate must not overwhelm routers that rate limit their responses. To overcome these challenges, the technique will have to integrate information from multiple measurement nodes, each with only partial visibility into routing behavior.

I assume that a routing failure between a pair of endpoints can be explained by a single problem. While addressing multiple failures is an interesting direction for future work, this dissertation focuses on single failures.

Overview of Failure Isolation

The failure isolation component of *LIFEGUARD* is a distributed system, using geographically distributed PlanetLab hosts to make data plane measurements to a set of monitored destinations. Drawing on Hubble's measurements, which showed that many outages were partial, *LIFEGUARD* uses vantage points with working routes to send and receive probes on behalf of those with failing paths. Because many failures are unidirectional, it adapts techniques from reverse traceroute to provide reverse path visibility. In the current deployment, vantage points send pings to monitor destinations, and a vantage point triggers failure isolation when it experiences repeated failed pings to a destination. This failure detection works like Hubble's ping monitors.

LIFEGUARD uses historical measurements to identify candidates that could potentially be causing a failure, then systematically prunes the candidate set with additional measurements. I outline these steps first before describing them in greater detail.

1. **Maintain background atlas:** *LIFEGUARD* maintains an atlas of the round-trip paths between its sources and the monitored targets to discern changes during failures and generate candidates for failure locations.

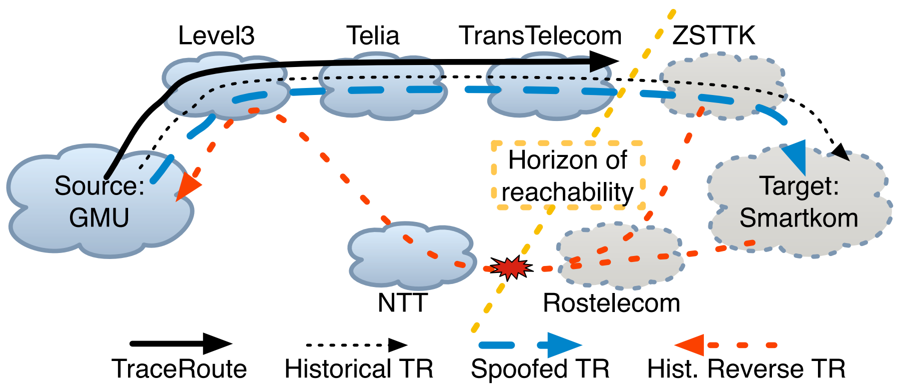


Figure 5.3: Isolation measurements conducted for an actual outage. With traceroute alone, the problem appears to be between TransTelecom and ZSTTK. Using spoofed traceroute, reverse traceroute, and historical path information, *LIFEGUARD* determines that the forward path is fine, but that Rostelecom no longer has a working path back to GMU.

2. **Isolate direction of failure and measure working direction:** After detecting a failure, *LIFEGUARD* uses Hubble's spoofed ping technique to isolate the direction of failure to identify what measurements to use to locate the failure. Further, if the failure is unidirectional, it measures the path in the working direction using one of two measurement techniques.

3. **Test atlas paths in failing direction:** *LIFEGUARD* tests which subpaths still work in the failing direction by probing routers on historical atlas paths between the source and destination. It then remeasures the paths for responsive hops, thereby identifying other working routers.

4. **Prune candidate failure locations:** Routers with working paths in the previous step are eliminated. Then, *LIFEGUARD* blames routers that border the "horizon of reachability." This horizon divides routers that have connectivity to the source from those that lack that connectivity.

Description of Fault Isolation

I illustrate the steps that *LIFEGUARD* uses to isolate failures using an actual example of a failure diagnosed on February 24, 2011. At the left of Figure 5.3 is one of *LIFEGUARD*'s vantage points, a PlanetLab host at George Mason University (labeled *GMU*). The destination, belonging to Smartkom in Russia, at the right, became unreachable from the source. For simplicity, the figure depicts hops at the AS granularity.

Maintain background atlas: In the steady state, *LIFEGUARD* uses traceroute and reverse traceroute to regularly map the forward and reverse paths between its vantage points and the destinations it is monitoring. During failures, this path atlas yields both a view of what recent paths looked like before the failure, as well as a historical view of path changes over time. These paths provide likely candidates for failure locations and serve as the basis for some of the isolation measurements I discuss below. Because some routers are configured to ignore ICMP pings, *LIFEGUARD* also maintains a database of historical ping responsiveness, allowing it to later distinguish between connectivity problems and routers configured to not respond to ICMP probes.

The figure depicts historical forward and reverse traceroutes with the dotted black and red lines, respectively. The thick, solid black line in Figure 5.3 depicts the traceroute from GMU during the failure. Traceroute can provide misleading information in the presence of failures. In this case, the last hop is a TransTelecom router, suggesting that the failure may be adjacent to this hop, between TransTelecom and ZSTTK. However, without further information, operators cannot be sure, since the probes may have been dropped on the reverse paths back from hops beyond TransTelecom.

Isolate direction of failure and measure working direction: *LIFEGUARD* tries to isolate the direction of the failure using spoofed pings, using Hubble's technique described in Section 3.1.4. In the example, spoofed probes sent from GMU to Smartkom reached other vantage points, but no probes reached GMU, implying a reverse path failure. When the failure is unidirectional, *LIFEGUARD* measures the complete path in the working direction. Extending the spoofed ping technique, *LIFEGUARD* sends spoofed probes to identify hops in the working direction while avoiding the failing direction. For a reverse failure, *LIFEGUARD* finds a vantage point with a working path back from *D*, then has *S* send a spoofed traceroute to *D*, spoofing as the working vantage point. In the example, GMU issued a spoofed traceroute, and a vantage point received responses from ZSTTK and the destination. The blue dashed edges in Figure 5.3 show the spoofed traceroute.

If the failure had been on the forward path, the system instead would have measured the working reverse path with a spoofed reverse traceroute from *D* back to *S*. Reverse traceroute, as described in Chapter 4, requires two-way connectivity between the source and destination. *LIFEGUARD* avoids that limitation by sending all probes from its vantage points, spoofing as the source experiencing the problem.

It is useful to measure the working direction of the path for two reasons. First, since the path is likely a valid policy-compliant path in the failing direction, it may provide a working alternative for avoiding the failure. Second, knowledge of the path in the working direction can guide isolation of the problem in the failing direction, as I discuss below.

Test atlas paths in failing direction: Once it has measured the working path, *LIFEGUARD* measures the responsive portion of the failing direction, as much as the failure allows. For forward and bidirectional failures, the source can simply issue a traceroute towards the destination in order to measure the path up to the failure.

For reverse failures, *LIFEGUARD* cannot measure a reverse traceroute from *D*, as such a measurement requires a response from *D* to determine the initial hops, even with the help of a spoofing node *V*. Instead, *LIFEGUARD* has its vantage points, including *S*, ping: (1) all hops on the forward path from *S* to *D* and (2) all hops on historical forward and reverse paths between *S* and *D* in its path atlas. These probes test which locations can reach *S*, which cannot reach *S* but respond to other vantage points, and which are completely unreachable. *LIFEGUARD* uses its atlas to exclude hops configured never to respond. For all hops still pingable from *S*, *LIFEGUARD* measures a reverse traceroute to *S*.

In the example, *LIFEGUARD* found that NTT still used the same path towards GMU that it had before the failure and that Rostelecom no longer had a working path. I omit the reverse paths from most forward hops to simplify the figure. In this case, *LIFEGUARD* found that all hops before Rostelecom were reachable (denoted with blue clouds with solid boundaries), while all in Rostelecom or beyond were not (denoted with light-gray clouds with dashed boundaries), although they had responded to pings in the past.

Prune candidate failure locations: Finally, *LIFEGUARD* removes any reachable hops from the suspect set and applies heuristics to identify the responsible hop within the remaining suspects. For forward outages, the failure is likely between the last responsive hop in a traceroute and the next hop along the path towards the destination. *LIFEGUARD*'s historical atlas often contains paths through the last hop, providing hints about where it is trying to route. *LIFEGUARD* checks the reachability of these hops with pings to settle on candidate locations.

For a reverse failure, *LIFEGUARD* considers reverse paths from *D* back to *S* that are in its atlas prior to the failure. For the most recent path, it determines the farthest hop *H* along that path that can still reach *S*, as well as the first hop *H'* past *H* that cannot. Given that *H'* no longer has a working path to *S*, contacting the AS containing *H'* or rerouting around it may resolve the problem.

If the failure is not in *H'*, one explanation is that, because *H'* lacked a route, *D* switched to another path which also did not work. In these cases, *LIFEGUARD* performs similar analysis on older historical paths from D, expanding the initial suspect set and repeating the pruning. Since Internet paths are generally stable [137], the failing path will often have been used historically, and there will often be few historical paths between the source and destination.

Because both historical reverse paths from unresponsive forward hops traversed Rostelecom, it seems highly likely this is the point of failure. This conclusion is further supported by the fact that all unreachable hops except Rostelecom responded to pings from other vantage points, indicating that their other outgoing paths still worked. I provide details on this and other examples at `http://lifeguard.cs.washington.edu`.

Case Studies

These techniques enable failure isolation at the granularity of an AS or router, within minutes of detecting an outage. I now present two case studies that illustrate how *LIFEGUARD* uses different measurements to isolate failures depending on the direction of the failure.

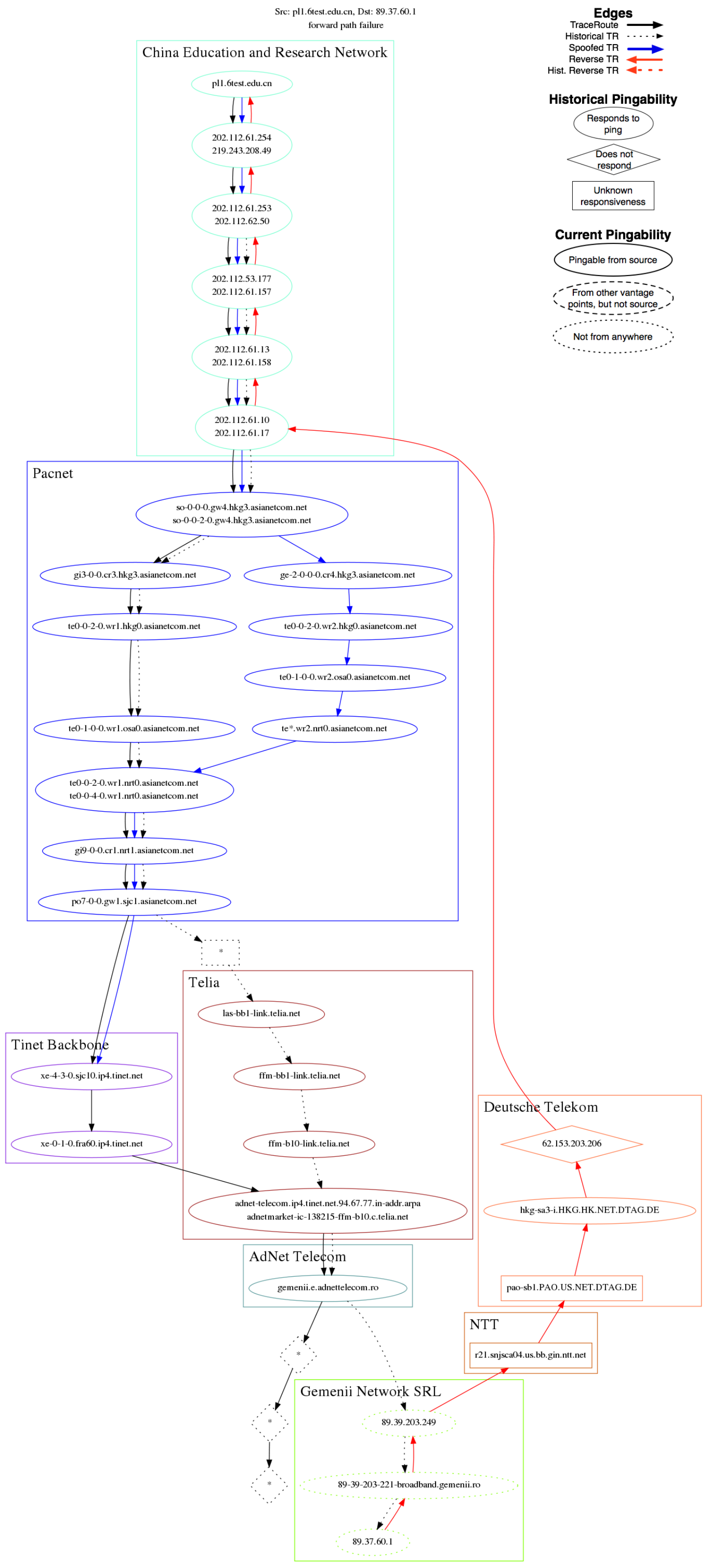

Src: pl1.6test.edu.cn, Dst: 89.37.60.1
forward path failure
Edges
TraceRoute
Historical TR
Spoofed TR
Reverse TR
Hist. Reverse TR
Historical Pingability
Responds to ping
Does not respond
Unknown responsiveness
Current Pingability
Pingable from source
From other vantage points, but not source
Not from anywhere
China Education and Research Network
pl1.6test.edu.cn
202.112.61.254
219.243.208.49
202.112.61.253
202.112.62.50
202.112.53.177
202.112.61.157
202.112.61.13
202.112.61.158
202.112.61.10
202.112.61.17
Pacnet
so-0-0-0.gw4.hkg3.asianetcom.net
so-0-0-2-0.gw4.hkg3.asianetcom.net
gi3-0-0.cr3.hkg3.asianetcom.net
ge-2-0-0-0.cr4.hkg3.asianetcom.net
te0-0-2-0.wr1.hkg0.asianetcom.net
te0-0-2-0.wr2.hkg0.asianetcom.net
te0-1-0-0.wr2.osa0.asianetcom.net
te0-1-0-0.wr1.osa0.asianetcom.net
te*.wr2.nrt0.asianetcom.net
te0-0-2-0.wr1.nrt0.asianetcom.net
te0-0-4-0.wr1.nrt0.asianetcom.net
gi9-0-0.cr1.nrt1.asianetcom.net
po7-0-0.gw1.sjc1.asianetcom.net
*
Telia
las-bb1-link.telia.net
ffm-bb1-link.telia.net
ffm-b10-link.telia.net
adnet-telecom.ip4.tinet.net.94.67.77.in-addr.arpa
adnetmarket-ic-138215-ffm-b10.c.telia.net
Tinet Backbone
xe-4-3-0.sjc10.ip4.tinet.net
xe-0-1-0.fra60.ip4.tinet.net
Deutsche Telekom
62.153.203.206
hkg-sa3-i.HKG.HK.NET.DTAG.DE
pao-sb1.PAO.US.NET.DTAG.DE
AdNet Telecom
gemenii.e.adnettelecom.ro
NTT
r21.snjsca04.us.bb.gin.ntt.net
*
*
*
Gemenii Network SRL
89.39.203.249
89-39-203-221-broadband.gemenii.ro
89.37.60.1

Figure 5.4: Isolation measurements for an actual forward path failure. The forward traceroute terminates in AdNet Telecom, but *LIFEGUARD* measures a working reverse path through NTT.

Isolating a forward path failure: *LIFEGUARD* identified a failure keeping one of its sources, located in China, from reaching a router in Romania. Figure 5.4 depicts some of *LIFEGUARD*'s measurements in a figure generated automatically by the system. The historical atlas revealed that the forward path to Romania went through AdNet Telecom, one of the destination's providers, whereas the reverse path left the destination AS through NTT, a different provider. Using spoofed pings, the system isolated the problem to the forward path. *LIFEGUARD* then successfully measured the entire (spoofed) reverse traceroute back from the destination, revealing that it still went via NTT. The figure depicts the identical historical and spoofed reverse paths as a single line. The Chinese source's current forward traceroute followed a similar path to the historical one to AdNet Telecom but terminated at a router in that AS. This result indicated a failure reaching the destination from AdNet. Furthermore, the only vantage points that could reach the destination during the outage used NTT as an ingress, supporting the hypothesis. Other vantage points could reach the destination via NTT, and the source could ping hops in NTT that appeared on the reverse path to China. If networks behind the failure could switch to paths through NTT, they could avoid the problem with AdNet.

Isolating a reverse path failure: Routes between *LIFEGUARD*'s source in Korea and a target router in Chicago had historically gone through Level 3 in both directions. When the source detected a problem, its traceroute to the target router did not receive any responses after Level 3's San Jose PoP. However, *LIFEGUARD* isolated the failure to the reverse direction. Figure 5.5 depicts some of *LIFEGUARD*'s measurements in a figure generated automatically by the system. When the source sent a forward traceroute spoofing as a vantage point that could reach the target, it reached the destination via Level 3's Los Angeles, Dallas, and Chicago PoPs, yielding hops past where the (non-spoofed) traceroute terminated. *LIFEGUARD*'s historical reverse traceroutes showed that the reverse path from the destination passed through Level 3 in Chicago, Level 3 in Denver, and then Level 3's San Jose PoP. The source could still successfully ping the San Jose, Los Angeles, Dallas, and Denver PoPs, but the Chicago PoP was pingable only from other vantage points. Other vantage points could also ping the destination. These results placed the "reachability horizon" in Level 3, with San Jose, Los Angeles, Denver, and Dallas on the reachable side (near the source) and Chicago on the unreachable side (near the destination). These measurements suggested a failure affecting some paths through Level 3 towards the Korean vantage point.

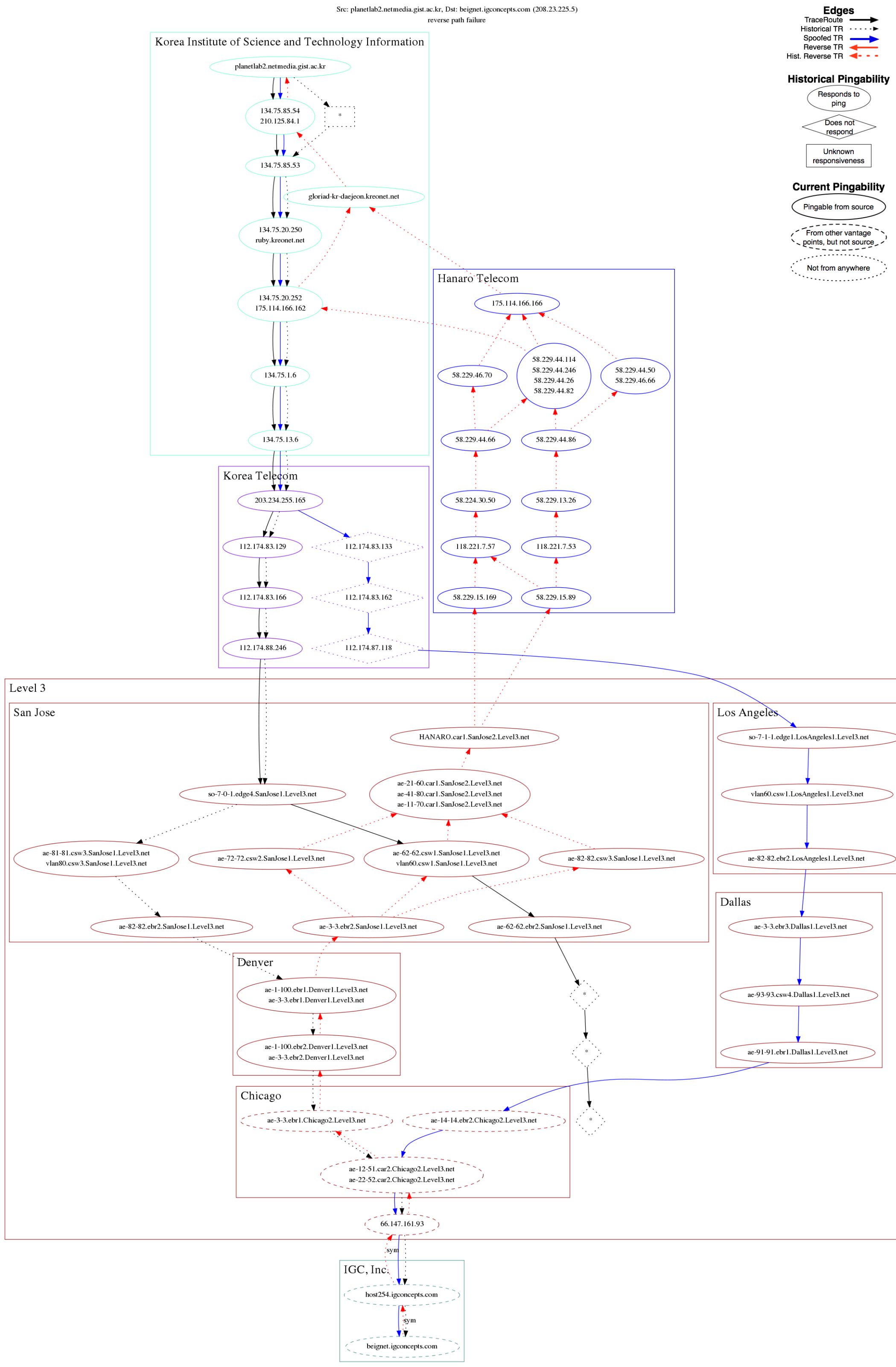

Src: planetlab2.netmedia.gist.ac.kr, Dst: beignet.igconcepts.com (208.23.225.5)
reverse path failure
Edges
TraceRoute
Historical TR
Spoofed TR
Reverse TR
Hist. Reverse TR
Historical Pingability
Responds to ping
Does not respond
Unknown responsiveness
Current Pingability
Pingable from source
From other vantage points, but not source
Not from anywhere
Korea Institute of Science and Technology Information
planetlab2.netmedia.gist.ac.kr
134.75.85.54
210.125.84.1
*
134.75.85.53
gloriad-kr-daejeon.kreonet.net
134.75.20.250
ruby.kreonet.net
134.75.20.252
175.114.166.162
134.75.1.6
134.75.13.6
Hanaro Telecom
175.114.166.166
58.229.46.70
58.229.44.114
58.229.44.246
58.229.44.26
58.229.44.82
58.229.44.50
58.229.46.66
58.229.44.66
58.229.44.86
58.224.30.50
58.229.13.26
118.221.7.57
118.221.7.53
58.229.15.169
58.229.15.89
Korea Telecom
203.234.255.165
112.174.83.129
112.174.83.133
112.174.83.166
112.174.83.162
112.174.88.246
112.174.87.118
Level 3
San Jose
HANARO.car1.SanJose2.Level3.net
so-7-0-1.edge4.SanJose1.Level3.net
ae-21-60.car1.SanJose2.Level3.net
ae-41-80.car1.SanJose2.Level3.net
ae-11-70.car1.SanJose2.Level3.net
ae-81-81.csw3.SanJose1.Level3.net
vlan80.csw3.SanJose1.Level3.net
ae-72-72.csw2.SanJose1.Level3.net
ae-62-62.csw1.SanJose1.Level3.net
vlan60.csw1.SanJose1.Level3.net
ae-82-82.csw3.SanJose1.Level3.net
ae-82-82.ebr2.SanJose1.Level3.net
ae-3-3.ebr2.SanJose1.Level3.net
ae-62-62.ebr2.SanJose1.Level3.net
Los Angeles
so-7-1-1.edge1.LosAngeles1.Level3.net
vlan60.csw1.LosAngeles1.Level3.net
ae-82-82.ebr2.LosAngeles1.Level3.net
Dallas
ae-3-3.ebr3.Dallas1.Level3.net
ae-93-93.csw4.Dallas1.Level3.net
ae-91-91.ebr1.Dallas1.Level3.net
Denver
ae-1-100.ebr1.Denver1.Level3.net
ae-3-3.ebr1.Denver1.Level3.net
ae-1-100.ebr2.Denver1.Level3.net
ae-3-3.ebr2.Denver1.Level3.net
*
*
*
Chicago
ae-3-3.ebr1.Chicago2.Level3.net
ae-14-14.ebr2.Chicago2.Level3.net
ae-12-51.car2.Chicago2.Level3.net
ae-22-52.car2.Chicago2.Level3.net
66.147.161.93
sym
IGC, Inc.
host254.igconcepts.com
sym
beignet.igconcepts.com

Figure 5.5: Isolation measurements for a reverse path failure. The forward traceroute terminates in Level 3, but *LIFEGUARD* reveals a working forward path and a reverse failure in Level 3.

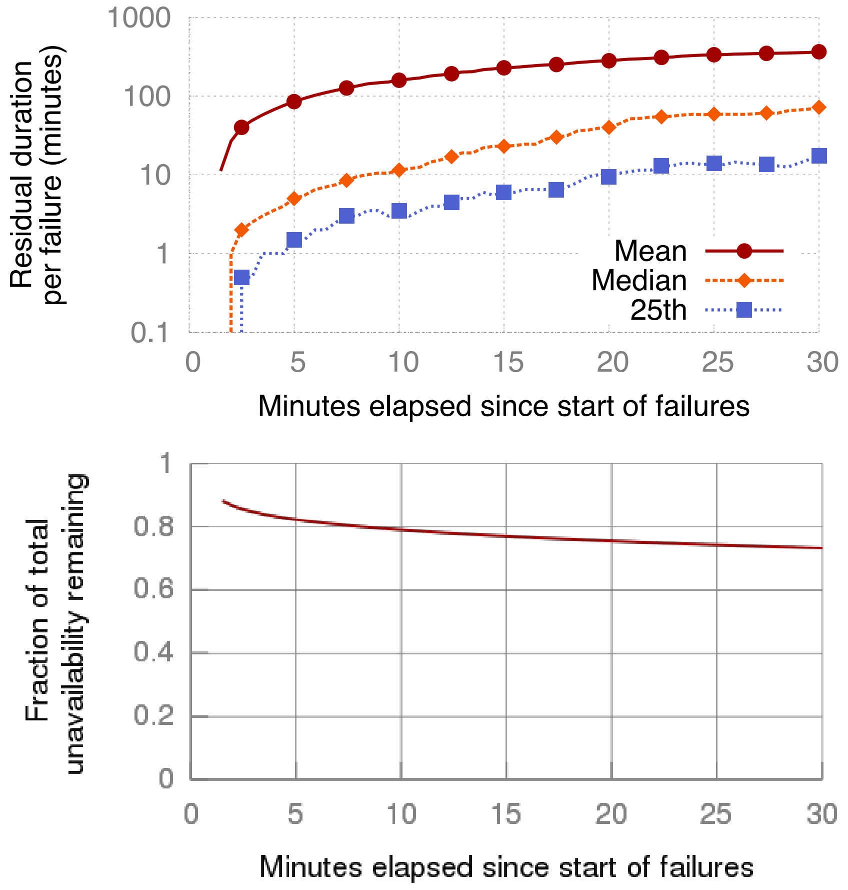


Figure 5.6: For the EC2 dataset, residual duration after outages have persisted for X minutes. The top graph shows that, once a problem has persisted for a few minutes, it will most likely persist for at least a few more minutes unless there is corrective action. The bottom graph gives the fraction of total unavailability in the study, including outages that resolved before time X. It shows that, even if *LIFEGUARD* does not trigger rerouting for a few minutes and the rerouting takes a few more minutes to converge, we can still avoid most of the total unavailability.

5.3.2 Deciding to Start and Stop Poisoning

Deciding whether to poison: As seen in Section 3.4, most outages resolve quickly. For the system to work effectively, it would be helpful to differentiate between outages that will clear up quickly and those that will persist. If routing protocols will quickly resolve a problem, then it would be better to wait in order to avoid causing further routing changes. If the protocols will not restore connectivity quickly on their own, then poisoning may be a better approach.

The following analysis of the EC2 outage data from Section 3.4 shows that it is possible to differentiate between these cases with high likelihood. The top graph in Figure 5.6 shows the residual duration of these outages, given that they have already lasted for X minutes. The median duration of an outage in the study was only 90 seconds (the minimum possible given the methodology). However, of the 12% of problems that persisted for at least 5 minutes, 51% lasted at least another 5 minutes. Further, of the problems that lasted 10 minutes, 68% persisted for at least 5 minutes past that. *LIFEGUARD* triggers isolation after multiple rounds of failed pings, and it takes an average of 140 seconds to isolate a reverse path outage (see Section 5.4.4). If a problem persists through both those stages, then the results suggest that the problem is likely to persist long enough to justify using poisoning to fix it. I will show in Section 5.4.2 that poisoned routes converge within a few minutes in almost all cases, with little loss during the convergence period. So, if there are alternative paths that avoid the problem *LIFEGUARD* locates, the system should quickly restore connectivity.

Long-lasting problems account for much of the total unavailability. As a result, even if *LIFEGUARD* takes five minutes to identify and locate a failure before poisoning, and it then takes two minutes for routes to converge, the system can still potentially avoid 80% of the total unavailability in the EC2 study (bottom graph in Figure 5.6). In Section 5.4.1, I will show that it is possible to determine (with high likelihood) whether alternate policy compliant paths will exist before deciding to poison an AS. If no paths exist, *LIFEGUARD* does not attempt to poison the AS.

Deciding when to unpoison: Once *LIFEGUARD* accurately identifies the AS *A* responsible for a problem, BGP poisoning can target it and cause other ASes to route around *A*. However, *A* will eventually resolve the underlying issue, at which point we would like to be able to revert to the unpoisoned path, allowing ASes to use paths through *A*, if preferred. When the poisoned announcement is in place, however, *A* will not have a path to the prefix in question.

LIFEGUARD uses a sentinel prefix to test reachability. Concerns such as aggregation and address availability influence the choice of sentinel. In the current deployment, the sentinel is a less specific prefix containing both the production prefix and a prefix that is not otherwise used. Responses to pings from the unused portion of the sentinel will route via the sentinel prefix, regardless of whether the hops also have the poisoned more-specific prefix. By sending active ping measurements from

Table 5.1: Key results of my evaluation of *LIFEGUARD*, demonstrating its viability for addressing long-duration outages.

Criteria	Summary	Experimental Result
Effectiveness (§5.4.1)	Most edge networks have routes that avoid poisoned ASes, and it is possible to calculate which do *a priori*	77% of poisons from BGP-Mux 90% of poisons in large-scale simulation
Disruptiveness (§5.4.2)	Routes that already avoid the problem AS reconverge quickly after poisoning	95% of paths converge instantly
	Minimal loss occurs during convergence	Less than 2% packet loss in 98% of cases
Accuracy (§5.4.3)	Locates failures as if it had traceroutes from both ends	Consistent results for 93% of inter-PlanetLab failures
	Isolates problems that traceroute alone misdiagnoses	40% of cases differ from traceroute
Scalability (§5.4.4)	Quickly isolates problems with reasonable overhead	140 seconds for poisoning candidates 280 probes per failure
	Induces reasonably small additional BGP update load on routers	$<$ 1% if 1% of ISPs use *LIFEGUARD* $<$ 10% if 50% of ISPs use *LIFEGUARD*

this prefix to destinations that had been unable to reach the production prefix prior to poisoning (e.g., *E* in Figure 5.1), the system can detect when to unpoison the production prefix. If the sentinel is a less-specific without any unused prefixes, *LIFEGUARD* can instead ping the destinations within the poisoned AS (e.g., *A*) or within captives of the poisoned AS (e.g., *F*).

5.4 LIFEGUARD Evaluation

To preview the results of my evaluation, I find that *LIFEGUARD*'s poisoning finds routes around the vast majority of potential problems, its approach is minimally disruptive to paths that are not experiencing problems, and its failure isolation can correctly identify the AS needing to be poisoned. Table 5.1 summarizes my key results. The following sections provide more details.

I deployed *LIFEGUARD*'s path poisoning using the BGP-Mux testbed [14]. The BGP-Mux testbed has its own AS number and prefixes, which *LIFEGUARD* uses. *LIFEGUARD* connected to a BGP-Mux instance at Georgia Tech, which served as the Internet provider for the BGP-Mux AS (and hence for *LIFEGUARD*). For the poisoning experiments in this section, *LIFEGUARD* announced prefixes via Georgia Tech into the commercial Internet. To speed my experiments, I announced multiple prefixes at once, poisoning a different AS on each one. Traffic to these prefixes routes through Georgia Tech to the machine hosting *LIFEGUARD*'s controller.

I assessed the effects of poisoning in the absence of failures. To obtain ASes to poison, I announced a prefix and harvested all ASes on BGP paths towards the prefix from route collectors [87, 105, 99, 1]. I excluded all Tier-1 networks, as well as Cogent, as it is Georgia Tech's main provider. In Section 5.4.2 and Section 5.6.1, I evaluate techniques to poison even these large ASes. For each of the remaining harvested ASes,[4] I first went from a baseline of O to a poisoned announcement O-A-O, then repeated the experiment starting from a baseline of O-O-O. I kept each announcement in place for 90 minutes to allow convergence and to avoid flap dampening effects. For the duration of the experiments, I also announced an unpoisoned prefix to use for comparisons.

5.4.1 Efficacy

Do ASes find new routes that avoid a poisoned AS? I monitored BGP updates from public BGP route collectors to determine how many ASes found an alternate path after I poisoned an AS on their preferred path. There were 132 cases in which an AS peering with a route collector was using a path through one of the ASes I poisoned. In 102 of the 132 cases (77%), the route collector peer found a new path that avoided the poisoned AS. Two-thirds of the cases in which the peer could not find a path were instances in which I had poisoned the only provider of a stub AS.

Do alternate policy-compliant routes exist in general? I analyzed a large AS topology to show that, in the common case, alternate paths exist around poisoned ASes. The topology, along with AS relationships, is from a dataset combining public BGP feeds with more than five million AS paths between BitTorrent (BT) peers [24]. To simulate poisoning an AS A on a path from a source S to an origin O, I remove all of A's links from the topology. I then check if S can restore connectivity while avoiding A (i.e., a path exists between S and O that obeys export policies).

[4] I announced my experiments on the NANOG mailing list and allowed operators to opt out their ASes. None did. A handful opted out of an earlier Georgia Tech study, and I honored their list.

I check policies using the three-tuple test [79], as in Section 3.5. This approach may miss alternate paths if the valley-free assumption is too strict,[5] and rarely used backup paths may not be in my topology. Conversely, it may identify valley-free paths not used in practice.

To establish the validity of this methodology, I simulated the Georgia Tech poisonings. In 92.5% of cases, the simulation found an alternate path if and only if the AS found such a path following my actual poisoning. In the remaining 7.5% of cases, the simulation found an alternate path, but in practice the AS failed to find one. In these cases, the source was an academic network connected to both a commercial provider and an academic provider (which only provides routes to academic destinations). These connections made the AS multi-homed in my simulation. In practice, however, the AS seems to reject routes from its commercial provider if the route contained an academic AS, and I had poisoned one.

Having established that my simulation closely predicts the results of actual poisonings, I simulated poisoning ASes on the BT and BGP feed paths. For each AS path with length greater than three (i.e., traversing at least one transit AS in addition to the destination's provider), I iterated over all the transit ASes in the path except the destination's immediate provider and simulated poisoning them one at a time.[6] An alternate path existed in 90% of the 10 million cases. I then simulated poisoning the failures isolated by *LIFEGUARD* in June 2011. Alternate paths existed in 94% of them.

LIFEGUARD uses a sentinel prefix to decide when a failure is repaired, making it safe to revert to an unpoisoned announcement. The effectiveness of this approach relies on assumptions that I test here.

Are poisoned more specifics aggregated into unpoisoned less specifics? If an AS aggregates a poisoned more-specific route into the unpoisoned less-specific sentinel, it would likely remove the poison. To verify that this aggregation does not generally happen, at least from *LIFEGUARD*'s

[5]It is well known that not all paths are valley-free in practice, and I observed violations in the BitTorrent traceroutes.

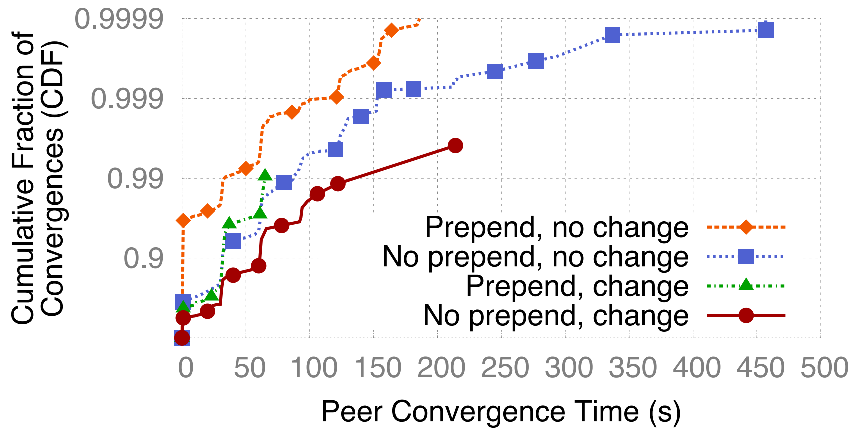
[6]A multi-homed destination can use selective advertising to avoid a particular provider. A single-homed destination can never avoid having paths traverse its provider.

current BGP deployment, I used the Georgia Tech BGP-Mux to announce two prefixes: (1) 184.164.224.0/24 with poison and (2) 184.164.224.0/23 without poison. I poisoned an edge AS, MIT, to guarantee that the prefix would propagate normally. I then checked how many RouteViews peer ASes exported both prefixes and how many only exported one. Out of 47 total ASes, 46 (98%) received both prefixes with identical paths, except for the poisoned insertion of MIT. Surprisingly, AS28571, a university in Brazil, received only the poisoned more-specific /24 prefix, but not the unpoisoned less-specific /23 prefix. Note that it does not affect the use of sentinel prefixes if ASes aggregate the more-specific prefix when it is not poisoned.

Do ASes choose the same path for the sentinel and the production prefix? For a sentinel prefix to detect when a problem has resolved, the sentinel needs to experience similar routing to what the production prefix would experience if unpoisoned. I now show that networks rarely use different paths for unpoisoned more specific and less-specific prefixes from the same origin. Networks could choose different paths to facilitate load-balancing. Via the Georgia Tech BGP-Mux, I announced three unpoisoned prefixes: (1) less-specific 184.164.244.0/23, (2) more-specific 184.164.244.0/24, and (3) more-specific 184.164.245.0/24. I then analyzed RIPE RIS, PCH, and RouteViews route table snapshots between September 22 and September 29, 2011, to see how many router collector peer ASes chose differing paths to the prefixes. During that period, six (out of 202) ASes showed different paths to the prefixes at least once. Five of these six ASes had different paths to the prefixes in fewer than 30% of the snapshots I analyzed. One AS (AS29449) chose different paths to more-specific /24 and less-specific /23 prefixes throughout the period. The other 97% of ASes chose the same paths to all three prefixes during all snapshots.

5.4.2 Disruptiveness

How quickly do paths converge after poisoning? I used updates from the BGP collectors to measure convergence delay after poisoning. I will show that, in most cases, if an AS was not routing through the AS I poisoned, it re-converges essentially instantly to its original path, requiring only a single update to pass on the poison. Global convergence usually takes at most a few minutes.



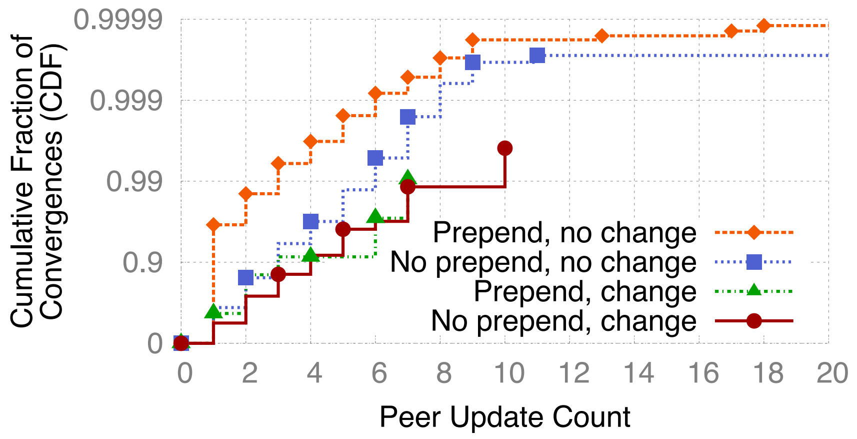


Figure 5.7: Convergence time (top) and number of updates issued (bottom) for route collector peer ASes after poisoned announcements. A data point captures the convergence for one peer AS after one poisoned announcement. *Change* vs. *no change* indicates if the peer had to change its path because it had been routing through the poisoned AS. For *Prepend*, the baseline announcement before poisoning was *O-O-O*, whereas for *No prepend* it was *O*. In both cases, the poisoned announcement was *O-A-O*. Prepending reduces path exploration by keeping path length consistent, resulting in reduced convergence delay.

First, I assess how quickly ASes that were using the poisoned AS settle on a new route that avoids it. I also assess how quickly routes converge for ASes that were not using the poisoned AS. As explained above, I poisoned each harvested AS twice each using different pre-poisoning baseline paths. After each announcement, for each AS that peers with a route collector, I measured the delay from when the AS first announced an update to the route collector to when it announced its stable post-poisoning route. I leave out (route collector peer, poisoned AS) pairs if the peer AS did not have a path to the *LIFEGUARD* prefix following the poisoning.

As seen in the top graph in Figure 5.7, a baseline announcement of *O-O-O* greatly speeds convergence. More than 95% of the time, ASes that were not using the poisoned AS converged essentially instantly upon receiving the poisoned update, as they did not need to change their path, and 99% of them converged within 50 seconds (*Prepend, No Change* line). In comparison, if I simply announce O as the baseline before poisoning, less than 70% of the unaffected ASes converged instantly, and 94% converged within 50 seconds (*No Prepend, No Change* line). Similarly, using O-O-O as the baseline helps ASes that had been routing via A settle on a new route faster: 96% converged within 50 seconds, compared to only 86% if I use O as the baseline. Prepending keeps the announcements a consistent length, which reduces path exploration for ASes not routing through A. In fact, as seen in the bottom figure, with prepending, 97% of unaffected ASes made only a single update, informing neighbors only of the insertion of the poisoned AS A into the route. Without prepending, only 64% of these ASes made only one update. The other 36% explored alternatives before reverting back to their original path.

Figure 5.7 captures per-AS convergence. Because announcements need to propagate across the Internet and are subject to protocol timers as they do so, different ASes receive a poisoned announcement at different times.

I also assessed how long global convergence took, from the time the first router collector receives an update for my prefix until all route collector peer ASes had converged to stable routes. With prepending, global convergence took at most 91 seconds in the median case, at most 120 seconds for 75% of poisonings, and at most 200 seconds in 90% of poisonings. In contrast, without prepending, the 50th, 75th, and 90th percentiles are 133 seconds, 189 seconds, and 226 seconds. Compared to the delay following a poisoning, it generally takes slightly less time for routes to converge globally when I remove the poison and revert to the baseline announcement. Because most ASes that were not using the poisoned AS reconverge to their original path without exploring other options, I would not expect them to experience transient loss during the global convergence period.

How much loss accompanies convergence? My results indicate the packet loss during convergence is minimal. I calculated loss rate during convergence following my poisonings that used a baseline of O-O-O. For each poisoning, I considered the period starting when the poisoned announcement first reached a route collector and ending when all route collector peers had converged. Every 10 seconds for the duration of the experiment, I issued pings from the poisoned and unpoisoned *LIFEGUARD* prefixes to all 308 working PlanetLab sites. I set the source address for each ping to an address in the *LIFEGUARD* prefix it was sent from, and so a particular response from a PlanetLab site would follow routes to that prefix. In general, many of the sites were not routing via any particular poisoned AS, and so this experiment lets us evaluate how much I disrupt working paths. I filtered out cases in which loss was clearly due to problems unrelated to poisoning, and I excluded a PlanetLab site if it was completely cut off by a particular poisoning.

Following 60% of poisonings, the overall loss rate during the convergence period was less than 1%, and 98% of poisonings had loss rates under 2%. Some convergence periods experienced brief spikes in loss, but only 2% of poisonings had any 10 second round with a loss rate above 10%.

Can poisoning shift routes off an AS link without completely disabling either AS? I demonstrate selective poisoning using two BGP-Mux sites, University of Washington (UWash) and University of Wisconsin (UWisc). Paths from most PlanetLab nodes to UWash pass through the Internet2 (I2) Seattle PoP, then to Pacific Northwest Gigapop, and to UWash. Most PlanetLab paths to UWisc pass through I2's Chicago PoP, then through WiscNet, before reaching UWisc. So, the BGP-Mux AS has UWash and UWisc as providers, and they connect via disjoint paths to different I2 PoPs.

I tested if I could shift traffic away from the I2 Chicago PoP, supposing the link to WiscNet experienced a silent failure. I advertised the same two prefixes from UWash and UWisc. I announced the first prefix unpoisoned from both. I poisoned I2 in UWisc's announcements of the second prefix, but had UWash announce it unpoisoned.

First, I show that paths that were not using I2 would not be disrupted. I looked at paths to the two prefixes from 36 RIPE RIS BGP peers. For 33 of them, the paths to the two prefixes were identical and did not use I2. The other three ASes routed to the unpoisoned prefix via I2 and WiscNet. For the selectively poisoned prefix, they instead routed via I2 and PNW Gigapop, as expected.

I then compare traceroutes from PlanetLab hosts to the two prefixes, to show that paths through I2 avoid the "problem." For the unpoisoned prefix, over 100 PlanetLab sites routed through I2 to

WiscNet. Focusing on just these PlanetLab sites, I assessed how they routed towards the other prefix, which I selectively poisoned for I2 from UWisc. For that prefix, all the sites avoided the link from I2 to WiscNet. All but three of the sites routed via PNW Gigapop and UWash, as I intended. The remaining three – two sites in Poland and one in Brazil – used Hurricane Electric to reach UWisc. Excepting these three sites, selective poisoning allowed us to shift paths within a targeted network without changing how networks other than the target routed.

Are Internet paths diverse enough for selective poisoning to be effective? This technique may provide a means to partially poison large networks, and services with distributed data centers may have the type of connectivity required to enable it. In my second selective poisoning experiment, I approximated this type of deployment, and I assess how many ASes I could selectively poison. To make this assessment, I need to identify whether ASes select from disjoint paths, which would allow us to selectively poison them. It is hard to perform this assessment in general, because I need to know both which path an AS is using and which paths it might use if that path became unavailable.

Route collectors and BGP-Mux let us experiment in a setting in which I have access to these paths. I announced a prefix simultaneously via BGP-Muxes at UWash, UWisc, Georgia Tech, Princeton, and Clemson. This setup functions as an AS with five PoPs and one provider per PoP. I iterated through 114 ASes that provide BGP feeds. For each pair of AS *A* and BGP-Mux *M* in turn, I poisoned *A* from all BGP-Muxes except *M* and observed how *A*'s route to my prefix varied with *M*. I found that selective poisoning allowed me to avoid 73% of the first hop AS links used by these peers, while still leaving the peers with a route to my prefix. In Section 5.1, I found that these five university providers allowed me to avoid 90% of these links on forward paths to these same ASes.

5.4.3 Accuracy

Having demonstrated that *LIFEGUARD* can often use BGP poisoning to route traffic around an AS or AS link without causing widespread disruption, I assess *LIFEGUARD*'s accuracy in locating failures. I show that *LIFEGUARD* seems to correctly identify the failed AS in most cases, including many that would be not be correctly found using only traceroutes. A tech report provides further analysis of *LIFEGUARD*'s failure isolation [108].

LIFEGUARD's failure isolation system is a prototype. I intend this chapter to describe the current system and to establish that it should be possible to locate many failures. The live deployment of *LIFEGUARD*'s failure isolation system has run continuously since January 2011. It currently uses a host at each of 12 PlanetLab sites as sources. It monitors paths between those sources and the following distinct sets of destinations, selected to provide an interesting set of paths on which to evaluate the system:

- one router in each of 83 highly-connected PoPs located in the core of the Internet based on iPlane's traceroute atlas [78].
- 185 targets located on the edge of the Internet, selected because each target had a path from at least one of the 12 sources that traversed at least one of the 83 centralized PoPs.
- 76 PlanetLab nodes. These targets provide a mechanism to validate results.

Because of the smaller number of sources and destinations relative to Hubble, each source pings every destination in every two minute round, in order to detect outages. I expect that it will be possible in the future to refine the techniques and implementations used to isolate failures, and part of this refinement may involve studying how to select targets that can serve as "canaries" signaling wider problems.

Are *LIFEGUARD*'s results consistent with what it would find with control of both ends of a path? In general, obtaining ground truth for wide-area network faults is difficult: emulations of failures are not realistic, and few networks post outage reports publicly. Due to these challenges, I evaluate the accuracy of *LIFEGUARD*'s failure isolation on paths between a set of PlanetLab hosts used as *LIFEGUARD* vantage points and a disjoint set of PlanetLab hosts used as targets.[7] Every

[7] I cannot issue spoofed packets or make BGP announcements from EC2, and so I cannot use it to evaluate my system.

five minutes, I issued traceroutes from all vantage points to all targets and vice versa. In isolating the location of a failure between a vantage point and a target, I only gave *LIFEGUARD* access to measurements from its vantage points. I checked if *LIFEGUARD*'s conclusion was consistent with traceroutes from the target, "behind" the failure. *LIFEGUARD*'s location was consistent if and only if (1) the traceroute in the failing direction terminated in the AS *A* blamed by *LIFEGUARD*, and (2) the traceroute in the opposite direction did not contradict *LIFEGUARD*'s conclusion. The traceroute contradicted the conclusion if it included responses from *A* but terminated in a different AS. The responses from *A* indicate that some paths back from *A* worked. I lack sufficient information to explain these cases, and so I consider *LIFEGUARD*'s result to be inconsistent.

I examined 182 unidirectional isolated failures from August and September 2011. For 169 of the failures, *LIFEGUARD*'s results were consistent with traceroutes from the targets. The remaining 13 cases were all forward failures. In each of these cases, *LIFEGUARD* blamed the last network on the forward traceroute (just as an operator with traceroute alone might). However, the destination's traceroute went through that network, demonstrating a working path from the network back to the destination.

Does *LIFEGUARD* locate failures of interest to operators? I searched the Outages.org mailing list [94] for outages that intersected *LIFEGUARD*'s monitored paths and found two interesting examples. On May 25, 2011, three of *LIFEGUARD*'s vantage points detected a forward path outage to three distinct locations. The system isolated all of these outages to a router in Level 3's Chicago PoP. Later that night, a network operator posted the following message to the mailing list: "Saw issues routing through Chicago via Level 3 starting around 11:05 pm, cleared up about 11:43 pm." Several network operators corroborated this report.

In the second example, a vantage point in Albany and one in Delaware observed simultaneous outages to three destinations: a router at an edge AS in Ohio and routers in XO's Texas and Chicago PoPs. *LIFEGUARD* identified all Albany outages and some of the Delaware ones as reverse path failures, with the remaining Delaware one flagged as bidirectional. All reverse path failures were isolated to an XO router in Dallas, and the bidirectional failure was isolated to an XO router in

Virginia. Several operators subsequently posted to the Outages.org mailing list reporting problems with XO in multiple locations, which is likely what *LIFEGUARD* observed.

Does *LIFEGUARD* provide benefit beyond traceroute? I now quantify how often *LIFEGUARD* finds that failures would be incorrectly isolated using only traceroute, thus motivating the need for the system's more advanced techniques. In the example shown in Figure 5.3, traceroutes from GMU seem to implicate a problem forwarding from TransTelecom, whereas my system located the failure as being along the reverse path, in Rostelecom. For the purposes of this study, I consider outages that meet criteria that make them candidates for rerouting and repair: (1) multiple sources must be unable to reach the destination, and these sources must be able to reach at least 10% of all destinations at the time, reducing the chance that it is a source-specific issue; (2) the failing traceroutes must not reach the destination AS, and the outage must be partial, together suggesting that alternate AS paths exist; and (3) the problem must not resolve during the isolation process, thereby excluding transient problems. During June 2011, *LIFEGUARD* identified 320 outages that met these criteria [108]. In 40% of cases, the system identified a different suspected failure location than what one would assume using traceroute alone. Further, even in the other 60% of cases, an operator would not currently know whether or not the traceroute was identifying the proper location.

5.4.4 Scalability

How efficiently does *LIFEGUARD* refresh its path atlas? *LIFEGUARD* regularly refreshes the forward and reverse paths it monitors. Existing approaches efficiently maintain forward path atlases based on the observations that paths converge as they approach the source/destination [34] and that most paths are stable [31]. Based on these observations, *LIFEGUARD*'s reverse path atlas caches probes for short periods, reuses measurements across converging paths, and usually refreshes a stale path using fewer probes than would be required to measure from scratch. In combination, these optimizations enable *LIFEGUARD* to refresh paths at an average (peak) rate of 225 (502) reverse paths per minute. It uses an amortized average per path of 10 IP option probes (compared to 35 measured using the basic reverse traceroute technique in Section 4.4.3) and slightly more than 2 forward traceroutes (including those to measure reverse traceroute's traceroute atlas). It may be possible to improve scalability in the future by focusing on measuring paths between “core” PoPs whose routing health and route changes likely influence many paths,[8] and by scheduling measurements to minimize the impact of router-specific rate limits.

What is the probing load for locating problems? Fault isolation requires approximately 280 probe packets per outage. *LIFEGUARD* isolates failures on average much faster than it takes long-lasting outages to be repaired. For bidirectional and reverse path outages, potential candidates for poisoning, *LIFEGUARD* completed isolation measurements within 140 seconds on average.

What load will poisoning induce at scale? An important question is whether *LIFEGUARD* would induce excessive load on routers if deployed at scale. The study above showed that, by prepending, *LIFEGUARD* reduces the number of updates made by each router after a path poisoning. In this section, I estimate the Internet-wide load my approach would generate if a large number of ISPs used it. While my results serve as a rough estimate, I find that the number of path changes made at each router is low.

The number of daily path changes per router my system would produce at scale is $I \times T \times P(d) \times U$, where I is the fraction of ISPs using my approach, T is the fraction of ASes each ISP is monitoring with *LIFEGUARD*, $P(d)$ is the aggregate number of outages per day that have lasted at least d minutes and are candidates for poisoning, and U is the average number of path changes per router generated by each poison. Based on my experiments, $U = 2.03$ for routers that had been routing via the poisoned AS, and $U = 1.07$ for routers that had not been routing via the poisoned AS. For routers using the poisoned AS, BGP should have detected the outage and generated at least one path update in response. For routers not routing through it, the updates are pure overhead. Thus, poisoning causes affected routers to issue an additional 1.03 updates and and unaffected routers an additional 1.07 updates. For simplicity, I set $U = 1$ in this analysis.

I base $P(d)$ on the Hubble dataset of outages on paths between PlanetLab sites and 92% of the Internet’s edge ASNs (Section 3.3.2). I filter this data to exclude complete outages, where poisoning

[8]For example, 63% of iPlane traceroutes traverse at least one of the most common 500 PoPs (0.3% of PoPs) [58].

Table 5.2: Number of additional daily path changes due to poisoning for fraction of ISPs using *LIFEGUARD* (I), fraction of networks monitored for reachability (T), and duration of outage before poisoning (d). For comparison, Tier-1 routers currently make between 255,000 and 315,000 updates per day.

		$d = 5$ minutes		$d = 15$ min.		$d = 60$ min.	
		$T = 0.5$	1.0	0.5	1.0	0.5	1.0
	0.01	393	783	137	275	58	115
I	0.1	3931	7866	1370	2748	576	1154
	0.5	19625	39200	6874	13714	2889	5771

is not effective. I assume that the Hubble targets were actually monitoring the PlanetLab sites (instead of vice versa) and define $P(d) = H(d)/(I_h T_h)$, where $H(d)$ is the total number of poisonable Hubble outages per day lasting at least d minutes, $I_h = 0.92$ is the fraction of all edge ISPs that Hubble monitored, and $T_h = 0.01$ is my estimate for the fraction of total poisonable (transit) ASes on paths from Hubble VPs to their targets. Because the smallest d that Hubble provides is 15 minutes, I extrapolate the Hubble outage distribution based on the EC2 data (Section 3.4) to estimate the poisoning load for $d = 5$.

Scaling the parameters estimates the load from poisoning under different scenarios. In Table 5.2, I vary the fraction of participating ISPs (I), the fraction of poisonable ASes being monitored (T), and the minimum outage duration before poisoning is used (d). I scale the number of outages linearly with I and T. I scale with d based on the failure duration distribution from my EC2 study.

LIFEGUARD could cause a router to generate from tens to tens of thousands of additional updates. For reference, a single-homed edge router peering with AS131072 sees an average of approximately 110,000 updates per day [13], and the Tier-1 routers I checked made 255,000 to 315,000 path updates per day [87]. For cases where only a small fraction of ASes use poisoning ($I \leq 0.1$), the additional load is less than 1%. For large deployments ($I = 0.5$, $T = 1$) and short delays before poisoning ($d = 5$), the overhead can become significant (35% for the edge router, 12-15% for Tier-1 routers). I note that reducing the number of monitored paths or waiting longer to poison can easily reduce the overhead to less than 10%.

5.5 LIFEGUARD Case Study

To demonstrate *LIFEGUARD*'s ability to repair a data plane outage by locating a failure and then poisoning to restore connectivity, I describe a case study of a failure affecting routing between the University of Wisconsin and a PlanetLab node at National Tsing Hua University in Taiwan. *LIFEGUARD* announced a production prefix and a sentinel prefix via the University of Wisconsin BGP-Mux. It also hosted an isolation vantage point at the University of Wisconsin. *LIFEGUARD* had monitored the PlanetLab node for a month, gathering historical atlas data. On October 3-4, 2011, I used nodes in the two prefixes to exchange test traffic with the PlanetLab node, and I sent traceroutes every 10 minutes from the PlanetLab node towards the test nodes to keep track of its view of the paths. As in the evaluation in Section 5.4.3, I use the measurements from the Taiwanese PlanetLab node to evaluate *LIFEGUARD*, but I did not give the system access to these measurements during failures.

After experiencing only transient problems for most of the day, around 8:15pm on October 3, the test traffic began to experience a persistent outage. When *LIFEGUARD* isolated the direction of the outage, spoofed pings from *LIFEGUARD*'s vantage point reached Taiwan along the forward path from Wisconsin, but spoofed pings towards Wisconsin failed, indicating a problem on the reverse path. A spoofed forward traceroute from Wisconsin revealed the forward path. *LIFEGUARD*'s atlas revealed two historical reverse paths from Taiwan back to Wisconsin. Prior to the outage, the PlanetLab node had been successfully routing to Wisconsin via academic networks. Older measurements in the historical atlas showed that this academic route had been in use since 3pm, but, prior to that, the path had instead routed through UUNET (a commercial network) for an extended period. The UUNET and academic paths diverged one AS before UUNET (one AS closer to the Taiwanese site). *LIFEGUARD* issued measurements to see which of the hops on these reverse paths still could reach Wisconsin. These measurements establish a reachability horizon with UUNET and the ASes before it behind the failure. During the failure, all routers along the academic path from the divergence point to Wisconsin were still responsive to pings, meaning they had a route to the University of Wisconsin. However, hops in UUNET no longer responded to probes. In fact, hand inspection of traceroutes from the PlanetLab node to Wisconsin showed that, at 8:15pm, the path had switched to go via UUNET, and the traceroutes had been terminating in UUNET.

Because the hops on the academic route had paths to Wisconsin, it was a likely viable alternative to the broken default path, and I used *LIFEGUARD* to poison UUNET. For a brief period after *LIFEGUARD* announced the poison, test traffic was caught in a convergence loop. After convergence, both the test traffic and a traceroute from Taiwan successfully reached the production prefix via the academic networks. Traceroutes to the sentinel prefix continued to fail in UUNET until just after 4am on October 4, when the path through UUNET began to work again. In summary, *LIFEGUARD* isolated an outage and used poisoning to re-establish connectivity until the outage was resolved. Once repaired, the poison was no longer necessary, and *LIFEGUARD* reverted to the baseline unpoisoned announcement.

5.6 Discussion

5.6.1 Poisoning Anomalies

Certain poisonings cause anomalous behavior. Some networks disable loop detection, accepting paths even if they contain their AS. Other networks do not accept an update from a customer if the path contains a peer of the network. I discuss these two issues separately.

Some networks with multiple remote sites communicate between sites across the public Internet, using the same AS number across sites. One approach is to disable BGP's loop prevention to exchange prefixes from the remote sites, even though they share an origin AS.[9] The paths for these prefixes allow the remote sites to route to each other using BGP. Fortunately, best practices mandate that, instead of disabling loop detection altogether, networks should set the maximum occurrences of their AS number in the path. For instance, AS286 accepts updates if it is already in the AS path. Inserting AS286 twice into the AS path, however, causes it to drop the update, thus enabling the use of poisoning. In general, I would expect ASes that need to use the public Internet to communicate

[9]Another option is to establish a virtual backbone using some form of tunneling.

between remote sites to be stubs (meaning they have no customers). *LIFEGUARD* uses poisoning to enable paths to bypass faulty transit networks, so has no need to poison stubs.

Some networks do not accept an update from a customer if the path contains one of the network's peers. For example, Cogent will not accept an update that includes one of its Tier-1 peers in the path, meaning that announcements poisoning one of these ASes via Georgia Tech did not propagate widely. However, I could poison them via BGP-Mux instances at universities that were not Cogent customers, and 76% of route collector peers were able to find a path that avoided a poisoned AS through which they had previously been routing. While the filtering reduced the coverage of my poisoning studies and will prevent poisoning from rerouting around some failures, it is likely not a big limitation. First, most failures occur outside of large networks (such as Cogent and Tier-1s) [108, 136]. Second, I used poisoning to experiment with AS avoidance, but most of the techniques and principles still apply to other implementations, such as a modification of BGP to include a new signed AVOID_PROBLEM(X,P) notification.

5.6.2 Address Use by the Sentinel Prefix

Above, I proposed using a less-specific prefix with an unused sub-prefix as a sentinel. I discuss the trade-offs when using three alternative approaches. First, absent any sentinel, it is not clear how to check for failure recovery or to provide a backup unpoisoned route. Second, if an AS has an unused prefix that is not adjacent to the production prefix, it can use the unused prefix as a sentinel, even though it may lack a super-prefix that covers both prefixes. This approach allows the AS to check when the failed route is repaired. However, this scheme does not provide a "backup" route to the production prefix for networks captive behind the poisoned AS. Third, if an AS lacks unused address space to serve as a sentinel, a less-specific prefix will ensure that the poisoned AS and ASes captive behind it still have a route, even though the less-specific prefix does not include an unused sub-prefix. Pings to the poisoned AS and its captives will return via routes to the unpoisoned less-specific prefix, and so they can be used to check for repairs.

Alternatively, a provider with multiple prefixes hosting the same service can use DNS redirection to test when a problem has resolved, without using additional addresses. Such providers often use

DNS to direct a client to a nearby data prefix. This scheme relies on clients using the same route to reach all prefixes in the absence of poison. To establish that this property holds, at least for Google, I resolved a Google hostname at 20 PlanetLab sites around the world, yielding a set of Google IP addresses from various data centers. I then issued traceroutes from the PlanetLab sites to the set of IP addresses. Each PlanetLab site used a consistent path to reach Google's network for all IP addresses. Google then routed the traffic to a particular data center. With this consistent routing, if a provider discovers a routing problem affecting a set of clients *C*, it could poison the prefix *P1* serving those clients. It need not poison its prefix *P2* that serves other clients (possibly from a different data center). Periodically, the provider's DNS resolvers could give a client from *C* an address from *P2* and an address from *P1*, with *P1* serving as a failover. By checking server logs, the provider could discern if the client was able to reach *P2*. When clients can reach *P2*, the provider can remove the poison on *P1*.

5.7 Summary

Increasingly, Internet and cloud-based services expect the Internet to deliver high availability. Nevertheless, partial outages that last for hours occur frequently. To improve their availability, edge providers would benefit from having a way to repair outages affecting them, even when the problem occurs in a transit AS outside their control. In this chapter, I presented *LIFEGUARD*, a system that locates the AS at fault and routes around it. I designed *LIFEGUARD* to focus on long-lasting partial outages, which account for a substantial portion of unavailability; to locate failures along either the forward or reverse path, since failures are often unidirectional; and to trigger BGP reroutes, because alternate policy-compliant paths frequently exist. Using multiple vantage points and spoofed probes, *LIFEGUARD* can identify the failing AS in the wild. I show that a provider can use BGP poisoning to cause routes to avoid an AS without disrupting working routes. *LIFEGUARD*'s mechanism approximates a BGP modification to notify ASes of problems. Because it uses existing protocols and devices, an ISP can deploy my approach unilaterally today. *LIFEGUARD* enables experiments that I hope will motivate the need for and design of new route control mechanisms.

Chapter 6

RELATED WORK

In this chapter, I present the existing work that this dissertation built upon.

6.1 Internet Measurement

6.1.1 Measurement Tools and Techniques

The traceroute tool [59], originally by Van Jacobson, influenced much of this work. I used it extensively in the systems and studies in this dissertation. Its usefulness to me and others motivated me to pursue a reverse path equivalent, and Jacobson's approach, to repurpose a packet's Time-To-Live (TTL) to reveal otherwise hidden information, inspired me to similarly repurpose other features of the Internet protocols. Section 2.3.2 discusses traceroute at length.

IP options: Researchers used the timestamp and record route IP options to measure various aspects of the Internet, inspiring my use of them in reverse traceroute. Fonseca *et al.* found that routers on half of the paths between pairs of PlanetLab sites dropped packets with the timestamp or record route IP options enabled [40]. Although the title of the report dramatically proclaimed that "IP options are not an option," the authors concluded that, based on the location of drops, options could form the basis for a wide-area system. In particular, they found that 90% of drops occurred at the edges of the Internet and that a small number of ASes accounted for most of the drops. In other words, the core of the Internet rarely dropped options packets, and reconfigurations in a small number of ASes could greatly improve overall support. Section 4.2.2 described how reverse traceroute uses source spoofing to avoid some filters.

The Passenger [116] and DisCarte [115] projects showed that the record route option, when set on traceroute packets, can reduce false links, uncover more routers, and provide more complete alias information. The approach has two benefits: record route and traceroute can discover different IP addresses for the same router, enabling alias discovery, and some routers may respond to one technique but not the other, with the combination supplying more complete data. The central challenge addressed by the techniques is how to align the two measurements in order to properly resolve the aliases. Reverse traceroute uses alias results from DisCarte to intersect traceroutes with record routes.

iPlane used the record route and timestamp options to determine if a path traversed a link symmetrically [76].[1] Reverse traceroute generalized and expanded upon iPlane's techniques in order to measure reverse paths in general, not just symmetric ones. Additionally, my colleagues and I studied timestamp support across the Internet and used the timestamp option to resolve aliases [114]. The results from that study led to refinements in how this dissertation uses the timestamp option.

IP source spoofing: While the work in this dissertation is the first I am aware of to use IP source spoofing to measure forward and reverse paths separately, others have investigated IP source spoofing in other contexts. Govindan and Paxson used source spoofing to conclude that most routers generated ICMP Time Exceded (traceroute) responses within a few milliseconds [44], demonstrating that traceroute-measured latencies accurately reflect the round-trip latency experienced by other packets. To arrive at this conclusion, they sent a packet from one vantage point to another. Then, they sent a TTL-limited packet from the first vantage point, spoofing as the other. Assuming the routing is destination-based, both the spoofed and non-spoofed probe follow the same one-way path between the vantage points. The researchers measured the one-way latency of the two probes by synchronizing the clocks on the two machines, then compared these two values, assuming the difference between them stemmed from ICMP generation. In Section 4.4.1, I adapted their approach to assess how often routers violate destination-based routing.

Spoofing can aid in nefarious acts, potentially enabling anonymity, indirection, and amplication exploits [12]. For this reason, some people have looked at restricting it. Strategies exist to enable

[1] I helped work on iPlane's IP options techniques, inspiring reverse traceroute.

routers to filter spoofed packets [39, 10], though it can be difficult to determine which packets have legitimate source addresses. Techniques that rely on lists of permissible addreses [39] are difficult to configure and maintain, especially given realities such as multi-homing. Strict uRPF filters a packet if it arrives over an interface other than the one that would be used to route towards the packet's source [10]. Given path asymmetry, however, this strategy is impractical in many networks, as it could filter non-spoofed traffic. Loose uRPF only allows packets with addresses present in the router's routing table, which is too lenient to prevent much spoofing [107]. Given these limitations, these approaches to filtering spoofed packets are practical only at the edge; in the core, strict uRPF is too restrictive, and loose uRPF is not restrictive enough. Because filtering is only feasible near the edge, reverse traceroute identifies which vantage points can and cannot spoof, then uses those that can to spoof as any source in the system.

The MIT Spoofer project offers software for users to download to test whether their provider allows spoofing [117]. According to the most recent published stufy, 31% of clients could spoof arbitrary routable IP addresses [12]. I found the fraction of PlanetLab sites with providers that do not filter spoofed packets to be similar.

BGP poisoning: Previous research used poisoning as a measurement tool to uncover hidden network topology [29] and to assess the prevalence of default routes [21]. In the former, researchers poisoned ASes on BGP paths to their prefix, in order to expose backup routes. They observed the new routes using BGP route collectors [105]. One of the motivations was to discover which routes would be used following failures. In the latter, researchers tested for the presence of a default route by poisoning an AS, then testing if the AS could still reach the poisoned prefix. A router uses a default route if it does not have a specific route for a prefix. By poisoning the AS for their prefix, the researchers guaranteed that the AS did not have the prefix (assuming that the AS used BGP loop prevention). They used ping to test if the AS still had a path to the prefix. Their results showed that most small ASes used default routes, and most large ASes did not.

Sentinel prefix: Entact used overlapping BGP prefixes to simultaneously measure alternative paths [139]. Bush *et al.* used an anchor prefix to determine whether a test prefix should be reachable [21]. Similarly, *LIFEGUARD* uses a sentinel prefix to test for repairs along the failing path while rerouting traffic to a path avoiding the outage.

Addressing limitations of traceroute: While this dissertation addressed the lack of reverse path visibility, the principal limitation of traceroute [122], traceroute has other limitations that are addressed by existing research.

Some routers along the path may not appear in the traceroute. *Hidden routers*, such as those inside some MPLS tunnels, do not decrement TTL and so are completely invisible to traceroute. Others, referred to as *anonymous routers*, turn up as '*'s in traceroute because they decrement TTL but do not send ICMP time-exceeded messages. Previous work identified the problem of anonymous routers and examined heuristics to overcome their effect on inferred topologies [134]. Because some routers may respond to record route and not traceroute, DisCarte aligns traceroute results with record route entries to discover many hidden and anonymous routers, yielding more accurate paths [115].

Rather than using a single path for all traffic to a particular destination, some networks load balance the traffic across multiple paths. In these cases, traceroute may fail to expose the multiple paths or, more problematically, may infer false links when probes to adjacent hops traverse different paths. Routers often balance packets across the multiple paths by hashing on some of the header fields in the packets. By systematically varying the values of these fields, Paris traceroute addresses these concerns, providing consistent paths and a way of exposing the multiple options given certain types of load balancing [6, 7].

Techniques to provide some reverse path visibility: While reverse traceroute is the first system to measure reverse paths from arbitrary hosts, other researchers recognized the problems introduced by path asymmetry and designed systems to provide some visibility into reverse paths or to identify cases of asymmetry. Section 2.3.3 mentioned that traceroute servers and loose source routing offer the ability to measure some paths, but both have very incomplete coverage.

Recognizing that these approaches did not provide a complete solution, Burch used reverse TTL values to estimate reverse routing [20]. The approach generated a view of the Internet's topology using forward traceroutes from a measurement vantage point and considered paths through this topology as candidates for the reverse path. Then, it examined the TTL value of the traceroute responses, known as the return TTL values. Since routers generally initialize the TTL field to one of a small set of common values and each router along the path generally decrements the value, the remaining TTL value when it reaches the vantage point suggests the number of routers on the reverse path. The technique used these reverse path lengths to eliminate candidate reverse paths. Using known paths from traceroute servers as ground-truth reverse paths (as I did in Section 4.4.1), Burch revealed the limitations of his novel approach. First, the forward topology included less than half of the links on the reverse paths, meaning that the technique could not possibly find the correct path if it contained one of these links. Second, the return TTL values often did not eliminate enough candidates, with multiple potential reverse paths remaining in most cases.

Other projects inferred information about reverse paths, without attempting to measure what routes were taken. For example, PlanetSeer included techniques to infer if a failure was on the forward [136]. PlanetSeer passively monitored clients of the CoDeeN content distribution network and launched active probes when it observed anomalies. PlanetSeer triggered measurements from multiple vantage points when it detected a problem. PlanetSeer inferred a forward failure in the following cases: (1) a traceroute to the destination returns an ICMP Destination Unreachable message; (2) TCP packets from the client suggest ACK losses on the forward path; or (3) a forward path route change accompanied the outage. In cases when these techniques did not apply, it could not determine the direction of failure. These techniques were attemping to solve the same problem as Hubble's source spoofing technique in Section 3.1.4. However, PlanetSeer's techniques could only infer a forward failure in 28% of cases. Further, in more than half of these cases, the system inferred the failure of an old forward path from observing a path change, but made no determination as to why the new path had failed. In Section 3.3.3, I presented results showing that, for sources able to spoof, Hubble isolated the direction of failure in 68% of cases.

Tulip identified packet reordering and loss along either the forward or reverse path [80]. When routers source new packets (as opposed to simply forwarding existing packets), they include an IP-ID value in the packet, and many routers use an incrementing counter for these values [62]. Tulip sent multiple ICMP requests to a single router. If the responses came back in a different order than the requests were sent, it determined that the reordering occurred on the reverse path if the response to the first packet sent had a lower IP-ID value (indicating it reached the destination first), and the reordering occurred on the forward path if the response to the first packet had a higher IP-ID value. Tulip included a similier IP-ID-based technique to isolate packet loss to the forward or reverse path.

Another line of research seeks to recruit traceroute vantage points in diverse locations to provide a rich view of Internet routing, similar to the role of public traceroute servers. Some of these deployments consist mainly of hosts at research institutions [101,5,4,98,120], available for Internet measurements or, in some cases, other experiments. Other deployments recruit users from outside academia to diversify the viewpoints [111,26]. Similar in spirit is Teixeira and Rexford's proposal, in which they argue for each AS to host servers, for distributed monitoring and querying of current forwarding path state [126]. This work complements reverse traceroute, and, in the future, it would be great if these efforts bundled reverse traceroute vantage point code to improve the system's coverage.

6.1.2 Systems that Would Benefit from Reverse Path Information

Many systems seem well-designed to make use of reverse path information, but, lacking it, make various substitutions or compromises. Reverse traceroute is the first technique to measure complete reverse paths from arbitrary destinations, and I now describe some recent systems that could benefit from this ability.

Some geolocation systems used delay and path information to estimate the geographic position of targets [61, 131], but the lack of reverse path information hurt the estimates. Geographic locations of hosts can be useful for targeted advertising and other applications. These approaches issue traceroutes from distributed vantage points to a target with an unknown location (and possibly to other hosts). Because packets must obey the speed of light, measured latencies limit how far apart hops along a path can be. The techniques attempt to constrain the location of the target enough to accurately estimate its geographic location. However, lacking reverse path data and accurate link latencies, the locations tended to be under constrained, often leading to inaccurate estimates with low confidence. For example, both techniques had errors of over 100 miles for more than one in ten targets.

iPlane estimates the latency between a pair of arbitrary Internet hosts [78], but the estimates are more accurate when the system can measure paths out from the hosts [77]. These latency predictions can enable performance optimization in systems in which a host can choose between multiple possible hosts to connect to. The system works by: (1) measuring traceroutes from PlanetLab vantage points to prefixes around the Internet to build an atlas of Internet routing; (2) composing segments of those traceroutes to predict the paths in both directions between the pair of hosts; and (3) estimating the latency of the composed paths using latency measurements from the PlanetLab vantage points [78]. iPlane has a better chance of predicting a path from a host correctly if it has access to even a few paths from the host, but it lacks access to vantage points in most networks [77]. Reverse traceroute could provide such paths from arbitrary hosts. Also, iPlane's latency estimates are more accurate in cases when it can accurately measure one-way link latencies [76], but it could only do so for links that were traversed symmetrically. My link latency measurement technique in Section 4.6.2 could provide iPlane with better latency values. My work benefitted greatly from iPlane, as I borrowed its atlas-based approach, as well as its tools and measurements.

iSpy attemped to detect an attacks known as prefix hijacks using traceroutes [140], but, in order to do so, assumed paths and failures were symmetric. In a prefix hijack, an attacking AS announces itself as the origin for another AS's prefix, in order to lure traffic to it and away from the legitimate owner. The researchers behind iSpy had the novel observation that the pattern of path outages caused by a hijack likely differs from the pattern of outages caused by a failure. In particular, in considering the tree of paths to a prefix, a failure in the tree only cuts off the subtree beyond that failure, whereas a hijack lures paths to the attacking AS, and so may cut off diverse parts of the tree. iSpy uses traceroutes from a prefix to distributed destinations in order to build this tree before any problems develop. Later, when the prefix experiences problems reaching some destinations, the system looks at which paths are disrupted to determine if a hijack or a failure likely explains the set of outages. This aspect of iSpy is similar to *LIFEGUARD*'s reachability horizon. However, a hijack disrupts only reverse paths back to the prefix, but iSpy builds the tree using forward traceroutes from the prefix. Most paths are asymmetric [51], and, in Section 3.3.3, I presented results showing that many failures are unidirectional. Therefore, the use of forward paths to detect a signature of cuts in reverse paths could be misleading. By incorporating reverse traceroute, iSpy could instead build its tree of paths using reverse paths. *LIFEGUARD* uses reverse traceroute and spoofed probes to more accurately determine the reachability horizon.

Likewise, intriguing recent work on inferring topology through passive observation of traffic based its technique on an implicit assumption that the hop counts of forward and reverse paths are likely to be the same [35]. The use of passive measurements could reduce the overhead required to measure the Internet's topology. This approach supplemented a limited number of traceroutes from vantage points with passive observations of IP traffic to the vantage points. It used the return TTL values in the passively captured packets to cluster them and to align them with the traceroutes. However, the traceroutes measure forward paths from the vantage points, whereas the TTL values come from packets to them. Asymmetric paths can have different hop counts. Using reverse traceroute as part of the approach would allow it to make both active and passive measurements along the same direction.

Netdiff sought to enable performance comparisons of paths across various ISPs [83], but it is unclear how accurate the comparisons are in cases of asymmetric routing. These comparisons could help applications or customers decide which ISP to use. The system uses traceroutes from end-hosts to measure the latencies from ingress points to egress points in transit networks along the paths. As shown in Section 4.6.2, traceroute can yield very inaccurate latency estimates when paths are asymmetric. Netdiff partially accounted for this by throwing out paths that are more than three hops longer in one direction than the other. However, the example in Section 4.5 shows that even paths with the same lengths can follow vastly different routes. Reverse traceroute could allow Netdiff to discard asymmetric paths, and the reverse traceroute-based technique for measuring link latencies in Section 4.6.2 could allow Netdiff to provide accurate comparisons even in cases of asymmetry.

6.2 Availability Problems

In this section, I survey relevant work on Internet availability problems. The background in Section 2.4 presented the most pertinent information on performance problems.

6.2.1 Characterizing and Locating Failures

A rich body of existing works explores various aspects of Internet outages, characterizing certain types of problems and locating them in various ways. In this section, I group them based on the approach used to detect the problems.

Passive BGP Monitoring: Numerous studies have modeled and analyzed BGP behavior to understand failures. For instance, Labovitz *et al.* [69] found that Internet routers may take tens of minutes to converge after a failure, and that end-to-end disconnectivity accompanies this delayed convergence. Mahajan *et al.* [82] showed that router misconfigurations could be detected with BGP feeds. Caesar *et al.* [22] proposed techniques to analyze routing changes and infer why they happen. Feldmann *et al.* were able to correlate updates across time, across vantage points, and across prefixes by assuming that the failure lies either on the old best path or the new best path. They can pinpoint the likely cause of a BGP update to one or two ASes [38]. Wang [130] examined how the interactions between routing policies, iBGP, and BGP timers lead to degraded end-to-end performance. BGP beacons [84] benefited this work and other studies. Together, these studies developed techniques to reverse-engineer BGP behavior, visible through feeds, to identify network anomalies.

Intradomain Troubleshooting: Shaikh and Greenberg proposed to monitor intra-domain routing announcements within an AS to identify routing problems [109]. Kompella *et al.* also developed techniques to localize faults with AS-level monitoring [64] and used active probing within a tier-1 AS to detect black holes [63]. Wu *et al.* used novel data mining techniques to correlate performance problems within an AS to routing updates [132]. Huang *et al.* [54] correlated BGP data from an AS with known disruptions; many were detectable only by examining multiple BGP streams.

Active Probing: Other studies used active probes to discover reachability problems. Paxson was the first to demonstrate the frequent occurrence of reachability issues [97]. Measuring Internet failures between vantage points using periodic pings and triggered traceroutes, Feamster *et al.*

correlated end-to-end performance problems with routing updates [37]. These and other studies [4, 129, 33, 43] probed between pairs of nodes in small deployments, allowing detailed analysis but limited coverage. Pervasive probing systems, such as *iPlane* [78] and DIMES [111], exist, but have been designed to understand performance and topology rather than to detect and diagnose faults.

6.2.2 Avoiding Failures

Detouring: A number of systems "detour" traffic around failures or slow paths [106, 4, 47, 15]. These approaches redirect traffic between two hosts through a third detour node. Within that general approach, the work varies studies assessing only pairwise paths between nodes in a testbed [106, 4], to academic systems capable of detouring traffic Web traffic [47], to commercial services [15]. Some solutions find detour paths by using all-pairs path monitoring [4], whereas others use random selection of alternative paths at the time of failure [47]. However, all the work found that better paths often exist, but are not found by existing protocols. In Sections 2.4 and 2.5, I summarized some of the results. The main drawbacks of detour-based solutions are the cost of maintaining the detour nodes and the fact that a composed detour path through an intermediary often is not policy-compliant as an end-to-end node. However, when economical, these approaches provide a great way to alleviate many problems.

Protocol modifications: Since transient loss frequently occurs during routing protocol convergence, a number of proposals suggest protocol modifications. These approaches include carrying information about failures in packets themselves, to signal that the packets should not be routed back towards the failures [71]; modifying BGP to include backup paths for use during convergence [68]; or modifying BGP to guarantee that ASes have consistent views of available routes [60], addressing the underlying cause of convergence loss.

MIRO proposes to enable AS avoidance and other functionality by allowing ASes to advertise and use multiple inter-domain paths [133]. While the proposal retains much of the quintessence of BGP and current routing policies, it does require modifications to protocols and to routers. The deployable solutions and results on failure avoidance in my dissertation could perhaps present some of the argument for the adoptation of MIRO-like modifications.

Chapter 7

CONCLUSIONS AND FUTURE WORK

In this chapter, I summarize the work presented in this dissertation, review the thesis it supports and the contributions made, and outline some lines of future work that could build upon my work.

7.1 Thesis and Contributions

In this dissertation, I supported the following thesis: *It is possible to build effective systems that can help service providers improve Internet availability and performance and that are deployable on today's Internet.*

The work presented in this dissertation made the following contributions:

Measurement studies characterizing long-lasting Internet outages. My system Hubble demonstrated that it is feasible to monitor outages on an Internet scale, with 15 minute granularity. I conducted a measurement study with Hubble, along with two related studies, to help understand long routing failures. The studies show that these prolonged issues contribute much of the unavailability observed by vantage points monitoring destinations around the Internet. Many of the problems do not manifest in the observable control plane, with available route collectors showing BGP routes to a destination even as data plane traffic fails to reach the destination. During these long-lasting failures, working alternate routes often seem to exist. Most of the problems during the studies were partial, with some vantage points able to reach destinations that were unreachable from other vantage points. Many of the failures were unidirectional, and measurements often reveal what seem to be functioning policy-compliant paths around the problems. These characteristics discovered by my studies guide my approach to addressing the problems.

The design and implementation of reverse traceroute. Because existing tools cannot measure reverse paths, I developed a distributed system that can measure the reverse path back from arbitrary hosts. The system works in many of the same cases as traceroute, a widely-used troubleshooting tool, and it supplements the forward path measurements traceroute provides. Reverse traceroute relies on the fact that Internet routing is generally destination-based, allowing the system to measure the path incrementally back from the destination. It uses IP options to measure each hop; limited source spoofing overcomes many of the shortcomings of relying on IP options. I demonstrate the benefit of reverse path information in understanding inflated latency and in measuring link latencies and peering links–Internet measurement goals that are challenging with existing tools. Reverse traceroute is also a key component of my technique to locate failures.

An approach to remediating persistent Internet outages by first locating the AS containing the failure, then redirecting routes around it. I designed and implemented *LIFEGUARD*, a two-component system to resolve long-lasting partial outages. Because unidirectional failures are common, *LIFEGUARD* builds on reverse traceroute to locate failures along either direction of a path. To have a solution that works today, I developed an approach that empowers a destination AS to use BGP poisoning in a controlled fashion to route others around a failure. I also described how this poisoning approximates the control the AS could achieve given a new BGP mechanism designed for this purpose.

7.2 Key Approaches

Some common approaches emerge from my work:

(Mis)use of Internet protocols. The Internet protocols have been hugely successful at supporting Internet growth, but this success is partly at the expense of providing rich functionality. Part of the Internet's success stems from the protocols' abilities to scale gracefully and to permit the Internet's expansion in terms of users, applications, and devices. The protocol designers facilitated this growth by stressing inter-operability and AS autonomy over other concerns like performance or transparency. These priorities complicate ASes' abilities to optimize performance and availability. For example, ASes make autonomous decisions with little visibility into or control over other networks, instead of having one centralized authority with a global view calculate Internet-wide routing. The relatively simple design eases adoption, allowing the Internet to grow in ways that the designers may not have explicitly planned for. So, paths in the early Internet were generally symmetric, leaving little need for visibility into reverse paths. Now, most paths are asymmetric, causing a need for bidirectional visibility.

Because the Internet protocols do not explicitly provide some of the needed functionality, I repurpose aspects of the protocol in pursuit of better performance and availability. For example, because the Internet does not enforce source authentication, I use source spoofing to isolate the two directions of a path when determining the direction and location of a failure. Similarly, I use the IP timestamp option not because I need the timestamp values from routers, but rather to determine the order in which a packet traverses a pair of routers. Because BGP does not provide explicit problem notifications or source authentication, I repurpose BGP's loop prevention mechanism to let an origin AS inform other ASes of a routing problem.

Separation of forward and reverse paths. When possible, I separate forward and reverse path considerations to make more precise measurements and decisions. I use source spoofing to probe the two directions of a path individually. Spoofing aids failure isolation and permits the use of vantage points in the best position to make reverse traceroute measurements. The principle also helps in failure avoidance. Many deployable approaches to failure avoidance reroute both the forward and reverse direction [4, 47], regardless of which failed. I use a unidirectional approach to repair unidirectional outages, allowing the working direction to continue to use its existing route.

Dynamic coordination of measurements from multiple vantage points. To obtain a view unavailable from any single vantage point, my measurement techniques rely on dynamic coordination of probes between multiple vantage points to provide richer data – the result of one vantage point's measurement may trigger a probe from another, or one might send a probe to be received by another. Taking advantage of topological relationships that vantage points have to each other and to destinations, I issue probes that approximate the effects of vantage points I lack. Recent measurement architectures focus on providing platforms for dynamically coordinated measurements [55, 2, 65], and my techniques seem like good fits.

All my probing systems relied on dynamic coordination. For example, my systems to characterize and isolate failures rely on one vantage point to identify a problem and notify others to issue further probes to understand the problem. Reverse traceroute uses measurements from various vantage points to stitch together a path that no one vantage point could measure on its own. In various places, I have one vantage point spoof the source address of another to decouple forward path measurements from reverse path measurements.

Historical views of Internet routing. I maintain historical atlases of Internet routes and topology, and I use these historical views to improve live measurements and to understand and contextualize current routes. The use of atlases in iPlane [78, 79] heavily influenced my work; in some instances, I use iPlane's measurements as the basis for my atlas. I use an atlas of the Internet's topology and business relationships to reason about alternate paths during failures. A historical view of routing before failures allows a comparison of paths during a failure to the routing before the failure. This information is particularly useful for isolating reverse path failures. Reverse traceroute uses atlas measurements to decide which vantage point is nearest to a destination and to generate candidate next hops for timestamp measurements.

7.3 Future Work

The work presented in this dissertation can be extended in several ways. Sections 7.3.1, 7.3.2, and 7.3.3 discuss how Hubble, reverse traceroute, and *LIFEGUARD* could be improved and better assessed. Section 7.3.4 suggests ways in which Internet protocols and routers might be modified to better support the goals of this dissertation.

7.3.1 Improving Understanding of Failures

Although the studies in Chapter 3 enrich our understanding of outages, we could learn more along a number of dimensions. *LIFEGUARD*'s failure isolation techniques could also be used to characterize outages in more detail that I did in Chapter 3.

It would be worthwhile to design a system to identify and understand fine-grained failures. To avoid introducing too much probe traffic, Hubble made a number of compromises that meant that it only identified fairly lengthy problems that affected multiple vantage points. If future systems probe more efficiently, they may be able to improve upon Hubble in this regard. Such systems could provide outage detection to feed into *LIFEGUARD*'s failure location and avoidance. Here, I describe the approaches Hubble took and how future systems might address them.

To find problems faster (and hence detect shorter-lived outages), systems could probe targets more frequently. Before failures, Hubble in aggregate only probed each destination every two minutes, which meant that it took awhile for individual vantage points to cycle through the targets. A number of strategies could reduce the cycle time. For one, a system could spread its monitoring of a single "target" across multiple destinations within the same prefix, such that the system probed the prefix more often without probing an individual host more often. Also, it might be possible to identify hosts in some prefixes that the system could probe at faster rates without inducing rate limiting or raising alarms. Finally, it may be possible to reduce the total number of targets by merging multiple prefixes into "atoms" that share routes [16], if it is the case that prefixes in an atom experience the same outages.

In addition to cycling through targets faster, a system might be able to cycle through them more intelligently, by predicting which failures might correlate. For example, an outage affecting paths to a core router might also affect paths through that router, suggesting that the system should recheck those paths. During a failure, each Hubble vantage point only issues one traceroute to the destination per 15 minutes, and so the system could miss changes (in routes or reachability) that happen at a finer granularity. Recent techniques may aid in designing adaptive probing rates to try to catch these changes [31], and light weight signals (such as probing only the TTL just past the last hop in the previous traceroute) could be used to determine when to issue a full traceroute.

Whereas Hubble used the same vantage points to monitor all targets, a system that chose them per target could optimize for path coverage. Hubble only considered failures affecting at least a few of its vantage points. It would be interesting to also consider more localized failures, or instances when a given source experienced simultaneous problems reaching multiple destinations (perhaps due to a shared reverse path).

7.3.2 Improving Reverse Traceroute

Reverse traceroute for all. While reverse traceroute can measure individual paths, the system as presented in this dissertation cannot support a huge number of measurements or sources. The current system only uses a small number of PlanetLab sites as sources. In this section, I discuss ongoing and future plans to improve the scalability of the system.

First I discuss why it would be helpful to be able to issue more measurements. The needs of operators and ASes partly motivated the need for a technique to measure reverse paths. As argued in this dissertation, reverse traceroute provides information valuable for improving performance and availability. If operators across the Internet could use it in their daily operations, just as they use traceroute today, it would improve their ability to provide high availability and good performance. To realize that benefit, the system needs to be available for use, which requires opening it and being able to support the measurement load. ASes could also use such an open system for other purposes, such as providing intelligence to guide peering decisions.

Reverse traceroute can also help researchers understand the Internet. As the system grows to support more measurements, it will be able to support more studies and richer studies with better coverage. A similar transformation occurred as researchers went from using traceroutes between a small number of testbed nodes to Internet-scale measurements, and as services like RouteViews [87] and RIPE RIS [105] made more BGP feeds available. A more scalable reverse traceroute could enable a number of research directions. Section 4.6.2 demonstrated that the system can be used to accurately measure backbone link latencies. A link latency map of the Internet would be useful for applications including comparing ISP performance [83], optimizing wide-area routes, and predicting end-to-end latencies [79]. Reverse traceroute could enable a novel approach to group IP addresses into routers or PoPs. Reverse traceroute enables direct measurement of how arbitrary hosts route towards reverse traceroute sources, essentially giving a partial view into the hosts' routing tables that could be used for clustering. Similarly, techniques to infer routing policy work by comparing observed paths to those that could exist, but are not observed - that is, by comparing taken paths to paths not taken [42, 124, 118]. By measuring paths back to reverse traceroute sources from locations around the Internet, a scalable reverse traceroute could provide uniquely detailed maps of the route and topology options available to ASes' policy engines. This ability to build a detailed routing tree of paths back to a given source could provide the basis for new observations about routing. Reasoning about changes over time to these reverse path maps could lead to techniques for inferring the underlying causes of routing changes. My own system *LIFEGUARD* would be able to monitor many more paths if it could issue reverse traceroute measurements at a much faster rate.

While one approach to enabling many more measurements would be for different ASes or research groups to run their own systems, I believe the community is best-served by a single unified system. First, if groups interested in using reverse traceroute contribute to such a system, instead of running their own, their vantage points contribute to the common good, improving coverage and accuracy for all, and improving measurement capacity. Public traceroute servers provide a model of ASes providing measurement vantage points for others to use. Second, as I discuss further below, because of the overlap of paths, reverse traceroute allows many opportunities for measurement reuse, and so a unified system may be more efficient. Third, such a system allows for consolidated probe rate limiting. Many routers have strict limits on the rate at which they reply to probes, especially those with IP options, and so careful accounting can increase the chances of replies. Finally, because reverse traceroute uses techniques rarely used elsewhere, it is beneficial to provide a central point of contact for router vendors and ASes. Although all reverse traceroute probes are protocol compliant, the ease of communication may come in handy if the unusual probes raise any concerns or questions. The trust this encourages may be useful in negotiating updates to standards or router configurarions in support of the system.

Most scalability improvements seem likely to come from shifting from focusing on measuring individual paths to measuring sets of paths. As sketched in Section 5.4.4, it is possible to refresh a stale path faster than measuring a new path from scratch, as the system can optimize the probes it sends to quickly determine whether or not parts of the path had changed. That evaulation used a very primitive version of this approach, reusing all measurements within the same fixed period. Instead, by accounting for how quickly routes change in different parts of the Internet, a future system could cache reverse paths and reverse path segments for appropriate periods of time, reusing them when possible. The system could also more quickly identify changes by, for example, using all four IP timestamp slots to confirm multiple hops on the path at once. To enable Internet-scale studies and systems, it may make sense to have the system continually map reverse hops from PoPs across the core of the Internet towards a set of sources, then use this atlas to piece together paths as needed. In building such an atlas, it would likely be useful to carefully stage the measurements, to maximize reuse and minimize the effects of rate limiting. Towards that end, it may be helpful to model or measure the rate limiting in place at routers – it may be possible to learn and account for the different configurations in use by different router models and different ASes.

Some other improvements to the system seem possible at the level of individual path measurements. For example, better alias information and better information about load balancers may allow a reverse traceroute measurement to intersect a known traceroute to the source sooner. A quicker intersection reduces the number of required measurements. Similarly, better understanding of issues related to the system's accuracy and coverage, as proposed below, could lead to fewer wasted probes.

Understanding limitations on reverse traceroute's accuracy. Section 4.4.1 demonstrated that reverse traceroute accurately measures most reverse paths. In that section, I also described the reasons why a reverse traceroute might yield a different path than a traceroute over the same path.

I can extend this analysis in a number of ways. First, while the current analysis captures how often the technique needs to assume symmetry, it would be useful to understand where the technique assumes symmetry and where the assumptions are correct. Preliminary analysis suggests that assuming symmetry within an individual AS rarely introduces path inaccuracies, at least at the AS level, whereas assuming symmetry across AS boundaries often results in incorrect paths. If these results hold, it may be possible to use the number of inter-AS assumptions of symmetry as a metric of confidence for reverse traceroute measurements.

Second, since reverse traceroute assumes that routing is destination-based, it would be helpful to have a better understanding of where this assumption does or does not hold. I performed some analysis of how often paths violate reverse traceroute's assumption that routing is destination-based. I plan to extend it to cover more paths, to classify which ASes violate the assumption, to understand the causes of the violations, and to distinguish which ASes use MPLS.

Third, reverse traceroute requires router support for IP options or vantage points with forward traceroutes that traverse the network in question, and so it would be helpful to understand which parts of the Internet are blind spots for the technique. This analysis could drive the recruitment of vantage points in advantageous positions.

7.3.3 Improving LIFEGUARD

Both *LIFEGUARD*'s failure location subsystem and the evaluation of that subsystem could improve substantially, with two main goals. First, I would like *LIFEGUARD* to be used operationally, and so the system needs to be improved to support that use. Second, with feedback from operators, I would like to use experiences with *LIFEGUARD* to motivate modifications to the Internet's protocols to better support high availability, good performance, and measurement.

Correlating failures across sources and destinations may enrich understanding of each individual failure. While I was able to compare *LIFEGUARD*'s isolation results to ground truth in a limited PlanetLab setting, the accuracy evaluation outside that setting was opportunistic and incomplete, relying on occassional overlaps with operator mailing lists. In the future, it would be great to verify more of the results by emailing operators at the ASes the system identifies as experiencing problems. Feedback from operators could establish whether the system identifies problems affecting real traffic and whether it pinpoints the failure accurately enough to speed recovery. Operators have access to ground truth information within their networks and frequently perform postmortem investigations of outages, and so they could compare the data provided during the failure by *LIFEGUARD* to the results of their investigations that use privileged information. Similar comments from operators provided useful evaluation of existing systems [119, 82].

A number of steps can improve *LIFEGUARD*'s poisoning-based failure avoidance and the evaluation of the system. First, to avoid poisoning in cases that will resolve quickly without that intervention, the system needs to be better able to predict whether an outage will persist. Integrating BGP collectors might help – if routes converge without resolving the problem, it may be likely to persist. A study could also determine if topological characteristics of an outage are helpful in predicting if it will persist. For example, historical measurements could reveal the presence or absence of backup routes. Second, in work in progress, I am checking whether sentinel prefixes experience the same failures as production prefixes – the system depends on this assumption to discern when to revert a poisoning. Third, the evaluation of end-to-end failure location and avoidance relied on a qualitative case study rather than quantitative evaluation. An evaluation of failure avoidance could use *LIFEGUARD*'s failure location to trigger its failure avoidance for many outages over an extended time period, using measurements before and after poisoning from both the sentinel and production prefix to test whether poisoning restored connectivity.

7.3.4 Improving Support for Measurements

Better protocol and router support could substantially aid the goals of this thesis. One important aspect of this support is convincing the operator community that best practices should encourage routers to reply to measurement techniques like the record route probes used by reverse traceroute, just as the community encourages operators to configure routers to reply to traceroute. The other aspect is adding new features to routers and to Internet protocols. I hope that the results in this dissertation and related work [76, 80, 114, 115] help inspire some of these changes. A partial list of potential modifications that would be useful, ordered loosely by how useful they might be, includes:

- **A modified record route option that allows the sender to specify that routers should start recording n hops into the path, rather than from the first hop.** This option would obviate the need to spoof in order to be within record route range. Figure 4.12 showed that many destinations are out of record route range of all reverse traceroute vantage points. This modified option would allow the system to measure reverse hops from all responsive destinations, and it would obviate the need to spoof in order to be within record route range. Note that my systems would still need to spoof to avoid filters and to isolate unidirectional failures.

- **A way to request from a router its next hop towards a specified destination.** This information would be very useful for troubleshooting outages, as it would indicate what the last router on a broken route was trying to do. ASes are unlikely to be willing to reveal that information in general. However, a version of the ICMP Time Exceded message that encoded the next hop might be more palatable, as then sources could only learn about paths they were routing along.

- **The** AVOID_PROBLEM(X,P) **announcement described in Chapter 5.** Although BGP poisoning provided a way to evaluate failure avoidance, it would be preferable if BGP provided more explicit support. Some operators likely would prefer to use an explicitly designed community than to poison. By inserting another AS into the path, an origin AS poisoning one of its prefixes potentially raises concerns that it is misrepresenting the path, that it does not have authority to include the other AS, or that it is disrupting other networks. While I think these concerns are minor, given that the poison only affects traffic to the origin's prefix, an AVOID_PROBLEM(X,P) announcement designed for the purpose would assuage many of these concerns. BGP communities likely provide the easiest path to adoption, though ASes would have to agree not to strip the communities on transit and, ideally, to standardize the format. The community would instruct the problem AS not to export the route (or not to export it if learned from a particular neighboring AS). Ideally, the origin would be able to sign the community to establish that it had not been added in transit; this authentication is in keeping with ongoing efforts to secure BGP [88, 9].

- **A query to have a router return all of its IP addresses (aliases).** Although many techniques exist to resolve aliases [11, 62, 48, 114, 115, 86], even in combination they do not provide complete coverage or accuracy. Combining the results of multiple techniques is problematic, as false positives compound. Further, many are heavyweight and only lend themselves to occasional offline analysis. Better alias information could help the work in this dissertation in multiple ways: reducing the overhead (and possibly increasing the coverage) of reverse traceroute by enabling faster intersection with atlas traceroutes; improving the analysis of reverse traceroute's accuracy by making it easier to compare the results of reverse traceroute with traceroute; and allowing more accurate joining of multiple outages or path measurements at the router level.

- **A ping variant that indicates to the destination that it should route its reply via the router that forwarded it the request (the last hop on the forward path).** Currently, measuring link latencies is very difficult. The constraint-based technique using reverse traceroute from Section 4.6.2 was accurate on the Sprint topology, but it requires many measurements and will not work on under-constrainted topologies. However, measuring the latency of a link that is traversed symmetrically is easy, as one needs only subtract the round-trip latencies to each end of the link. By "bouncing" a ping back symmetrically over the last link, this proposed probe would let us use this subtraction technique for any link. It would also be useful to test connectivity during unidirectional failures; if *H* appears on the forward traceroute but the next hop *H'* only appears on a spoofed forward traceroute, this ping would test whether *H'* could route via *H* to use the working path back from *H*. The bounce would not substantially alter the routing of the ping, as the router it bounced back to would route it normally.

- **"Fast path" support for measurements, and a higher default rate limit configuration.** Many core Internet routers have strict rate limits for how often they reply to IP options packets. Operators configure them with these limits because routers process options in software, rather than on the "fast path" that handles forwarding most option-free packets. The software processing occurs on a typically underpowered processor, which may also be responsible control plane processing [122]. Rate limiting can keep IP options packets from consuming resources needed for other purposes. This dissertation demonstrates how useful options can be, which could help convince router vendors to improve how efficiently they are processed. If routers process options more efficiently, operators can configure them to reply to more. Higher rate limits would enable my systems to probe faster, allowing them to diagnose more problems faster. My systems' productive use of options could also help persuade more operators to support options in their networks, just as best practices today call for routers to support traceroute.

- **An error message a router can send to a source to indicate that the router has exceded its rate-limit for responding to probe packets.** Currently, a lack of response could indicate filtering, congestion loss, a routing failure, or rate-limiting at any router along the path. Knowing that a particular router was rate-limiting would enable measurement systems to be network-friendly by probing at a high, but acceptable rate. Of course, this message itself would be strictly rate-limited.

7.4 Summary

As we depend on the Internet more and more, it is increasingly important that the Internet deliver high availability and good performance. In this dissertation, I presented systems to aid ASes in meeting these goals. Whereas currently operators try to troubleshoot performance problems with a view of only half the path, my reverse traceroute system gives round-trip visibility. My studies characterizing failures demonstrated that long-lasting partial outages contribute much of the unavailability, and my system *LIFEGUARD* provides an approach to addressing them, by locating the failure and rerouting around it. While my current techniques supply useful tools to improve Internet reliability, multiple avenues exist to enhance their applicability and scalability. I hope in the future operators and researchers can depend on reverse traceroute just as they use traceroute today, and prolonged outages decline as our understanding of them and techniques to address them improve.

BIBLIOGRAPHY

[1] Abilene Internet2 network. http://www.internet2.edu/network/.

[2] Mark Allman and Vern Paxson. A reactive measurement framework. In *PAM*, 2008.

[3] Amazon EC2. `http://aws.amazon.com/ec2/`.

[4] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP*, 2001.

[5] Archipelago measurement infrastructure. `http://www.caida.org/projects/ark/`.

[6] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clemence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *IMC*, 2006.

[7] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the Internet. In *IMC*, 2007.

[8] Brice Augustin, Balachander Krishnamurthy, and Walter Willinger. IXPs: Mapped? In *IMC*, 2009.

[9] Rob Austein, Steven Bellovin, Randy Bush, Russ Housley, Matt Lepinski, Stephen Kent, Warren Kumari, Doug Montgomery, Kotikalapudi Sriram, and Samuel Weiler. BGPSEC protocol. `http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol`.

[10] Fred Baker and Pekka Savola. Ingress filtering for multihomed networks. RFC 3704, 2004.

[11] Adam Bender, Rob Sherwood, and Neil Spring. Fixing Ally's growing pains with velocity modeling. In *IMC*, 2008.

[12] Robert Beverly, Arthur Berger, Young Hyun, and k claffy. Understanding the efficacy of deployed Internet source address validation filtering. In *IMC*, 2009.

[13] The BGP Instability Report. http://bgpupdates.potaroo.net/instability/bgpupd.html.

[14] BGPMux Transit Portal. http://tp.gtnoise.net/.

[15] Claudson Bornstein, Tim Canfield, and Gary Miller. Akarouting: A better way to go. In *MIT OpenCourseWare 18.996*, 2002.

[16] Andre Broido and kc claffy. Analysis of RouteViews BGP data: policy atoms. In *Proceedings of the ACM SIGMOD/PODS Workshop on Network-Related Data Management*, Santa Barbara, CA, May 2001.

[17] Martin A. Brown, Clint Hepner, and Alin C. Popescu. Internet captivity and the de-peering menace. In *NANOG*, 2009.

[18] Jake Brutlag. Speed matters for Google web search. Technical report, Google Inc., 2009.

[19] Tian Bu, Nick Duffield, Francesco Lo Presti, and Don Towsley. Network tomography on general topologies. In *SIGMETRICS*, 2002.

[20] Hal Burch. *Measuring an IP network in situ*. PhD thesis, Carnegie Mellon University, 2005.

[21] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Internet optometry: assessing the broken glasses in Internet reachability. In *IMC*, 2009.

[22] Matthew Caesar, Lakshminarayanan Subramanian, and Randy H. Katz. Towards localizing root causes of BGP dynamics. Technical report, University of California, Berkeley, 2003.

[23] Hyunseok Chang, Sugih Jamin, and Walter Willinger. Inferring AS-level Internet topology from router-level path traces. In *SPIE ITCom*, 2001.

[24] Kai Chen, David R. Choffnes, Rahul Potharaju, Yan Chen, Fabián E. Bustamante, Dan Pei, and Yao Zhao. Where the sidewalk ends: Extending the Internet AS graph using traceroutes from P2P users. In *CoNEXT*, 2009.

[25] Yan Chen, David Bindel, Hanhee Song, and Randy H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *SIGCOMM*, 2004.

[26] David Choffnes and Fabian E. Bustamante. Taming the torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *SIGCOMM*, 2008.

[27] David R. Choffnes, Fabián E. Bustamante, and Zihui Ge. Crowdsourcing service-level network event monitoring. In *SIGCOMM*, 2010.

[28] Mark Coates, Alfred Hero, Robert Nowak, and Bin Yu. Internet tomography. *IEEE Signal Processing Magazine*, 2002.

[29] Lorenzo Colitti. *Internet Topology Discovery Using Active Probing*. PhD thesis, Roma Tre University, 2006.

[30] Russ Cox, Frank Dabek, Frans Kaashoek, Jinyang Li, and Robert Morris. Practical, distributed network coordinates. In *HotNets*, 2003.

[31] Italo Cunha, Renata Teixeira, and Christophe Diot. Predicting and tracking Internet path changes. In *SIGCOMM*, 2011.

[32] Mike Dahlin, Bharat Chandra, Lei Gao, and Amol Nayate. End-to-end WAN service availability. *IEEE/ACM TON*, 2003.

[33] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *CoNEXT*, 2007.

[34] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *SIGMETRICS*, 2005.

[35] Brian Eriksson, Paul Barford, and Robert Nowak. Network discovery from passive measurements. In *SIGCOMM*, 2008.

[36] Facebook statistics. `https://www.facebook.com/press/info.php?statistics`, viewed March 3, 2011.

[37] Nick Feamster, David G. Andersen, Hari Balakrishnan, and M. Frans Kaashoek. Measuring the effects of internet path faults on reactive routing. In *SIGMETRICS*, 2003.

[38] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet routing instabilities. In *SIGCOMM*, 2004.

[39] Paul Ferguson and Daniel Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2827, 2000.

[40] Rodrigo Fonseca, George Manning Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. IP options are not an option. Technical report, EECS Department, University of California, Berkeley, 2005.

[41] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM TON*, 2001.

[42] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM TON*, 2001.

[43] Fotis Georgatos, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Henk Uijterwaal, and Rene Wilhem. Providing active measurements as a regular service for ISPs. In *PAM*, 2001.

[44] Ramesh Govindan and Vern Paxson. Estimating router ICMP generation delays. In *PAM*, 2002.

[45] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet map discovery. In *INFOCOM*, 2000.

[46] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based geolocation of Internet hosts. *IEEE/ACM TON*, 2006.

[47] Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *OSDI*, 2004.

[48] Mehmet H. Gunes and Kamil Sarac. Resolving IP aliases in building traceroute-based Internet maps. *IEEE/ACM TON*, 2008.

[49] Omer Gurewitz, Israel Cidon, and Moshe Sidi. One-way delay estimation using network-wide measurements. *IEEE/ACM TON*, 2006.

[50] Md. Ahsan Habib and Marc Abrams. Analysis of sources of latency in downloading web pages. In *WebNet*, 2000.

[51] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the Internet. In *Autonomic Networks Symposium in Globecom*, 2005.

[52] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. A systematic framework for unearthing the missing links. In *NSDI*, 2007.

[53] Urs Hengartner, Sue Moon, Richard Mortier, and Christophe Diot. Detection and analysis of routing loops in packet traces. In *IMW*, 2002.

[54] Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim (Jun) Xu. Diagnosing network disruptions with network-wide analysis. In *SIGMETRICS*, 2007.

[55] Young Hyun. Archipelago update and analyses. In *ISMA AIMS*, 2009.

[56] Internet address hitlist dataset, PREDICT ID USC LANDER internet_address_hitlist_it28w-beta20090914. `http://www.isi.edu/ant/lander`.

[57] Internet World Stats. `http://www.internetworldstats.com/stats.htm`, viewed January 3, 2012.

[58] iPlane. `http://iplane.cs.washington.edu`.

[59] Van Jacobson. Traceroute. `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[60] John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, and Arun Venkataramani. Consensus routing : The Internet as a distributed system. In *NSDI*, 2008.

[61] Ethan Katz-Bassett, John P. John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. Towards IP geolocation using delay and topology measurements. In *IMC*, 2006.

[62] Ken Keys, Young Hyun, Matthew Luckie, and kc claffy. Internet-scale IPv4 alias resolution with MIDAR: System architecture. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2011. Under preparation.

[63] Ramana Kompella, Jennifer Yates, Albert Greenberg, and Alex Snoeren. Detection and localization of network black holes. In *INFOCOM*, 2007.

[64] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C. Snoeren. IP fault localization via risk modeling. In *NSDI*, 2005.

[65] Balachander Krishnamurthy, Harsha V. Madhyastha, and Oliver Spatscheck. ATMEN: A triggered network measurement infrastructure. In *WWW*, 2005.

[66] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize CDN performance. In *IMC*, 2009.

[67] D. Richard Kuhn. Sources of failure in the public switched telephone network. *IEEE Computer*, 1997.

[68] Nate Kushman, Srikanth Kandula, and Dina Katabi. R-BGP: Staying connected in a connected world. In *NSDI*, 2007.

[69] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet routing convergence. In *SIGCOMM*, 2000.

[70] Craig Labovitz, Abha Ahuja, and Farnam Jahanian. Experimental study of Internet stability and wide-area network failures. In *FTCS*, 1999.

[71] Karthik Kalambur Lakshminarayanan, Matthew Chapman Caesar, Murali Rangan, Thomas Anderson, Scott Shenker, and Ion Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.

[72] Anthony Lambert, Mickael Meulle, and Jean-Luc Lutton. Revisiting interdomain root cause analysis from multiple vantage points. In *NANOG*, 2007.

[73] Greg Linden. Make data useful. `http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt`, 2006.

[74] Matthew Luckie, Young Hyun, and Bradley Huffaker. Traceroute probe method and forward IP path inference. In *IMC*, 2008.

[75] Matthew Luckie and Tony McGregor. IPMP: IP measurement protocol. In *PAM*, 2002.

[76] Harsha V. Madhyastha. *An Information Plane for Internet Applications*. PhD thesis, University of Washington, 2008.

[77] Harsha V. Madhyastha, Thomas Anderson, Arvind Krishnamurthy, Neil Spring, and Arun Venkataramani. A structural approach to latency prediction. In *IMC*, 2006.

[78] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *OSDI*, 2006.

[79] Harsha V. Madhyastha, Ethan Katz-Bassett, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane Nano: Path prediction for peer-to-peer applications. In *NSDI*, 2009.

[80] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson. User-level Internet path diagnosis. In *SOSP*, 2003.

[81] Ratul Mahajan, Neil Spring, David Wetherall, and Tom Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.

[82] Ratul Mahajan, David Wetherall, and Thomas Anderson. Understanding BGP misconfiguration. In *SIGCOMM*, 2002.

[83] Ratul Mahajan, Ming Zhang, Lindsey Poole, and Vivek Pai. Uncovering performance differences among backbone ISPs with Netdiff. In *NSDI*, 2008.

[84] Z. Morley Mao, Randy Bush, Timothy G. Griffin, and Matthew Roughan. BGP beacons. In *IMC*, 2003.

[85] Measurement Lab website. `http://www.measurementlab.net/`.

[86] Pascal Merindol, Virginie Van den Schrieck, Benoit Donnet, Olivier Bonaventure, and Jean-Jacques Pansiot. Quantifying ASes multiconnectivity using multicast information. In *IMC*, 2009.

[87] David Meyer. RouteViews. http://www.routeviews.org.

[88] Pradosh Mohapatra, John Scudder, David Ward, Randy Bush, and Rob Austein. BGP prefix origin validation. http://tools.ietf.org/html/draft-ietf-sidr-pfx-validate.

[89] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. In *POPL*, 2012.

[90] Mike Muuss. Ping. http://ftp.arl.army.mil/pub/ping.shar.

[91] North American Network Operators Group mailing list. http://www.merit.edu/mail.archives/nanog/.

[92] NANOG 45, 2009.

[93] Ricardo Oliveira, Dan Pei, Walter Willinger, Beichuan Zhang, and Lixia Zhang. In search of the elusive ground truth: The Internet's AS-level connectivity structure. In *SIGMETRICS*, 2008.

[94] Outages mailing list. http://isotf.org/mailman/listinfo/outages.

[95] Jean-Jacques Pansiot and Dominique Grad. On routes and multicast trees in the Internet. *SIGCOMM CCR*, 1998.

[96] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Z. Morley Mao, and Y. Charlie Hu. A measurement study of Internet delay asymmetry. In *PAM*, 2008.

[97] Vern Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM TON*, 1997.

[98] Vern Paxson, Jamshid Mahdavi, Andrew Adams, and Matt Mathis. An architecture for large-scale Internet measurement. *IEEE Communications*, 36, 1998.

[99] Packet clearing house. http://www.pch.net/home/index.php.

[100] Personal communication with Brice Augustin of Laboratoire d'informatique de Paris 6.

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

[102] Francesco Lo Presti, N. G. Duffield, Joe Horowitz, and Don Towsley. Multicast-based inference of network-internal delay distributions. *IEEE/ACM ToN*, 2002.

[103] Bruno Quoitin and Olivier Bonaventure. A survey of the utilization of the BGP community attribute. Internet draft, draft-quoitin-bgp-comm-survey-00, 2002.

[104] RIPE 58, 2009.

[105] RIPE Routing Information Service (RIS). `http://www.ripe.net/ris/`.

[106] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of Internet path selection. In *SIGCOMM*, 1999.

[107] Pekka Savola. Experiences from using unicast RPF. IETF Internet draft, 2008.

[108] Colin Scott. LIFEGUARD: Locating Internet Failures Effectively and Generating Usable Alternate Routes Dynamically. Technical report, Univ. of Washington, 2012.

[109] Aman Shaikh and Albert Greenberg. OSPF monitoring: Architecture, design and deployment experience. In *NSDI*, 2004.

[110] Stanislav Shalunov, Benjamin Teitelbaum, Anatoly Karp, Jeff W. Boote, and Matthew J. Zekauskas. A one-way active measurement protocol (OWAMP). RFC 4656 (Proposed Standard), 2006.

[111] Yuval Shavitt and Eran Shir. DIMES: Let the Internet measure itself. *SIGCOMM CCR*, 2005.

[112] Yuval Shavitt, Xiaodong Sun, Avishai Wool, and Bulent Yener. Computing the unmeasured: An algebraic approach to Internet mapping. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 22(1), 2004.

[113] Yuval Shavitt and Udi Weinsberg. Quantifying the importance of vantage points distribution in internet topology measurements. In *INFOCOM*, 2009.

[114] Justine Sherry, Ethan Katz-Bassett, Mary Pimenova, Harsha V. Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. Resolving IP aliases with prespecified timestamps. In *IMC*, 2010.

[115] Rob Sherwood, Adam Bender, and Neil Spring. Discarte: A disjunctive Internet cartographer. In *SIGCOMM*, 2008.

[116] Rob Sherwood and Neil Spring. Touring the Internet in a TCP sidecar. In *IMC*, 2006.

[117] The Spoofer project. `http://spoofer.csail.mit.edu/`.

[118] Neil Spring, Ratul Mahajan, and Thomas Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.

[119] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM*, 2002.

[120] Neil Spring, David Wetherall, and Thomas Anderson. Scriptroute: A public Internet measurement facility. In *USITS*, 2003.

[121] Sprint IP network performance. `https://www.sprint.net/performance/`.

[122] Richard A Steenbergen. A practical guide to (correctly) troubleshooting with traceroute. In *NANOG 45*, 2009. `http://www.nanog.org/meetings/nanog45/presentations/Sunday/RAS_traceroute_N45.pdf`.

[123] Stoyan Stefanov. Yslow 2.0. In *CSDN SD2C*, 2008.

[124] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *INFOCOM*, 2002.

[125] Renata Teixeira, Keith Marzullo, Stefan Savage, and Geoffrey M. Voelker. In search of path diversity in ISP networks. In *IMC*, 2003.

[126] Renata Teixeira and Jennifer Rexford. A measurement framework for pinpointing routing changes. In *NeT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, 2004.

[127] Yolanda Tsang, Mehmet Yildiz, Paul Barford, and Robert Nowak. Network radar: Tomography from round trip time measurements. In *IMC*, 2004.

[128] UCLA Internet topology collection. `http://irl.cs.ucla.edu/topology/`.

[129] Feng Wang, Nick Feamster, and Lixin Gao. Quantifying the effects of routing dynamics on end-to-end Internet path failures. Technical report, Univ. of Massachusetts, Amherst, 2005.

[130] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. In *SIGCOMM*, 2006.

[131] Bernard Wong, Ivan Stoyanov, and Emin Gun Sirer. Octant: A comprehensive framework for the geolocalization of Internet hosts. In *NSDI*, 2007.

[132] Jian Wu, Zhuoqing Morley Mao, Jennifer Rexford, and Jia Wang. Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In *NSDI*, 2005.

[133] Wen Xu and Jennifer Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.

[134] Bin Yao, Ramesh Viswanathan, Fangzhe Chang, and Daniel Waddington. Topology inference in the presence of anonymous routers. In *INFOCOM*, 2003.

[135] Jennifer Yates and Zihui Ge. Network Management: Fault Management, Performance Management and Planned Maintenance. Technical report, AT&T Labs, 2009.

[136] Ming Zhang, Chi Zhang, Vivek Pai, Larry Peterson, and Randy Wang. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, 2004.

[137] Yin Zhang, Vern Paxson, and Scott Shenker. The stationarity of Internet path properties: Routing, loss, and throughput. *ACIRI Technical Report*, 2000.

[138] Ying Zhang, Z. Morley Mao, and Ming Zhang. Effective diagnosis of routing disruptions from end systems. In *NSDI*, 2008.

[139] Zheng Zhang, Ming Zhang, Albert Greenberg, Y. Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, 2010.

[140] Zheng Zhang, Ying Zhang, Y. Charlie Hu, Z. Morley Mao, and Randy Bush. iSpy: detecting IP prefix hijacking on my own. In *SIGCOMM*, 2008.

[141] Xiaolian Zhao, Dan Pei, Lan Wang, Dan Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. An analysis of BGP multiple origin AS (MOAS) conflicts. In *IMW*, 2001.

VITA

Ethan Katz-Bassett was born in Northampton, Massachusetts. He received his Bachelor of Arts degree in Computer Science and Mathematics from Williams College in 2001 and his Master of Science degree in Computer Science and Engineering from the University of Washington in 2006. In 2012, he graduated with a Doctor of Philosophy in Computer Science and Engineering from the University of Washington. His research interests broadly include building networked systems and understanding the behavior and properties of such systems, especially the Internet.