Network Troubleshooting from End-Hosts

Renata Cruz Teixeira

Habilitation à Diriger des Recherches
Université Pierre et Marie Curie – Paris 6

Network Troubleshooting from End-Hosts

presented by

Renata Cruz Teixeira

Submitted in total fullfilment of the requirements
of the degree of Habilitation à Diriger des Recherches

Commitee:

Prof. Ernst Biersack	**Eurecom**
Prof. Serge Fdida	**UPMC Paris Universitas**
Prof. Paul Francis	**Max Planck Institute for Software Systems**
Prof. Jim Kurose	**University of Massachusetts Amherst**
Prof. Vern Paxson	**University of California Berkeley**
Prof. Jean-Pierre Verjus	**INRIA**

Abstract

Troubleshooting faults or performance disruptions in today's Internet is at best frustrating. When users experience performance or connectivity problems, there is little they can do. The most common attempt to solve the problem is to reboot or call the provider's hot line. Network providers have more information to use in diagnosing problems in their networks, but their network troubleshooting is often mostly manual and ad-hoc. We argue that automatically troubleshooting network faults or performance disruptions requires monitoring capabilities deployed at end-hosts (in or close to the customer's premises). End-host monitoring is necessary to detect the problems that affect end-users. In addition, when problems happen outside the control of the network administrator or the end-user performing the troubleshooting, end-host monitoring is the only approach to identify problem location. The goal of our research is to make network troubleshooting more transparent by designing tools that require as little as possible human involvement (both end-users and administrators).

This document presents our initial steps towards automatic network troubleshooting from end-hosts as well as our long-term objectives. Our main contributions are the design of more accurate and efficient end-host measurement methods. We work both on techniques to detect faults and performance disruptions, and to identify the location of the problem. For fault identification, we improve the two basic techniques using end-to-end measurements: traceroute and network tomography. First, we show that traceroute, the most widely used diagnosis tool, reports erroneous paths in the presence of routers that perform load balancing. We build Paris traceroute to correct these errors. Second, we design measurement methods for accurate fault identification using network tomography. Network tomography assumes up-to-date and correlated measurements of the status of end-to-end paths and of the network topology, which are hard to get in practice. We design techniques to track correlated path reachability and network topology. Finally, we design techniques for lightweight detection of faults and performance disruptions. We minimize the overhead of active end-to-end probing for fault detection and design a tool to collect passive measurements at end-hosts.

Key Words:

Network troubleshooting, network performance diagnosis, network tomography, traceroute, network topology measurements, end-host data collection

Table of contents

Chapter 1

Introduction

The Internet is a part of many of our daily activities. It is easy to understand people's frustration when they cannot connect to their favorite web site, when their download is too slow, when their voice-over-IP call is choppy, or when the movie they have paid for keeps restarting. This frustration only gets worse if people try to understand the source of the problem in the hopes of solving the problem. The average Internet user may restart the application or reboot the machine and modem. As a last resort, users may call the help line of their service providers, who are often not in a much better position to diagnose the problem.

A simple example illustrates the complexity of identifying the cause of Internet faults. When the user clicks on a web page, several elements need to cooperate for the web content to be successfully delivered. At the end hosts, the operating system and all layers of the protocol stack need to work properly. The users' local environment can also be fairly complex (users are often behind NATs or firewalls and connected over a wireless LANs). At the core of the network, all routers and links in the path from the user to the server and vice-versa need to forward packets to the correct destinations. The user also needs to contact a Domain Name System (DNS) server to translate the name of the web site into an IP address and may also need to contact multiple hosts, because different parts of the Web page (e.g., text, images) may come from different servers in different locations and from HTTP caches. Faults, misconfigurations, or poor performance of any of these elements can cause the user's request to fail, or suffer high delay or losses.

Unfortunately, when a failure happens none of the parties involved is in a position to easily diagnose the problem. The user can quickly detect that something is wrong, but she has no control of most of the elements involved in accessing the web page. The Internet Service Providers (ISPs) that manage the networks in the path have direct access only to

the routers and links inside their networks. ISPs often monitor their network equipment and most equipment raise alarms only under a limited set of faults and performance disruptions. However, these datasets provide no detailed view into the performance of individual end-to-end paths, so it is hard for ISPs to know when their customers experience problems. Moreover, in some cases a customer may be experiencing a failure, but none of the provider's datasets explicitly reports a fault: either because the problem is elsewhere (at another network or end-system) or because the problem doesn't appear in any of the ISPs' alarm systems [1]. Similarly, the content provider may have no signs of faults or disruptions.

To circumvent the fact that no single entity has direct access to all elements of a communication path, network operators and users often resort to end-to-end measurement techniques. Network operators routinely monitor the performance of paths traversing their network from dedicated monitoring hosts either inside their network (located at the provider's points of presence) or by subscribing to monitoring services such as Keynote [2] and RIPE TTM [3]. Path performance monitoring helps verify that the network performance complies with the service-level agreements (SLAs) contracted with customers as well as to detect faults or anomalous behavior before customers complain. Content and application providers also monitor the quality of end-to-end paths to deliver the best performance to users. The most knowledgeable end-users deploy end-to-end monitoring tools to identify the cause of faults or performance disruptions [4–6], verify that the network performance they get is what they are paying for [4–9], and to test whether their provider is blocking or rate-limiting any of their applications [10–12]. Despite the recent advances in end-host monitoring and troubleshooting, network performance and fault diagnosis are mostly manual and completely ad hoc. Often troubleshooting only starts after the end-user or the network administrator notice a problem.

The long-term goal of our research is to remove the human (both the end-user and the ISP) from the loop and automatically detect and troubleshoot network faults and performance disruptions. We believe that monitoring from end-hosts (in or close to the customer's premises) is essential for troubleshooting. As a starting point, we focus on troubleshooting network-related problems (or end-to-end performance problems originating at routers and links). During the past five years, we have worked on two initial steps in network troubleshooting from end-hosts: *detection* that a problem has occurred and *identification* of the cause of the fault or disruption. After a brief background on detection and identification techniques, we summarize our contributions.

Detection. Fast detection of faults or performance disruptions requires continuous monitoring of network paths through either active or passive measurement techniques. The goal is to detect problems automatically, hopefully before disruptions are noticed by end-users. *Active probing* relies on sending a probe message and waiting for the response (the most

popular active probing tool is ping). Active probing can be used from any end-host, so network operators can deploy active probing from dedicated servers in their networks and end-users can use active probing from their machines. *Passive monitoring* observes users' incoming and outgoing traffic and monitors the status of active TCP connections or UDP flows [13, 14]. Packet capture at end-hosts is often done with a tool such as tcpdump or pcap [15]. In backbone networks link rates are too high, so even with dedicated hardware (such as a DAG card [16]) it is not feasible to use packet capture to detect faults or performance disruptions in individual flows. Passive monitoring reduces probing overhead and captures the performance of active connections, but it requires tapping users' traffic, which may not always be possible. The main challenge of fault and performance disruption detection using both active and passive measurements is the measurement overhead. For active monitoring, fast detection requires sending probes at a high frequency to a large number of destinations. Passive monitoring can potentially cause CPU and storage overhead on the end-user's machine (to tap the user's traffic).

Identification. There are two basic techniques for fault identification using end-to-end measurements: traceroute-like probing and network tomography.

- Traceroute [17] sends probes to a destination with an increasing Time-To-Live (TTL) to force routers along the path to send an error message, which reveals the IP address of the router's interface that issued the error message. Traceroute can identify some forwarding anomalies such as loops or unreachable networks [14, 18, 19]. However, traceroute's output may be inaccurate or incomplete. First, network operators often use the load-balancing capabilities of routers, but traceroute only probes a single path. This mismatch leads to the identification of false links and incomplete paths. Second, routers may not respond to traceroute probes (i.e., probes can be lost or rate-limited). In these cases, traceroute outputs a star ('*') to indicate that the hop is unknown. Third, tunneling (e.g., MPLS) may hide parts of the path.

- Network tomography refers to the practice of inferring unknown network properties from measurable ones [20]. It correlates end-to-end measurements of a set of paths to infer the links responsible for a fault [1, 21], high delay [22, 23], or high loss [23–27]. Network tomography assumes up-to-date and correlated measurements of path reachability or performance and of the network topology. However, measurement errors together with the lack of synchronized measurements can lead to inconsistent end-to-end measurements, which in turn result in inference errors. Similarly, a monitor can take a long time to probe a full topology [28]. This long probing delay can result in an inferred topology that is out of date.

These two techniques are complementary. Traceroute works with access to the source end-host alone, but it can only identify the effect of fault not its cause (a router may no longer have a route to a destination, because of the failure at some other router or network). Network tomography works in an environment where multiple end-hosts collaborate, so it correlates information from different vantage points to infer the link or router that most likely caused the failure.

Contributions In the past five years, our research has focused on the design of measurement methods for network troubleshooting from end-hosts, improving troubleshooting accuracy and efficiency. We have made the following contributions:

- **Improvements to traceroute under load balancing (Chapter 2).** We built a new traceroute, called *Paris traceroute*, that eliminates most of the false links that arise because of load balancing. Paris traceroute also infers all paths between a source and destination in the presence of load balancing. Paris traceroute is distributed with Debian Linux and it has become the tool-of-choice for topology tracing (for example, Skitter/Ark and Dimes now use Paris traceroute).

- **Methods for accurate fault identification using network tomography (Chapter 3).** We designed a probing strategy that distinguishes persistent path failures from transient packet losses, and allows consistent merging of the status of paths (even though it is not possible to synchronize measurements). To keep the network topology up to date, we designed *DTrack*, a probing scheme that quickly detects topology changes and remaps the parts of the topology that have changed. Then, we designed *NetDiagnoser*, an algorithm based on binary tomography that can handle partial failures and incomplete topologies. NetDiagnoser also introduces mechanisms to combine routing messages collected at an ISP network with end-to-end probing data to improve the diagnosis accuracy.

- **Methods for fast and lightweight fault detection using end-to-end measurements (Chapter 4).** There is a large probing cost to monitor paths among a large set of monitors and destinations at a very high frequency. We proposed algorithms to minimize the probing cost needed to detect faults. We also discuss preliminary work on passive detection of end-to-end performance disruption at end-hosts.

This document first presents our contributions in these three areas. Then, Chapter 5 concludes with a discussion of our plans for applying these techniques in two scenarios: directly at end-users' laptop or desktops as well as at home gateways (i.e., DSL or cable modems).

Chapter 2

Paris Traceroute

Traceroute [17] is one of the most widely deployed network measurement and diagnosis tools. It reports an IP address for each network-layer device along the path from a source to a destination host in an IP network. Network operators and researchers rely on traceroute to diagnose network problems and to infer properties (e.g., topology) of IP networks. We now show how traceroute introduces measurement artifacts in the presence of routers that perform load balancing on packet header fields. These artifacts lead to incorrect route inferences that may result in erroneous Internet topologies being reported and incorrectly identified forwarding anomalies.

Network administrators employ load balancing to enhance reliability and increase resource utilization. Load balancing routers (or *load balancers*) use three different algorithms to split packets on outgoing links [29, 30]: *per destination*, which forwards all packets destined to a host to the same output interface (similar to the single-path destination-based forwarding of classic routing algorithms); *per flow*, which uses the same output interface to all packets that have the same *flow identifier* (described as a 5-tuple: IP source address, IP destination address, protocol, source port, and destination port); or *per packet*, which makes the forwarding decision independently for each packet. Per-packet load balancing equally splits load, but has potentially detrimental effects on TCP connections, because packets can be reordered.

Under load balancing, the original traceroute fails to accurately trace all possible routes. Our explanation of the problems draws on the example in Fig. 2.1. L is a load balancer at hop 6 from the traceroute source, S. On the left, we see the true router topology at hops 6 through 9. Circles represent routers, and each router interface is numbered. Black squares depict probe packets sent with TTLs 6 through 9. They are shown either above the topology, if L directs them to router A, or below, if L directs them to router B. On

the right, we see the topology that would be inferred from the routers' responses.

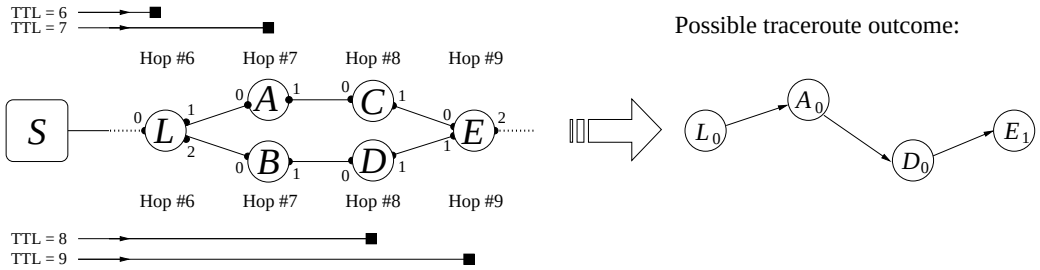

Figure 2.1: Missing nodes and links, and false links.

Missing nodes and links. Because routers B and C send no responses, nodes B_0 and C_0 are not discovered, and links such as (L_0, B_0) and (B_0, D_0) are not identified.

False links. L directs the probe with initial TTL 7 to A and the one with initial TTL 8 to B, leading to the mistaken identification of a link between A_0 and D_0.

These errors occur even under per-flow load balancing, because traceroute systematically varies the flow identifier of its probes. Traceroute needs a probe identifier, or a way to match routers' responses with the probes that elicited them. A router that sends an ICMP Time Exceeded response encapsulates only the IP header of the packet that it discarded, plus the first eight octets of data [31, p.5], which, in the case of UDP or ICMP Echo probes, is equivalent to the transport-layer header. Hence, for UDP, traceroute identifies probes by varying the destination port; and for ICMP probes, the Sequence Number field (which in turn varies the Checksum field, which is in the first four octets). Our experiments show that varying any field in the first four octets of the transport-layer header amounts to changing the flow identifier for each probe [32]. Thus, traceroute reports false links both under per-flow and per-packet load balancing.

We provide a publicly-available traceroute tool, called *Paris traceroute* [1], which resolves both problems identified above. First, we discuss how Paris traceroute controls packet header contents to correctly probe a single route. Measurements comparing Paris traceroute with classic traceroute show that Internet topologies measured with classic traceroute may contain "loops", "cycles", and "diamonds" that are pure artifacts of traceroute's behavior under load balancing. Then, we extend Paris traceroute with the Multipath Detection Algorithm (MDA), a stochastic probing algorithm that adapts the number of probes to send on a hop-by-hop basis in order to enumerate all reachable interfaces at each hop. We use the MDA to measure load balancing in hundreds of thousands of Internet paths. Paris traceroute is the thesis of Brice Augustin, who I co-advise with Timur Friedman.

[1] Paris traceroute is free, open-source software, available from `http://www.paris-traceroute.net/`.

2.1 Removing False Links

"Avoiding traceroute anomalies with Paris traceroute", IMC 2006 [33]
"Detection, understanding, and prevention of traceroute measurement artifacts", Computer Networks 2008 [32]

The key innovation of Paris traceroute is to control the probe packet header fields in a manner that allows all probes towards a destination to follow the same path in the presence of per-flow load balancing. It also allows a user to distinguish between the presence of per-flow load balancing and per-packet load balancing. Unfortunately, due to the random nature of per-packet load balancing, Paris traceroute cannot perfectly enumerate all paths in all situations. But it can flag those instances where there are doubts.

Maintaining certain header fields constant is challenging because Paris traceroute still needs to be able to match response packets to their corresponding probe packets. Paris traceroute does this by varying header fields that are within the first eight octets of the transport-layer header, but that are not used for load balancing. For UDP probes, Paris traceroute varies the Checksum field. This requires manipulating the payload to yield the desired value, as packets with an incorrect checksum are discarded. For ICMP Echo probes, Paris traceroute varies the Sequence Number field, as does classic traceroute, but also varies the Identifier field, so as to keep constant the value for the Checksum field.

Paris traceroute also sends TCP probes, unlike classic traceroute, but like *tcptraceroute* [34]. For TCP probes, Paris traceroute varies the Sequence Number field. No other manipulations are necessary in order to maintain the first four octets of the header field constant. Paris traceroute's TCP probing is not innovative in the same way as its UDP and ICMP Echo probing, as tcptraceroute already maintains a constant flow identifier. However, no prior work has examined the effect, with respect to load balancing, of maintaining a constant flow identifier for probe packets.

To study traceroute artifacts, we conduct side-by-side measurements with classic traceroute and Paris traceroute from one source to 5,000 destinations. (We refer the reader to our papers [32, 33] for a detailed description of these measurements.) We study three topology artifacts, which we call loops, cycles, and diamonds. A *loop* happens when the same node appears twice or more in a row in a measured route. A *cycle* happens when an IP address, r, appears at least twice in a measured route, but separated by at least one other address distinct from r. A *diamond* happens when probes to the same hop and destination observe different IP addresses. We perform back-to-back measurements with classic traceroute and Paris traceroute. Our results [32] show that false links due to per-flow load balancing are the cause of more than 70% of loops, almost 40% of cycles, and more than 50% of diamonds that appear in the topology measured with classic traceroute. As observations from other vantage points towards other destination sets would reveal different numbers, the particular values from this study cannot be considered statistically

representative. These experiments form a case study showing how Paris traceroute finds more accurate routes and yields knowledge concerning the causes of each artifact.

2.2 Finding All Paths

" Multipath tracing with Paris traceroute", End-to-end Monitoring Workshop 2007 [35]
"Measuring load-balanced paths in the Internet", IMC 2007 [36]
"Failure control in multipath route tracing", INFOCOM 2009 [37]

We specify an adaptive, stochastic probing algorithm, the Multipath Detection Algorithm (MDA), to report all paths that traffic could follow between a source and a destination. The MDA proceeds hop-by-hop, and explores the IP-level graph by enumerating the next-hops of each interface discovered, until it reaches the destination along all paths. For a given interface r at hop $h-1$, MDA generates a number of random flow identifiers and selects those that will cause probe packets to reach r. It then sends probes with these identifiers, but one hop further, in an effort to discover the successors of r at hop h. We call this set of interfaces, $s_1, s_2, \ldots, s_n$ the *nexthops* of r. The MDA selects the number of probes to send at each hop adaptively according to the set of discovered nexthops and as a function of the upper bound on the probability of failing to discover the entire multipath route [37]. The MDA also has mechanisms to deal with unresponsive routers and for identifying per-packet load balancers and routing changes.

We perform experiments from a single source to 5,000 destinations to explore the tradeoff between overhead and actual success rates of MDA in a real world environment. Our results [37] show that even with a failure probability bound of 50% the MDA can find all the discoverable routes for more than 90% of multipath routes in our traces. On the other hand, we find that classic traceroute (sending the default three probes per hop) misses at least one link for 84% of multipath routes. This more complete route knowledge comes at the cost of higher probing overhead. In extreme cases, the MDA may send more than a thousand probes to find all links of the multipath route when failure is bounded at 1%. We are currently exploring techniques to reduce MDA's overhead. For instance, knowing the hash functions used by per-flow load balancers should make it possible to reduce probing overhead, with the possibility to revert to the MDA in cases where no predictable pattern can be identified with high certainty.

We use the MDA to quantify multipath routes observed from 15 RON nodes [38] to 68,000 destinations. In our dataset [36], the paths between 39% of source-destination pairs traverse a per-flow load balancer and 70% traverse a per-destination load balancer, but only approximately 2% traverse a per-packet load balancing. Some paths traverse more than one load balancer.

2.3 Summary

This chapter identified a problem with one of the most used internet measurement tools (that arises because of changes in modern routers) and developed new techniques to avoid this problem. We identified the roles that packet header fields play in load balancing, and their interactions with classic traceroute. We showed that load balancing is the cause of a number of measurement artifacts such as "loops", "cycles", and "diamonds". Then, we designed Paris traceroute, a new traceroute tool that allows more precise measurements of a route, and determined the causes of many of the artifacts of the classic traceroute tool. Paris traceroute is distributed with Debian linux and most topology measurement systems now use Paris traceroute (for example, Dimes and Skitter/Ark). Paris traceroute also includes a Multipath Detection Algorithm (MDA) to discover multipath routes. MDA represents a major advance for traceroute users. Instead of blindly using the default policy of sending three probes per hop, users will now be able to configure their probing algorithm with the ability to trade off the completeness of multipath routes against low probing overhead. Finally, we use the MDA to characterize multipath routes in the Internet. The high fraction of paths that traverse a load balancer has important implications on end-to-end performance and topology measurement techniques. Designers of such tools should use Paris traceroute's technique, which controls the flow identifier of a series of probes, to avoid measurement artifacts.

There are a number of possible extensions to Paris traceroute. The knowledge of the algorithm used by per-flow load balancers to hash flow-ids into paths should make it possible to reduce the probing overhead of MDA. There is considerable scope for improving the current (loose) failure bound, thereby saving probes. Other open questions are how to accurately trace multipath routes behind per-packet load balancers and to infer the hops within tunnels (for instance, due to MPLS or GRE).

Chapter 3

Network Tomography for Fault Diagnosis

Network tomography algorithms have been proposed for numerous applications. For example, to infer link delays [22, 23] and loss rates [23–27], or to determine the location of faulty equipment [1, 21]. Most relevant to Internet troubleshooting is the work on binary tomography [21]. Instead of estimating the properties of individual links (which requires a level of correlation among the measurements of path status that can rarely be achieved in practice), the goal of binary tomography is to separate links into "good" or "bad". For example, estimating the loss rates of each link in a network is more challenging than just pinpointing the set of "lossy" links (given a threshold that separates high and low loss rates). The assumption is that if a link is bad, then all paths that cross the link experience bad performance. The *path status* refers to the quality of the end-to-end connectivity between the source and the destination of a path. In this chapter, we apply binary tomography to fault diagnosis. We define a path to be good if it is reachable; otherwise, the path is bad.

Fig. 3.1 shows an example of applying binary tomography to fault diagnosis. *Monitors* periodically probe destinations. A *coordinator* combines the results of probes from all monitors and runs a tomography algorithm. When a link fails, a unique set of end-to-end paths from monitors to destinations experiences the failure. The goal of binary tomography algorithms is to use the set of paths that experience end-to-end losses to identify the failed link.

The original binary tomography formulation by Duffield [21] considers a topology consisting of paths from a single monitor to multiple destinations, and assumes that this topology is accurately known. Duffield [21] presents the "Smallest Common Failure Set"

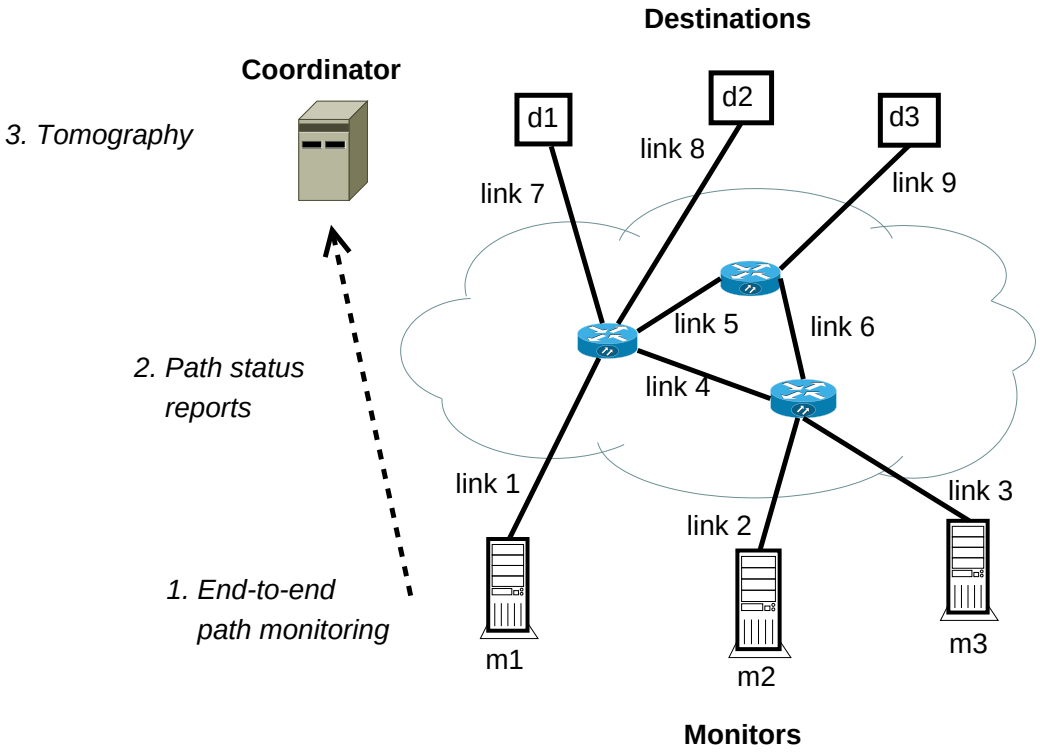

Figure 3.1: Example of binary tomography.

(SCFS) algorithm, which designates as bad only those links nearest the source that are consistent with the observed set of bad paths. For example, consider a single-source from monitor m_2 to destinations d_1 and d_2 in Fig. 3.1. If link 7 fails, then path m_2 to d_1 will fail, while path m_2 to d_2 will remain working. In this case, SCFS will correctly infer that link 7 has failed.

Binary tomography has been explored extensively in theory, simulations, and offline analysis [1, 21, 39]. Unfortunately, binary tomography algorithms are difficult to apply directly in practice [40]. Assuming that every lost probe indicates a failure leads to many *false alarms*—cases where the tomography algorithm claims that a link has failed when it has not. For instance, the naive application of tomography in our measurements from PlanetLab would trigger almost one alarm per minute; in our measurements from an enterprise network, it would raise one alarm every three minutes. Such alarm rates are much too high to be useful in practice.

False alarms arise because the inputs to the tomography algorithm are inaccurate. Binary tomography requires accurate and up-to-date measurements of the network topology and the status of end-to-end paths (formalized as a *reachability matrix*, which indicates

whether each path is up or down). In the example in Fig. 3.1, suppose that all links are working, but that because of a measurement error (for instance, a probe lost due to rate limiting) the reachability matrix views the path between m_2 and d_1 as down and m_2 and d_2 as up; binary tomography would raise a false alarm with link 7 as down. Similarly, if at some time the path from m_2 to d_1 is re-routed to follow the path traversing links 2, 6, 5 and 8, but the coordinator still has the old topology, the failure of links 5 or 6 would lead to a false alarm with link 2 or 4 as down. The thesis of Italo Cunha (co-advised with Christophe Diot) designs methods to build consistent reachability matrices (Section 3.1) and to keep real-time network topologies (Section 3.2).

Even with more accurate inputs, the original version of the binary tomography algorithm faces difficulties when diagnosing reachability problems in an environment with multiple autonomous networks (or Autonomous Systems–ASes) such as the Internet. These difficulties arise because links can fail "partially" (in the case of router misconfigurations) and topologies collected with traceroute are often incomplete (because some routers do not respond to probes). In addition, when an ISP deploys these techniques to identify faults, it can make use of additional information such as routing messages to improve diagnosis. Hence, we propose NetDiagnoser (Section 3.3), a binary tomography algorithm that includes features designed to overcome these limitations of binary tomography. NetDiagnoser was the topic of the internship of Amogh Dhamdhere at Technicolor (co-advised with Constantine Dovrolis and Christophe Diot).

3.1 Tracking path reachability

"Measurement Methods for Fast and Accurate Blackhole Identification with Binary Tomography", IMC 2009 [41]

The path reachability matrix can be inconsistent for two reasons: (1) *detection errors:* when there is no failure, but a path was mistakenly detected as down; and (2) *lack of synchronization:* for instance, in Fig. 3.1, supposed that link 2 has failed, but when m_2 probed d_1 the link was still up, and when it probed d_2 the link was down. Type 1 errors occur because packet losses are often bursty, and hence monitors can easily misinterpret a transient but bursty loss incident as a persistent failure. Type 2 errors arise because it is practically impossible to guarantee that probes issued by different monitors to different destinations will cross a link at the same time. Early tomography algorithms [24] assumed multicast probes from a single monitor to achieve synchronization, but multicast is not widely deployed on an end-to-end basis. To reduce these errors, we first design and evaluate a probing strategy for *failure confirmation* that distinguishes persistent path failures from transient packet losses. Then, we develop and validate *aggregation strategies* to address errors due to lack of synchronization. These strategies introduce a delay to verify that measurements are stable before producing inputs for binary tomography.

3.1.1 Distinguishing persistent failures from transient losses

A lost probe might indicate a persistent failure, but it can also indicate a transient loss due to congestion, routing convergence, or overload at hosts. In this section, we develop a probing method to distinguish persistent failures from congestion-based transient losses. We refer to this method as *failure confirmation*. When a monitor observes a single lost probe along a path, it sends additional failure confirmation probes to determine whether lost packets are caused by a failure. A *confirmed failure* happens when all confirmation probes are lost. We aim to confirm failures as quickly as possible while reducing the overall number of *detection errors*—cases where we misclassify a transient loss as a failure. Additional probing can reduce detection errors at the cost of increasing the time to detect a failure (perhaps by as much as tens of seconds, depending on the overall probing rate).

Detection errors are unavoidable in real deployments, and we would need to send an infinite number of confirmation probes to achieve perfect detection. Our objective is to make the detection-error rate, F, as small as possible while still keeping detection time low. We define κ as the number of confirmation probes and T as the time to perform failure confirmation. We model path losses using the *Gilbert model* [42], because Zhang et al. [43] showed that this model accurately captures the burstiness of congestion-based losses observed on Internet paths. The rest of this section examines the probing process, rate, and number of probes needed to provide fast detection and low overall detection-error rate.

Probing process. We show that a periodic probing process minimizes the detection-error rate, F, given κ and T. Minimizing detection errors is equivalent to minimizing the probability that all confirmation probes fall within loss bursts. Our goal is to find the intervals between probes, $\mu_1, \cdots, \mu_{\kappa-1}$, that minimize F by solving an optimization problem constrained by the total time available to run confirmation, i.e., $\sum_\kappa \mu_i < T$. The solution to this optimization occurs when all μ_i are equal. One way to prove this result is by showing that if there exists $\mu_i > \mu_j$, then decreasing μ_i by δ and increasing μ_j by δ decreases the value of F.

Although sending periodic probes is shown to minimize the detection-error rate if losses follow a Gilbert model, this method performs poorly in the unlikely case of periodic losses. To avoid the possibility of *phase locking*, i.e., losing all confirmation probes in periodic loss episodes, we use the method suggested by Baccelli *et al.* [44], where probe inter-arrival times are uniformly distributed between $[(1-\gamma)\mu, (1+\gamma)\mu]$, with $0 \leq \gamma < 1$.

Number of probes and rate. The second part of our analysis assumes confirmation probes have inter-arrival times between $[(1-\gamma)\mu, (1+\gamma)\mu]$ and takes as input a target

detection error rate, F. When the number of probes, κ, is too large, probes will interfere with the network (perhaps inducing additional losses); when κ is too small, detection errors increase. The objective is to find values of κ and T that achieve the target F. We formulate two optimization problems for selecting κ and μ, where μ is the average interval between probes. The first optimization model minimizes T, subject to the target F and a maximum probing rate of $1/\mu_{\min}$ packets per second. The second optimization model minimizes the total number of confirmation probes needed to achieve F.

We evaluate the failure confirmation scheme that minimizes the number of probes in a controlled environment using Emulab and in wide-area experiments using PlanetLab. Controlled experiments allow us to measure the accuracy of our technique, because we have ground truth; whereas wide-area experiments allow us to test our method under actual loss scenarios. Our results show that spacing probes significantly reduces detection errors compared to sending back-to-back probes.

3.1.2 Methods for correlated path reachability

Our goal in this section is to design an *aggregation strategy*, i.e., a method to combine measurements of the status of different paths into a consistent reachability matrix with small aggregation delay. If monitors could probe all paths instantaneously, then the resulting reachability matrix would be consistent. Unfortunately, synchronous measurements are impossible in practice. First, measurements from a single monitor are not instantaneous, because probing many destinations takes time (in our experiments, this process takes from tens of seconds to minutes). Second, each monitor probes a different set of paths, so it is impossible to guarantee that two probes cross a given link simultaneously. We call the process of probing all paths a *cycle*. Monitors have different cycle lengths as each monitor probes a different set of paths, and machines have different processing power and available bandwidth. The overall cycle length is the time it takes the slowest monitor to probe all its paths. We show that the overall aggregation delay is a function of both the cycle length and the aggregation strategy. Another reason for the reachability matrix to be inconsistent are detection errors from failure confirmation (Sec. 3.1.1). These errors may create situations where a path is considered to have failed when it has not.

We propose and evaluate three strategies for aggregating path measurements into a reachability matrix. The *basic strategy* waits for a full cycle for monitors to re-probe all paths after a failure is detected. If the failure lasts longer than a cycle, this simple strategy guarantees that the reachability matrix is consistent. However, it can build inconsistent matrices if failures are short or if there are detection errors. We then consider two enhancements that wait longer (n cycles, where n is a parameter) to build matrices, but achieve higher consistency. The *MC strategy* is conservative; it waits for n cycles with identical

measurements. *MC-path* is more tolerant to detection errors. We present models that capture how detection errors and unsynchronized measurements may introduce inconsistency and derive the expected consistency for each of these schemes. We validate our models using controlled experiments in Emulab. We also apply our strategies to measurements collected on both the PlanetLab testbed and across a geographically distributed enterprise network to check how useful they are in practical scenarios.

Our controlled experiments show that combined confirmation and aggregation achieve high identification rate with low false alarms. These empirical results (as our analytical results) show that both detection errors and short failures reduce the accuracy of aggregation strategies and that these methods quickly and accurately identify all failures that are longer than two measurement cycles, with few false alarms. In PlanetLab and the enterprise network experiments, our techniques decrease the number of alarms by as much as two orders of magnitude. Compared to the state of the art in binary tomography, our techniques increase the correct identification rate while avoiding hundreds of false alarms.

This section presented techniques to track the reachability matrix assuming a known network topology. Next, we discuss how to track the network topology using traceroutes.

3.2 Tracking the evolution of IP topologies

"Tracking the Evolution of IP Topologies: Change Detection and Local Remapping",
Technicolor Tech Report 2010 [45]

Network topologies are critical to a wide variety of tasks in network operations and research. For example, aside from fault and performance diagnosis, network operators need the network topology for traffic engineering and capacity planning; whereas researchers need the network topology to study properties of the Internet and to test new algorithms and protocols under realistic scenarios. When access to routers is not possible, operators and researchers typically resort to traceroute-like probes to infer IP links and then combine these links into a topology [28, 46–51].

Successive topology measurements with traceroutes capture the evolution of the network to some extent. However, these measurements cannot capture topology changes in real-time because they do not take instantaneous pictures of the network. Measuring a topology can sometimes take more than a day [28] depending on the size of the network and the probing capacity (which is constrained by the monitor's link capacity and CPU). A topology inferred with traceroutes is then an inconsistent view of the network, analogous to a blurred picture because the exposure time was too long. Advances in topology discovery techniques [49,51] have reduced the time required to infer a topology by sending fewer probes to cover all interfaces. For instance, Tracetree [51] takes only four minutes to discover a topology with more than 11,000 IP addresses from one dedicated monitor. Nevertheless, four minutes

is still too long to capture all the topology changes and to build a consistent view of the topology.

We argue that an approach that simply probes all interfaces in a network topology is fundamentally limited in its ability to track topology changes in real time. Even if we are able to design the most effective probing strategy, which sends exactly one probe to discover each interface, it is not possible to probe all interfaces of a topology exactly at the same time. For instance, Skitter limits its monitors to 300 probes/s, Dimes to 20 probes/s, and Tracetree (which has no explicit limit) can achieve at most 900 probes/s, so if the topology is larger than 1,000 IP interfaces (which is almost always the case), it is impractical to probe all interfaces within the same second. Even if monitors and links become faster in the future, topologies will also get larger. In addition, our measurements show that Internet routes are mostly stable, hence a systematic probing of all interfaces to build a topology may be overkill. We need to design new probing strategies to quickly capture topology changes.

Instead of periodically re-probing the full topology, we design a new probing strategy that combines two processes: one that detects topology changes, and the other that re-maps only the area that has changed. Our objective is to detect as many changes as possible. We study the scenario where one monitor (i.e., the source of the probes) tracks the topology to a set of destinations. Given an initial topology, we issue the probes in such a way that we quickly detect route changes. Upon detecting a change, we only remap the parts of the topology that have changed by iteratively probing the interfaces close to the detected change. Our key challenge is to detect these routes that are the most likely to change in the near future without systematically re-probing the topology.

Characterization and prediction of topology changes. We analyze topology changes in traceroute measurements collected from 71 PlanetLab nodes during five weeks. We use Paris traceroute so that our measurements are correct in the presence of load balancing. Note that load balancing represents forwarding dynamics, not topology dynamics. The routing topology is still the same, and load balancers only select among the available routes to use for any given packet. We only have measurements of a source-destination pair every 45 minutes, so we cannot observe route changes at shorter time scales. Our characterization of topology changes shows that the topology is mostly stable: the routes between less than 6% of the source-destination pairs change between 95% of consecutive topology maps. We also observe that the connectivity between source-destination pairs goes through periods of instability, during which it changes routes multiple times. If the connectivity between a source-destination pair is experiencing many route changes we call it a *hot path*; otherwise we call it *cold path*. Our results indicate that we can predict changes in hot paths based

on route duration and on whether the route is dominant (i.e., a route that is active for more than 80% of the history of measurements). Predicting changes in cold paths is more challenging, because when paths have been stable for sometime the distribution of the time until the next change is almost uniform (i.e., the next change can happen at any time with similar probability).

dtrack: probing strategies to track topology changes. We are currently designing DTRACK, a probing strategy to maximize the number of topology changes detected given a probing budget. DTRACK runs at one monitor and takes as input the topology between the monitor and a set of destinations and a probing budget, λ_{budget}. The probing budget captures the probing capacity of the monitor. DTRACK alternates between two functions: *route change detection* or the *topology remapping*. The goal of route change detection is to issue probes at rate λ_{budget} such that it maximizes the detection of route changes. Whenever a change is detected, DTRACK temporarily stops the detection process and allocates the full rate λ_{budget} to remap just the part of the topology that has changed.

The detection process decides *when* and *where* to send probes to maximize the number of topology changes detected. We probe hot and cold paths using two distinct detection processes that run in parallel to detect route changes. To avoid detecting topology changes because of load balancing, we keep a load-balancer database (obtained using Paris traceroute's MDA) and we do not consider as a route change if the changed interface is known as a load-balanced hop.

The hot and cold processes make two decisions: how to split probes among the set of destinations and how to split probes for different hops in a route to a destination. Currently, these processes work as follows.

- *Cold detection process:* Given that it is hard to predict change in cold paths, the cold process spreads probes equally across paths. It then splits probes for different hops in a route based on two observations from our characterization. First, more than 40% of the changes affects the route length, so we probe the ASes at the end of routes more frequently than intermediary ASes to detect changes that affect the route length. We distribute probes to ASes in the middle of the route uniformly to directly detect changes that do not affect the route length. Second, many topology changes affect multiple consecutive interfaces inside a single AS. Thus, we send probes to a single hop for each AS in a route at a time.

- *Hot detection process:* Our characterization allow us estimate, $\mathbb{P}_{change}$, the probability that the route to a destination will change in the next time window, based on the history of route changes for this destination. The hot process allocates probes to the

destinations according to $\mathbb{P}_{change}$. Probes for each destination are then distributed across ASes according to the probability of detecting a topology change by probing an AS.

When DTRACK detects a topology change, it performs a "local" remapping, i.e., it only re-probes the hops that have changed. Remapping receives as input the probe that detected the change, say (d, h) or the probe to destination d with hop distance h. Then, we probe downstream of h (i.e., we send probes with increasing h, similar to traceroute) until we find the route change's joining point (an interface that is both in the old and new routes). Similar, we probe upstream of (d, h) (i.e., probes with decreasing values of h) until we find the route change's branch point.

We are currently evaluating metrics to predict route changes and techniques to optimize the probe allocation so that we can maximize the number of topology changes detected. Trace-driven simulations comparing the current version of DTRACK to Tracetree [51], which is the fastest known technique to measure a topology map, and traceroute are encouraging. These preliminary results show that, given the same probing budget, DTRACK detects a higher fraction of topology changes with lower detection delay than Tracetree and than classic traceroute.

3.3 NetDiagnoser

"NetDiagnoser: Troubleshooting Network Unreachabilities Using End-to-end Probes and Routing Data", CoNEXT 2007 [52]

The techniques presented in the two previous sections—failure confirmation, aggregation strategies, and DTRACK—improve the quality of the inputs to binary tomography algorithms, but some issues are yet to be solved.

1. Links can fail "partially". Router misconfigurations such as incorrectly set BGP policies or packet filters [53,54] may cause a link to fail only for a subset of the paths using that link. Binary tomography cannot detect such failures because it assumes that if a link is up, then each path using that link is up.

2. There is life after link failures. Routing protocols (either IGP or BGP) try to reroute around failed links. Binary tomography does not explicitly consider the paths obtained after routing converges to new stable routes.

3. Inference using only end-to-end measurements can be inaccurate. Binary tomography uses only end-to-end probing. In cases in which an ISP deploys these techniques, we can improve the diagnosis by using routing messages (which directly report when links are up or down).

4. Some ASes block traceroute. If traceroutes are incomplete, then binary tomography does not have access to the complete topology.

We address these issues with NetDiagnoser [52], a set of algorithms that builds on binary tomography to identify the location of faults in a multi-AS environment. We propose two version of NetDiagnoser. ND-edge uses only end-to-end probing and can be used, for example, by a third-party troubleshooting service without the cooperation of ISPs. ND-bgpigp combines end-to-end probing with routing messages, and hence is appropriate for use by ISPs. We also extend NetDiagnoser to deal with incomplete topology information due to blocked traceroutes.

Locating router misconfigurations and using rerouted paths. ND-edge addresses the first two issues listed above. To handle failures due to such misconfigurations, ND-edge extends the network topology with a set of logical links to represent each interdomain link. We focus on interdomain links because usually operators only apply BGP policies at border routers. To capture router configurations and policies at the finest granularity, we should ideally have logical links on a per-prefix basis. However, this could result in a very large topology (tier-1 ISPs have more than 280,000 prefixes in their routing tables). Further, BGP policies are usually set on a per-neighbor basis, rather than on a per prefix basis [55], which means that logical links on a per neighbor basis should be sufficient. We can then apply binary tomography on this extended topology to identify misconfigurations. Binary tomography assumes a fixed topology. It uses the topology before the failure, but not the topology after routing protocols have converged to new stable routes. Therefore, it is not able to use information from paths that were rerouted, and work after the failure. If a path is rerouted but still works after the failure, then every link on the new path are working; we can then safely remove the links on this path from the set of candidate failed links.

Using routing messages. ND-bgpigp uses ND-edge, but it pre-processes the initial set of candidate failed links based on the routing messages exchanged by a network. Using IGP messages is straightforward, as these messages directly indicate the status of IGP links. Whenever ND-bgpigp receives a link down message, it directly marks the link as failed. We can also use BGP withdrawals to help narrow down the set of failed links. If a router, R, receives a BGP withdrawal message, then clearly the failure happened between R and the destination. Hence, we can add the links between R and the destination as part of the set of candidate failed links.

Dealing with blocked traceroutes. ND-edge and ND-bgpigp assume that the topology is complete. However, we use traceroutes to measure the topology and not all routers

respond to traceroute probes (in fact, traceroute stars are fairly common [56]). If the failed link falls in an AS that blocks traceroute, then it is impossible to exactly determine that link. We make the assumption that if an AS blocks traceroutes, then no router in that AS will respond, and if an AS allows traceroutes, each router in that AS will respond with a valid IP address. We disregard the case where only a few routers in an AS do not respond due to ICMP rate limiting. This problem can be solved by repeating the traceroute for the source-destination pair. We introduce a feature in NetDiagnoser that can be used to identify the AS(es) with failed links, when the topology contains stars. We call this algorithm ND-LG, because it uses information from Looking Glass servers [57]. Looking Glass servers located in an AS allow queries for IP addresses or prefixes, and return the AS path as seen by that AS to the queried address or prefix. The ND-LG algorithm proceeds in two steps: First, we map each star to an AS. Then, we cluster links with stars that could actually be the same link.

We evaluate multiple variations of NetDiagnoser algorithms using simulations based on realistic inter-AS topologies. Our results show that even the simple versions of these algorithms can successfully identify a small set of links, which almost always includes the actually failed links. We show that the use of routing messages is essential to narrow down the set of candidate links. Our troubleshooting algorithms obtain useful results even in the presence of ASes that block traceroute-like probes.

3.4 Summary

Binary tomography algorithms hold great promise for locating network failures. Unfortunately, there are a number of practical challenges to obtain accurate reachability matrices and topologies for binary tomography in practice: (1) the inability to distinguish persistent failures from bursty, congestion-related losses; (2) the lack of synchronized end-to-end measurements; and (3) the long delay to infer the network topology. This chapter has designed and evaluated a failure confirmation method and aggregation strategies to address the first two problems, respectively; and we are currently designing DTRACK, a probing strategy to track topology changes. Then, we proposed NetDiagnoser, which enhances binary tomography algorithms with techniques that can take advantage of the information obtained from rerouted paths, BGP and IGP messages, and Looking Glass servers. NetDiagnoser can also locate failures due to router misconfigurations and in ASes that block traceroutes.

Our next step is to combine these measurement methods with NetDiagnoser to build a real-time, tomography-based monitoring system that can quickly detect and locate network faults. Such a system will be more effective if it can also detect and identify other types of failures—for instance, intermittent failures and performance disruptions. We plan to couple tomography-based identification with the automatic detection techniques discussed in the next chapter. One issue we have not yet addressed is how to disambiguate if a failure happened in the forward or the reverse path when we have no control of the destination of probes. We will incorporate the spoofing technique proposed in the Hubble system [19]. In a large-scale deployment, consolidating all measurements in a central coordinator represents a bottleneck. We will explore data aggregation techniques to reduce this overhead and a distributed solution to avoid a communication bottleneck at a centralized coordinator.

Chapter 4

Reducing Detection Overhead

The fault *identification* techniques described in the two previous chapters pinpoint the location of faults once a problem is *detected*. This detection could be manual. For example, when an end-user realizes that she can no longer reach a web site, she could launch a traceroute to start identifying the cause of the problem. Similarly, an operator could launch traceroutes and network tomography after a customer's complaint. Manual detection is far from ideal; operators would prefer to avoid any customer complaints and end-users would prefer to avoid the frustration that comes with trying to diagnose a problem. Hence, identification techniques would be considerably more effective if coupled with automatic detection techniques.

Fast detection is challenging, because it requires *continuously* monitoring the status of end-to-end paths. There is a large probing cost to monitor paths among a large set of monitors and destinations at a very high frequency. One approach to reduce probing overhead is to infer path status by passively monitoring traffic, instead of issuing measurement-specific probes. However, we cannot completely eliminate active probing. Network operators cannot passively track the status of all individual TCP and UDP flows in real-time in high speed links, because of the large processing and memory overhead to track a large number of active flows. It is easier for network operators to deploy dedicated measurement servers in multiple locations in their networks and perform active measurements to detect instances of faults or bad performance.

Passive analysis is promising for end-host monitoring, because the volume of traffic to monitor on end-hosts is lower than on backbone links. Passive techniques only observe traffic, so there is no overhead due to injected probes. Another advantage is that passive traffic analysis detects the problems that affect the user's traffic. With active measurements, it is often hard to determine the set of paths that should be probed. The main issues with

passive analysis when running directly at end-user's machines is that it is only available when the machine is on, that it raises privacy concerns, and that it may overload the machine's resources.

We argue that both passive and active detection techniques are complementary in practice. Passive techniques are more practical at end-user machines or content servers where traffic volumes are lower; whereas active probing is necessary from dedicated measurement servers where there is no user traffic. Hence, we explore both active and passive detection techniques. Section 4.1 designs algorithms to minimize probing cost for detecting faults and Section 4.2 develops a passive monitoring tool that runs on end-user's machines to detect performance disruptions that affect users' experience.

4.1 Minimizing active probing cost

"Minimizing Probing Cost for Detecting Interface Failures: Algorithms and Scalability Analysis", INFOCOM 2009 [58]

We use active probing to detect faults that cause at least some paths to become unreachable. Paths may become unreachable for many reasons such as fiber cuts, router crashes, or network blackholes (failures that do not appear in the network's alarm system). Blackholes may be caused by router software bugs [59], errors in the interaction between multiple routing layers [60], or router misconfigurations [53, 54]. In such cases, end-to-end packet loss or outright loss of reachability are the only indication that a link has failed [1].

We consider an active monitoring system like the one presented in Fig. 3.1 with monitors that probe a set of destinations and send the results of probes to a coordinator. Our goal is to quickly detect unreachabilities at a *target network* with minimum probing overhead. We can think of the target network as the network of an ISP that deploys the system or that subscribes to monitoring services like Keynote [2] and RIPE TTM [3].

A simple approach to detecting unreachabilities in a network is to issue active probes from all monitors to all destinations. Unfortunately, this approach has serious scalability issues. First, probes consume network bandwidth and monitors' CPU. Second, quick detection needs frequent measurements of the status of each link in the network. Consequently, each monitor has a limited time to complete measurements to all destinations. The common approach to deal with this scalability problem is to select the smallest set of paths that covers all the links in the network [61, 62]. In the example in Fig. 3.1 say that the target network corresponds to the links 4, 5, and 6. Three paths (m_2 to d_1, m_1 to d_3, and m_3 to d_3) are sufficient to cover the three links. This path selection approach can effectively detect all *fail-stop faults*, which are faults that affect all packets that traverse the faulty equipment. However, this approach does not take into account *path-specific faults*, which are faults that only affect packets being forwarded toward a sub-set of the destination hosts. For instance, a misconfiguration of the route to d_2 would go unnoticed.

Our work [58] designs algorithms to optimize probing for detecting faults (both fail-stop and path-specific) in a target network. This work was Hung Nguyen's internship at Technicolor (co-advised with Patrick Thiran and Christophe Diot). Instead of selecting the minimum set of paths, we propose to select the frequency to probe each path in a way that probes cover a target network but do not generate too much overhead. Our key insight is that we can combine lower frequency per-path probing to achieve a high frequency probing of the links of the target network. Returning to the example in Fig. 3.1, say that m_2 and m_3 probe each target once every eight minutes. If we select the timing to issue these probes carefully, we can detect fail-stop faults of link 4 at a significantly higher frequency than per-path failures (once every two minutes).

Problem statement. We formalize this novel definition of the probe optimization problem as follows. We take as input the set of paths between all the monitors and destinations, $\mathcal{P}$, and the set of links in the target network, $\mathcal{I}$. For every link, we consider two types of failures: fail stop or path specific. The detection of fail-stop failures is easier; any probe that traverses the failed link will detect the fault. On the other hand, the detection of path-specific failures requires monitoring all paths that cross a link. At the same time, fail-stop failures affect a larger number of paths, and consequently should be detected more quickly. We incorporate this difference into our model by requiring that the active monitoring service detect all fail-stop failures that last for at least $\Delta\tau_1$ seconds, whereas path-specific failures should last for at least $\Delta\tau_2$ seconds to be detected. In practice, some fail-stop and path-specific failures may be shorter than $\Delta\tau_1$ and $\Delta\tau_2$, respectively. Instead of focusing on these short failures that recover automatically and fast, we choose to ensure the detection of persistent failures, which often require the operator's intervention for recovery. We expect $\Delta\tau_2 > \Delta\tau_1$, but this is not a requirement in our model. The output of our optimization is the set of probing rates, λ_k, for each path $P_k \in \mathcal{P}$ that achieves the minimum probing cost, where the probing cost of a given active monitoring system is defined as

$$C = \sum_{k=1}^{|\mathcal{P}|} \lambda_k. \tag{4.1}$$

Algorithms. The original problem formulation of selecting the smallest set of paths that covers a target network is an instance of the set-cover problem, known to be NP-hard [63]. Our new formulation of the probe optimization problem allows us to find the optimal solution by using linear programming (LP). Our solution requires that monitors probe each path P_k at rate at least $\lambda_k = 1/\Delta\tau_2$ (to detect path-specific failures) and that the sum of the probing rates of all paths traversing every link i be at least $1/\Delta\tau_1$, i.e., monitors can coordinate to probe a link. This LP solution, however, has an implicit overhead because

it requires synchronization among monitors. Thus, we consider an alternative probabilistic scheme where monitors independently send probes on a path P_k as a Poisson process with rate λ_k.

Analysis of the scaling law. We develop analytical models that consider the optimal probing cost achieved by the LP solution to show that random power-law graphs are the most costly to probe: the cost grows linearly with the size of the target network. This result is significant for practical Internet scenarios, even if the actual Internet topology is not power law [64]. The reason is that a diagnosis system based on active monitoring can only detect faults in the graph probed by its set of monitors; and topologies resulting from measurements of Internet paths often exhibit a power-law degree distribution [65, 66]. We validate our analytical models and evaluate the sensitive of our solutions to the inputs with traceroute data collected from PlanetLab [67] and RON [38] nodes.

4.2 Passive monitoring at end-hosts

"Perspectives on Tracing End-Hosts: A Survey Summary", CCR 2010 [68]
"Peeking without Spying: Collecting End-Host Measurements to Improve User Experience", LIP6 Technical report 2009 [69]

Passive monitoring is promising for detecting the problems that affect end-users, because such monitoring directly observes user's traffic. In addition, it introduces no probing overhead. It is possible to track round-trip times (RTTs) and retransmissions just by observing active TCP connections. Repeated retransmissions indicate loss of reachability [14]. Similarly, very large RTTs or packet losses indicate performance disruptions. Loss of reachability should directly affect the user's perception of network performance. Inferring the user's perception is more challenging when looking at metrics like large delays or losses. For example, an increase in RTT may go unnoticed if users are watching a video, because all video players use play-out buffers; whereas the same delay change may result in an SSH connection to become unusable. The goal of the thesis of Diana Joumblatt is to develop passive techniques that run on end-hosts to detect the network performance disruptions that affect end-users.

Unfortunately, the study and development of such techniques requires end-host measurements, which are fundamentally hard to collect. By "end-host measurements", we mean passive measurements that are collected *directly* on a host computer, and not those collected at access points [70]. Collecting data at endhosts allow us to understand application-network interactions from the user's perspective since we can get data about the user's context, such as processes running on a machine and CPU utilization. By placing the data collection close to the user, we also open up the opportunity to get the user's feedback,

which is essential in studying which performance disruptions affect the user's quality of experience. There are a few datasets collected directly on end hosts [8, 11, 71, 72], but none of them contains the user's feedback on the network performance.

The scarcity of end-host measurement datasets arises for both technical and psychological reasons. Monitoring tools can consume enough processing and storage resources for users to observe a slowdown in machine performance. The second reason has to do with the fear that someone studying their data will find something private and that this information will end up in the wrong hands. The privacy issue is vastly complicated because personal views on privacy differ across generations, cultures and countries. On the one hand, privacy laws do not pose a constraint when a measurement tool is installed by a user on her machine, since she explicitly gives consent when downloading and installing the tool. However, the fear of users not being willing to participate in such measurement studies has discouraged many in our community from pursuing the development of such tools and consequently the research that relies on such tools.

We are working in collaboration with Nina Taft and Jaideep Chandrashekar to build *HostView*, a data collection tool that runs on end-hosts. Many design tradeoffs arise such as the utility or benefit of collecting a given piece of data—the user's comfort with that data being monitored, and the overhead incurred via the technique used to collect the data. We combine both qualitative and quantitative elements to drive our own decisions for designing such a tool: we conducted an online user survey of 400 computer scientist to understand user comfort issues [68], and we carry out evaluations of the overhead and utility of different techniques for collecting various data.

Based upon our empirical analysis and the user survey, HostView incorporates the following features:

- **Network data.** The sniffer module collects packet headers with anonymyzed IP sources, and extracts the content-type from HTTP packet responses (whether the HTTP content is audio, video, image or text). An application lookup module, based on the gt toolkit [73], periodically logs applications (process names) associated with open network sockets. HostView launches a traceroute to www.google.com whenever the local machine acquires a different source IP address and maps the IPs of the first three hops to ASes using Team Cymru's IP-to-AS mapping [74]. We only export the AS numbers and names from the local machine. HostView also logs whether the active network interface is wired or wireless. In the latter case, a pop-up questionnaire asks the user to describe the wireless environment (work, airport, coffee shop, etc.). To protect each user's identity, only a cryptohash of the SSID is recorded along with the user description of the wireless network.

- **Machine performance data.** The sysperf module takes care of sampling system performance measurements. Currently, we only log CPU load, but the module is extensible and other types of measurements can be added easily. Additionally, the module also registers user (in)activity by recording timestamps of mouse and keyboard activity at a coarse level.

- **User feedback.** We capture the user's perception of performance with the user feedback module. This module incorporates two different mechanisms: an "I am annoyed!" button and a system-triggered feedback form. The *"I am annoyed!" button* is always displayed at the edge of the screen, so users can click on it when they are not happy with their network performance. The *system-triggered feedback form* prompts the user no more than three times per day to respond to a questionnaire about their network experience in the 5 minutes preceding the questionnaire. The system-triggered questionnaires are a form of "Experience Sampling Method (ESM)", which originates from the field of psychology [75] and has been adopted within the HCI community [76]. We design an experience-sampling algorithm that uses weighted random sampling to get user feedback with higher probability when network load is high. The questionnaire has 5 short questions and should take roughly 1 minute. The system-triggered questionnaire is configurable, so that users can turn it off.

A number of features have also been incorporated to make the tool more appealing to users. First, we remove any host identifying information: users are only identified with a randomly generated id, used to construct the trace file names; even source IP addresses, in the sniffer module, are anonymized using a SHA-256 hash. Second, the trace upload (which is done regularly so as to not use up too much disk space at the end-host), is done via secure file transfer. The files are stored on a server with restricted access (to a few individuals who are explicitly named on the project webpage). Third, the tool incorporates a "pause" mechanism that lets users turn off all logging (in half hour increments) when they carry out some activity that they do not want to be recorded. A beta version of the tool (for MacOS and Linux) can be found at: http://cmon.lip6.fr/EMD/EMD/Home.html.

We are testing HostView with a group of students in our lab. These tests reveal one more challenge of developing end-host measurement tools, namely, that of developing portable software to run on a diverse set of machine and system configurations. After this testing phase, we plan to release HostView and perform a one-month data collection campaign with a group of volunteers. We plan to recruit volunteers from the people who gave us their email addresses when they filled out the survey and word of mouth. Although our survey participants were mostly computer scientists, we hope to get enough volunteers to experience the diversity of users' tolerance to network performance disruptions.

4.3 Summary

This chapter addressed two problems for the automatic detection of faults and performance disruptions. First, we studied the problem of minimizing the number of probes required to detect failures in a network. We proposed a more practical formulation for this problem that incorporates both "fail-stop" and "path-specific" failures. Our formulation allows us to find a solution that minimizes probing cost in polynomial time using linear programming. Then, we described our efforts to build HostView, an end-host data collection tool. HostView is our first step towards passive detection of performance disruptions. In particular, we incorporate a user feedback module to help infer the performance disruptions as perceived by end-users.

We have identified many interesting directions for future work. The first immediate one is to implement an active monitoring system based on these probing techniques and deploy it in operational networks. We plan to extend our optimization techniques to minimize probing cost to detect other types of faults or performance disruptions. In the area of passive detection, we will first release HostView to obtain end-host measurement datasets annotated with user feedback. One issue we are facing is that of incentives for volunteers to participate in our experiment, because the current version of the tool does only data collection. We plan to extend HostView to report some simple statistics like the fraction of bandwidth utilization per application. After we get these end-host traces, the challenge will be to interpret the user feedback and correlate it with network performance metrics. Finally, we will study how to combine active and passive detection techniques and integrate them with the identification techniques described in the previous chapters.

Chapter 5

Perspectives

Our work in the past five years has just scratched the surface of the problem of network troubleshooting from end-hosts. We have designed measurement methods and tools that achieve more accurate network measurements and inferences. However, we are still far from a system that automatically detects a problem, identifies its root cause, and hopefully bypasses or fixes it. Our ultimate goal is to make network management transparent to users. In the near future, we plan to get closer to the end-users by applying the techniques developed so far in two platforms: directly at end-user laptops or desktops and at home gateways (for instance, DSL or cable modems).

5.1 Monitoring and troubleshooting from end-user machines

End users can deploy monitoring and troubleshooting software in their personal machines to track the performance of networked applications and launch a troubleshooting procedure upon the first sign of trouble. Our goal is to evolve HostView into such a tool. First, we want to develop techniques to detect problems as perceived by users (i.e., the quality of experience). Then, we can couple these detection techniques with tools to identify the origin of the problem (for instance, we could use a tool like netalyzr [4] or apply tomography if multiple users collaborate). Besides troubleshooting, we can also report raw performance and performance per application, so that users can verify whether they are getting what they paid for and whether their providers discriminate against any of the applications they use. We could imagine extending the SLAs to include clauses on the experience users get with different services, instead of just a promise of download and upload capacity.

End users can also collaborate to infer the performance of residential access providers (such as DSL and cable) and how providers treat their customers' traffic. For instance,

grenouille [7] is a French nationwide project to measure the performance of access links. Today, grenouille has thousands of active members, covering all major ISPs and cities in France. The grenouille client reports basic performance metrics to end-users. The grenouille server combines the measurements from multiple users to build the "weather forecast" of broadband access providers (a sun indicates that the performance matches the SLA, a cloud that there is some degradation, and so on). This provider rating helps end-users pick the best provider and SLA in each city. We have a cooperative project financed by the French National Research Agency (ANR) in collaboration with grenouille and other research labs in France. This project gives us the opportunity to quickly deploy our techniques thereby enhancing grenouille with more accurate performance-inference techniques and other types of statistics (like the providers that are responsible for more failures or that filter certain applications).

We are also analyzing grenouille's current datasets to understand the factors that affect the throughput and latency experienced by users of broadband access networks. A better understanding of how a user's choice of ISP and SLA affect performance can help users make better decisions to improve both reliability and performance. In addition to ISP and SLA, understanding how performance varies per city can also help the designers of networked services (e.g., overlay networks, content distribution networks) to decide where to replicate content and services to avoid paths that experience simultaneous performance degradations.

The main limitation of measuring access network performance with tools that run at end-users' machine (like in grenouille) is that it is not easy to distinguish between the access network performance from the performance of the home network. In fact, home networks are becoming more and more complex with n-play service (which brings TV, video-on-demand, and phone over IP), wireless LANs, and many devices competing for bandwidth. Thus, end-users may be experiencing problems just because of a poorly configured or over-utilized home. We plan to develop techniques to identify faults and performance disruptions inside the home network. Such tools will give users more confidence when reporting problems to their access ISP as well as help end-users manage their home networks.

5.2 Monitoring and troubleshooting from home gateways

Residential access networks are seeing steady deployment. Over the past decade, Internet usage has grown by more than 3.5 times, to about 1.6 billion users, about 300 million of which are broadband subscribers [77]. We expect that this growth will continue and that the home will become a central place for end-users to access the Internet and to store and share their personal content. To implement this vision, we will work on a home-

centric networking architecture. We have just been awarded funding from the European Commission to conduct this research.

Given that the home gateway connects the home with the rest of the Internet, we see it as the ideal place to implement the functionalities of this new architecture. The home gateway can serve both to manage the home network and to help monitor the access ISP network. It is simpler to distinguish between faults or performance disruptions originated in each of these networks from the gateway. Gateways are often controlled by the access ISP; they are the only vantage point ISPs have close to the customer. We plan to deploy the techniques described in this document on home gateways to pinpoint faults or performance disruptions at access networks in behalf of ISPs. ISPs can also sell home management service to their customers. In this scenario, the gateway will monitor and control the home network in behalf of home users. Nevertheless, this gateway-centric architecture also poses new challenges. We need to deploy both active and passive monitoring techniques continuously and online inside the gateway, but home gateways have limited resources. We will work on designing monitoring techniques that can be efficiently embedded at gateways.

The grenouille project represents an immediate opportunity for a large-scale deployment, whereas a modified home gateway will take longer to reach tens of thousands of homes. We will learn from our experience with the grenouille deployment to guide the design of measurement tools and their deployment in home gateways. These deployment scenarios represent new opportunities to improve the networking experience of home users. The focus of research in network monitoring, troubleshooting, and management should not only be on large networks with expert administrators, but also on end-users and their everyday networking experience.

References

[1] R. Kompella, J. Yates, A. Greenberg, and A. Snoeren, "Detection and Localization of Network Blackholes," in *Proc. IEEE INFOCOM*, (Anchorage, AK), 2007.

[2] "Keynote Systems – The mobile and Internet performance authority." `http://www.keynote.com/`.

[3] RIPE, "Test Traffic Measurements Service." `http://www.ripe.net/ttm/`.

[4] "Netalyzr." `http://netalyzr.icsi.berkeley.edu/`.

[5] M. Mathis *et al.*, "Network Path and Application Diagnosis." `http://www.psc.edu/networking/projects/pathdiag/`.

[6] R. Carlson, "Network Diagnostic Tool." `http://e2epi.internet2.edu/ndt/`.

[7] Grenouille, "Grenouille." `http://www.grenouille.com/`.

[8] C. R. S. Jr. and G. F. Riley, "Neti@home: A distributed approach to collecting end-to-end network performance measurements," in *Proc. PAM*, 2004.

[9] "Speed Test." `http://www.dslreports.com/stest`.

[10] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi, "Detecting bittorrent blocking," in *Proc. IMC*, 2008.

[11] M. B. Tariq, M. Motiwala, and N. Feamster, "Detecting Network Neutrality Violations with Causal Inference," in *Proc. CoNEXT*, 2009.

[12] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu, "Glasnost: Enabling end users to detect traffic differentiation," in *Proc. USENIX NSDI*, 2010.

[13] V. Padmanabhan, L. Qiu, and H. Wang, "Server-based Inference of Internet Link Lossiness," in *Proc. IEEE INFOCOM*, (San Francisco, CA), 2003.

[14] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-area Services," in *Proc. USENIX OSDI*, (San Francisco, CA), 2004.

[15] "tcpdump/libpcap." See `http://www.tcpdump.org/`.

[16] J. Cleary, S. Donnelly, I. Graham, T. McGregor, and M. Pearson, "Design principles for accurate passive measurement," in *Proc. PAM*, 2000.

[17] V. Jacobson, "traceroute," Feb 1989.

[18] V. Paxson, "End-to-end Routing Behavior in the Internet," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, no. 5, pp. 601–615, 1997.

[19] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson, "Studying Black Holes in the Internet with Hubble," in *Proc. USENIX NSDI*, (San Francisco, CA), 2008.

[20] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network Tomography: Recent Developments," *Statistical Science*, vol. 19, no. 3, no. 3, pp. 499–517, 2004.

[21] N. G. Duffield, "Network Tomography of Binary Network Performance Characteristics," *IEEE Trans. Information Theory*, vol. 52, no. 12, no. 12, pp. 5373–5388, 2006.

[22] Y. Tsang, M. Coates, and R. Nowak, "Network delay tomography," *IEEE Trans. Signal Processing*, vol. 51, pp. 2125–2136, 2003.

[23] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Accurate and Efficient SLA Compliance Monitoring," in *Proc. ACM SIGCOMM*, (Kyoto, Japan), 2007.

[24] R. Caceres, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-based Inference of Network-internal Loss Characteristics," *IEEE Trans. Information Theory*, vol. 45, no. 7, no. 7, pp. 2462 – 2480, 1999.

[25] N. Duffield, J. Horowitz, and F. Prestis, "Adaptive Multicast Topology Inference," in *Proc. IEEE INFOCOM*, (Anchorage, AK), 2001.

[26] H. Nguyen and P. Thiran, "Using End-to-End Data to Infer Lossy Links in Sensor Networks," in *Proc. IEEE INFOCOM*, 2006.

[27] H. Nguyen and P. Thiran, "Network Loss Inference with Second Order Statistics of End-to-end Flows," in *Proc. IMC*, (San Diego, CA), 2007.

[28] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and K. Claffy, "The CAIDA IPv4 Routed /24 Topology Dataset."

[29] Cisco, "How does load balancing work?." See `http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094820.shtml`.

[30] Juniper, "Configuring load-balance per-packet action." From the JUNOS 7.0 Policy Framework Configuration Guideline, see `http://www.juniper.net/techpubs/software/junos/junos70/swconfig70-policy/html/policy-actions-config11.html`.

[31] J. Postel, "Internet control message protocol." RFC 791, Sep 1981.

[32] F. Viger, B. Augustin, X. Cuvellier, C. Magnien, M. Latapy, T. Friedman, and R. Teixeira, "Detection, understanding, and prevention of traceroute measurement artifacts," *Computer Networks*, vol. 52, no. 5, no. 5, pp. 998–1018, 2008.

[33] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proc. IMC*, (Rio de Janeiro, Brazil), 2006.

[34] M. Toren, "tcptraceroute," Apr 2001. See `http://michael.toren.net/code/tcptraceroute/`.

[35] B. Augustin, T. Friedman, and R. Teixeira, "Multipath Tracing with Paris Traceroute," in *Proc. Workshop on End-to-End Monitoring (E2EMON)*, May 2007.

[36] B. Augustin, T. Friedman, and R. Teixeira, "Measuring load-balanced paths in the Internet," in *Proc. IMC*, 2007.

[37] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira, "Failure Control in Multipath Route Tracing," in *Proc. IEEE INFOCOM*, (Rio de Janeiro, Brazil), 2009.

[38] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, no. 5, pp. 131–145, 2001.

[39] M. Rabbat, M. Coates, and R. Nowak, "Multiple Source, Multiple Destination Network Tomography," in *Proc. IEEE INFOCOM*, (Hong Kong, China), 2004.

[40] Y. Huang, N. Feamster, and R. Teixeira, "Practical Issues with Using Network Tomography for Fault Diagnosis," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, no. 5, pp. 53–58, 2008.

[41] I. Cunha, R. Teixeira, N. Feamster, and C. Diot, "Measurement Methods for Fast and Accurate Blackhole Identification with Binary Tomography," in *Proc. IMC*, 2009.

[42] E. Gilbert, "Capacity of a Burst-Noise Channel," *Bell Systems Technical Journal*, vol. 39, no. 5, no. 5, pp. 1253–1265, 1960.

[43] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," in *Proc. IMW*, (San Francisco, CA), 2001.

[44] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot, "The Role of PASTA in Network Measurement," in *Proc. ACM SIGCOMM*, (Pisa, Italy), 2006.

[45] I. Cunha, R. Teixeira, and C. Diot, "Tracking the Evolution of IP Topologies: Change Detection and Local Remapping," *Technicolor Technical Report CR-PRL-2010-03-0001*, 2010.

[46] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet Map Discovery," in *Proc. IEEE INFOCOM*, (Tel Aviv, Israel), 2000.

[47] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), 2002.

[48] R. Sherwood, A. Bender, and N. Spring, "DisCarte: a Disjunctive Internet Cartographer," in *Proc. ACM SIGCOMM*, (Seattle, WA), 2008.

[49] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient Algorithms for Large-scale Topology Discovery," in *Proc. ACM SIGMETRICS*, (Banff, Canada), 2005.

[50] Y. Shavitt and E. Shir, "DIMES: Let the Internet Measure Itself," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, no. 5, pp. 71–74, 2005.

[51] M. Latapy, C. Magnien, and F. Ouédraogo, "A Radar for the Internet," in *Proc. First Inter. Workshop on Analysis of Dynamic Networks*, (Pisa, Italy), 2008.

[52] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "NetDiagnoser: Troubleshooting Network Unreachabilities Using End-to-end Probes and Routing Data," in *Proc. ACM CoNEXT*, (New York, NY), 2007.

[53] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP Misconfiguration," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), 2002.

[54] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. USENIX NSDI*, (Boston, MA), 2005.

[55] W. Muhlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an AS-topology Model that Captures Route Diversity," in *Proc. ACM SIGCOMM*, 2006.

[56] M. H. Gunes and K. Sarac, "Analyzing Router Responsiveness to Active Measurement Probes," in *Proc. PAM*, 2009.

[57] NANOG, "Looking Glass Sites." `http://www.nanog.org/lookingglass.html`.

[58] H. X. Nguyen, R. Teixeira, P. Thiran, and C. Diot, "Minimizing Probing Cost for Detecting Interface Failures: Algorithms and Scalability Analysis," in *Proc. IEEE INFOCOM*, 2009.

[59] M. Caesar and J. Rexford, "Building Bug-tolerant Routers with Virtualization," in *Proc. ACM SIGCOMM PRESTO workshop*, (Seattle, WA), 2008.

[60] L. Fang, A. Atlas, F. Chiussi, K. Kompella, and G. Swallow, "LDP Failure Detection and Recovery," *IEEE Communication Magazine*, vol. 42, no. 10, no. 10, pp. 117–123, 2004.

[61] Y. Bejerano and R. Rastogi, "Robust Monitoring of Link Delays and Faults in IP Networks," *IEEE/ACM Trans. Networking*, vol. 14, no. 5, no. 5, pp. 1092–1103, 2006.

[62] H. Nguyen and P. Thiran, "Active Measurement for Multiple Link Failure Diagnosis in IP Networks," in *Proc. PAM*, (Antibes-Juan les Pins, France), 2004.

[63] M. R. Garey and D. S. Johnson, *Computers and Intractibility: A guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[64] L. Li, D. Alderson, and J. C. Doyle, "Towards a theory of scale-free graphs: Definition, properties, and implications," *Internet Math.*, vol. 2, pp. 431–523, August 2005.

[65] A. Lakhina, J. Byers, M. Crovella, and P. Xie, "Sampling biases in IP topology measurements," in *Proc. IEEE INFOCOM*, 2003.

[66] A. Achlioptas, A. Clauset, D. Kempe, and C. Moore, "On the bias of traceroute sampling or, power-law degree distribution of regular graphs," in *Proc. STOC*, 2005.

[67] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in *Proc. USENIX OSDI*, (Seattle, WA), 2006.

[68] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft, "Perspectives on Tracing End-Hosts: A Survey Summary," *ACM SIGCOMM Computer Communication Review*, Apr 2010.

[69] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft, "Peeking without Spying: Collecting End-Host Measurements to Improve User Experience," *LIP6 Technical Report RP-LIP6-2009-10-31*, 2009.

[70] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proc. IMC*, 2009.

[71] S. Guha, J. Chandrashekar, N. Taft, and D. Papagiannaki, "How Healthy are Today's Enterprise Networks?," in *Proc. IMC*, October 2008.

[72] E. Cooke, R. Mortier, A. Donnelly, P. Barham, and R. Isaacs, "Reclaiming Network-wide Visibility Using Ubiquitous Endsystem Monitors," in *Proc. USENIX Annual Technical Conference*, 2006.

[73] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. Claffy, "GT: Picking up the truth from the ground for Internet traffic," in *ACM SIGCOMM Computer Communication Review*, 2009.

[74] Team Cymru, "IP to ASN Mapping." `http://www.team-cymru.org/Services/ip-to-asn.html`.

[75] M. Csikszentmihalyi and R. Larson, "Validity and Reliability of the Experience-Sampling Method," *Journal of Nervous and Mental Disease*, no. 175, no. 175, pp. 526–536, 1987.

[76] S. Consolvo and M. Walker, "Using the Experience Sampling Method to Evaluate Ubicomp Applications," *IEEE Pervasive Computing Magazine*, vol. 2, no. 2, no. 2, 2003.

[77] "Internet World Stats." `http://www.internetworldstats.com/dsl.htm`.