# END-TO-END INFERENCE OF INTERNET PERFORMANCE PROBLEMS

A Thesis
Presented to
The Academic Faculty

by

Partha Kanuparthy

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

College of Computing
Georgia Institute of Technology
December 2012

# END-TO-END INFERENCE OF INTERNET PERFORMANCE PROBLEMS

Approved by:


Dr. Constantine Dovrolis, Advisor
College of Computing
*Georgia Institute of Technology*

Dr. Mostafa Ammar
College of Computing
*Georgia Institute of Technology*

Dr. Ellen Zegura
College of Computing
*Georgia Institute of Technology*

Dr. Kimberly Claffy
CAIDA
*UC San Diego*

Dr. Konstantina Papagiannaki
Telefonica Research



Date Approved: 10 November 2012

*To Appa and Amma, for their support and encouragement.*

# ACKNOWLEDGEMENTS

I owe my deepest gratitude to my *guru*, advisor and mentor, Constantine. Constantine accepted me as his student right after I entered Georgia Tech as a fledgeling masters student. Thanks to Constantine for believing in my abilities. Constantine is a *guru* in the true sense of the word - he mentors his students and helps them grow. He has always been there to encourage me when I was doing fine, to teach me when I was walking down the wrong alley, and to patiently remind me whenever I slowed down. It is my privilege and fortune to have worked with an advisor so closely and to constantly learn from him for six years. I will miss the intense, focused and rich technical discussions I have had in each and every meeting with him. And I will look forward to many more in the future.

My thesis committee played a significant role in shaping my dissertation. Thanks to Mostafa for a very enjoyable collaboration on my first research project of this dissertation. Thanks to Ellen for taking me as her student in my first semester at GT - and for introducing me to the NTG group. Thanks to Kc for being a great mentor and a sounding board for my research over the course of my Ph.D. This thesis would not be what it is without her valuable and detailed comments. I would also like to thank Kc for the many invitations to CAIDA for the AIMS workshop, which is a great occasion to interact with the Internet measurement community, and has been my yearly ritual.

I have had two memorable summer internships during Ph.D. I can't thank Dina enough for the opportunity to work with her and for patiently teaching me the basics of wireless research. I would also like to thank Dina for her constant support and encouragement over the course of my Ph.D. Thanks to Peter and Srini for taking the time off their busy schedules at CMU to brainstorm and discuss about the WLAN-probe project. A big thank you to Christos and Thomas for hosting me at MSR Cambridge and for introducing me to problems in home networks early on. Thanks again to Christos, and to Prateek, James and Minh for a fun summer at Cambridge!

The Internet studies in this dissertation could not have been done without the support of many people. First and foremost, thanks to the folks at Measurement Lab for creating a wonderful platform for Internet measurements, and for inviting me to be a part of it all. I would like to thank all the users of my tools for running the tools, for recommending them, and for taking the time to email me about bugs, ports, fixes and feedback. This dissertation has immensely benefited from your support. The thesis has also benefited from conversations with the Federal Communications Commission and ISPs (including Comcast, ESnet and Internet2).

The NTG group at Georgia Tech is a wonderful place to grow up with during Ph.D. years. Thanks to the NTG faculty and colleagues for creating an intellectually stimulating environment. Thanks to Amogh and Murtaza for the friendship, and the innumerable discussions about research and about life over the years. Thanks to Chuck, Srikanth, Anirudh, Alireza, Abhinav, Hyojoon, Sam, Shuang, Valas, Yogesh, Bilal, Aemen, Nan, Maria and Mehmet for being wonderful colleagues and lab mates. Thanks to Ahmed for the discussions, to Saamer and Ilias for the fun times, and to Samantha for organizing lab events. Thanks to Sridhar for introducing me to NTG. Thanks to Manish, Ruomei, Ravi and Mukarram for the support as NTG "seniors".

My grad school years were enriched by colleagues and friends outside the lab. Special thanks to Hrishi, Vishal and Mahendra for the valuable friendship and for being there whenever I needed a break from networking and work; and for the many discussions about systems research. Thanks to Ahmed (Elmokashfi) and Amund for the fun discussions about networking and research in general. Thanks to Warren for working with me on the DNS load balancing study, and for his work on deploying Pythia. Warren's work gave me a valuable testbed to deploy, test and improve Pythia.

Thanks to the professors at CoC and ISyE who taught me about their areas. Thanks to my undergrad professor, C. S. Kumar, who had encouraged me to explore networking.

It was at NTG that I was fortunate enough to meet my best friend and dear wife, Saeideh. I am deeply indebted to her for her love, constant encouragement and immense support ever since we met. Saeideh has always been behind me and has unconditionally

supported everything that I did or wanted to do, even if it meant to sacrifice the time we spent together. She has patiently taken care of the countless occasions that demanded attention during the last two years, often at the expense of her own work. She would patiently listen to my comments about stress at work. Her presence gave me the strength to navigate the ups and downs in the last two years of graduate school.

Last but not the least, I would like to thank my parents and my brother for always being on my side, and for being pillars of support for whatever I wanted to pursue in life. Thanks to my father for being an inspiration, for encouraging me to pursue a Ph.D., for always asking me and discussing about Ph.D. life, research and thesis progress, and for always reminding me to keep high standards and integrity in life. Thanks to my mother for the unconditional love, for backing me at every stage of my career and life, and for patiently waiting for my visit to India. Without your support, it would not have been possible to reach this stage in my career. This thesis is dedicated to my parents. Thanks to my brother for always being proud of my smallest accomplishments. Thanks also to Babai and Mava for encouraging my interest in CS. I am grateful to them.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# SUMMARY

Inference, measurement and estimation of network path properties is a fundamental problem in distributed systems and networking. We consider a specific subclass of problems which do not require special support from the hardware or software, deployment of special devices or data from the network. Network inference is a challenging problem since Internet paths can have complex and heterogeneous configurations.

Inference enables end users to understand and troubleshoot their connectivity and verify their service agreements; it has policy implications from network neutrality to broadband performance; and it empowers applications and services to adapt to network paths to improve user quality of experience. In this dissertation we develop end-to-end user-level methods, tools and services for network inference.

Our contributions are as follows. We show that domain knowledge-based methods can be used to infer performance of different types of networks, containing wired and wireless links, and ranging from local area to inter-domain networks. We develop methods to infer network properties:

1. Traffic discrimination (DiffProbe),
2. Traffic shapers and policers (ShaperProbe), and
3. Shared links among multiple paths (Spectral Probing).

We develop methods to understand network performance:

1. Diagnose wireless performance pathologies (WLAN-probe), and
2. Diagnose wide-area performance pathologies (Pythia).

Among our contributions: We have provided ShaperProbe as a public service and it has received over 1.5 million runs from residential and commercial users, and is used to check service level agreements by thousands of residential broadband users a day. The Federal

Communications Commission (FCC) has recognized DiffProbe and ShaperProbe with the best research award in the Open Internet Apps Challenge in 2011. We have written an open source performance diagnosis system, Pythia, and it is being deployed in ISPs such as the US Department of Energy ESnet in wide-area inter-domain settings. The contributions of this dissertation enable Internet transparency, performance troubleshooting and improving distributed systems performance.

# CHAPTER I

# INTRODUCTION

Understanding the performance and characteristics of the Internet is a fundamental problem. Internet paths are highly heterogeneous in the nature of forwarding entities, and this makes studying and diagnosing Internet performance a hard problem. This is primarily due to lack of instruments (methods and tools) that enable estimation and diagnosis of Internet paths. This dissertation aims to build a body of research dedicated to building these methods and tools, and to discover Internet performance *at scale* using them.

End-to-end Internet paths are a complex and diverse mix of links from different networks (Autonomous Systems or ASes) which connect to deliver information (as packets, for example, in packet-switched networks). To illustrate the diversity, consider a conferencing session between two computers $A$ and $B$ connected to the Internet from homes via their respective Internet Service Providers (ISPs). ISPs place a device (e.g., a modem) inside the user's home which serves as a gateway between machines inside the home and the ISP's network. Many users deploy a wireless network (*802.11*) in their home which connects to the modem. The conferencing information that is generated traverses a diverse set of components or links between $A$ and $B$: (1) the operating system stacks, (2) wireless links, if any, (3) the modems that $A$ and $B$ connect to, (4) routers and links in the ISP and in other ASes, and (5) the link between the ISP and the neighbor AS and links between other ASes, called inter-domain links.

Many of the links that a packet traverses have *configurable properties*. For example, the home user may change the frequency band that the wireless link uses, the ISP may configure the modem to do traffic shaping, or the ISP may classify traffic and treat applications based on their class. Such configurations can affect the *Quality of Experience* (QoE) that applications (e.g., conferencing) can deliver. For example, users may notice a drop in QoE of an audio conference when packets are delayed in the network, while file downloads suffer

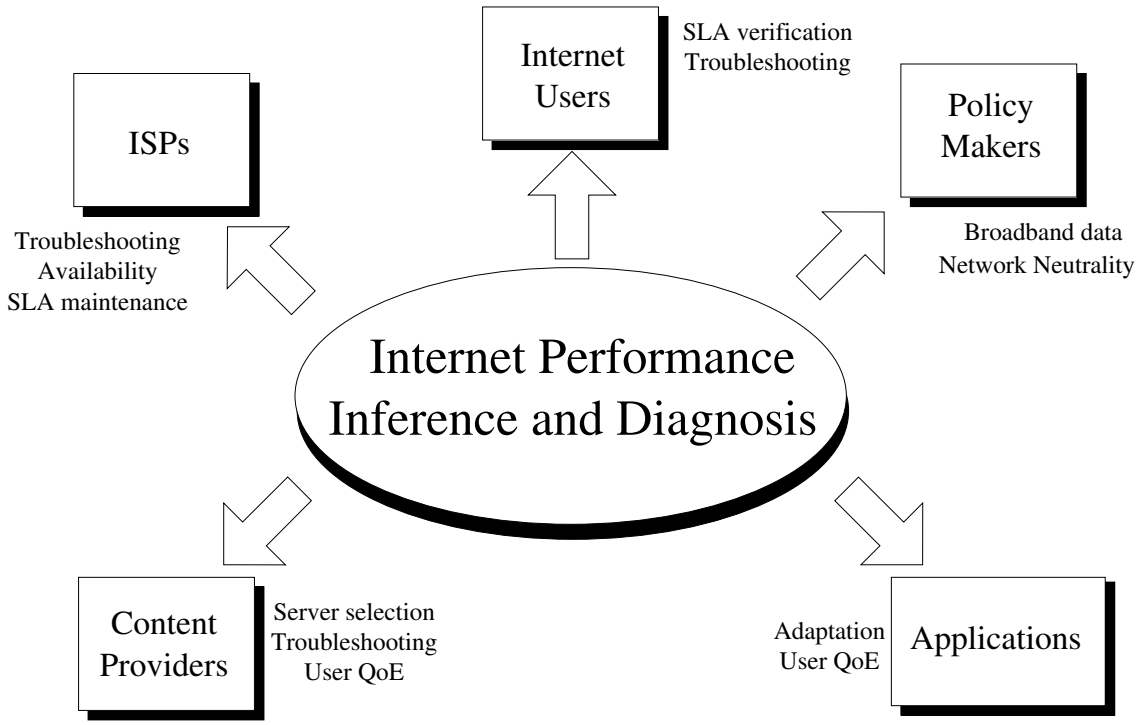



**Figure 1:** Applications of the thesis. Section 1.5 summarizes the thesis impact.

when packets are dropped. Further, network pathologies such as congestion (due to traffic from other users) and faulty network devices degrade the performance of applications. It is important to diagnose pathologies, since they affect QoE. For example, wireless interference inside the home due to neighborhood networks, a faulty router line card or fiber will induce packet losses, while congested links can induce large delays and jitter. We build methods to discover properties and to diagnose performance of Internet paths.

There exist many tools such as "speed tests" that provide users with a single end-to-end measurement such as throughput. A detailed inference, however, of the properties and performance of the Internet is necessary, and such inference can have several applications (see Figure 1). Network inference enables end users to understand and troubleshoot their connectivity and verify their Service Level Agreements (SLAs). Inference has policy implications from network neutrality to broadband performance measurement, since there is a dearth of scientific tools and data for policy making (e.g., broadband mapping). Inference empowers applications and services to adapt to network paths to improve user quality of experience. In this thesis, we refer to *inference* to imply both the discovery of the existence

of a network property of interest, and estimation of associated parameters. Examples of inference include traffic shapers and the associated token bucket parameters, and hidden terminals in wireless networks (with no associated parameters).

### *1.1 Problem Statement*

In this dissertation we ask the question:

> *Can we infer and diagnose end-to-end performance of the Internet at user-level?*

We focus on a wide range of networks - from wired to wireless, and from local area to the wide area inter-domain Internet. Using our inference methods, we also undertake a large-scale study of Internet performance.

> *We show that domain knowledge can be used to infer Internet properties and to diagnose performance problems using end-to-end methods* - without requiring any source of external data (e.g., measurement data from routers).

Note that external sources of data are usually not available on end-to-end paths that traverse multiple administrative domains.

Our terminology in the above description is as follows. We refer to user-level techniques to imply techniques that use userspace APIs exposed by commodity operating systems, and which do not require specialized hardware, operating system support, drivers or data (that a user or operator may not have access to). We use the word properties to refer to network configurations; for example, traffic shapers, schedulers, buffer managers and shared links. Figure 2 shows an end-to-end model of the Internet, depicting different forwarding elements (and their configurations) along a path; it highlights the different Internet properties and performance diagnoses that we look at in this dissertation.

We note that the properties that we are interested in are also violations of conventional assumptions about the Internet. Conventional wisdom has been that an end-to-end Internet path is a sequence of work-conserving First-In-First-Out (FIFO) queues having a finite and constant capacity (service rate) and a finite and constant buffer size; and that these buffers

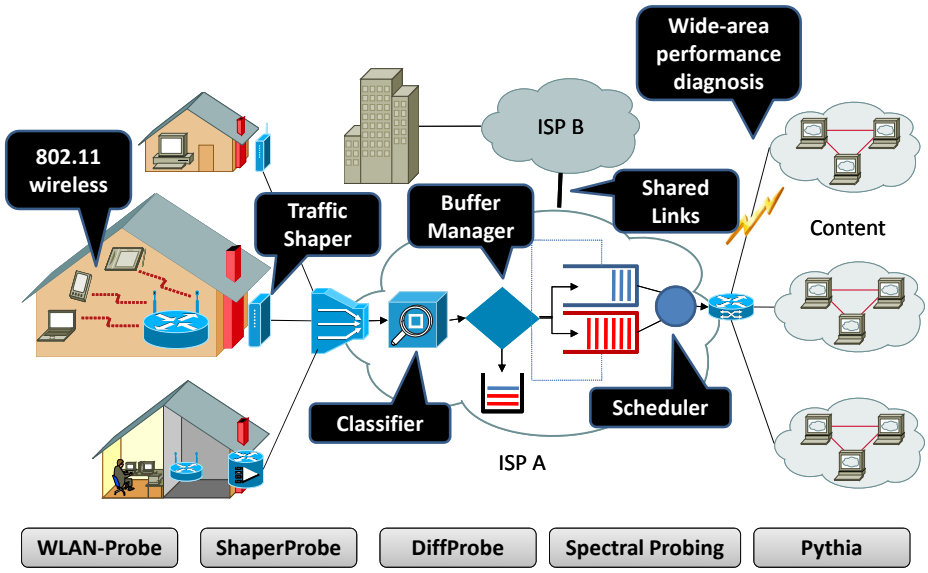



**Figure 2:** Model of the Internet abstracting out important network elements that can affect application performance. The components of the thesis are also shown as labels. This dissertation shows that it is possible to infer and diagnose performance using domain knowledge-based models.

drop incoming packets when they are full. These assumptions are increasingly being violated as ISPs deploy traffic management mechanisms in their networks.

Using the techniques, we develop inference tools, and build and deploy systems that enable end users, network operators and applications to understand network performance. We collect and study large-scale data using the tools to understand Internet performance and properties. Among our findings, we show a significant deployment of traffic shaping in residential broadband networks, and we show that the nature of performance pathologies can be different between the two dominant access technologies, cable and DSL, in residential broadband networks.

## *1.2 Requirements and Challenges*

There are several challenges and subtle tradeoffs in building inference tools. First and foremost, is inference *accuracy*. It is prudent to report that "no inference can be made" if the tool finds that it cannot operate under the conditions, or if the assumptions (if any) of the underlying methods are not valid. It is necessary to consider biases when designing estimators for inferring a path property. A common case of bias is to use TCP throughput

(*bulk-transfer capacity*) as an estimator of narrow link capacity - employed by many "speed tests" today. Bias in this case arises due to several factors; for example, if the narrow link has a TCP bulk-transfer flow, the measured throughput will *always* be less than the capacity. Another bias is to measure "speed" without considering the presence of traffic shapers. It is also important to understand the effect of and account for co-existent network properties - such as noisy wireless links and cross traffic - when designing methods.

Second is *usability* aspect of the tool. A residential user will not have access to sources of data such as NetFlow records or router syslogs; while these may be available to a network operator, we cannot expect the operator to record information about every (including layer-1) device on the network, or have access to flow records or logs from neighboring networks. Another usability consideration is whether the tool is capable of running over commodity hardware and operating systems, or if it requires specific privileges. A common example of non-commodity approaches is in the context of wireless diagnosis tools, which assume features from a specific chipset or modify a specific driver (e.g., MadWifi). Such approaches do not scale to commodity and "closed" devices and common use cases. For tools that do not run in the background, we will need to design methods that take a reasonably short duration to complete.

The combination of accuracy and usability brings many systems challenges when implementing inference tools. The ability to probe and sample an Internet path at a required granularity (from the userspace) in the presence of environmental noise in commodity hardware and operating systems is necessary. Such noise also makes it necessary to design robust inference algorithms. Portability to many commodity platforms is a useful feature to have. A large-scale deployment of an inference service for users brings scalability requirements such as the need for load balancing infrastructure to handle flash crowds, and data handling capabilities.

Finally, a critical component in the design of an active probing method is *non-intrusiveness*. Tools should not adversely affect the performance of existing traffic. They should continuously check the path performance and should either backoff or abort probing if they find drastic drops in performance (e.g., packet losses).

The inference methods in this dissertation address the above challenges. In particular, we show that domain knowledge-based methods can be designed to address them in performance inference of many types of networks.

## *1.3 Approach*

The work in this dissertation combines modeling, statistical inference, and systems implementation. We build and deploy services using the tools, collect large-scale raw data from users and analyze the data to understand Internet behavior. For each property that we want to infer or diagnose, we first create a domain knowledge-based input-output model of an end-to-end Internet path or a network that makes realistic assumptions (for example, the model accounts for Internet paths that include wireless links and end-host noise), while abstracting away components on the path that do not affect inference of that property. We then use this model to design the inference method. In designing an active probing tool, there are three design "degrees of freedom" to tackle: (1) the *probing structure and pattern*: What packet sizes to use? The packet timing? How many packets? The payload and flow ID? Probing patterns interact with the network in specific ways; (2) the *packet measurements*: What per-packet parameters do we record?; and, (3) the *inference*: using the measurements to infer the network property with statistical significance. For passive methods, we use the model to design inference algorithms. Note that the model may need to be refined based on observations of the tool in real conditions.

## *1.4 Thesis Contributions*

This dissertation develops methods, tools and systems for understanding and troubleshooting performance of the Internet. The dissertation also studies Internet performance and properties using these tools. The tools and techniques are useful to Internet users, network operators, content providers, policy makers and distributed applications.

Internet measurement is a significant area of research that can be viewed as comprising two distinct research threads, which depend on each other. First, the sub-area of measurement and inference tools. This body of work builds techniques, tools and systems that

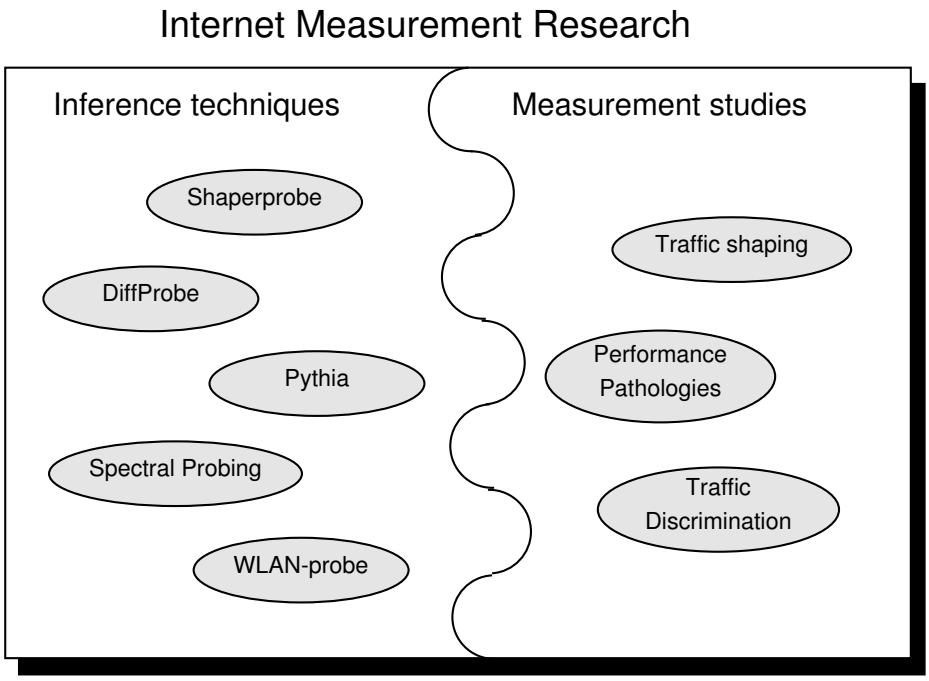



**Figure 3:** The two threads of Internet measurement research, and where this disseration's contributions fit.

enable understanding performance of the Internet. The techniques have long-term potential to enable understanding evolution of Internet performance, and follow-up work may "revise" them as the Internet evolves. Second, the sub-area of measurement studies. This body of work uses these tools and systems to collect measurements and study the Internet. Measurement studies may need to be repeated to have an up-to-date picture as the Internet evolves and changes. This dissertation contributes largely to the former thread, and it also studies large-scale Internet performance towards the latter thread (see Figure 3).

The contributions of this dissertation are the following.

### 1.4.1 Detecting traffic discrimination

We design techniques to detect different types of service discrimination in ISPs. We show a taxonomy of discrimination using a network model - delay and loss discrimination - depending on how an ISP implements discrimination. We show conditions under which it is not possible to detect discrimination, and the probing and pre-processing requirements for accurate detection. We implement the detection methods in a tool, DiffProbe, and test for discrimination in residential ISPs. Our work on DiffProbe highlights some important design

considerations such as probing structures and pre-processing algorithms to ensure low false detection rates.

### 1.4.2 Detecting and estimating traffic shaping and policing

We design the first active and passive methods to detect traffic shapers and policers in ISPs, and to estimate the token bucket configuration. The active probing tool, ShaperProbe, is a public domain service that has been deployed as a public service since 2009. We describe some of the design and implementation decisions that we took to have a tool that works on commodity hardware and in heterogenous paths that include wireless access links. The passive tool works on an offline trace or a live TCP flow to detect and estimate (application-specific) traffic shaping. The detailed diagnosis of the passive tool addresses limitations of the popular TCP-based "speed test" tools which usually give the user a single throughput number that can be biased by traffic shapers.

### 1.4.3 Spectral Probing and detecting shared links

We design methods to detect shared links between many Internet paths (which can be a source of performance problems) using active probing (without requiring any form of congestion in shared links - a limitation of prior work). Spectral probing is a probing methodology that works in the frequency domain and is based on analog communication concepts. We show that an end-to-end Internet path can be modeled as a linear channel, and this can lead to interesting and useful properties in the frequency domain. We show that traffic on Internet paths with shared links exhibits the concepts of crosstalk and frequency multiplexing. We use this framework to detect shared links among several Internet paths using a single probing session. We show other applications of spectral probing that are based on analog communication concepts such as frequency modulation.

### 1.4.4 Wide-area performance diagnosis

We design distributed methods to detect and diagnose performance problems in wide-area inter-domain networks in near real time. Our methods do not require any form of data from the network such as Netflow, SNMP data or customer tickets - which are used by

prior diagnosis systems that rely on data mining techniques (to construct implicit models). We use domain knowledge-based network models to define pathologies and the associated symptoms. Further, our system can be easily configured using domain knowledge of the network operator to improve diagnosis. We implement and deploy the methods in a scalable inter-domain monitoring system called Pythia.

### 1.4.5 Wireless performance diagnosis

We design active probing methods to diagnose the root cause of performance problems in 802.11 networks, which are commonly deployed in homes. We show that active probing methods can be designed for the 802.11 link model, which violates many assumptions of models used by probing methods in wired networks. Our work also highlights limitations of active probing that arise due to propreitary and non-standard implementations in 802.11 network cards. Our tool, WLAN-probe, uses a single wireless end-point for probing and does not require multiple specialized monitoring devices or any form of superuser privileges.

### 1.4.6 Understanding Internet performance

Using some of our methods, we perform a large-scale study of Internet performance. The highlights of our study are as follows. First, we did the first large-scale study of traffic shaping deployments on the Internet. We found traffic shaping in several residential ISPs - many of them cable ISPs - from ShaperProbe data collected in 2009 - 2011. Second, we found no evidence (in 2009) in residential ISPs of discrimination against Skype and Vonage traffic (relative to generic applications) using DiffProbe. Third, we did the first large-scale study of the nature of performance problems in different network types on the Internet: residential broadband, academic and research, and commercial networks, using Pythia. We show differences in performance problems between cable and DSL providers. Finally, using spectral probing, we show evidence of shared links between Internet paths when traceroute does not show any sharing (either because traceroute does not complete in some cases, or due to router aliasing).

## *1.5 Impact*

Over the course of this dissertation, we have provided two public domain tools as services. *ShaperProbe* for inferring traffic shaping: ShaperProbe has received over 1.5 million user runs since 2009 and receives about 1,000-3,000 runs a day. ShaperProbe has helped bring awareness of traffic shaping practices (terms such as "PowerBoost" and "sustained rate") and service level verification to home Internet users (over the traditional "speed test" approach); it is recommended by home users as well as technicians in online discussion forums for troubleshooting. *Pythia* is an open source performance problem detection and diagnosis system that is deployed in operational networks such as the US Department of Energy ESnet and the GA K-12 school networks.

The dissertation research has been recognized with the Best Research Award by the Federal Communications Commission (FCC) in 2011 for work on traffic discrimination and shaping detection (and has been presented to the FCC Chairman, Mr. Julius Genachowski, in 2011). The work has received widespread attention in the media (Ars Technica, Slashdot, O'Reilly Media, PC World, Network World, among others; tech policy sites such as New America Foundation; and several blogs including ISP blogs such as the Comcast Blog).

# CHAPTER II

# RELATED WORK

We discuss related work in the context of each of the end-to-end Internet properties and diagnoses that this thesis tackles.

## *2.1 Schedulers and Buffer Managers*

There has been some recent interest in active and passive methods for detecting traffic discrimination. Perhaps the closest in spirit to our work is Zhang et al.'s NetPolice [127], an active probing methodology that replays application traces with limited TTL values to solicit TTL-expired messages from intermediate routers. They compare loss rates with an HTTP flow as a baseline, and show discrimination in backbone ISPs. We are concerned about the validity of this conclusion mostly for two reasons. First, two simultaneous flows (say HTTP and BitTorrent) can observe very different loss rates if they do not "sample" a lossy queue with packets of the same size and at about the same time. We dealt with this issue using paired statistics, considering packets of the same size that were also sent at about the same time. Second, NetPolice relies on router-generated ICMP responses; the generation of such packets is subject to rate-limiting and vendor-specific lower-priority processing. DiffProbe replays the application traffic and this allows it to see similar treatment in the ISP as what the actual application would experience at that point of time.

Some of the prior work on passive and active discrimination detection methods complements our work. These methods focus on throughput as the metric to identify discrimination. A passive detection method proposed by Bin Tariq et al., NANO [114], uses throughput observations from many end-hosts to detect discrimination. They use causal inference on the client data, and information about confounding variables, to group clients according to performance. BTTest [47] focuses on detecting BitTorrent traffic blocking by ISPs using forged TCP RST packets. It emulates BitTorrent flows, and correlates client and server packet traces to detect RST messages. The authors show that ISPs mostly block

BitTorrent in the upstream direction and classify based on payloads. A follow-up work [46] by the authors emulates other types of applications and compares throughput with that of a "baseline" flow. They show evidence of throughput discrimination against BitTorrent traffic in ISPs. DiffProbe creates a similar baseline flow while probing for discrimination. In contrast to these methods, our work focuses on detecting a taxonomy of discrimination practices; this is useful in practice, since the two types of discrimination can result in very different application behavior.

There has been prior work on detecting loss discrimination. Lu et al. designed POPI [85] to detect priority based forwarding. They use high-rate packet trains to induce losses, and observe loss rates to analyze forwarding. They run POPI on the PlanetLab network and validate their results using hop-by-hop probing for loss rate, and contacting ASes that showed loss discrimination. We show that loss discrimination can be detected using non-intrusive end-to-end methods. We also show how to check for non-identifiability conditions, which is important to reduce false negatives.

Kuzmanovic et al. [77] use passive monitoring to observe service rates at different timescales and infer the parameters of a WFQ scheduler; their methods, however, have been designed for a single-hop path. In an extended abstract, Biczók et al. [29] proposed a method to detect discrimination with prior information about the classifier type. They send a single flow and observe performance difference between the sender and receiver sides. Mahajan et al. use ICMP probes and associate performance with geography to measure differences in backbone ISPs in their work on NetDiff [89]. There are limitations, however, in using ICMP to measure performance (see related work on NetPolice earlier in this section).

There has been some work on discrimination detection after DiffProbe. In a follow-up to DiffProbe, Weinsberg et al. devised techniques to infer queue weights of a discriminatory scheduler [120]. Recently, Kaminsky proposed a spoofing-based method to discover performance differences in flows due to ISP discrimination [60].

DiffProbe constructs an artificial flow that is expected to be classified as "normal" priority; it is hence important to look at the problem of classification. Automated traffic classification often relies on machine learning and statistical techniques. Traffic classification

methods have been compared in prior work [73]. Commercial classification products include (but not limited to) Sandvine PTS8210 and Cisco NBAR. The Traffic Morphing project [121] shows that it is possible to avoid certain kinds of classifiers by altering flow characteristics. DiffProbe alters the packet payload, port and protocol to avoid the flow being classified as low priority.

## *2.2 Traffic Shapers and Policers*

Traffic shaping and policing are available in many network devices that work at L2 and above [8, 116]. There has not been, however, prior work on tools to detect traffic shaping, or to study shaping deployments at scale. Lakshminarayanan et al. recorded initial observations of traffic shaping in residential ISPs in 2003 [81]. Further observations of downstream traffic shaping were recorded in a residential access network study done by Dischinger et al. in 2007 [45]; they used a 10Mbps train for 10s to measure received rate, and did not find evidence of upstream traffic shaping.

Follow-up work to ShaperProbe focused on the effects of shaping deployments ("Powerboost") in cable ISPs [27, 112]. In particular, Sundaresan et al. [112] discuss the effect of shaping on speed tests. Bauer et al. [27] discuss two variants of Powerboost deployments in ISPs - *capped* and *uncapped* - and mention that shaping may not significantly impact applications such as web browsing, while it could impact gaming and video streaming. Both studies refer to advertised tier descriptions by ISPs for ground truth.

Capacity estimation is a critical component in determining traffic shaper configuration. Capacity estimation in non-FIFO scheduling environments such as wireless and cable upstream received attention in the literature, since existing bandwidth estimation techniques assume FIFO scheduling. Lakshminarayanan et al. [80] propose Probegap for *available bandwidth* estimation in cable upstream and in 802.11 links. Portoles-Comeras et al. study the impact of contention delays on packets in a packet train [100]; they show that the contention delay process, under certain conditions, will result in the 802.11 capacity overestimates. We have observed this in downstream capacity estimates while testing ShaperProbe in home wireless networks (the ISP upstream link rate is typically lower than that of 802.11, and

hence capacity estimates are likely to be correct).

### *2.3 Spectral Probing and Shared Links*

We discuss related work in two contexts, which are relevant to the Spectral Probing (SP) framework, and to the shared link detection problem:

**Spectral analysis of Internet traffic:** He et al. analyzed the spectral characteristics imposed by bottleneck links on aggregate traffic [52, 53]. Specifically, they applied frequency-domain analysis on the interarrivals of a packet trace. The basic idea is that a bottleneck link imposes distinct signatures on the underlying traffic; their method detects those signatures at a downstream monitoring point, attempting to estimate the capacity of upstream bottlenecks. Research at BBN also used spectral analysis techniques to extract timing information, such as TCP connection RTTs or topological information from aggregated traffic traces [42]. This work uses the SP framework's notion of crosstalk to detect mobility in a wireless network. Broido et al. used spectral techniques to detect periodicities in the timeseries of DNS updates for private (RFC 1918) addresses received by an authoritative nameserver [32]. The previous approaches are based on offline spectral analysis of passive measurements, while our method is based on active probing in the frequency domain. A 2003 patent from Partridge and Cousins [98] proposed covert channels in packet networks using hidden frequencies in the packet interarrivals. This approach is similar to the frequency covert channel application of spectral probing.

Signal processing methods have also been used for efficient detection of traffic anomalies such as flash crowds, denial-of-service attacks and network outages [26]. Hussain et al. [58] used spectral analysis to detect and classify denial-of-service attacks as single and multi-source attacks. Related work in signal processing-based anomaly detection includes work by Cheng et al. [38] and Huang et al. [56].

The work of San-qi Li and his group looks at frequency domain properties of network queues, and is closely related to spectral probing. They have worked extensively on the spectral analysis of queueing processes (see their paper [84] and references therein). An important result of their analytical work is that, in any queueing system, there is a break

frequency, under which the low-frequency traffic stays intact as it crosses the queueing system. The characteristics of this low-frequency traffic can have a significant impact on downstream queues. Our work is related to this result given that the probing signals we create can go through a sequence of queues as long as their frequency is sufficiently low (see Equation 23 in the thesis).

**Tomography and the shared congestion problem:** Network tomography aims to infer internal link characteristics using end-to-end measurements. Scalar tomography attempts to infer metrics, such as the loss probability or the delay of a network link. Boolean tomography aims to infer a binary state ("good" versus "bad") for each network link. A recent work focused on the identifiability problem in scalar tomography [37], and it used a Fourier domain technique to infer link delays. Rabbat et al. proposed a multiple source tomography scheme that can be used to detect sharing (at any store-and-forward device) among two paths [103]. Their technique is based on the timing characteristics with which packet-pairs arrive at the two destinations. Their method is based on time-domain analysis, while ours is based on frequency-domain analysis.

The problem of identifying *shared congestion* among two or more paths has received significant attention in the last decade [44, 72, 74, 75, 107]. The basic idea in most related techniques is to examine the cross-correlation of end-to-end delays, losses or interarrival measurements between paths to infer whether those paths share the same bottleneck. Our approach is significantly different because the shared link does not need to be congested in order to detect path sharing. The only constraint is that the prober should be able to cause queueing at the shared queue when it sends probing trains.

## *2.4 Wide-area Performance*

There has been significant prior work on detection and diagnosis of performance problems. The work can be categorized into two classes; we present representative work in each class.

**Data-oriented methods:** These are diagnosis methods that use significant amount of data sources usually available in enterprises and single administrative domains, but *not* in a wide-area inter-domain setting. In such settings, a detailed and fine-grained diagnosis is feasible.

There is a tradeoff, however, between how *detailed* the diagnosis can be and the wide-area applicability (*generality*) of a diagnosis method. A summary of some of these methods follows. G-RCA [122] works on data sources within a single network such as SNMP traps, syslogs, alarms, router configurations, topology and end-to-end measurements. It mines dependency graphs from the data and constructs diagnosis *rules* from the graphs. Giza [91] uses multiresolution analysis and mines dependencies on similar data. SyslogDigest [102] mines faults from router syslogs. NetMedic [61] and Sherlock [25] diagnose faults in an enterprise by profiling end-host variables and by mining dependencies from historic data. NICE [90] enables troubleshooting by analyzing correlations across logs, router data and loss measurements. META [118] looks at both spatial and temporal features of network data to learn fault signatures; and a follow-up work [119] matches these signatures using network-wide data. NetPrints [21] learns decision trees of configuration data. A recent study [115] used email logs and network data to analyze routing-based failures. It should be noted that learning approaches may require frequent training.

Data-oriented methods have also been used to diagnose specific pathological behavior in the context of a single network. Feather et al. look at diagnosing soft failures in a LAN using domain knowledge on a pre-defined set of features [49]. We take a similar approach to diagnosis using domain knowledge, but in the more general wide area inter-domain context. Lakhina et al. used unsupervised clustering methods on packet-level features in packet traces to classify performance anomalies [79]; this method requires packet traces of traffic, however, it can complement Pythia by classifying problems that cannot be diagnosed using domain knowledge. Huang et al. use structural properties of network packet traces to detect performance problems in a LAN [56]. Huang et al. showed [57] that inter-domain routing anomalies can be identified using BGP updates. Mercury [92] uses a statistical change point detection method to detect changes in the network. There has been extensive work on structural methods to detect anomalies in volume data in an ISP; for example, the influential work by Lakhina et al. used dimensionality reduction on volume data [78].

**Active probing methods:** These methods rely on active probing, and are typically based

on domain knowledge. They can reveal detailed and accurate diagnosis, since they provide the choice of carefully crafting probing structures based on domain knowledge. It is, however, hard to widely deploy a new active probing tool in a large monitoring network, especially if some of the monitors are in other ASes; moreover, active probing methods are tailored to diagnose specific pathologies and not the overall network health. We summarize a few representative tools below. Netalyzr [76] probes to help a user with troubleshooting information. Tulip [88] diagnoses and localizes reordering, loss and congestion using a single end-point. PlanetSeer [124] uses a combination of active and passive methods to monitor path failures.

Prior work designed probing tools for specific diagnoses such as: to detect Ethernet duplex mismatch [108], measure bottleneck buffer size [41], detect buffering problems [93] and reordering [86] using TCP. In the context of 802.11 wireless LANs, our work on WLAN-probe [66] and the work from Rayanchu et al. on AirShark [104] diagnose 802.11-specific root causes of performance problems, while Jigsaw [40] diagnoses root causes of TCP performance problems using distributed 802.11 monitors in an enterprise.

**Comparison with our approach:** In contrast to related work, our approach uses basic end-to-end active measurements and can easily plug into an existing monitoring infrastructure (which collects such measurements using widely-used tools such as *ping* or a simple implementation of one-way probing such as *One-Way Ping*). Our methods are applicable in the wide area inter-domain setting, since they do not require network-specific or ISP customer data such as syslogs, SNMP data, routing tables or trouble tickets and alerts.

## *2.5 Wireless Performance*

There is significant prior work in the area of 802.11 Wireless LAN (WLAN) monitoring and diagnosis. To the extent of our knowledge, however, there is no earlier attempt to diagnose WLAN problems using exclusively user-level active probing, without any information from 802.11 devices and other layer-2 monitors. User-level active probing has been used to estimate *conflict graphs* and hidden terminals, assuming that the involved devices cooperate in the detection of hidden terminals [22,95,97]. Instead, with our approach (WLAN-probe),

hidden terminals may not participate in the detection process (and they may be located in different WLANs). Passive measurements have also been used for the construction of conflict graphs [33, 50, 117].

Earlier diagnosis systems require multiple 802.11 monitoring devices [39, 40, 87], NIC-specific or driver-level support for layer-2 information [36, 110], and network configuration data [21]. Model-based approaches model transmissions from the NIC to predict interference [71, 83, 101, 105]. Signal processing approaches decode PHY signals to identify the type of interference [82]; some commercial spectrum analyzers [18, 19] deploy such monitoring devices at vantage points.

# CHAPTER III

# INFERRING SCHEDULERS AND BUFFER MANAGERS

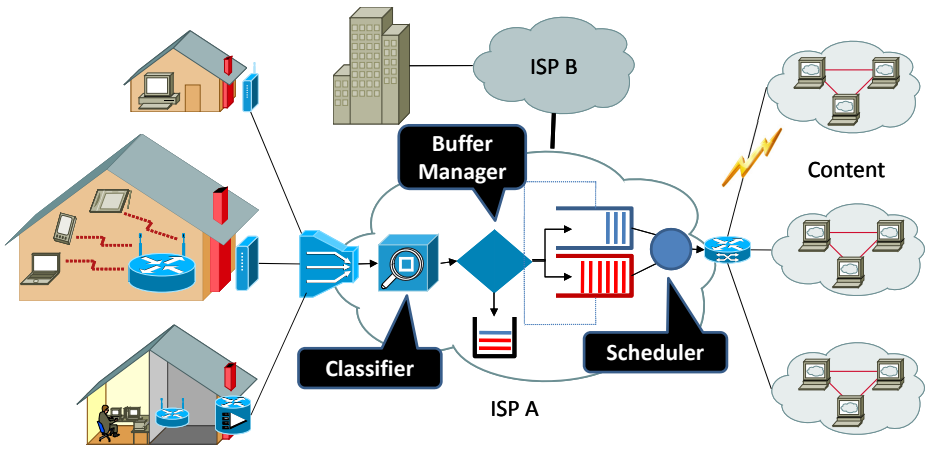



**Figure 4:** Internet model showing discriminatory scheduling and buffer management in an ISP. The classifier marks flows into different priorities.

## *3.1 Introduction*

There is significant interest recently about the so-called "Network Neutrality" debate [55]. Users are concerned that their access ISPs will degrade the network performance that is offered to certain applications, such as peer-to-peer file sharing, or "over-the-top" services (such as Skype, Vonage or Joost) that may be competing with similar services offered by the ISP. Prior work [47] showed evidence that some ISPs are discriminating against BitTorrent traffic by rate limiting or blocking such flows.

We propose an active probing method, referred to as *Differential Probing* or DiffProbe, that can be used to detect delay and/or loss discrimination of given traffic flows. Such discrimination can be easily performed by ISPs today, given that most routers allow real-time classification of traffic[1] and provide packet scheduling and buffer management mechanisms that can be used for service discrimination (such as Strict Priority (SP) scheduling, Weighted

[1]Examples of commercial classifiers include the Sandvine PTS8210 and Cisco NBAR.

Fair Queueing (WFQ), or Weighted RED packet dropping (WRED)). We also show how to distinguish between SP and WFQ scheduling variants.

The detection problem that we focus on can also be viewed as an instance of a new class of network tomography [35] problems. Instead of estimating internal link delays or losses or the topology of the network, in this class of tomography problems the objective is to identify the type of forwarding modules that a packet flow goes through. In this chapter, we consider two packet scheduling forwarding modules (SP and WFQ) as well as discriminatory packet dropping schemes such as WRED. Our objective is to design and evaluate the probing and statistical methods for the detection of these forwarding modules.

**Chapter organization:** Section 3.2 presents our model for ISP service discrimination. Section 3.3 gives the basic idea of DiffProbe and describes the probing pattern. Sections 3.4 and 3.5 focus on the delay and loss discrimination detection problems. We have implemented and tested our tool on several ISPs, as described in Section 3.6. Section 3.7 evaluates the detection accuracy with simulation and controlled emulation experiments.

## *3.2 Basic Model and Definitions*

We consider the basic model illustrated in Figure 5. A number of users are connected to an ISP $I$ through access links. The user traffic goes through a classifier $M$, which marks flows as High ($H$) priority or Low ($L$) priority. We assume that the classification is done at the granularity of IP flows, even though our method is agnostic to the exact classification scheme (whether it is payload-based, port-based, a behavioral method like BLINC [70], etc).

If the ISP discriminates against low priority traffic, the classified traffic goes through a *discriminatory ISP forwarding module* that applies different packet scheduling and buffer management policies to the two classes of traffic. Most routers today implement at least two discriminatory schedulers: Strict Priority (SP), and variants of Weighted Fair Queueing (referred to as WFQ in the rest of this paper). SP services a packet from the $L$ queue only if the $H$ queue is empty when the link becomes available. A WFQ scheduler guarantees a minimum bandwidth share to each class[2]. Even though many other scheduling algorithms

[2]The many variants of WFQ differ in how accurate this allocation is across flows in small timescales [116], an issue that is not important in our context.

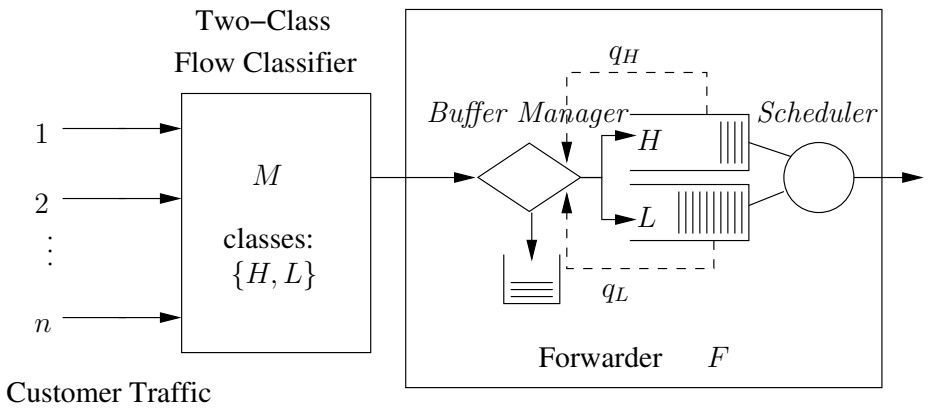



**Figure 5:** Model of access ISP discrimination.

have been proposed in the literature, SP and WFQ variants are the main schedulers that are available in routers today [116].

In order to perform buffer management and loss discrimination, most routers today support Weighted RED (WRED) [4], allowing incoming $L$ packets to be dropped with a higher probability than incoming $H$ packets. Another form of loss discrimination can be performed using the Drop-Longest-Queue policy, which removes a potentially backlogged packet from the longest queue when there is no buffer for an incoming packet.

If the ISP does not perform discrimination, we expect that the scheduling discipline will be First-Come First-Served (FCFS), there will be a single queue for all traffic, and the buffer management policy will be Drop-Tail (DT) (i.e., drop an arriving packet if there is no space to store it). Note that in some cases, an ISP may conduct loss discrimination but not delay discrimination (e.g., to use FCFS scheduling and WRED on a single queue), or delay discrimination but not loss discrimination (e.g., to use SP scheduling and DT buffer management with both queues sharing the same pool of packet buffers). Our high-level objective is to enable a user of ISP $I$ to detect whether $I$ performs any type of service discrimination on her traffic using an active probing methodology.

Note that it is possible that an ISP deploys discriminatory mechanisms, but without really affecting the user traffic. All previously mentioned forwarding mechanisms (SP, WFQ, WRED, etc) are identical to FCFS-DT when there are no backlogged packets at the discriminatory link, which is often the case under low load conditions. Obviously, we are not

interested in such low load conditions because *there is no effective discrimination* in such cases. Besides, the ISP would not have the incentive to deploy discriminatory mechanisms if they would remain idle. Instead, we aim to detect discrimination when it actually affects user traffic. This would typically happen during time periods, potentially short, of high load at the discriminatory link. This does not mean that we assume that the ISP network is heavily loaded. The more relevant question is whether a user observes any service discrimination even when she receives (or sends) traffic at the maximum possible rate that her access link allows. If not, for all practical purposes the ISP does not deploy service discrimination on her traffic.

### *3.3 Differential Probing*

The basic idea in Differential Probing is to generate two flows: an *Application (A) flow* that may be classified by the ISP as low priority, and a *Probing (P) flow* that should be classified as normal traffic and thus will not be discriminated against. The user first sends (and then receives) these two flows through the network simultaneously, and then compares their delay and loss characteristics. Discrimination is detected when the two flows have experienced statistically significant queueing delays and/or loss rate.[3] We assume that $A$ and $P$ flows are generated between the same end-points. The proposed architecture may be extended to cover cases where the A and P flows traverse different paths but share a discriminatory link; we do not consider such cases in this work.

The $A$ flow can be generated by an actual application, or it can be an application packet trace that the user replays. It represents traffic that the user is suspecting the ISP may be discriminating (e.g., BitTorrent or Skype traffic). The $P$ traffic is a synthetic flow that is created by DiffProbe under two constraints. First, it should be sufficiently different than the $A$ flow so that it does not get classified in the same class as $A$. Second, it should be sufficiently similar with $A$ so that it observes the same network performance, statistically speaking, in terms of delays or packet losses if there were no discrimination.

[3]When we refer to "delays" in this chapter we mean the end-to-end delays of a flow, after subtracting the minimum observed measurement from the raw end-to-end delay measurements. The presence of a clock offset does not influence these measurements because we focus on relative delays and not absolute delays.

For example, if the ISP classifies traffic based on port numbers, the $P$ flow can be identical with the $A$ flow but it should use different ports. If the classification is based on packet payload information (e.g., specific HTTP strings), the $P$ flow can have randomized payloads. If the classification is behavioral-based, focusing on specific flow features such as packet sizes, packet interarrivals, port numbers, etc, the $P$ flow can be created in principle as a sufficiently randomized version of the $A$ flow (e.g., with distorted packet sizes, average rate, rate fluctuations, etc). Wright et al. [121] show that it is possible to defeat statistical traffic classification by changing flow characteristics. Our DiffProbe implementation generates $P$ flows from Skype and Vonage $A$ flows using a combination of port, payload, packet size and rate randomization. We expect that this combined randomization would be sufficient for most traffic classifiers today.[4]

To ensure that the two flows see similar network performance when the ISP does not perform discrimination, we rely on the following two techniques. First, we consider only those $P$ packets that have been sent very close in time with a corresponding $A$ packet. Thus, even if the $P$ flow can include many more packets than the $A$ flow, with different sizes and interarrivals, we rely on *paired statistics* and consider $(A, P)$ packet pairs that have "sampled" the ISP discriminatory link at about the same time. Second, a $P$ packet that is sent shortly after an $A$ packet has the same size as the latter. This ensures that the network transmission delays of the $(A, P)$ packet pairs that we consider are equal.

### 3.3.1 Probing pattern

Differential Probing works by sending the $A$ and $P$ flows through the ISP network simultaneously, and then comparing their delay variations and packet losses. DiffProbe creates the following probing pattern that consists of two phases, a Balanced Load Period ($BLP$) followed by a Load Increase Period ($LIP$). In the following, we denote by $\lambda_A(t)$ the nominal rate of the $A$ flow and by $\lambda_P(t)$ the nominal rate of the $P$ flow, at time $t$. The flows are of variable bitrate, and so these rates vary with time.

[4]Most commercial classifiers are based on port numbers and payload information. Behavioral classification is viewed as not sufficiently accurate.

- $BLP$: we send both flows at their nominal rates.
- $LIP$: we scale up the rate $\lambda_P(t)$ (by scaling down the packet interarrivals) of the $P$ flow by a factor $g(t) > 1$.

We generate a $LIP$ period for the following reason. Our objective is to maximize the chances that there is some queueing in the (potentially discriminatory) ISP network. As previously discussed, without having some queueing in the ISP's network it is not important whether a delay and/or loss discrimination mechanism is deployed. Given that the user's access link is probably of lower capacity than the ISP's links, the highest rate that the user can generate is that of her access link. We cannot modify the rate of the $A$ flow, however, as that may affect its classification by the ISP. Consequently, during a $LIP$ period, we artificially increase the rate of the $P$ flow to the point that, together with the $A$ flow, the two flows almost saturate the user's access link. Specifically, we dynamically adjust the $P$ flow rate so that: $\lambda_A(t) + g(t)\lambda_P(t) \approx C_a\,(1-\epsilon)$, where $C_a$ is the user's access link capacity (in the upstream or downstream direction, depending on the direction of probing), and $\epsilon > 0$. Thus, our goal is to *not* introduce queueing at the user's access link, and focus instead on the delays/losses that take place at the ISP's network. In practice, we use $\epsilon = 0.1$ and calculate $g(t)$ with a sliding window estimate of $\lambda_A(t)$. To avoid probing intrusiveness, we increase $\epsilon$ if we observe significant losses in the $P$ flow. The $LIP$ duration is chosen so that we have a sufficiently large sample of $(A, P)$ packet pairs to detect loss discrimination (see Section 3.5 for sample size requirements of our statistical tests). We assume that the ISP does not deploy a mechanism that marks down the priority of the $P$ flow during $LIP$; this is a valid assumption, since the rate $C_a\,(1-\epsilon)$ is sufficiently low. The reason behind the $BLP$ period is described next.

### 3.3.2 Unidentifiability

The $BLP$ is used to identify cases in which we cannot detect whether the ISP deploys service discrimination mechanisms. We compare the higher delays of $P$ packets during the $LIP$ period with the average delays of $P$ packets during the $BLP$ period. If the former are not significantly larger than the latter, the stimulus that we generated during the $LIP$ period

was not sufficiently high to trigger a significant increase in the queueing delays of the $P$ flow. We view these cases as *unidentifiable*, given that even if the ISP deploys discriminating mechanisms, those mechanisms would have no significant effect on the user's traffic.

We compare the 90th percentile of $P$ flow delay distribution during the $LIP$ period ($\mathcal{D}_{0.9}(P)$) with median delay of the same flow during the $BLP$ period ($\mathcal{D}_{0.5}^{BLP}(P)$):

$$\mathcal{D}_{0.9}(P) > (1+\delta)\mathcal{D}_{0.5}^{BLP}(P) \tag{1}$$

where $\delta > 0$. We choose $\delta = 0.1$ based on empirical observations of the $P$ delay distribution's variability and user-perceived discrimination (i.e., the difference between $A$ and $P$ delays). We say that delay discrimination is unidentifiable if the above condition is *not* true. We account for any clock skew that may exist between the $BLP$ and $LIP$ periods. The $BLP$ duration is chosen reasonably large (at least 10s) to ensure a sufficiently large number of samples for computing and comparing percentiles in equation 1.

### *3.4 Delay Discrimination Detection*

Under FCFS scheduling, the two flows will be serviced by the same queue. So, at least in statistical terms, the $A$ and $P$ flows would observe similar delay distributions. On the other hand, a Strict Priority scheduler will provide lower delays to the $P$ flow packets as long as there is some backlog in the discriminatory link, since the scheduler will always serve the high priority queue until it is not non-empty (before serving the low priority queue). The WFQ scheduler serves the queues based on the weights assigned to them by the ISP. The WFQ scheduler can be used to provide delay discrimination only if the bandwidth share that is provided to the high priority class is larger than the bandwidth share provided to the low priority class, relative to the traffic load of the two classes. Specifically, suppose that the WFQ weights $\phi_H$ and $\phi_L$ are assigned to high priority and low priority traffic, respectively (with $\phi_H + \phi_L = 1$). Let $\lambda_i$ be the offered load in class $i$. To achieve delay discrimination in favor of the high priority, the ISP should make sure that the ratio of weight to offered load is higher for $H$ than the ratio for $L$:

$$\alpha_H > \alpha_L \tag{2}$$

where

$$\alpha_i = \frac{\phi_i}{\lambda_i}, i \in \{H, L\}$$

Note that if $\alpha_H \gg \alpha_L$, a WFQ scheduler would exhibit a behavior similar to SP, i.e., it would service low priority packets only when there are no backlogged high priority packets.

We detect delay discrimination as follows. Recall that the $A$ flow is classified as low priority, while $P$ is classified as high priority. We observe the empirical distribution of delays of the $A$ and $P$ flow packets during the $LIP$ period; call them $\mathcal{D}(A)$ and $\mathcal{D}(P)$, respectively. We detect delay discrimination (SP or WFQ) when:

$$\mathcal{D}(A) \gg \mathcal{D}(P) \tag{3}$$

On the other hand, we detect no delay discrimination, i.e., FCFS scheduling, when:

$$\mathcal{D}(A) \approx \mathcal{D}(P) \tag{4}$$

**Test for Equality of Delay Distributions:** We first perform the test of equality of distributions (4) as follows. Our test is based on the non-parametric *Kullback-Leibler (KL) divergence*, and it is motivated by the test used in a prior work [113]. The KL-test does not assume any priors about the input delay distributions. We have not used the well-known Kolmogorov-Smirnov (KS) test, since that test can be inaccurate when the underlying distributions exhibit *discontinuities*. The delays of Internet paths usually include a large number of samples close to the sum of the propagation and transmission delays, causing significant discontinuities.

The empirical distributions $\mathcal{D}(A)$ and $\mathcal{D}(P)$ are constructed from the measured delay timeseries $\{t_i^A, d_i^A\}$ and $\{t_j^P, d_j^P\}$ of $A$ and $P$ flows respectively. Timestamps $t^A$ and $t^P$ are taken at the sender. We first pre-process the two timeseries to form a *paired* sample $\mathcal{D}$ as follows. For each delay sample $(t_i^A, d_i^A)$, we find the nearest sample $(t_j^P, d_j^P)$ in time, such that $|t_i^A - t_j^P| \leq \tau$ for a threshold $\tau$ defined as the transmission time of an MTU-sized packet in the bottleneck link. If there exists no such sample $(t_j^P, d_j^P)$, we discard $(t_i^A, d_i^A)$. Otherwise, we add the delay tuple $(d_i^A, d_j^P)$ to $\mathcal{D}$ and continue with the next sample $(t_{i+1}^A, d_{i+1}^A)$. After pairing, $\mathcal{D}$ will consist of sample pairs that form $\mathcal{D}(A)$ and $\mathcal{D}(P)$.

We also discard from $\mathcal{D}$ delay values close to the propagation delay, that is those values that are less than $\tau$ time units above the propagation delay. We then subtract the propagation delay (computed as the minimum of all delay samples for that flow) from each delay sample in $\mathcal{D}$. Thus, our statistical analysis focuses on queueing delays, not absolute end-to-end delays. Note that clock skew does not affect this test because the difference in one-way delays of a paired sample, even with delay discrimination, would be small compared to the timescales in which clock skew is significant (many seconds).

The next step is to construct a non-parametric hypothesis test for the null hypothesis that $\mathcal{D}(A)$ and $\mathcal{D}(P)$ come from the same underlying distribution. For two discrete probability distributions $X$ and $Y$, the KL-divergence of $Y$ from $X$ is defined as:

$$D\left(X\|Y\right) = \sum_i X(i) \log_2 \frac{X(i)}{Y(i)}$$

The KL-test on $\mathcal{D}(A)$ and $\mathcal{D}(P)$ proceeds as follows:

1. Estimate the probability mass functions $X^A$ and $X^P$ from the samples $\mathcal{D}(A)$ and $\mathcal{D}(P)$ defined on the same set of bins. The bin width $w = 2n^{-1/3}I$ is determined using the inter-quartile range $I$ of the joint sample $\mathcal{D}(A) \cup \mathcal{D}(P)$, where $n$ is the length of the joint sample. We merge bins with their neighbors if the number of measurements from both samples is less than 1%.

2. Calculate the KL-divergence $D\left(X^A\|X^P\right)$.

3. *Bootstrapping:* randomly partition without replacement $\mathcal{D}(A)$ into two samples $\mathcal{S}_i^1$ and $\bar{\mathcal{S}}_i^1$. Estimate their probability mass functions $X_i^1$ and $\bar{X}_i^1$ using the binning procedure in (1). Calculate the KL-divergence $D\left(X_i^1\|\bar{X}_i^1\right)$. Repeat this a number of times (we use 200) to estimate the distribution of $D\left(X_i^1\|\bar{X}_i^1\right)$; call it $\mathcal{D}\left(X_i^1\|\bar{X}_i^1\right)$.

4. Reject the null hypothesis if $D\left(X^A\|X^P\right)$ is *large* compared to the distribution $\mathcal{D}\left(X_i^1\|\bar{X}_i^1\right)$. More precisely, define the p-value as:

$$p = Prob\left[D\left(X^A\|X^P\right) \leq \mathcal{D}\left(X_i^1\|\bar{X}_i^1\right)\right]$$

   and reject the null hypothesis with this p-value if $p < 0.05$.

**Inequality of Delay Distributions:** We detect delay discrimination using a test that checks whether one of the two distributions is consistently larger than the other (equation 3). Our test of inequality is as follows. We first require that the KL-test rejects the null hypothesis that $\mathcal{D}(A) \approx \mathcal{D}(P)$; otherwise we say that the distributions are equal and we detect FCFS. We then consider several $p$-percentiles $\mathcal{D}_p(A)$ and $\mathcal{D}_p(P)$ of the distributions $\mathcal{D}(A)$ and $\mathcal{D}(P)$ using their empirical Cumulative Distribution Function (CDF) estimates. We say that $\mathcal{D}(A) \gg \mathcal{D}(P)$ if:

$$\mathcal{D}_p(A) > \mathcal{D}_p(P) \text{ for all } p \in [0.5, 0.95] \tag{5}$$

We choose the above range for $p$ since the lower percentiles may not be affected by queueing delays, and they can be close to zero for both flows. The percentiles $p$ are determined from the empirical CDF of $\mathcal{D}(A)$, and for each such $p$ we use nearest-neighbor interpolation to find $\mathcal{D}_p(P)$. We also give the user a measure of the *delay difference* between the two flows as $\mathcal{D}_{0.75}(A) - \mathcal{D}_{0.75}(P)$.

Note that there could be a case where the ISP prioritizes the $A$ flow over the $P$ flow. We run our inequality test by swapping $A$ and $P$ as inputs to detect if the $P$ delays are higher than the $A$ delays.

### 3.4.1 Distinguishing WFQ from SP

After we have detected a discriminatory scheduler that treats $A$ as low priority and $P$ as high priority, we examine whether that scheduler is SP or a WFQ variant. The intuition behind this method is the following. Consider a two-class discriminating link that services high priority and low priority packets. A packet experiences propagation, transmission, and potentially queueing delays at that link. We distinguish between the queueing delay due to a packet that is currently being transmitted from queueing delays due to other backlogged packets; the former is referred to as the *non-preemption delay* and it can affect all packets irrespective of their priority. The basic idea of the proposed method is that an arriving $P$ packet at an SP scheduler *may* experience non-preemption delay if another packet is currently being transmitted, but it will *never* experience queueing delays due to backlogged low priority packets. In the WFQ scheduler, on the other hand, an arriving $P$

packet may also experience queueing delays due to backlogged low priority packets. At the same time however, we need to consider that $P$ packets may experience queueing delays at both schedulers when they are backlogged behind other high priority packets. So, if we had a way to identify those $P$ packets that were not backlogged behind other high priority packets, we expect that their queueing delays would be bounded by the non-preemption delay at the link, while those delays may be much higher in the case of a WFQ scheduler.

In practice, we have no way to know which $P$ packets are backlogged behind high priority packets from other sources. We can identify, however, bursts of $P$ packets sent by DiffProbe. From such bursts, we only consider the first $P$ packet because that packet is more likely to *not* be backlogged behind other high priority packets.

Further, we limit our sample to those $P$ packets that were *received* shortly after $A$ packets. The reason is that those $P$ packets are more likely to arrive at the link during a busy period. Otherwise, if a $P$ packet arrives at an idle scheduler, it will experience zero queueing delay independent of the scheduler type. Such packets would not help us detect the scheduler's type.

After we have identified the subset of $P$ packets as previously described, we examine the variability of their delay distribution. The basic idea is that, with SP scheduling, the selected $P$ packets will have very small delay variability. In the discriminatory link, their queueing delays will be practically zero (at most the non-preemption delay, which is the MTU divided by the capacity of that link). In practice, of course, we need to use a larger threshold because of possible queueing delays at other links. The selection of that threshold is not critical, however, because the corresponding delay variability with a WFQ scheduler will be significantly higher. Figure 6 shows the distribution of one-way delays of the selected $P$ packets for simulated SP and WFQ schedulers. Notice that, with SP, the delay variability of the selected $P$ packets is practically zero; on the other hand, WFQ leads to significant delay variability. In DiffProbe, we measure the delay variation of the selected $P$ packets as the 95th-5th percentile difference. Denote the $p$th percentile delay of the selected $P$ packets as $\check{\mathcal{D}}_p(P)$. We declare that the scheduler is SP if:

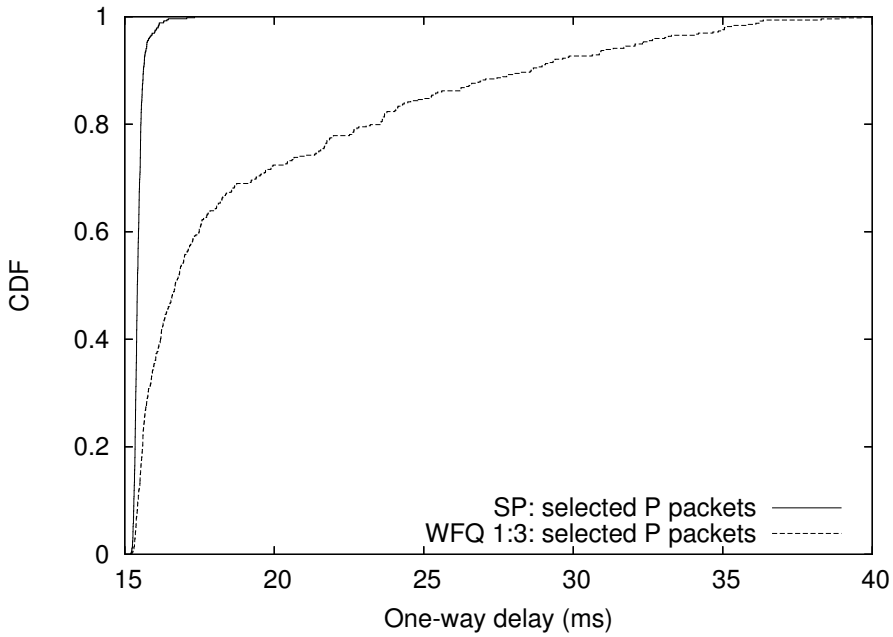



**Figure 6:** Identifying SP and WFQ: distribution of selected $P$ packet delays. The difference in distributions is due to non-preemption delay in the case of SP and delay backlogs in the case of WFQ-like schedulers.

$$\check{\mathcal{D}}_{0.95}(P) - \check{\mathcal{D}}_{0.05}(P) < \kappa$$

The threshold $\kappa$ is estimated as the MTU packet size divided by the capacity of the access link (given that we do not know the capacity of the discriminatory link). In Section 3.7 we show that the accuracy of this detection method does not depend critically on $\kappa$.

### *3.5 Loss Discrimination Detection*

Discriminatory buffer managers drop packets, already backlogged or arriving, considering the class of those packets. This is different than DropTail or RED, which do not consider the class of the packets they drop. Consider the WRED discriminatory buffer manager in the case of two traffic classes $H$ and $L$. In practice the ISP would configure the queue thresholds in order to differentiate between traffic classes [3], such that:

$$\hat{q}_{\min}(H) \gg \hat{q}_{\min}(L)$$

Hence, after the average queue length has exceeded the lower threshold for the low priority class, $\bar{q} > \hat{q}_{\min}(L)$, packets of that class will be dropped with a higher probability than

packets of the high priority class. In the case of the Drop-Longest-Queue policy, backlogged packets from the longest queue are dropped when needed. Thus, if the low priority class has a longer queue, due to a lower service priority or rate, it will also tend to see a higher loss rate.

We detect *loss discrimination* during the $LIP$ period as follows. We estimate the loss rates of $A$ and $P$ flows (as fraction of packets lost) during the $LIP$ period; denote them as $\ell(A)$ and $\ell(P)$ respectively. We declare that there is loss discrimination when the loss rate of $A$ is much higher than that of $P$:

$$\ell(A) \gg \ell(P) \tag{6}$$

Conversely, for a non-discriminatory buffer manager, we would have the equality:

$$\ell(A) \approx \ell(P) \tag{7}$$

We describe the specific statistical tests to perform loss rate comparisons next.

**Equality of Sampled Loss Rates:** We first pre-process the sender-side sequence number timeseries to create a *paired* sample of $A$ and $P$ packets sent almost simultaneously, as done in the pairing procedure for the delay timeseries described in Section 3.4. Consider the measured loss rates $\ell_A$ and $\ell_P$ in the paired samples of the $A$ and $P$ flow, respectively. Let us denote the number of samples sent by the two flows as $s_A$ and $s_P$ respectively.

We use a two-tailed version of the well-known *two-proportion z-test* for equal population proportions of two independent samples [109]. Our null hypothesis is that the $A$ and $P$ flows sample the same loss process, while the alternate hypothesis is that the two flows sample a different loss process (we do not assume that the P flow will necessarily see a lower loss rate in the alternate hypothesis, as it may be that the ISP is discriminating in favor of the $A$ flow). The two-proportion test uses the $z$-score statistic:

$$z = \frac{\ell_A - \ell_P}{\sqrt{l\,(1-l)\left[\frac{1}{s_A} + \frac{1}{s_P}\right]}}, l = \frac{\ell_A s_A + \ell_P s_P}{s_A + s_P}$$

The $z$-score has an asymptotic standard Normal ($\mathcal{N}(0,1)$) distribution when null hypothesis is true. The $p$-value is computed as:

$$p = Prob\left[|z| < \mathcal{N}(0,1)\right]$$

The test rejects null hypothesis with this $p$-value if $p < 0.05$.

A rule of thumb for Normal approximation of $z$ statistic in the two-proportion test is that $s_A$ and $s_P$ both include at least 10 dropped packets. We diagnose loss discrimination as *unidentifiable* if this is not the case. To help with the identifiability objective, we make the $LIP$ duration sufficiently large (always maintaining $\lambda_A(t) + g(t)\lambda_P(t) \approx C_a\left(1-\epsilon\right)$), so that we can observe a sufficiently large number of packet losses in this period.

We determine the $LIP$ duration as follows. We observe the loss rate of the $A$ flow during the $BLP$; call it $\ell_A^{BLP}$. Suppose that the rate of $A$ packets sent during $BLP$ is $\lambda_A^{BLP}$ (packets per unit time). The minimum $LIP$ duration is then chosen so that the expected number of $A$ losses is at least 10. The $LIP$ duration $\Delta_{LIP}$ is thus:

$$\Delta_{LIP} \geq \frac{10}{\ell_A^{BLP}\lambda_A^{BLP}}$$

For example, using a G.711 (voice) $A$ flow that sends about 33 packets per second, and with a 1% loss rate during the $BLP$ period, we would determine that the $LIP$ should be at least 30s long.

### *3.6 Implementation and Test Runs*

We have implemented *DiffProbe* in a completely automated tool. The current version is about 7,500 lines of C code and it has been tested on Linux platforms so far. In this section we describe the tool, and we also show some test runs at large access ISPs.

DiffProbe consists of two endpoints, the client ($\mathcal{CLI}$), and the server ($\mathcal{SRV}$). $\mathcal{CLI}$ is run by a user connected to the target ISP. DiffProbe operates in two phases. In the first phase, $\mathcal{CLI}$ sends timestamped probing streams to $\mathcal{SRV}$. For each probing structure, $\mathcal{SRV}$ collects one-way delay timeseries of $A$ and $P$ flows. In the second phase, the roles of $\mathcal{CLI}$ and $\mathcal{SRV}$ are reversed.

### 3.6.1 Capacity Estimation

Before probing, we make a rough estimate of the upstream and downstream capacities at the end-to-end path using packet trains of back-to-back packets over UDP. We use this estimate to decide the $LIP$ probing rate, since the end-to-end capacity is the highest rate we can use for probing without overflowing the bottleneck queue (a high probing rate increases the likelihood of inducing a backlog in the normal traffic class during $LIP$). More precisely, we send $K$ packet trains of length $L$ packets, each of size $S$. At the receiver, and for each train, we measure the dispersion $\Delta$ and estimate the path capacity as: $C_a = \frac{(L-1)S}{\Delta}$. Finally, we take the median of the $K$ trains. $C_a$ is an estimate of the capacity of the *narrow link* between $\mathcal{CLI}$ and $\mathcal{SRV}$. For residential ISPs, this link is most likely the home access link in both upstream and downstream directions. In the current implementation, we set $K = 10$, $L = 50$, and $S = 1450B$, and send the trains over a port which is not likely to be classified low-priority by an ISP.

### 3.6.2 Probing

Each probing session consists of the $BLP$ and $LIP$ probing periods. After we probe the upstream direction, we repeat that sequence in the downstream direction. Each probing packet of the $A$ flow is replayed according to a pre-recorded application flow trace. We maintain the same port number(s), transport protocol, packet sizes, inter-packet gaps and payload as in the trace file while replaying the $A$ flow. We overwrite the last four bytes of the payload with the sender timestamp for one-way delay measurement.

We create $P$ flow using the last sent $A$ packet size. The payload is randomized (excluding the sender timestamp) and we use port numbers that are not likely to be classified as low priority. The user can choose between two UDP Skype voice traces (taken from prior work on Skype classification [31]) and two UDP Vonage voice traces (each 10min. long) to test for discrimination. In our implementation, we use a single probing session. Unless otherwise mentioned, the tool uses the following parameters: $LIP$ and $BLP$ durations 30s, $LIP$ probing rate is estimated with $\epsilon = 0.1$.

An example output of the DiffProbe client is shown below.

```
DiffProbe alpha release. April 2009.

Using tracefile skype-upstream.pcap.
Using device: eth0.
sleep time resolution: 1.99 ms.
Connected to server 123.231.123.231.

Estimating capacity:
 Upstream: 10800.39 Kbps.
 Downstream: 37127.07 Kbps.

Checking for traffic shapers:
 Upstream: Burst size: 5402 KB; Shaping rate: 1008.00 Kbps.
 Downstream: No shaper detected.

*** Upstream ***
 ......
 sending measurement data to server..done.
 Analyzing measurements.

 Results:
 --------
 Delay discrimination:
 Delay discrimination detected.
 Application traffic classified low priority: delay between flows: 6.24 ms.

 Loss discrimination:
 No loss discrimination detected.

*** Downstream ***
 ......
```

```
sending measurement data to server..done.
Analyzing measurements.

Results:
--------
Delay discrimination:
No delay discrimination detected.

Loss discrimination:
Not detectable.

For more information, visit: http://www.cc.gatech.edu/~partha/diffprobe
```

### 3.6.3 Test runs

We have run DiffProbe at some large residential ISPs. Note that it is not possible to know the ground truth in such experiments. We can, however, say that there is no discrimination between the $A$ and $P$ flows if no significant delay differences have been *perceived* by the user, or if the KL-test reports a high $p$-value. All experiments were done in April and July of 2009. The $LIP$ duration in these experiments is 10s while the $BLP$ duration is 5s.

Table 1 shows access ISP locations and the $p$-values from our KL-test for delay discrimination. We test for discrimination against the four Skype and Vonage traces once for each ISP. The table shows that we do not detect payload and/or port based discrimination in these ISPs (all tests were identified as "detectable")[5]. An exception is the case for ISP-1: 1 out of 4 downstream trials and all upstream trials showed discrimination. Upon visual inspection, we noticed that the two flows follow different paths in the ISP-1 network, while one of the paths introduces higher queueing delays than the other. The traceroute tool can be used to detect such routing differences between the $A$ and $P$ flows (DiffProbe does not test for this condition).

[5]We were not able to collect data for two downstream cases due to NAT issues.

**Table 1:** Access ISP test runs: $p$-values across Skype and Vonage tests. Some runs on ISP-1 showed routing differences between the two flows, which explains the low $p$-values.

| ISP | Upstream | Downstream |
|---|---|---|
| ISP-1 (US) | 0.01-0.04 | 0.0-1.0 |
| ISP-2 (Switzerland) | 1.0 | 0.28-1.0 |
| ISP-3 (US) | 0.54-1.0 | 1.0 |
| ISP-4 (US) | 0.87-1.0 | 0.17-1.0 |
| ISP-5 (Belgium) | 1.0 | - |
| ISP-6 (Norway) | 0.25-0.98 | - |
| ISP-7 (US) | 0.82 | 0.98 |

### *3.7 Simulation/Emulation Evaluation*

In this section we first evaluate the accuracy of Differential Probing using simulations, and then show some realistic emulation experiments.

We evaluate the accuracy of the discrimination detection methods using NS2 simulations. The simulation topology is as follows. The discriminating link capacity is 100Mbps. The $A$ and $P$ flows are generated from a server and they are sent to a residential client. All servers and residential users have access links of 1Gbps and 10Mbps, respectively. The capacity of the discriminatory link is 100Mbps. We simulate 200 residential clients generating closed-loop ("interactive") TCP sessions by downloading Pareto-sized heavy-tailed content from 200 randomly chosen servers. These well-provisioned servers are connected to the discriminating link through links of different propagation delays. We provision all link buffers according to the bandwidth-delay product. The setup for reverse-direction cross traffic is similar. We perform at least 96 trials for each utilization point of the discriminating link, so that we have an error margin of 2% at 95% confidence assuming a prior proportion of 0.9. A utilization of $U$% encompasses all trials in the interval $(U-5, U+5]$%.

Unless otherwise mentioned, we use the following parameters. Cross-traffic is classified at the access links on the basis of the generating source as low or high priority with probability 0.5. We use a Skype iSAC packet trace as the $A$ flow. We use $\epsilon = 0.1$ to adjust the $LIP$ rate; the $LIP$ and $BLP$ durations are 30s long. We consider three weight ratios of WFQ, 1:1.5, 1:3 and 1:10. We will see that the first weight ratio is small and performs similar to FCFS-DT, while the third case performs similar to SP; the second ratio is realistic,

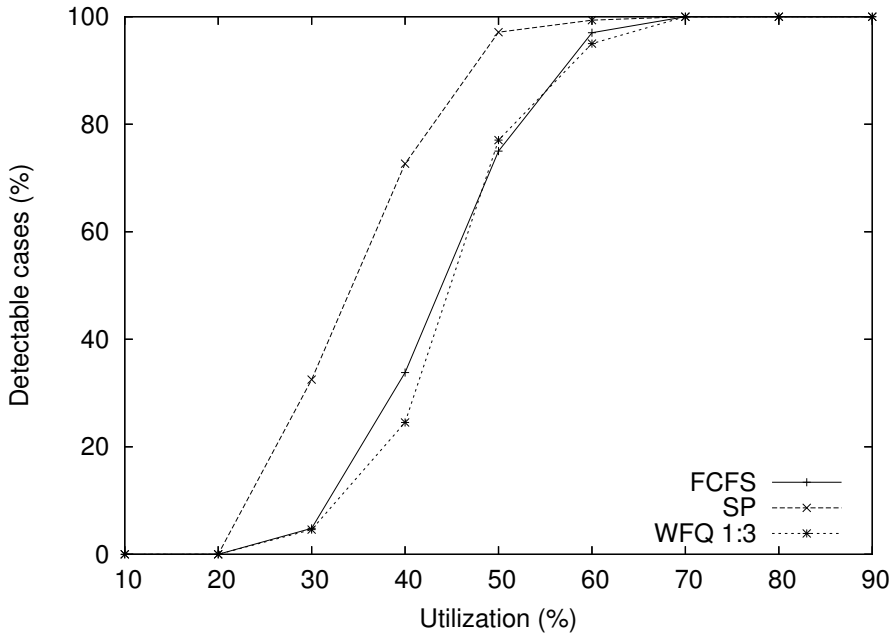



**Figure 7:** Fraction of detectable trials: delay discrimination. Low utilization in the discriminating link may not create significant backlogs in the normal traffic queue to perceive discrimination.

given that half of cross-traffic utilization comes from high priority flows. We start with an evaluation of delay and loss discrimination detection accuracy.

### 3.7.1 Delay Discrimination

In this subsection, we evaluate detection accuracy of FCFS, SP, and WFQ schedulers.

**Detectability:** A detection threshold factor (the ratio $\mathcal{D}_{0.9}(P)/\mathcal{D}_{0.5}^{BLP}(P)$ in eq. (1) of 1.3 is sufficient to get detection accuracy higher than 90%. Using this threshold, we show the fraction of detectable trials at each utilization range in Figure 7. Note that at low utilization ($\leq 40\%$) we are not able to detect the majority of trials. This also implies that there is no *user-perceived delay discrimination* at low utilization in the discriminating link, because it is then unlikely that the user traffic will observe any queueing at the discriminating link. We also found that without this detectability condition, we only get 90%+ detection accuracy when the utilization exceeds 50%.

**Detection accuracy:** Figure 8 shows discrimination and no-discrimination detection accuracy with utilization for FCFS, SP, and WFQ (weight ratio 1:3). We get high detection accuracy at all utilizations of the discriminating link. Note that false positives would

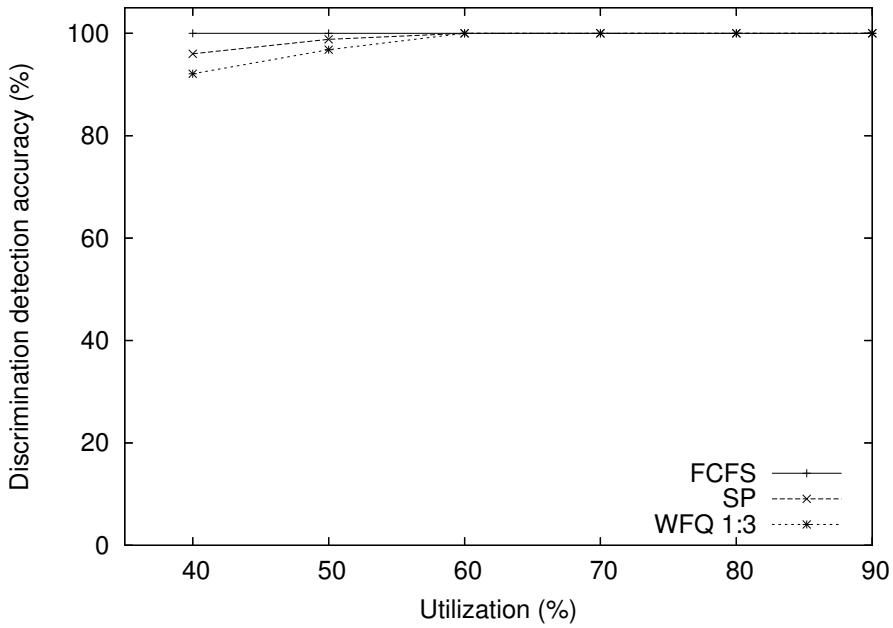



**Figure 8:** Delay discrimination detection accuracy is close to 100%.

correspond to inaccurate detection of FCFS - but we see that there are no such cases.

**WFQ weight ratio:** The effect of the WFQ weight ratio on delay discrimination detection accuracy is shown in Figure 9. The weight ratio 1:10 performs similar to SP and leads to high detection accuracy, while the ratio 1:1.5 performs close to FCFS (no significant delay discrimination) and hence it leads to low detection accuracy.

**WFQ and SP:** Figure 10 shows the accuracy of distinguishing the SP and WFQ schedulers for a threshold $\kappa = 0.7$ms. We see that low utilization leads to low accuracy; this is expected, since, although the difference in the delay distributions of $A$ and $P$ flows is large enough to show the presence of discrimination, WFQ and SP service $P$ packets quite similarly. A large WFQ weight ratio (1:10) makes this scheduler similar to SP, and so the detection accuracy is lower than for lower weight ratios. We also found that for more reasonable weight ratios (e.g. 1:3), the detection of SP and WFQ can be done accurately with a wide selection of $\kappa$ values.

### 3.7.2 Loss Discrimination

In this subsection, we evaluate detection accuracy of DropTail, WRED, and drop from longest queue (Drop-Longest-Queue) buffer managers. We use the DT buffer manager in

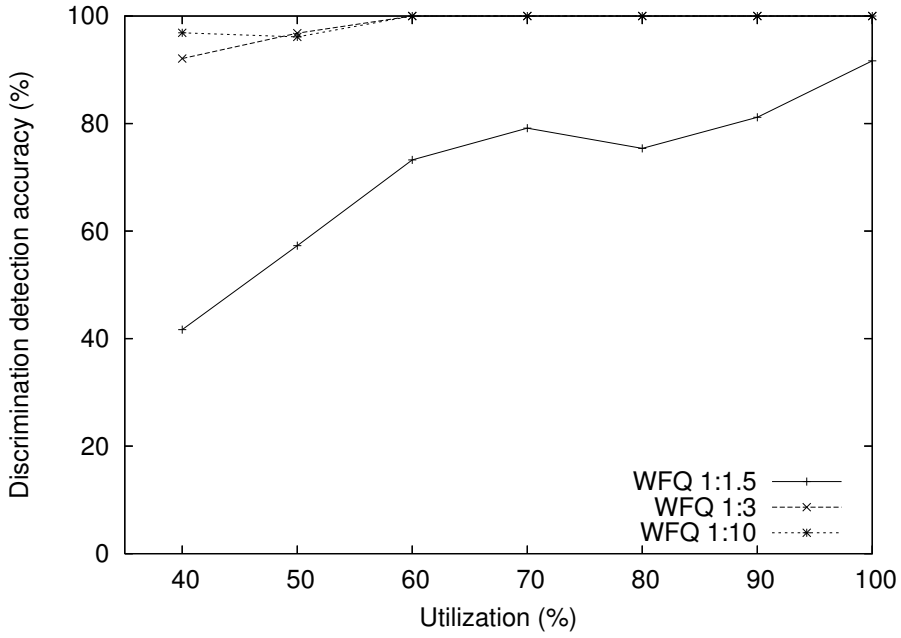



**Figure 9:** Effect of WFQ low-to-normal traffic weight ratio on delay discrimination detection accuracy. A weight ratio close to one will perform similar to FCFS, while a weight ratio close to zero will perform similar to SP in terms of delays.

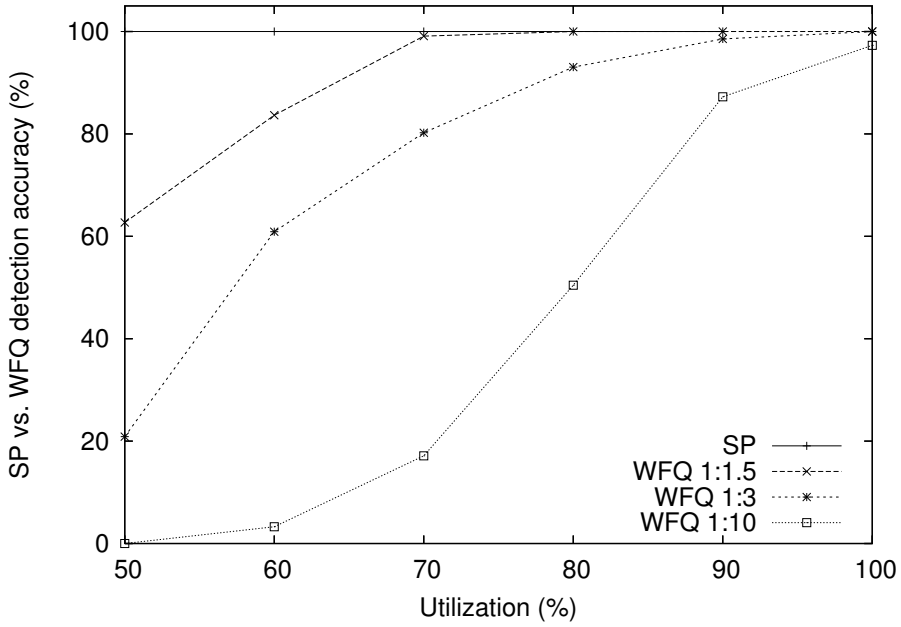



**Figure 10:** Distinguishing SP from WFQ: effect of utilization and WFQ weight ratio ($\kappa$=0.7ms). Low utilization cases may not create sufficient backlogs in the normal traffic queue to observe queueing delays.

**Table 2:** Loss discrimination detection accuracy.

| Buffer mgr. | DropTail | | Drop from longest (WFQ) | | |
|---|---|---|---|---|---|
| | FCFS | SP | 1:1.5 | 1:3 | 1:10 |
| Accuracy | 100% | 98.57% | 33.75% | 98.75% | 100% |

FCFS and SP, while for our WFQ implementation, we use a discriminatory buffer manager that drops packets from the longest queue. Note that we get false positives when the accuracy is less than 100% in the case of DropTail (none in FCFS-DT and less than 2% in SP-DT).

**DT and Drop-Longest-Queue:** Table 2 shows the accuracy of DropTail and Drop-Longest-Queue buffer managers for detectable trials. We see a high detection accuracy for both discriminatory and non-discriminatory buffer managers. The table shows Drop-Longest-Queue detection for three different WFQ weight ratios. A WFQ ratio of 1:1.5 is close to DropTail in terms of loss discrimination, and yields low accuracy.

**WRED accuracy:** We choose the following WRED parameters for $L$ and $H$ traffic: $\hat{q}_{\max}(H) = \hat{q}_{\max}(L) = 500$ (buffer size of discriminating link in packets; avg. packet size is 1000B); $\hat{q}_{\min}(L) = \hat{q}_{\max}(L)/2$; $\hat{q}_{\min}(H) = \hat{q}_{\min}(L)\,[1+f]$; $p_H = 0.15$; $p_L = 0.20$. We vary the parameter $f$, which quantifies the difference between the $H$ and $L$ classes. Figure 11 shows the effect of $f$ on the detection accuracy. At low values of $f$ ($\leq 0.3$), the detection accuracy is low since the loss discrimination between the two priorities is not significant.

### 3.7.3 Discrimination Emulations

In this subsection, we evaluate the tool in a realistic emulation setup. Our emulated discrimination scenario is as follows. Our testbed is connected to a residential cable ISP in Atlanta, GA (US). The DiffProbe client runs inside the residence, and the server is hosted in the Georgia Tech campus. We emulate the discriminating link on a multihomed Linux router that connects to the cable modem, and serves the client machine connected through a Fast Ethernet interface. Note that the narrow link in this case is between the cable modem and CMTS, which at the time of experiments was a 10Mbps upstream and 17Mbps downstream DOCSIS link. Cross traffic is generated using two Pareto sources (mean gap

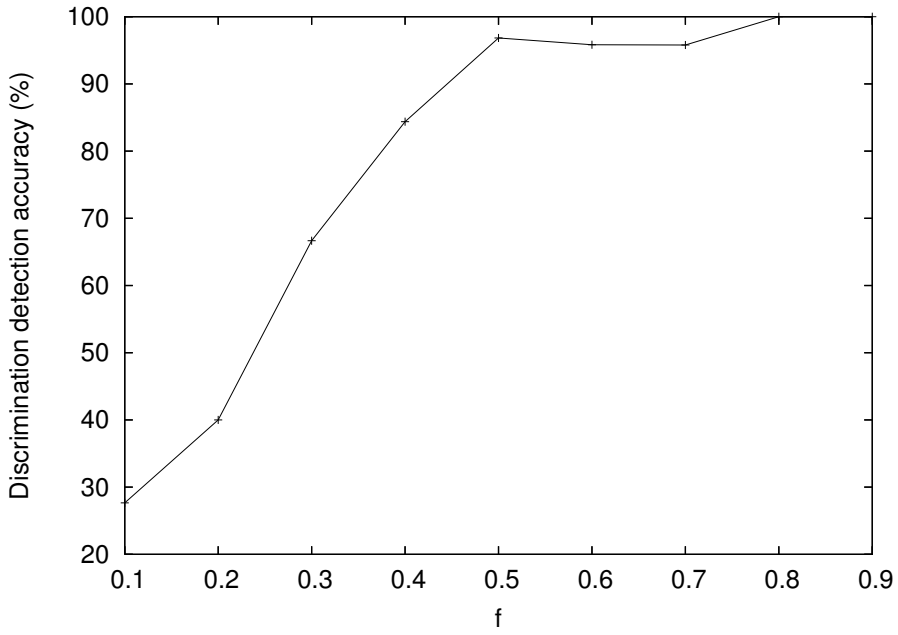



**Figure 11:** WRED: effect of $f$ on detection accuracy. The parameter $f$ quantifies the difference in WRED configuration between the $H$ and $L$ traffic classes; low values of $f$ will result in DropTail-like buffer management.

100ms and shape 1.5), with a packet size of 600B, over ICMP. Our DiffProbe tests on this ISP (without emulated discrimination) show that the ISP does not discriminate against the $A$ flow. We show results for at least 10 trials for each experiment.

**FCFS:** We start with the no-discrimination case. We pair $A$ and $P$ samples when they are in 1ms send-time proximity. DiffProbe does not reject the null hypothesis of equal $A$ and $P$ distributions, with $p$-values in $[0.86, 0.92]$. Note that we reject the null hypothesis if $p < 0.05$.

**SP and WRR:** We use the the Linux Advanced Routing and Traffic Control framework [12] to implement scheduling and buffer management on a 2.6.22 kernel. We classify traffic using protocol, port numbers and destination IP address. The classifier is built out of *tc* filter rules. One of the cross traffic sources and the $P$ flow are classified as high priority, while the other source and the $A$ flow are classified as low priority.

We implement SP using LARTC's *prio* scheduler. We also limit the service rate to 1Mbps using a token bucket of small depth. The queue size for each class was 50KB. DiffProbe rejects the null hypothesis of equal distributions with $p = 0$ for all trials.

We also implement Weighted Round Robin (WRR) scheduling using LARTC's `CBQ` scheduler with a weight ratio of 1:3 (link capacity of 1Mbps). The queue size for each class was 50KB. Note that although WRR is implemented in most network devices, it does not account for packet sizes during scheduling and hence it is not as fair in weighted rate allocation as WFQ or DRR. DiffProbe rejects the null hypothesis of equal distributions with $p = 0$ for all trials.

**Loss discrimination:** The SP and WRR implementations use separate physical queues for each priority, and incoming packets are dropped using the Drop-Longest-Queue policy. For illustration, we consider one WRR 1:3 trial, in which the estimated loss rates were 1.68% for the $A$ flow and 0.15% for the $P$ flow. DiffProbe rejects the null hypothesis of equal loss rates with $p = 0$.

### *3.8 Summary*

In this chapter, we presented Differential Probing, a general method for the detection of delay and loss discrimination. We focused on the accurate detection of two packet scheduling mechanisms, SP and WFQ, as well as on the detection of loss discrimination, in the case of two classes of service. The simulation and emulation experiments showed that the detection methods are accurate, as long as our probing traffic can create some queueing at the discriminatory link and when the ISP-configured delay and/or loss differentiation is non-negligible (in other words, discrimination is *perceivable* by user traffic). DiffProbe can also distinguish between SP and WFQ, as long as the weight ratio in the latter is not so high that would make WFQ behave similarly to SP. Our test runs at some major access ISPs show that, at least so far (2009), there is no delay and loss discrimination against Skype and Vonage traffic at those ISPs.

The DiffProbe home page is at `http://www.netinfer.net/diffprobe/`. Parts of this work appeared in a prior publication [62].

# CHAPTER IV

# INFERRING TRAFFIC SHAPERS AND POLICERS

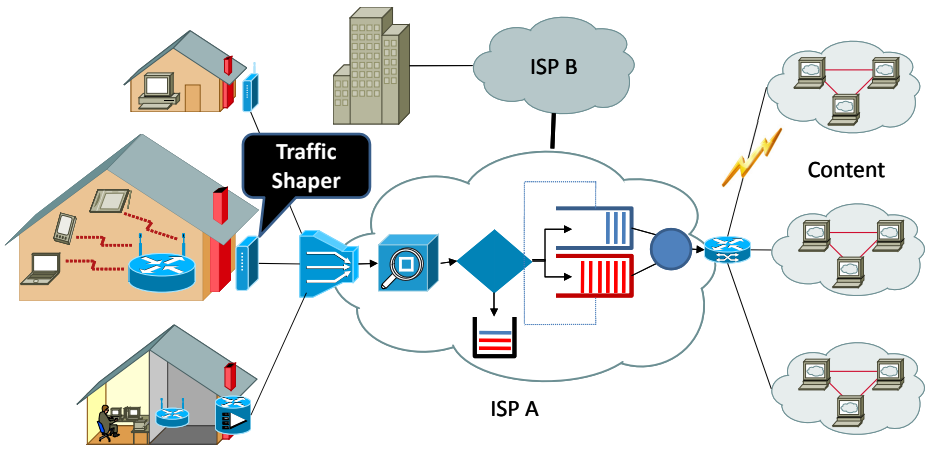



**Figure 12:** Internet model showing a traffic shaper deployment by an ISP.

## *4.1 Introduction*

The increasing penetration of broadband access technologies, such as DSL, DOCSIS and WiMAX, provides users with a wide range of upstream and downstream service rates. Broadband users need to know whether they actually get the service rates they pay for. On the other hand, ISPs now have an extensive toolbox of traffic management mechanisms they can apply to their customers' traffic: application classifiers, schedulers, active queue managers etc. In this chapter we focus on a class of such mechanisms referred to as *traffic shapers* or *traffic policers*.[1]

A traffic shaper is a single-input single-output packet forwarding module that behaves as follows. Consider a link of capacity $C$ bps, associated with a "token bucket" of size $\sigma$ tokens. Whenever the bucket is not full, tokens are generated at a rate $\rho$ tokens per second, with $\rho < C$. The link can transmit an arriving packet of size $L$ bits only if the token

[1]When it is not important to distinguish between shaping and policing, we will simply refer to such mechanisms as "traffic shapers" or just "shapers".

bucket has at least $L$ tokens - upon the transmission of the packet, the shaper consumes $L$ tokens from the bucket. So, if we start with a full token bucket of size $\sigma$ tokens, and with a large burst of packets of size $L$ bits each (suppose that $\sigma$ is an integer multiple of $L$ for simplicity), the link will be able to transmit $k$ of those packets at the rate of the capacity $C$, with $k = \frac{\sigma/L}{1-\rho/C}$. After those $k$ packets, the link will start transmitting packets at the token generation rate $\rho$. Usually $\rho$ is referred to as the "shaping rate", the capacity $C$ is also referred to as the "peak rate", and $\sigma$ is referred to as the "maximum burst size". Another way to describe a traffic shaper is by specifying that the *maximum number of bits* that can be transmitted in any interval of duration $\tau$, starting with a full token bucket, is:

$$\hat{A}(\tau) = \min\{L + C\tau, \sigma + \rho\tau\}$$

The difference between a traffic shaper and a traffic policer is that the former has a buffer to hold packets that arrive when the token bucket is empty [8,116]. A policer simply drops such "non-conforming" packets. In other words, a shaper delays packets that exceed the traffic shaping profile ($\sigma$, $\rho$), while a policer drops them.[2] Policers can cause excessive packet losses and so shapers are more common in practice - we focus on the latter in the rest of the chapter.

Why would a residential ISP deploy traffic shaping? First, to allow a user to exceed the service rate that he/she has paid for, for a limited burst size. In that case the user pays for $\rho$ bps, with the additional service capacity $(C - \rho)$ marketed as a free service enhancement. This is, for instance, how Comcast advertises their PowerBoost traffic shaping mechanism [7]. Second, an ISP may want to limit the service rate provided to the aggregate traffic produced or consumed by a customer, or to limit the service rate consumed by a certain application (e.g. BitTorrent). This form of shaping is relevant to the *network neutrality* debate. Third, certain ISPs prefer to describe their service rates as upper bounds for what the user will actually get, e.g., a downstream rate of *at most* 6Mbps. In that case, a shaper can be used to enforce the upper bound of the service rate.

The contribution of this chapter is threefold. First, we develop an *active end-to-end*

[2] A shaper also drops packets once its droptail buffer is full.

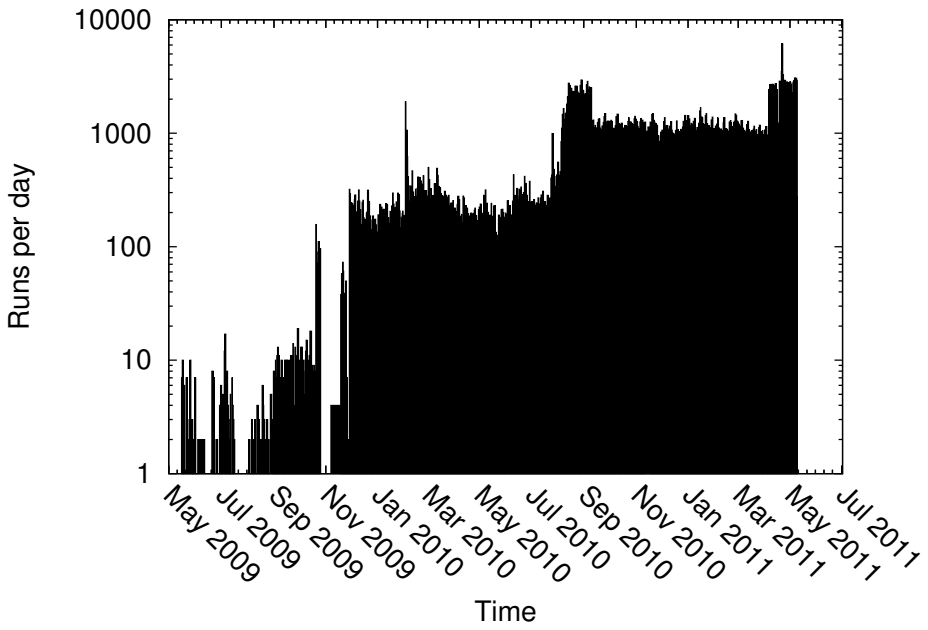



**Figure 13:** ShaperProbe: volume of runs. The gaps in time show downtime due to tool enhancements.

*detection mechanism*, referred to as *ShaperProbe*, that can infer whether a particular path is subject to traffic shaping, and in that case, estimate the shaper characteristics $C$, $\rho$ and $\sigma$. Second, we analyze results from a large-scale deployment of ShaperProbe on M-Lab [16] between May 2009 and May 2011, detecting traffic shaping in several major ISPs. Our deployment received about one million runs over the two year period from more than 5,700 ISPs; we have seen 2,000-3,000 runs per day (see Figure 13). Figure 14 shows the locations of ShaperProbe clients. All data collected through ShaperProbe runs is publicly available through M-Lab [10].[3] Third, we develop passive detection and estimation methods for traffic shaping. These methods work on a live or an offline TCP trace, and can also be used to detect application-specific traffic shaping deployments.

Traffic shaping detection and estimation methods can be used in different ways: *as a library (API)*; and as a *service* that enables users/administrators to detect or verify their SLAs/shaping configurations. In this work, we focus on the latter. The ShaperProbe client is a download-and-click userspace binary (no superuser privileges or installation needed) for

[3]We log per-packet send and receive timestamps and sequence numbers for all probing phases, and client IP address and server timestamp (UTC) for each run.

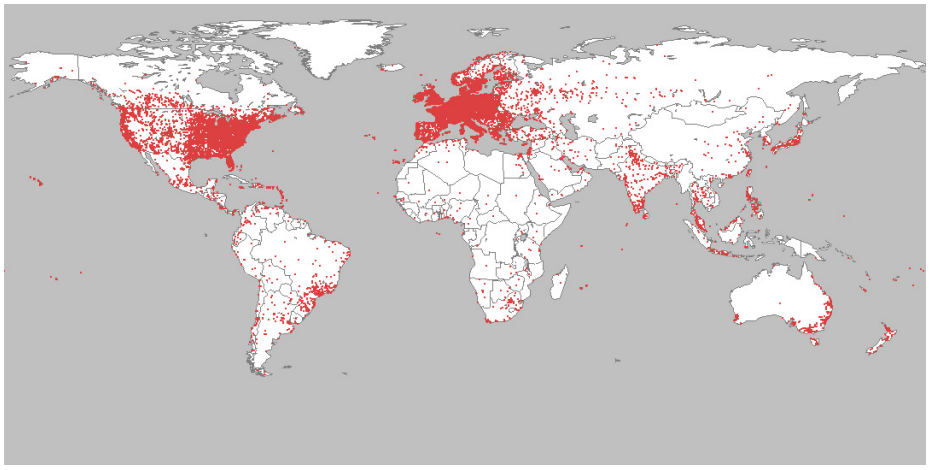


**Figure 14:** ShaperProbe: location of clients.

32/64-bit Windows, Linux, OS X and BSD; a plugin is also available for the *Vuze* BitTorrent client. The non-UI logic is about 6000 lines of open source native code.

There are several challenges that one needs to tackle when designing an active measurement service that can scale to thousands of users per day, including *accuracy*, *usability* and *non-intrusiveness*. Even though these challenges are often viewed as not significant, at least from the research perspective, they have greatly influenced several design choices and parameter values in ShaperProbe.

**Chapter outline:** We describe the active detection method in Section 4.2, and implementation and deployment of ShaperProbe in Section 4.3. We look at ShaperProbe data using case studies of four ISPs in Section 4.4. Section 4.6 describes the passive detection and estimation methods.

## *4.2 Active Probing Method*

The active probing method is an end-to-end process in which a sender $\mathcal{SND}$ sends packets on the network path to the receiver $\mathcal{RCV}$. We detect the presence of traffic shaping in the path $\mathcal{SND} \rightarrow \mathcal{RCV}$ at $\mathcal{RCV}$.

Suppose that the narrow link's capacity on the path is $C$, and that the sender probes at a *constant bit rate* $R_s = C$. The ShaperProbe capacity estimation process is described in Section 4.3. The receiver $\mathcal{RCV}$ records the received rate timeseries $R_r(t)$. We compute $R_r(t)$ by discretizing time into fixed size non-overlapping intervals of size $\Delta$. For simplicity,

assume that the probing starts at $t = 0$, and that intervals are numbered as integers $i \geq 1$. The $i$'th interval includes all packets received in the interval $[(i-1)\,\Delta, i\Delta)$, where packet timestamps are taken at $\mathcal{RCV}$ upon receipt of each packet. The discretized received rate timeseries $R_r(i)$ is estimated as the total bytes received in interval $i$ divided by $\Delta$. Note that this estimator of $R_r(t)$ can result in an error of up to $\epsilon = \pm S/\Delta$ where $S$ is the MTU packet size. By choosing a reasonably large $\Delta$, we can reduce the magnitude of $\epsilon$ relative to the true received rate.

In the presence of a token bucket traffic shaper (or policer) on $\mathcal{SND} \rightarrow \mathcal{RCV}$, there exists a value of $i > 1$ at which the received rate timeseries $R_r(i)$ undergoes a *level shift to a lower value*. Our goal is to detect the presence of a level shift, and estimate the token bucket parameters using $R_r(i)$.

### 4.2.1 Detection

We want to detect a level shift in $R_r$ in *real-time*, i.e., as we compute the received rate for each new interval. Note that the receiver $\mathcal{RCV}$ is also receiving new packets during the level-shift detection process, and so our method should be fast and computationally light-weight to avoid the introduction of timestamping jitter. The detection method is rather simple and relies on nonparametric rank statistics [54] of $R_r$ so that it is robust to outliers.

We compute ranks online (every time we estimate $R_r$). Suppose that we have estimated $n$ values of $R_r$ so far. At the start of the new interval $n+1$ (i.e., after the receipt of the first packet in that interval), we compute $R_r(n)$ and update the ranks $r(i)$ of $R_r(i)$ for $i = 1 \ldots n$. We identify $\tau$ as the *start of level shift* if it is the first interval index that satisfies the following three conditions.

First, all ranks at the left of $\tau$ are *equal to or higher* than all ranks at the right of $\tau$:

$$\min_{i=1\ldots\tau-1} r(i) \geq \max_{j=\tau+1\ldots n} r(j) \tag{8}$$

Second, we have observed a minimum time duration before and after the current rate measurement:

$$n_L < \tau < n - n_R \tag{9}$$

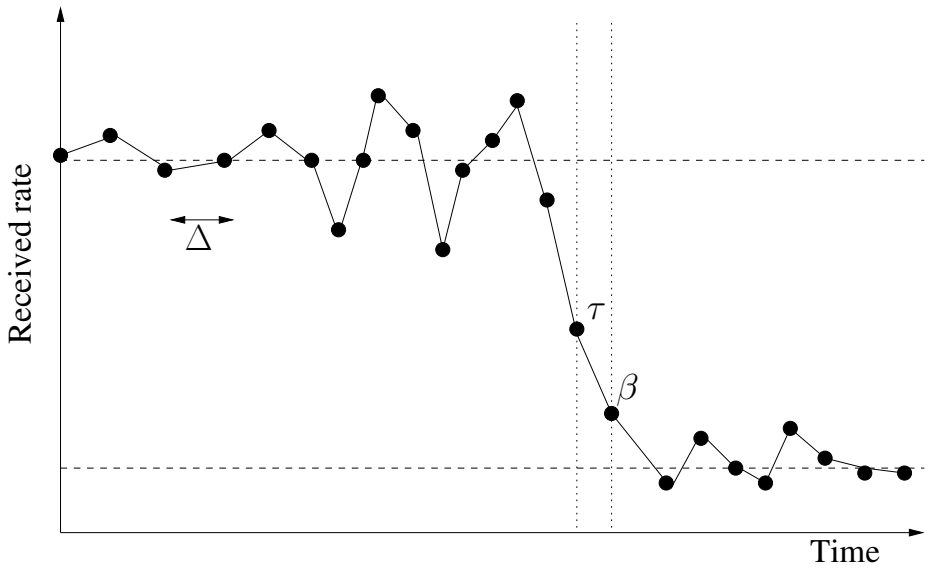



**Figure 15:** Active probing: The receiver-estimated rate timeseries showing level shift detection parameters.

The value of $n_L$ is chosen based on empirical observations of typical burst durations in ISP deployments; more specifically, it is chosen to be large, but smaller than typical configurations of burst durations in ISPs (see Figure 16) so that we minimize false negative detections. The value of $n_R$ is a sanity check to ensure that the drop in rate is not just a temporary variation (e.g., due to cross traffic).

Third, we require that there is a *significant drop* in the median rate at point $\tau$:

$$\underset{i=1\ldots\tau}{\tilde{R}_r(i)} > \gamma \underset{j=\tau\ldots n}{\tilde{R}_r(j)} \tag{10}$$

where $\tilde{R}_r$ denotes the median, and $\gamma$ is a suitable threshold. We select $\gamma$ based on empirical observations of ISP capacities and shaping rates in practice (see Section 4.2.3).

Similarly, we detect the *end of a level shift* index (or time) $\beta$ such that $\beta \geq \tau$ and $\beta$ is the *last* point which satisfies the rate condition in Equation 8. Figure 15 illustrates the two level shift indices.

### 4.2.2 Estimation

After the detection of a level shift, we estimate the token bucket parameters from the rate timeseries $R_r$ as follows. The token generation rate (shaping rate) $\rho$ is estimated as the

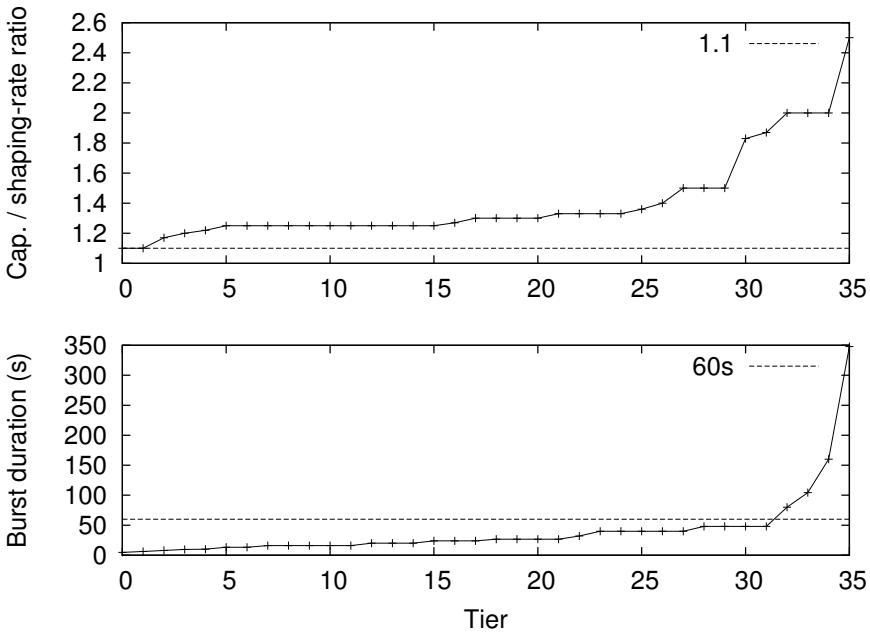



**Figure 16:** Advertised Comcast and Cox service tiers on the ISP websites. We use this information to choose the level shift ratio $\gamma$ and the probing duration $\Lambda$.

*median* (to be robust to outliers) of the received rate measurements *after* $\beta$:

$$\hat{\rho} = \underset{i=\beta+1\ldots n}{\tilde{R}_r(i)} \tag{11}$$

We estimate the token bucket depth (burst size) $\sigma$ based on the number of bytes sent till the $\tau$'th time interval. We estimate a range for $\sigma$, since we discretize time into intervals of size $\Delta$, based on the estimate $\hat{\rho}$ of $\rho$ and the received rates:

$$\hat{\sigma} = \left( \sum_{i=1}^{\tau} [R(i) - \hat{\rho}]\Delta \right) \pm \frac{[R(\tau) - \hat{\rho}]\,\Delta}{2} \tag{12}$$

### 4.2.3 Parameter Selection

As in any other measurement tool that is used in practice, there are some parameters that need to be tuned empirically. In ShaperProbe, the key parameters are the factor $\gamma$, the probing duration $\Lambda$, and the interval duration $\Delta$. We have selected the values of these parameters based on the detection of actual shaper deployments in broadband ISPs for which we knew the ground truth from service tiers advertised on ISP websites.

Figure 16 shows the ratio of the capacity over the shaping rate $C/\rho$ and the maximum burst duration (in seconds) for 36 advertised traffic shaper deployments at Comcast and

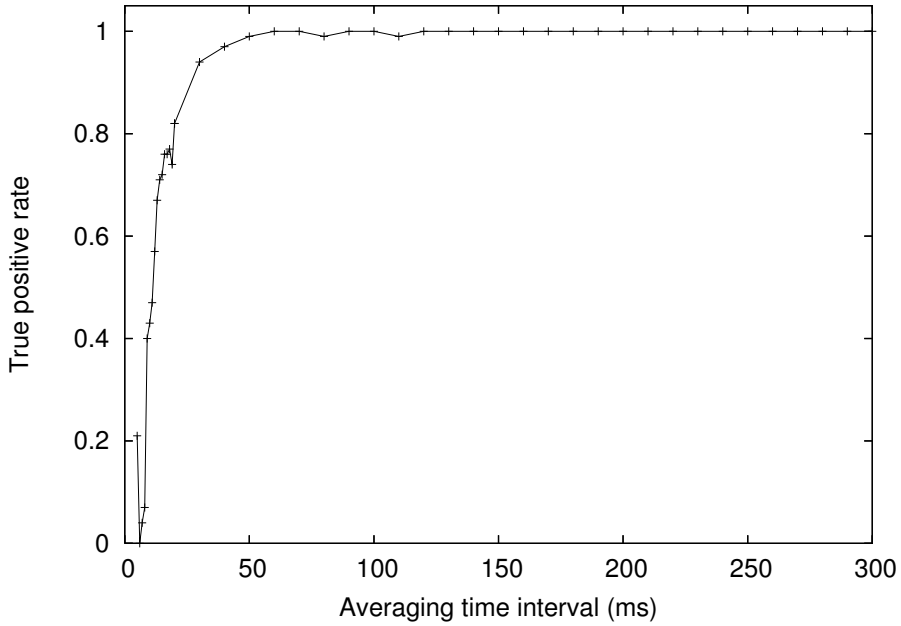



**Figure 17:** Effect of the averaging time interval for rate estimation, $\Delta$, on the shaping detection accuracy for a run on Comcast upstream.

Cox in metro Atlanta in October 2010. Note that all tiers have a capacity-to-shaping rate ratio of 1.1 or higher[4]; in the current implementation of ShaperProbe we use $\gamma = 1.1$.

The probing duration $\Lambda$ should be sufficiently long so that it can detect as many ISP shaping configurations as possible, while at the same time keeping the total probing duration reasonably short when there is no shaping. Figure 16 shows that the burst duration is at most 48s, except for 4 out of 36 configurations. $\Lambda$ is set to 60s in the current implementation. A typical ShaperProbe run in a residential network lasts for 2-3 minutes.

The averaging window size $\Delta$ should be sufficiently large to keep the estimation noise in $R_r$ low, and sufficiently short so that $\Lambda$ includes several rate samples. We have performed 100 trials in the upstream direction of a Comcast residential connection, whose SLA we know (4.5Mbps shaped to 2Mbps). Figure 17 shows the shaping detection rate for different values of $\Delta$. We found that for $\Delta \geq 50$ms, the shaping detection rate is 100%; as $\Delta$ approaches the inter-packet gap, the detection rate drops significantly. We set $\Delta$ to 300ms so that we can detect shaping even in low capacity links.

[4]The ISP does not have an incentive to deploy traffic shaping with a capacity-to-shaping rate ratio close to one.

## *4.3 ShaperProbe Implementation*

The design of a tool that works well on a wide variety of network conditions, OS platforms and broadband link technologies is challenging. A first challenge is that ShaperProbe requires a fast and accurate estimate of the narrow-link capacity between the sender and receiver; this estimate is the ShaperProbe probing rate. ShaperProbe uses packet train dispersion for estimating capacity; it additionally probes using a longer train to be robust to wireless link effects. Second, the probing method should be able to generate traffic at a *constant rate*, even with a coarse-grained userspace OS timer granularity. At the same time, the transmission of packets should not impose heavy load on the CPU resources at the sender. ShaperProbe sends small periodic packet trains, and times the inter-train gaps such that busy-wait loops are minimized. Third, the ShaperProbe client should be non-intrusive. The client and server abort the probing process if they observe losses on the path. Finally, cross traffic on the path may lead to temporary drops in the received rate $R_r$; we need to incorporate a filtering mechanism that can remove *outliers* from $R_r$. ShaperProbe filters outliers using recorded observations from the local neighborhood of the $R_r$ timeseries.

Figure 18 shows a screenshot of the ShaperProbe Windows client with shaping detection output of an ISP deployment.

We have assumed in the discussion in Section 4.2 that we have an estimate of the narrow link capacity. In practice, we can have cross traffic in the path, last mile wireless links, and end-host effects, which can add significantly to probing and measurement noise. We describe how ShaperProbe addresses the previous challenges and implementation below.

### 4.3.1 Capacity Estimation

We require an estimate of the narrow link capacity on the $\mathcal{SND} \rightarrow \mathcal{RCV}$ path before we probe for traffic shapers, since it determines our probing rate. We need an accurate estimate so that: (1) we minimize intrusiveness, i.e., not create persistent queue backlogs in the path due to our probing, and (2) we are able to consume tokens in the experiment duration. We implement capacity estimation as a *pre-probing* phase.

We estimate capacity using UDP packet trains. Specifically, $\mathcal{SND}$ sends a packet train

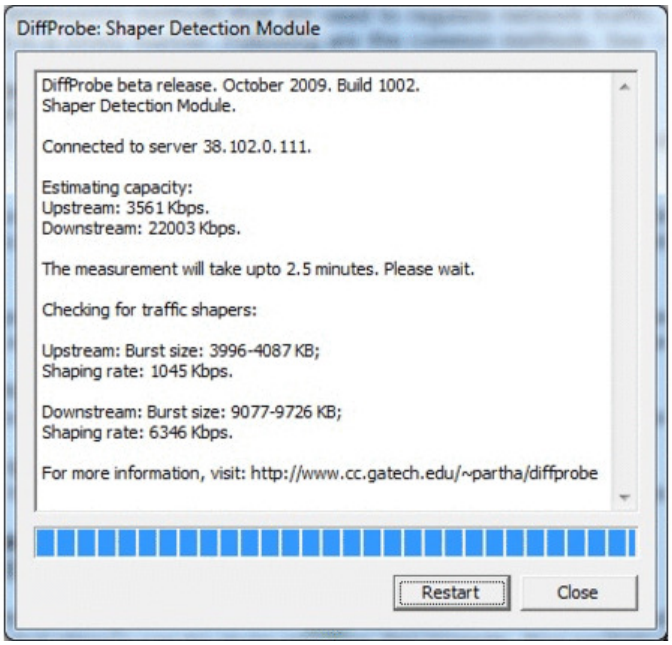



**Figure 18:** ShaperProbe client: screenshot from a user run.

of $N$ MTU-sized (size $S$) back-to-back packets $\{p_1, p_2 \ldots p_N\}$ to $\mathcal{RCV}$. After receiving the train, $\mathcal{RCV}$ estimates the narrow link capacity using the train dispersion $\delta$ (calculated as the difference between received timestamps of $p_N$ and $p_1$):

$$\hat{C} = \frac{(N-1)S}{\delta}$$

We repeat this measurement for $K$ trains and determine the path capacity as the median of the $K$ train estimates to reduce underestimates due to cross-traffic. In our implementation, we use $N = 50$ packets and $K = 10$ trains. Note that if we do not receive a part of the train[5], we use $N$ as the number of received packets, and $\delta$ is computed as the dispersion of the *received* train.

In practice, we found that the above methodology works well for wired end-hosts, but tends to incorrectly estimate the narrow link capacity in the downstream direction when the last mile is a wireless (802.11a/b/g) link - in cases where the narrow link is not the wireless link. This overestimation[6] occurs because of the non-work-conserving contention

[5]We use a timeout of 1s to receive each packet of the train.

[6]Typical downstream train overestimates on 802.11g are in the range 25-35Mbps, close to the L2 channel throughput.

delay process in CSMA/CA access links, due to which a downstream train may have to wait until the channel is free [100]. We also observed differences in overestimates with some 802.11g NICs depending on the operating system/driver implementation. In order to avoid such overestimates, we send a *long* packet train after we determine the train-based capacity estimate. Specifically, if the train-based estimate is $\hat{C_T}$, we send a UDP stream of MTU-sized packets at a rate $\hat{C_T}$ on $\mathcal{SND} \rightarrow \mathcal{RCV}$ for a duration of 5s and revise our capacity estimate to be the aggregate received rate in that duration. Note that this UDP stream will only induce large queue backlog if we have a significant capacity overestimate.

### 4.3.2 Probing and Non-intrusiveness

It is important that we send the probing stream for shaping detection at a constant rate that is close to the capacity $C$ for two reasons. First, we may detect false negatives if we cannot send at a rate equal to the path capacity. Second, end-host effects such as operating system noise and context switches due to the client environment can lead to drops in the sending rate at $\mathcal{SND}$, leading to potential false positives. We probe using MTU-sized (size $S$) packets, and send all probing traffic over UDP as follows.

**Probing:** The goal of the sender is to send a sequence of packets at a rate close to $C$, for a maximum of $\Lambda = 60$s. For a given sending rate, implementations of packet trains typically use a CPU-intensive busy-wait loop to maintain inter-packet gaps at userspace. Busy-wait loops can lead to a drop in send rate, since a sender process running for *extended periods* would yield to other processes. To avoid such scenarios, we send periodic *short-duration trains* of back-to-back packets to minimize time spent in busy-waits.

We determine the length of short-duration trains at runtime as follows. The idea is to minimize the duration spent on busy-waits. We first measure the userspace sleep resolution period $t_s$ on $\mathcal{SND}$; this gives us the minimum duration the send process can sleep without the need for a busy-wait loop. The train length $N_p$ (packets) is then determined as the value which minimizes the residual delay $d(N_p) = N_p g - \lfloor N_p g/t_s \rfloor t_s$ where $g = S/C$ (the inter-packet gap). We upper bound $N_p$ to 30 packets to accommodate OSes with a low sleep time resolution (e.g., some flavors of Windows). We maintain the probing rate by: (1)

sending train of size $N_p$, (2) sleeping for a duration $t_s$ for $\lfloor N_p g/t_s \rfloor$ (integer) times; followed by (3) a *short* busy-wait for $d(N_p)$.

**Non-intrusiveness:** The sender, $\mathcal{SND}$, stops probing if either of the following is true: (1) $\mathcal{RCV}$ detected a traffic shaper, (2) $\mathcal{SND}$ probed for the experiment duration $\Lambda$, or (3) $\mathcal{RCV}$ detected packet losses. We record the packet loss rate in time windows of $\Delta$. We abort on packet losses if we either see a low loss rate (more than 1%) for a small time period, or if we see a high loss rate (more than 10%) for a smaller time period (a few consecutive time windows - it is necessary that this duration is longer than $n_R\Delta$ to avoid aborting probing in the presence of shaping).

### 4.3.3 Filtering Rate Noise

The received rate timeseries $R_r$ may have temporal variations ("noise") at $\Delta$-timescales in practice, even if $\mathcal{SND}$ sends at a constant rate, due to cross traffic variations in intermediate links or due to end-host noise at $\mathcal{RCV}$. Such noise manifests as *outliers* in $R_r$. We pre-process the $R_r$ timeseries with a simple filtering heuristic before running the level shift algorithm. Each time we compute a new rate value $R_r(n)$, we condition a previous timeseries value $R_r(n-n_f)$ by looking at rate points in its neighborhood as follows (assuming $n > 2n_f$). We compute the median of $n_f$ rate points to the left (say $\tilde{R}^l_{n_f}$) and $n_f$ points to the right (say $\tilde{R}^r_{n_f}$) of the point $n-n_f$. We modify $R_r(n-n_f)$ if it is either greater than or less than *both* $\tilde{R}^l_{n_f}$ and $\tilde{R}^r_{n_f}$. Under this condition, we modify the rate value to the mean of the left and right medians, $R_r(n-n_f) = (\tilde{R}^l_{n_f} + \tilde{R}^r_{n_f})/2$. Note that we would condition each rate point at most once during the experiment.

When we modify a rate point, we recompute ranks of all rate values that fall between the old value and the new value of $R_r(n-n_f)$. Note that this filtering method would *not change* rate values that lie in the $n_f$-neighborhood of the level shift points $\tau$ and $\beta$, since the median condition on the $n-n_f$ point would not be satisfied for those points. We say that a rate $R_a$ is larger than $R_b$ if $R_a > \gamma R_b$ where $\gamma$ is the rate threshold in Equation 10.

### 4.3.4 Service Deployment

We have implemented the ShaperProbe client in userspace for 32 and 64-bit Windows, Linux, OS X and BSD[7] operating systems. The non-GUI server-client functionality is about 6000 lines of open source native code. The client is a download-and-click binary and does not require superuser privileges or installation. A version of the ShaperProbe client has also been ported as a plugin to the *Vuze* BitTorrent client and is actively used.

The server and client establish a TCP control channel for signaling. The client starts with capacity estimation in both upstream and downstream directions, followed by shaping detection in each direction. We log per-packet send and receive timestamps and sequence numbers for all probing phases at the server, as well as the client IP address and server timestamp (UTC) for each run.[8] A typical run of ShaperProbe on a residential connection lasts for about 2-3 minutes.

ShaperProbe has been provided as a service on M-Lab since May 2009 [16]. We currently run ShaperProbe server replicas on 49 M-Lab hosts connected directly to tier-1 ASes with gigabit connectivity. Each server contains eight CPU cores and 8GB physical memory and runs Linux kernel 2.6.22; the servers currently host 10 measurement tools in virtual machines. The servers have public IP addresses. Servers are monitored for resource usage of the virtual machines to keep usage low. The servers are located amongst 27 sites in the following countries: North America (10 sites), Greece (2), Italy (2), Netherlands (2), United Kingdom (2), Australia (1), Austria (1), France (1), Germany (1), Ireland (1), Japan (1), New Zealand (1), Slovenia (1) and Spain (1).

For measurement accuracy, we allow only one client at each server replica at any time. Incoming client requests are first served at random by one of three load balancer replicas, which redirects the client to a server instance randomly. If a server is busy, the client uses a fall-through mechanism to retry a random server from the remaining servers; after three failed retries, the client aborts with a message to the user.

[7] The BSD source is distributed with the FreeBSD Ports collection.
[8] All ShaperProbe data is publicly available through M-Lab [10].

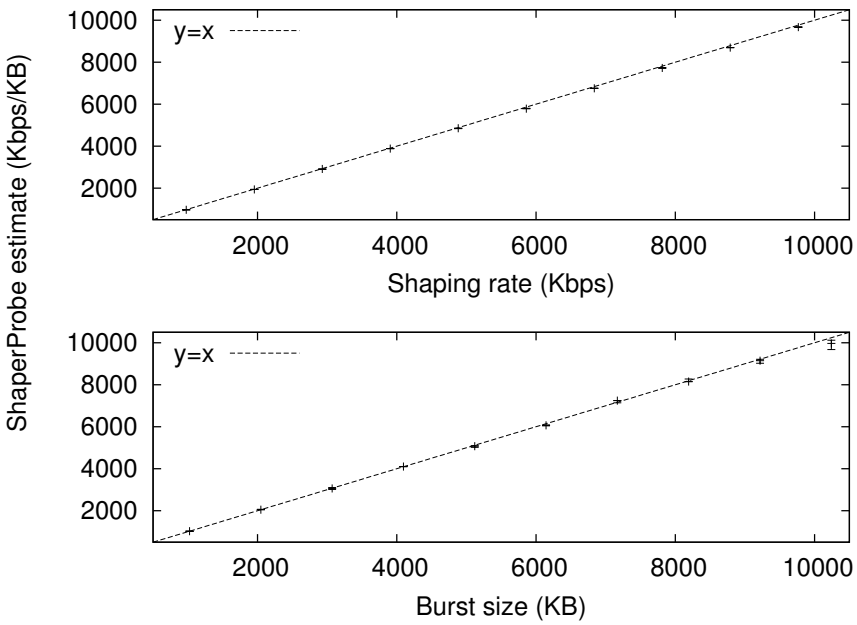



**Figure 19:** Accuracy: ShaperProbe estimates of token bucket parameters vs. ground truth using shaping emulation experiments.

## *4.4 Results*

In this section, we take a first look at results from the ShaperProbe deployment at M-Lab. We first examine accuracy using two ISPs for which we know the shaping ground truth and from emulation experiments.

### 4.4.1 Accuracy

We test the latest version (2011) of ShaperProbe on two residential ISPs, AT&T and Comcast, at two homes in metro Atlanta. We use the High-Speed Internet service of Comcast, and the DSL service of AT&T. At the time of these experiments, the Comcast configuration was: {10Mbps up, 22Mbps down} shaped to {2Mbps up, 12Mbps down} [7], while the AT&T configuration did not use shaping ({512Kbps up, 6Mbps down}) [2]. Out of 60 runs, we did not observe any shaping detection errors in either direction at the AT&T connection, while we observed two upstream false negatives at the Comcast connection due to capacity underestimation.

We also emulated token bucket shaping on a wide-area path between a residential Comcast connection and a server deployed at the Georgia Tech campus. We use the LARTC `tc`

tool on Linux with a 2.6.22 kernel on a dual-NIC 1GHz Celeron router with 256MB RAM. Over 20 experiments for each token bucket configuration and 10 configurations, we found that ShaperProbe detects the shaper in all (200) experiments; it also accurately estimates the shaping rate and bucket depth for all configurations. Figure 19 shows the Wilcoxon median estimate and confidence intervals for ShaperProbe's token parameter estimates, based on 20 repetitions for each token bucket configuration in the downstream direction.

**Data preprocessing** In the following, we analyze data collected from the ShaperProbe M-Lab service. First, we consider runs from the ShaperProbe collected between 20th October 2009 and 9th May 2011 (total of 845,223 runs). Each run's trace contains per-packet timestamps and sequence numbers for the upstream and downstream probing "half runs". Second, we say that a half run is "unfinished" if no shaping was detected and the run lasted for less than 50s - we discard such runs. All completed half runs which are not diagnosed as shaping are considered *no-shaping* cases. Recall that ShaperProbe probes each direction for 60s, and terminates a half run if it either detected shaping or if it observed packet losses during probing. A half run can also be unfinished if the user aborted the client before it could run to completion. After preprocessing, we have a total of 281,394 upstream and 236,423 downstream finished half runs.

### 4.4.2 Geography and ISPs

First, we look at estimated capacity and shaping detections across the Internet. We use the MaxMind IP geolocation database (the GeoLite city version [20]) to find the latitude-longitude pair for the client IP address for a run. Note that there are two limitations to our geography visualizations: (1) IP geolocation approaches are not always accurate (a recent study [99] found high accuracy at the level of countries but not at the level of cities), and (2) the IP address that ShaperProbe records is the address that is visible to the server - this can be a problem if the client is behind a NAT.

Figure 20 shows the a heatmap of download capacities from ShaperProbe runs across the Internet. We show one point for each finished half run. Figure 21 shows the percent of shaping detections for each integer latitude-longitude pair (we round off latitude and

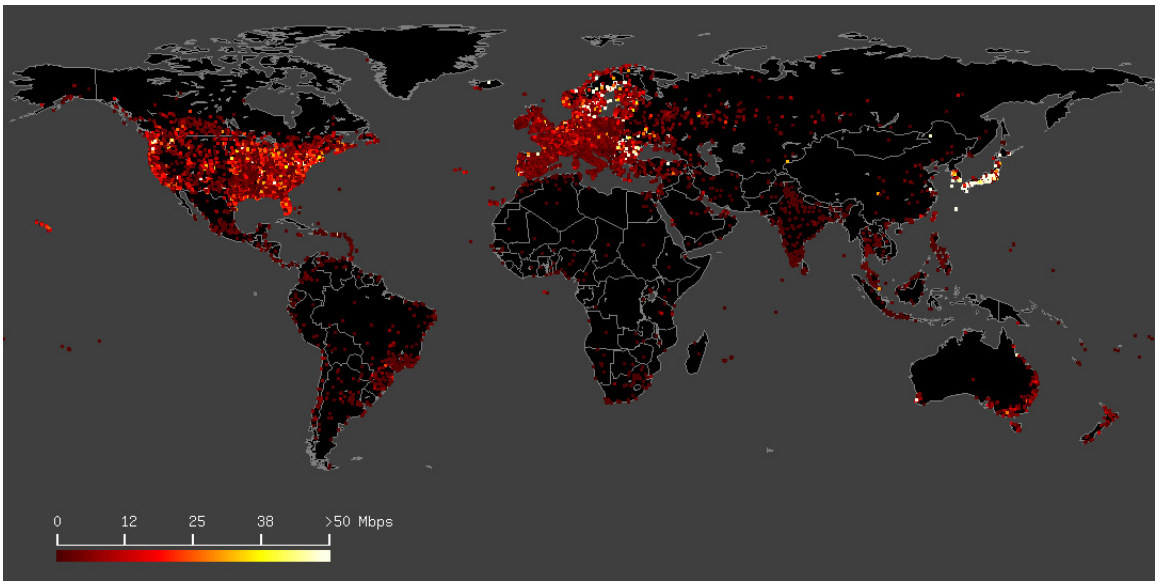



**Figure 20:** ShaperProbe data: download capacity estimates across the Internet. We show one point on the heatmap for each finished half run. The scale of the heatmap is from 0 to 50 Mbps (and higher).

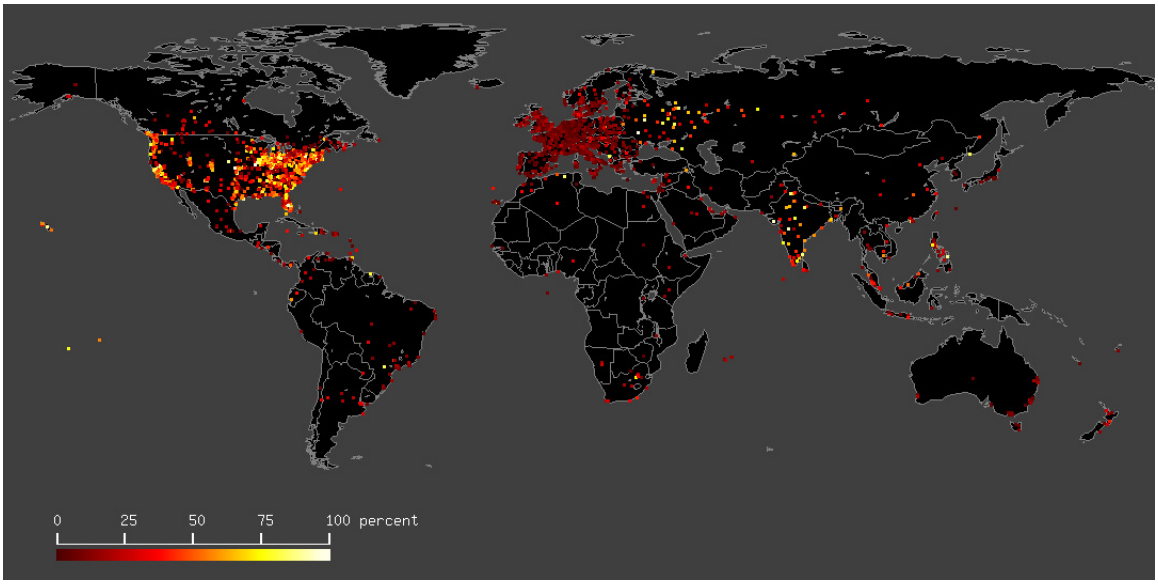
**Table 3:** Shaping detections: top-5 ISPs in terms of ShaperProbe runs. For each ISP we show percentage of runs with detected shaping and number of total runs.

| ISP | Upstream (%) | Dwnstrm. (%) |
|---|---|---|
| Comcast | **71.5** (34874) | **73.5** (28272) |
| Road Runner | 6.5 (7923) | **63.9** (5870) |
| AT&T | 10.1 (8808) | 10.9 (7748) |
| Cox | **63** (5797) | **47.4** (4357) |
| MCI-Verizon | 5.6 (8753) | 8.4 (7733) |

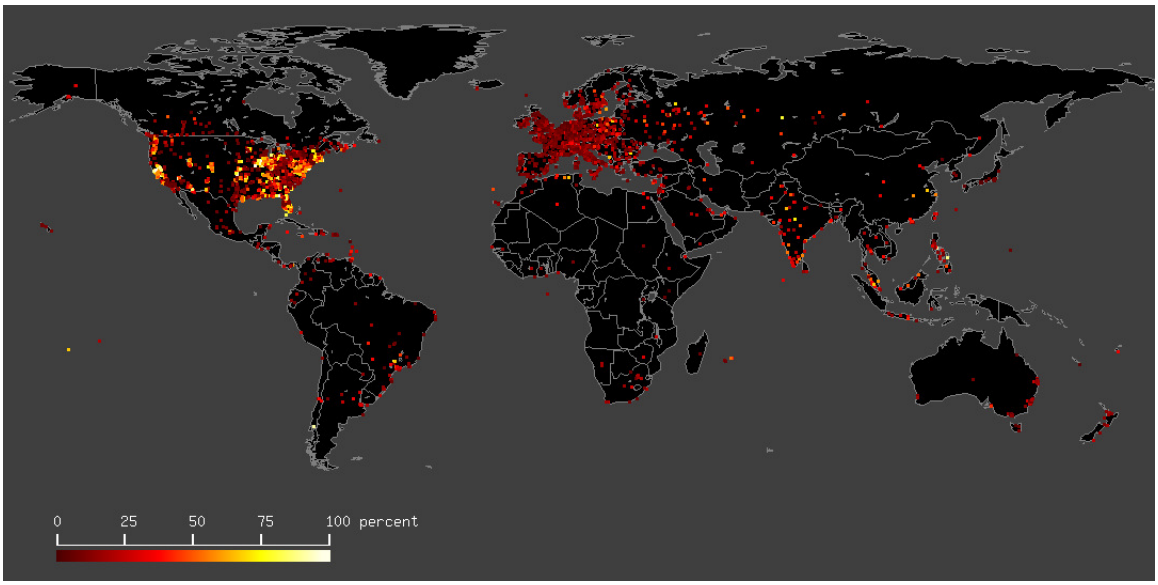
longitude to the nearest integer values). We only show those latitude-longitude points for which we have at least 10 finished half runs. We see that many shaping deployments are in the US, while many parts of Europe have a small fraction of shaping detections. Moreover, in the US, the shaping deployments are less prevalent in the upstream direction compared to the downstream.

Next, we cluster AS numbers into ISPs using their `whois` AS names. The AS information was obtained from Cymru's whois database in May 2011. Runs which passed the pre-processing checks come from 5,167 distinct ISPs. The top five ISPs in terms of the number of runs as well as the fraction of shaping detections are shown in Table 3.

It should be noted that there are several factors that influence the fraction of shaping




(a) Downstream




(b) Upstream

**Figure 21:** ShaperProbe data: percent shaping detections across the Internet. Each point shows the percentage of shaping detections for the integer latitude-longitude pair. The scale of the heatmap is from 0 to 100%.

**Table 4:** Comcast: detected shaping properties in ShaperProbe data (2009-2011).

(a) Upstream.

| $C$ (Mbps) | $\rho$ (Mbps) | $\sigma$ (MB) | Burst duration (s) |
|---|---|---|---|
| 3.5 | 1 | 5 | 16.7 |
| 4.8 | 2 | 5, 10 | 15.2, 30.5 |
| 8.8 | 5.5 | 10 | 25.8 |
| 14.5 | 10 | 10 | 18.8 |

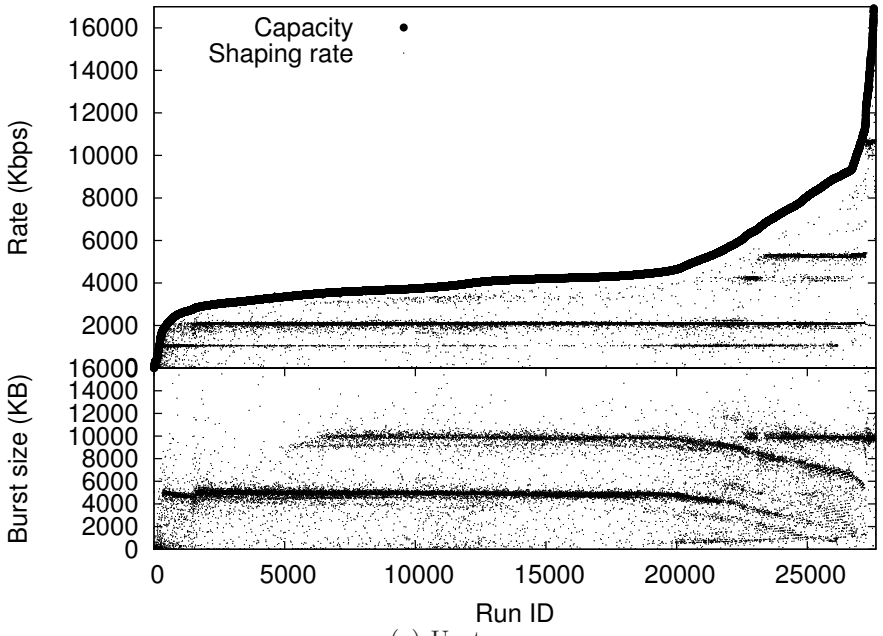
(b) Downstream.

| $C$ (Mbps) | $\rho$ (Mbps) | $\sigma$ (MB) | Burst duration (s) |
|---|---|---|---|
| 19.4 | 6.4 | 10 | 6.4 |
| 21.1 | 12.8 | 10 | 10.1 |
| 28.2 | 17 | 20 | 14.9 |
| 34.4 | 23.4 | 20 | 15.3 |

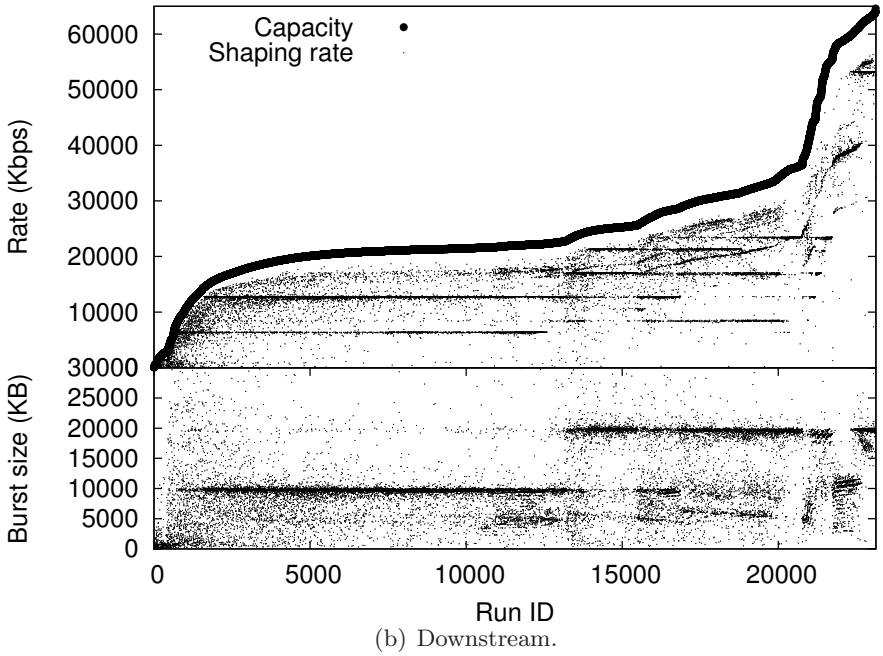
detections in an ISP. First, ISPs provide multiple tiers of service; some tiers may not use shaping, while service tiers change frequently. Second, an ISP may not deploy shaping in all geographic markets. Third, the access link type can be a factor: a DSL provider can dynamically change the link capacity instead of doing shaping, while a cable provider is more likely to use shaping since DOCSIS provides fixed access capacities. Fourth, for a given connection, the shaping parameters can be dynamically adjusted based on time or load conditions in the ISP. Fifth, an ISP $A$ can originate the BGP prefixes of a smaller ISP $B$ that deploys shaping (while $A$ does not) - we cannot distinguish $A$ from $B$ based on BGP prefix-to-ASN mapping. We study some of these factors in ISP case studies in the sub-sections below. Some ISPs disclose their traffic shaping configurations (e.g., Comcast, Road Runner and Cox, which we cover in this chapter); in such cases, we can validate our observations and shaping estimates.

### 4.4.3 Case Study: Comcast

Comcast offers Internet connectivity to homes [7] and enterprises [5], and uses two types of access technologies: cable (DOCSIS 3.0) and Ethernet. In each access category, it offers multiple tiers of service. Comcast shapes traffic using the PowerBoost technology [6].

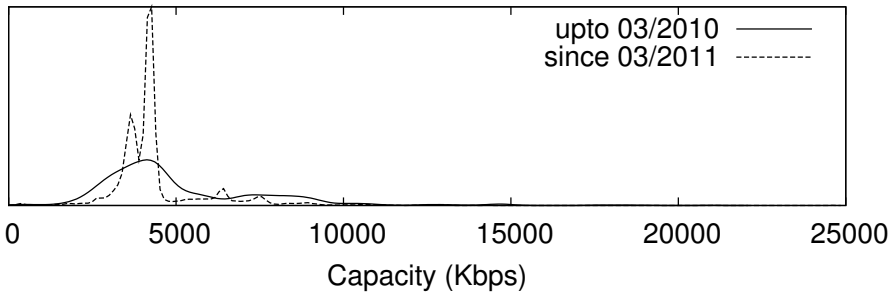


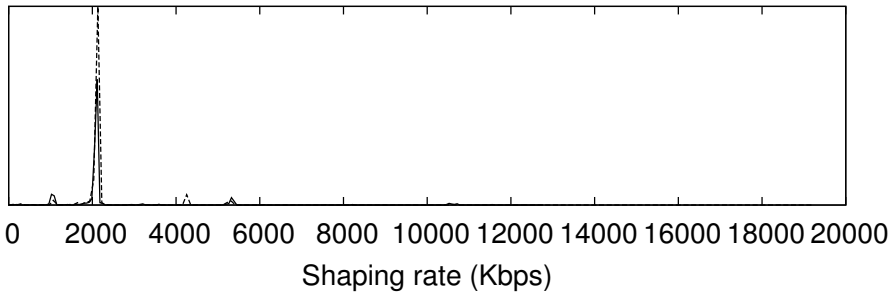

(a) Upstream.



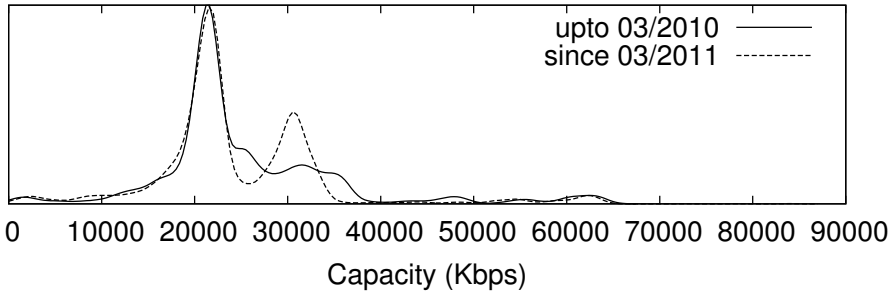

(b) Downstream.

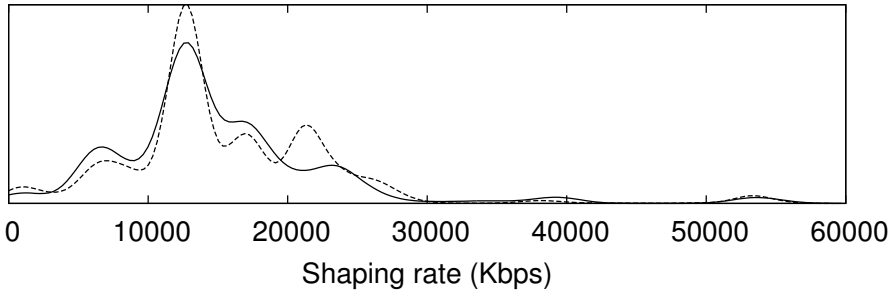
**Figure 22:** Comcast: Visualizing shaping deployment characteristics. For each completed half run, we show the estimated shaping configuration with three points - burst size in the bottom panel, and the capacity and shaping rate in the top panel.




(a) Upstream.




(b) Downstream.

**Figure 23:** Comcast: histogram of shaping rate and capacity, comparing estimates from 2010 and 2011.

**Shaping profiles** We observed many shaping configurations at Comcast between October 2009 and May 2011. Figure 22 shows the shaping configuration of each run (ordered by capacity). For each run, designated by an "ID", we plot two points in the top panel for the capacity and the shaping rate; and a point in the bottom panel for the burst size. The capacities form an *envelope* of the shaping rate points. We see that there are strong *modes* in the data; Table 4 is a summary of these modes. For higher capacities, we see a larger number of modes in the shaping rate. However, at the tail of the capacity distribution there is only one shaping rate that corresponds to the highest service tier provided by Comcast. We verified our observations with the Comcast website listings [5, 7]. Note that we may not observe all service tiers in that web page, depending on the number of ShaperProbe users at each service tier. We also observe two or three burst sizes that are used across all tiers; the PowerBoost FAQ mentions 10MB and 5MB burst sizes [6].

Note that the capacity curves do not show strong modes, unlike the shaping rates. This is due to the underlying DOCSIS access technology. The cable modem uplink is a non-FIFO scheduler; depending on the activity of other nodes at the CMTS, the capacity can vary due to customer scheduling and DOCSIS concatenation. A DOCSIS downlink can also influence the dispersion-based capacity estimates under heavy traffic load conditions because it is a broadcast link. On the other hand, DSL access links are point-to-point links, and the capacity estimates for DSL providers are less likely to be noisy.

**Did shaping configurations change during the two years?** We compare Comcast data collected from 1st October 2009 to 31st March 2010 with data from 1st March to 31st May 2011. Figure 23 shows estimates of the capacity and shaping rate distributions using a Gaussian kernel density estimator. In the upstream direction, the capacity and shaping rates (the modes of the corresponding distributions) have not changed significantly. The downstream links show a new capacity mode of 30Mbps and a shaping rate mode of 22Mbps in 2011. We did not find significant changes in the burst size during the last two years.

**Did shaping parameters change with time of day?** We compare runs at 1200 and 0000 hours UTC (timestamps are taken at the server) in Figure 24; the data is taken

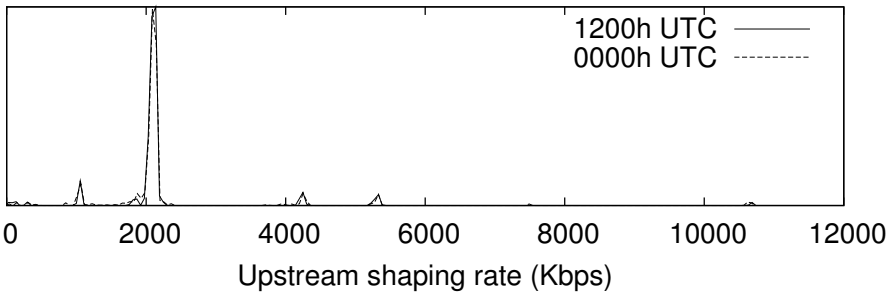
from March-May 2011. We see that the upstream shaping rates have a similar distribution between the two times; the downstream rates show a slight difference in densities - the lower shaping rates show a higher density at evenings in US times (PST/EST). The burst sizes show a similar trend - we see a higher density of lower burst sizes in the evenings than in the mornings. Note that the above analysis assumes that *the user sample (in terms of tier) that we get at different hours of day are identically distributed*.

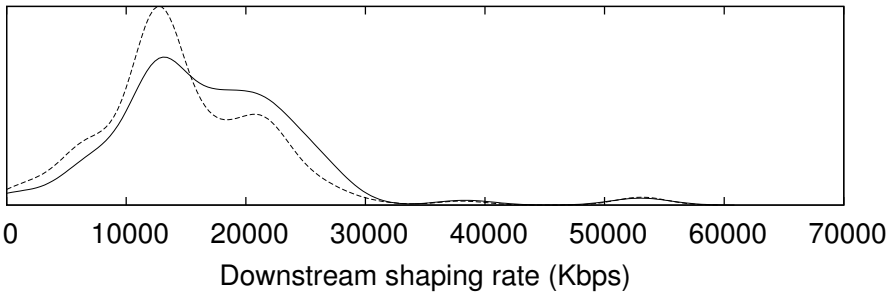
**Non-shaped runs** We examine runs in which ShaperProbe did not detect shaping. Figure 27 compares the capacity distribution in such runs with the shaping rate distribution in shaping runs. The non-shaped capacity distributions are *similar* to the shaping rate distributions. Non-shaping runs occur due to the following two reasons. First, Comcast provides service tiers that do not include PowerBoost, but have capacities similar to PowerBoost service tiers (e.g., the Ethernet 1Mbps and 10Mbps business service). Second, it is possible that cross traffic resulted in an empty token bucket at the start of the measurement, and so the capacity that ShaperProbe estimated was equal to the shaping rate; we would not detect shaping in that case.

### 4.4.4 Case Studies: Road Runner and Cox

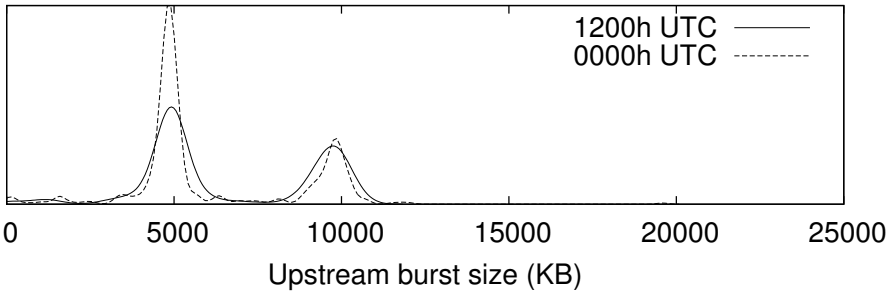
Road Runner (RR) is a cable ISP. A unique aspect of RR is that we have found evidence of downstream shaping, but *no evidence* of upsteam shaping in any service tier on their web pages. The ShaperProbe measurements for RR support this observation - 94% of the upstream runs *did not* detect shaping, while 64% of the downstream runs did. Another interesting aspect of RR is that shaping depends on the geographic region of the customer; for example, in Texas, RR provides four service tiers: the lower two are not shaped while the upper two are shaped [15]. Figure 25 shows the downstream shaping properties in the ShaperProbe RR runs. We see three dominant shaping rates and a single dominant burst size. Under the hypothesis that RR does not shape upstream traffic, we can say that our false positive detection rate for their upstream links is about 6.4%.

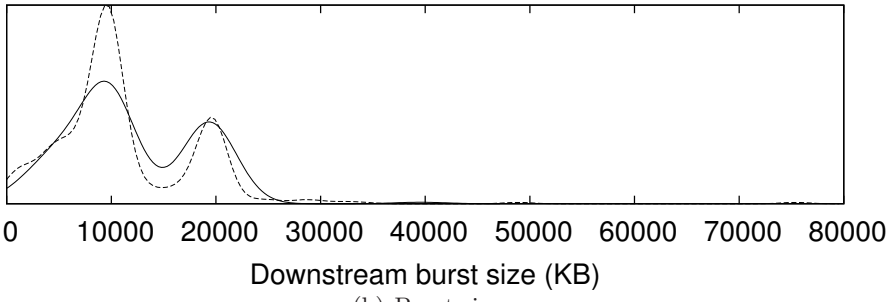
The capacity distribution of non-shaped RR runs is shown in Figure 26 (the x-axis is truncated). Note that unlike Comcast the downstream capacity mode of 750Kbps is not




(a) Shaping rate.




(b) Burst size.

**Figure 24:** Comcast: histogram of shaping rate estimates at different times of day.

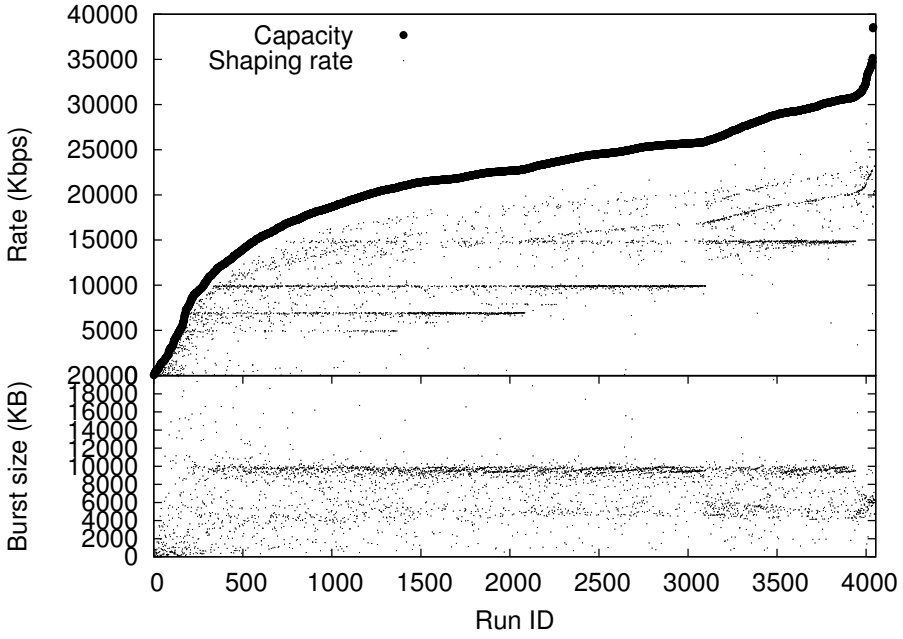



**Figure 25:** Road Runner: visualizing downstream shaping configuration estimates.

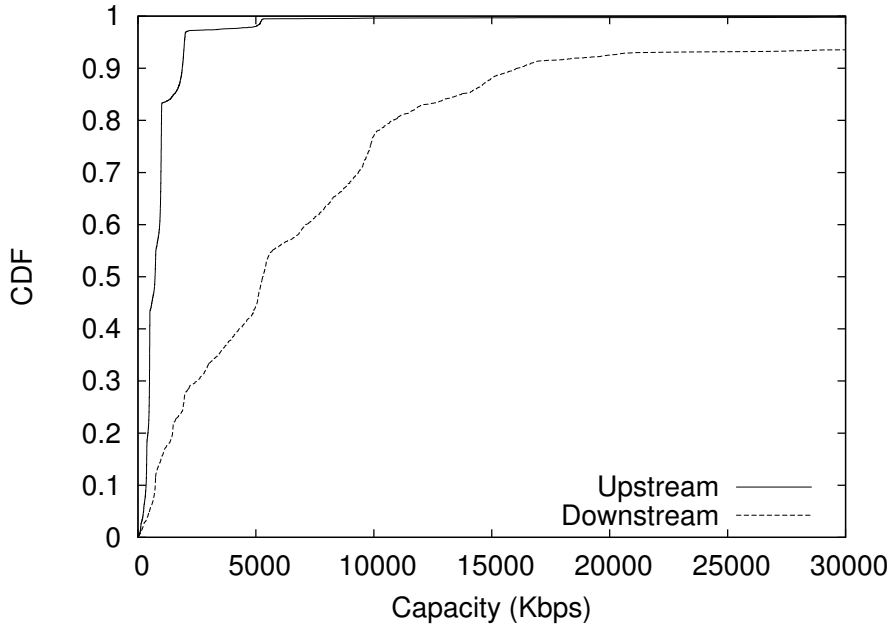



**Figure 26:** Road Runner: distribution of capacity of non-shaped runs.

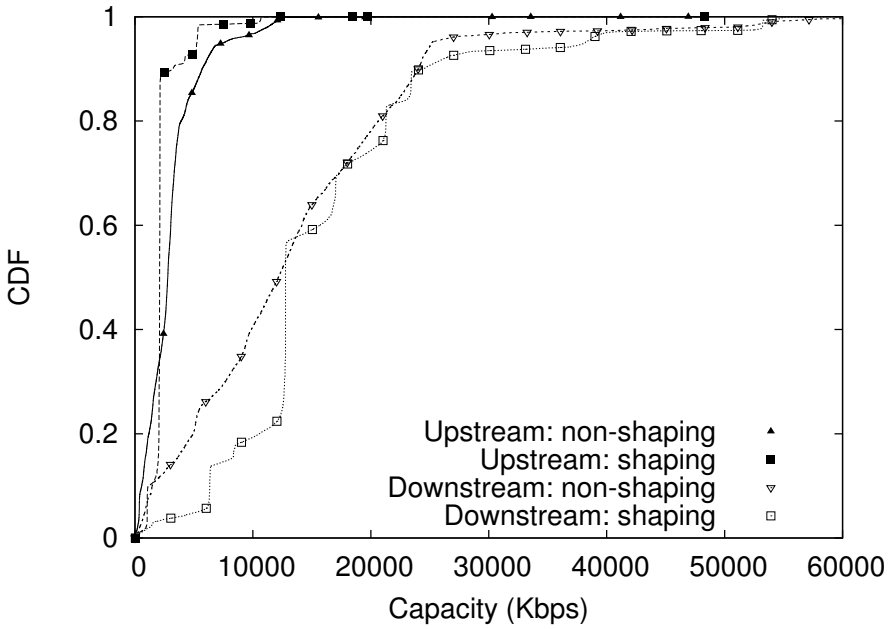



**Figure 27:** Comcast: comparing distribution of capacity in non-shaping runs with shaping rate in shaping runs. The non-shaping runs could arise due to either: (1) service tiers without shaping but having similar capacities as sustained rates in shaped tiers, or (2) an empty token bucket at start of probing.

equal to any of their shaping modes, which may mean that this tier is not shaped.

Cox provides residential and business Internet access using cable and Ethernet access. The website [9] mentions that the residential shaping rates and capacities depend on the location of the customer. We gathered residential shaping configurations from the residential pages [9]. The upstream shaping properties of Cox runs in Figure 28 agree with some of the ground truth information we found: $(C, \rho)$Mbps: (1, 0.77), (1.3, 1), (2, 1.5), (2.5, 1), (2.5, 2), (3, 2), (3.5, 3), (5, 4) and (5.5, 5). Note that the previous ground truth was collected in October 2010, while the ShaperProbe data covers two years. We also found a single burst size mode in the Cox data.

#### 4.4.5 Case Study: AT&T

Our final case study is that of an ISP for which we do not see frequent shaping detections (10% or less). AT&T provides Internet access to a wide range of customers, from homes and small businesses to enterprises (including other ISPs). Their residential service includes four DSL service tiers [1, 2]. We did not find any mention of traffic shaping in the AT&T

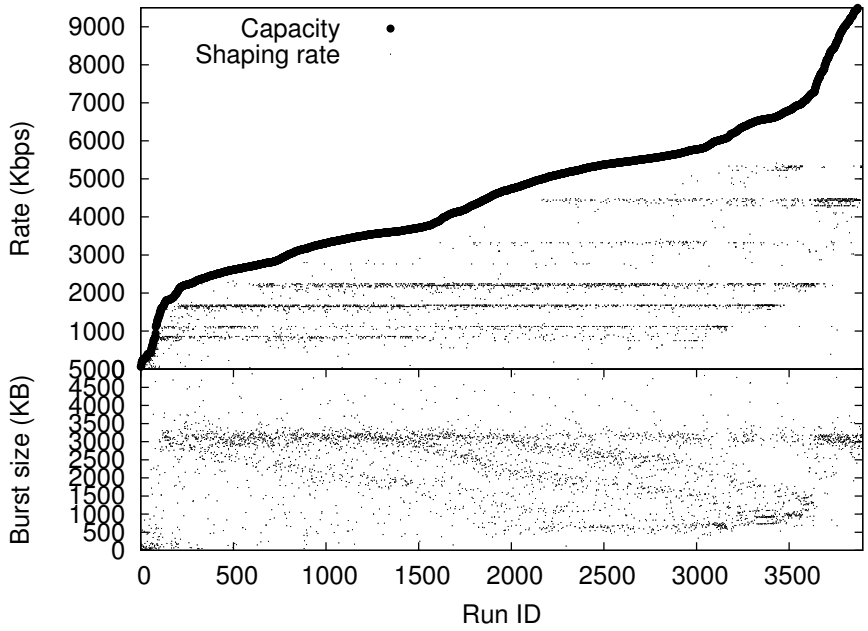



**Figure 28:** Cox: visualizing upstream shaping configuration estimates.

service descriptions [1, 2].

**Capacity** We first look at the 90% of the 7,000 to 8,000 runs that did not see shaping. The capacity distribution of non-shaped runs is shown in Figure 29. Given the point-to-point nature of DSL links, ShaperProbe estimates the narrow link capacity more accurately than in cable links. Hence, the capacity distributions show several modes: {330Kbps, 650Kbps, 1Mbps, 1.5Mbps} upstream, and {1Mbps, 2.5Mbps, 5Mbps, 6Mbps, 11Mbps, 18Mbps} downstream. We did not observe significant changes in the capacity modes between 2009 and 2011.

**Shaping runs** We look at the 10% of AT&T runs that were *probably mis-diagnosed* as shaping. We found that about a third of these runs exhibit strong shaping rate modes and an associated burst size mode (Figure 30). About 80% of 333 runs that have the shaping rate modes come from hostnames that resolve to the domain `mchsi.com`, owned by the cable ISP Mediacom [13]. So, it is possible (though we cannot be certain) that these shaping detections were *not* errors afterall.

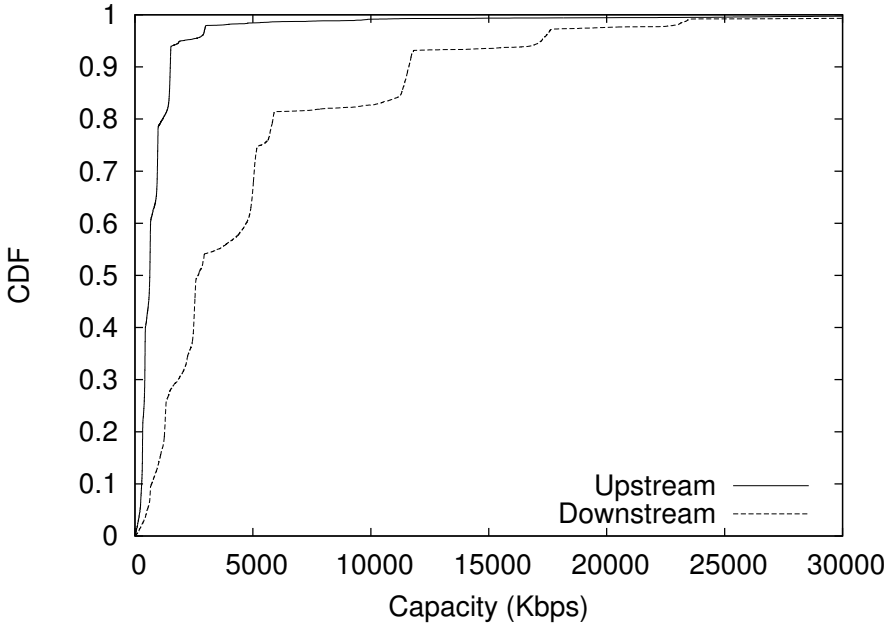
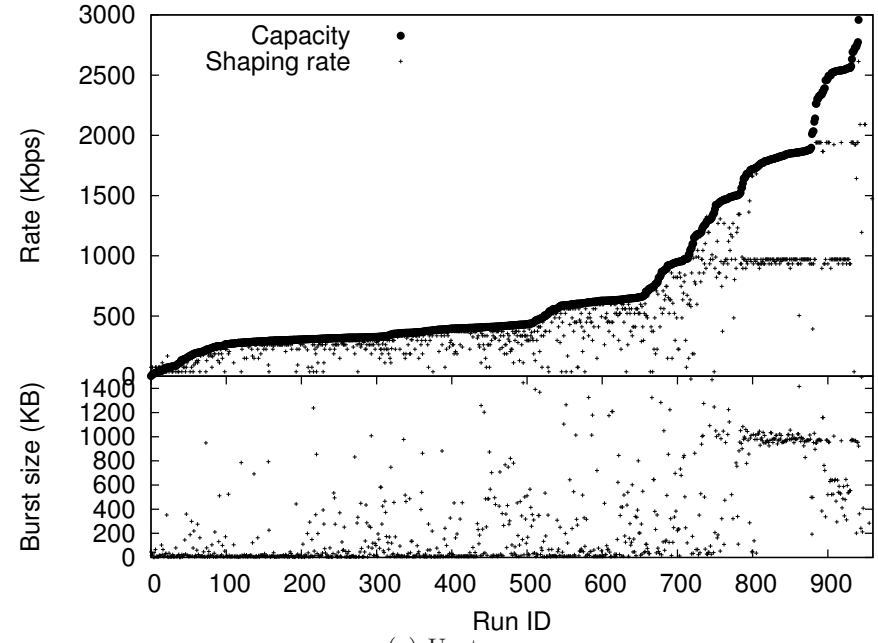



**Figure 29:** AT&T: distribution of capacity of non-shaping runs.

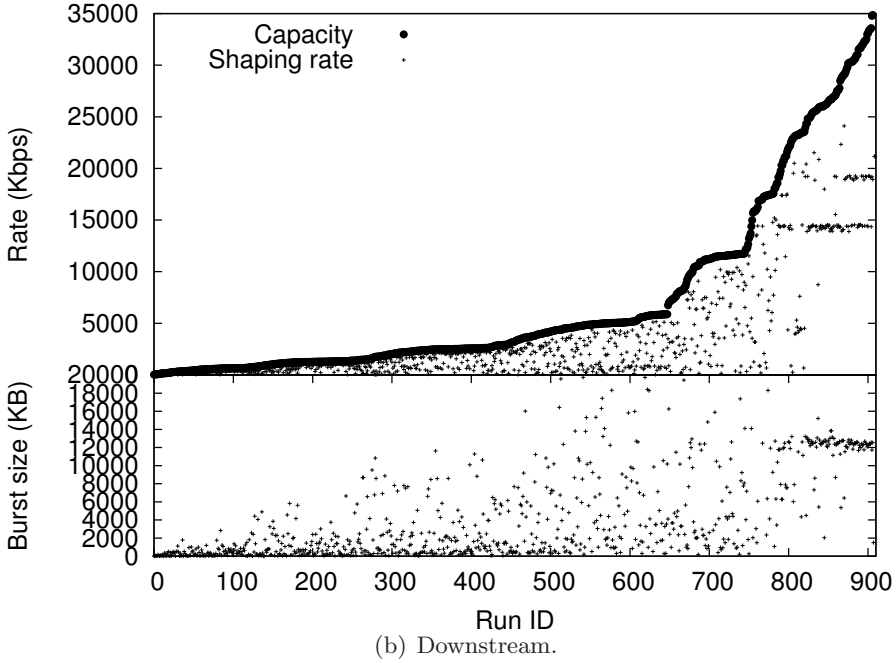
**Did capacity change with time of day?** We look at the capacity distribution at two one-hour periods of day separated by 12h; and consider non-shaping traces in the period March-May 2011. Figure 31 shows a pdf of the capacity at the two UTC times. We see that the *relative densities* of link capacities did not change significantly between different times of day.

### *4.5 Discussion: Limitations*

We have made two assumptions in designing ShaperProbe. First, we assume that the bottleneck link on an end-to-end path is close to the home, and that the traffic shaping module is also deployed close to the home (or in the ISP). In other words, we assume that the selection of a server for probing and measurements will not affect the inference result. Second, we assume that the token bucket is full when we start probing for traffic shaping, and that there is no significant competing cross traffic that consumes tokens. Violations of these assumptions can lead to false detections and incorrect estimates. We can check at runtime whether these assumptions are true by cross-checking inference results with one or more tests at different times and against different servers (which have non-overlapping paths between the server and the ISP). In practice, however, it is not likely that these




(a) Upstream.




(b) Downstream.

**Figure 30:** AT&T: Visualizing shaping configuration estimates. We found that runs falling in the distinct shaping rate modes come from a cable ISP, Mediacom.

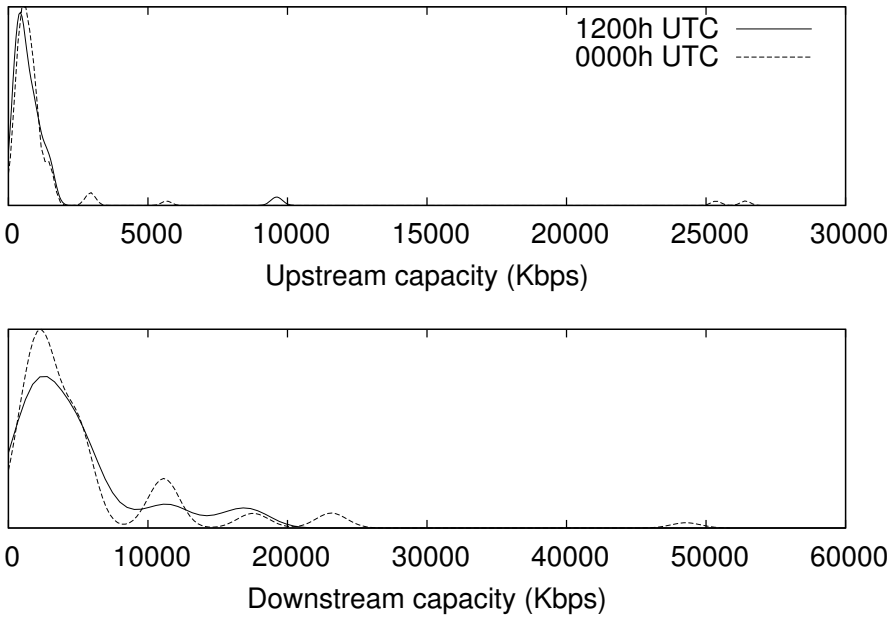


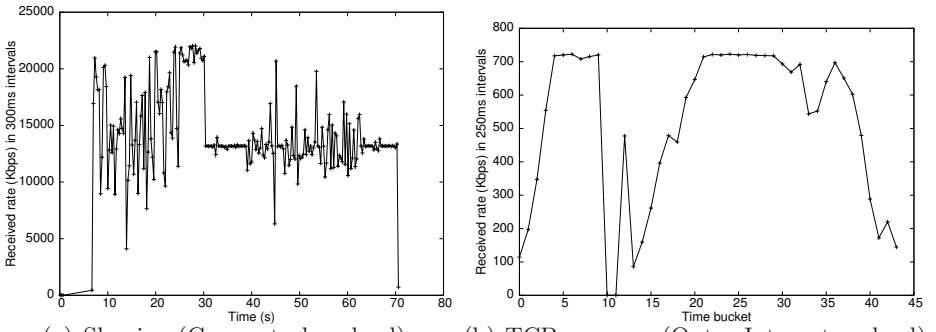

**Figure 31:** AT&T: distribution of capacity estimates at different times of day.

assumptions are violated, as we see low variability in shaping estimates in our data. Note that additional probing will incur longer runtime and higher chances of being intrusive.

### *4.6 Passive Method*

In this section, we design an end-to-end detection mechanism for inferring whether a particular TCP-based application's traffic is subject to traffic shaping using passively observed traces; and in such cases, estimate the shaper parameters $C$, $\rho$ and $\sigma$. This approach can be used to detect if an ISP shapes the traffic of specific applications. We apply this passive method in the Network Diagnostic Tool (NDT) traces [14] collected at M-Lab. NDT is a TCP speed test and diagnosis tool, which collects packet traces for 10s TCP bulk-transfers in upload and download directions.

We have several requirements in designing a shaping detection method. First, our method should be robust to the presence of "noise" induced by TCP flavour-specific dynamics, cross-traffic, and link types (DOCSIS, 802.11, WiMAX, etc.). Second, our method should detect and estimate shaping in *real-time*. Third, our method should be *user friendly*; in other words, it should work with or without administrative privileges (an online packet capture or an offline trace).




(a) Shaping (Comcast, download). (b) TCP recovery (Optus Internet, upload).

**Figure 32:** Rate level shifts can arise from either TCP dynamics or traffic shapers.

The passive inference technique takes as input an application packet trace (either real-time or offline) at a sender $\mathcal{SND}$ or the receiver $\mathcal{RCV}$[9]. The passive inference method is useful over active probing to detect cases where an ISP is shaping certain classes of traffic based on parameters such as destination/source which may not always be feasible to replicate with active probing.

We consider the specific case of a bulk-transfer application that uses TCP. Detecting shaping on TCP traffic is challenging for a number of reasons. First, TCP throughput can change with time depending on network conditions, and a level shift in TCP throughput occurs every time TCP decreases its window due to timeouts or packet losses. Second, TCP does not send a constant-rate stream, and so it harder to estimate the number of tokens in the token bucket. There can be time periods in which the TCP connection's throughput is below the shaping rate, causing accumulation of tokens. Third, after shaping kicks-in, TCP backoffs and timeouts can lead to accumulation of tokens (bringing the link service rate temporarily back to the capacity) - hence, the link rate may not be bounded by $\rho$. We cannot use the active detection method as-is on TCP traces for the above reasons. Our technique needs to distinguish throughput level shifts due to a shaper from those due to TCP backoffs. Figure 32 shows examples of the two types of level shifts.

The passive detection method works on the received rate timeseries $R_r(t)$, and uses two properties of $R_r(t)$ when there is a shaper: (1) there will be a time $t$ at which $R_r(t)$

[9]Since a TCP end-point can be both sender and receiver (depending on the application), we distinguish sender from receiver based on user input.

undergoes a level shift, and (2) after the level shift, the received rate is *almost constant* (for a duration that depends on the connection's round-trip time and the link buffer size).

**Rate estimation:** We begin by constructing a received rate timeseries $R_r(i)$, $i \geq 1$, by dividing time into discrete non-overlapping intervals of size $\Delta$. It is important that we choose a suitable value for $\Delta$ so that we have sufficient number of rate measurements to detect a level shift, and at the same time have low *noise*. We estimate $\Delta$ based on the TCP trace, since the inter-packet gaps can vary depending on TCP backoffs and timeouts. In the NDT traces, we have observed that some token bucket implementations generate tokens in periodic intervals ($\delta_{tb}$). Depending on the length of this interval, packets buffered in a shaper that has no tokens will be served in short bursts (of size $\rho\delta_{tb}$ bytes) at the link capacity as long as there is a backlog. In order to reduce *variance* in the post-shaping rate timeseries, we estimate $R_r(i)$ as follows.

We record the start and end timestamps of received packets in the $i$'th interval: call them $s(i)$ and $e(i)$. If we received $B(i)$ bytes in the $i$'th interval, we estimate $R_r(i)$ using the inter-burst gap $\delta_b$:

$$R_r(i) = \frac{B(i)}{\delta_b}$$

where, $\delta_b = \max\{s(i+1)-s(i), e(i)-e(i-1)\}$. The max. condition is to avoid overestimates of $R_r(i)$ by using a small $\delta_b$, especially in the post-shaping estimates. If $B(i) = 0$ for some $i$, we set $R_r(i) = 0$. We ignore the TCP handshake and termination exchanges when computing $R_r$.

Note that in the above description, we assumed that $R_r(i)$ is estimated using the application packet trace at $\mathcal{RCV}$. We have implemented the previous method for a trace captured at $\mathcal{SND}$ by observing sequence numbers of TCP ACKs from $\mathcal{RCV}$. This works even in the case of delayed ACKs, since we treat each ACK as a *logical packet* at $\mathcal{RCV}$, whose size is determined by difference in ACK sequence numbers, and whose receive timestamp is estimated as the receive timestamp of the corresponding ACK. We ignore duplicate ACKs in this computation.

Note that it is unlikely that the TCP ACKs will be paced due to shaping in the other direction during a TCP transfer, since the MSS-to-ACK size ratio (about 28.8 without L2

headers) is much larger than the capacity ratio in asymmetric links.

### 4.6.1 Detection

We use a sliding window of size $w$ points on the rate timeseries $R_r$ to detect shaping. The value of $w$ is chosen small so that we can observe level shifts and constancy of a few points after the level shift (without noticing TCP-based rate drops). Suppose that $w$ is odd and let $w = 2k + 1$ points. We say that there is a level shift *due to shaping* at a given point $\tau$ if all of the following conditions hold true in a window $w$ centered at $\tau$ (we require that $\tau > k$).

First, $k$ rate values before $\tau$ are larger than the $k$ rate values after $\tau$:

$$\min_{i=(\tau-k)\ldots(\tau-1)} R(i) \geq \max_{j=(\tau+1)\ldots(\tau+k)} R(j) \tag{13}$$

Second, the *aggregate* throughput of $w$ points before $\tau$ is significantly larger than the *aggregate* throughput of $w$ points after $\tau$: [10]

$$\bar{R}_r(i)_{i=(\tau-w)\ldots(\tau-1)} > \gamma \; \bar{R}_r(j)_{j=(\tau+1)\ldots(\tau+w)} \tag{14}$$

for a threshold $\gamma > 1$. We use aggregate throughput and not median, since $w$ is small. Third, we have received at least one packet in each $\Delta$ interval after $\tau$ in a $w$-window:

$$R_r(i) \neq 0,\ i = (\tau + 1) \ldots (\tau + w) \tag{15}$$

This condition allows us to discard some cases of TCP timeouts. Finally, we require *constancy of rate* in the $w$ points after $\tau$. We elaborate on this condition next.

**Rate constancy:** This condition differentiates TCP recovery dynamics after a loss/timeout from the output of a shaper just after it runs out of tokens. The property we test is that just after the shaper runs out of tokens, incoming packets are buffered and serviced at the rate $\rho$ (for a sufficiently small duration), since the incoming rate is larger than $\rho$. In other words, in this period, the inter-packet gap will be about $S/\rho$, where $S$ is the current packet size. In practice, however, some token bucket implementations serve fixed-size bursts of packets during a shaping period. The size of this burst depends on the inter-token gap $\delta_{tb}$ and the

[10] If $\tau$ is such that we have less than $w$ points before or after it, we consider only those points.

token rate $\rho$, and each burst is served at the link capacity $C$, with a large inter-burst gap. We account for this temporal burstiness in our rate constancy heuristic.

The basic idea is to look at "variability" of a timeseries of cumulative bytes, $B_c(t)$, received at $\mathcal{RCV}$. The variability would be higher for TCP recovery than for output of a shaper. In order to construct $B_c(t)$, we identify bursts of packets served at the link capacity. We identify a burst as the largest set of consecutive packets in which each packet pair is received at a rate higher than $0.9C$. For each identified burst, we add a point $(t, B)$ to $B_c(t)$, where $t$ is the timestamp of the first packet in the burst, and $B$ is the cumulative bytes received up to the end of that burst. At the end of the window $\tau + w$, if $B_c(t)$ contains less than 5 points, we say that the post-shaping rate is *not constant*.

Once we construct $B_c(t)$, we sample uniform randomly pairs of points $(t_1, B_c(t_1))$ and construct the slope $m_1$ of the line connecting them. We construct 500 such samples (with replacement), and estimate the *nonparametric variability* of the set of slopes $M = \{m_1 \dots m_{500}\}$. The variability estimate $v(M)$ of $M$ is defined as $(M_{0.9} - M_{0.1})/M_{0.5}$, where $M_p$ is the $p$-percentile value in $M$. We avoid using the parametric equivalent, coefficient of variation, since it can be biased by few "outlier" slope samples. We say that the rate post-shaping is *constant* if, for all packets received in the time interval $[\tau + 1, \tau + w]$, $v(M) < 0.15$. We found that a threshold of 0.15 yields a low false positive and false negative detection rate in our experiments with NDT traces from various ISPs (for details on the methodology, see Section 4.6.4).

### 4.6.2 Estimation

Once we detect the presence of shaping at point $\tau$, we determine the shaping parameters as follows. The shaping rate $\rho$ is estimated as the aggregate received rate in a window of $w$ points after point $\tau$ (or $m$ points if there are $k \leq m < w$ points after $\tau$):

$$\hat{\rho} = \underset{j=(\tau+1)\dots(\tau+w)}{\bar{R}_r(j)} \tag{16}$$

The link capacity $C$ is estimated as the aggregate received rate in $w$ points before $\tau$:

$$\hat{C} = \underset{i=(\tau-w)\dots(\tau-1)}{\bar{R}_r(i)} \tag{17}$$

It is relatively challenging to estimate the token bucket depth $\sigma$ since the TCP send rate can be variable, and the arrival rate at the shaper can in fact be lower than $\rho$ even before shaping starts. Given the received rate timeseries $R_r(i)$, the size of tokens in the token bucket at time $i$ is a function of the size of tokens at time $i-1$, the received rate at time $i$, and the token rate $\rho$ (and bounded by the bucket depth $\sigma$). It is given by the recursive relation:

$$\hat{\sigma}(i) = \max\left\{\min\left\{\hat{\sigma}(i-1) - \left[R_r(i) - \rho\right]\Delta, \sigma\right\}, 0\right\} \tag{18}$$

where, $\sigma$ is the bucket depth, $\hat{\sigma}(0) = \sigma$, and $\hat{\sigma}(\tau+1) = 0$. We estimate the bucket depth as $\hat{\sigma} = \hat{\sigma}(0)$.

Equation 18 is nonlinear and depends on $\sigma$ itself. We solve for $\sigma$ as follows. We construct a function $\hat{\sigma}(\tau+1) = f_\sigma(\rho, \tau, R_r, \Delta, \hat{\sigma})$ that gives the value of token bytes after shaping, $\hat{\sigma}(\tau+1)$, given a value for $\sigma = \hat{\sigma}$. The function $f_\sigma$ has the following properties: (1) it is non-decreasing as a function of $\hat{\sigma}$, (2) $f_\sigma \approx 0$ for $\hat{\sigma} \leq \sigma$ and (3) $f_\sigma$ increases monotonically with $\sigma$ for $\hat{\sigma} > \sigma$. We solve for $\sigma$ using binary search on an interval $[\sigma_{\min}, \sigma_{\max}]$ such that $\sigma$ is the maximum value of $\hat{\sigma}$ for which $f_\sigma \approx 0$.

In our implementation, we use $\sigma_{\min} = 10$KB, $\sigma_{\max} = 100$MB, and define the approx. relation above as $f_\sigma < 5$KB (close to the minimum bucket size on commercial implementations, as seen in NDT traces). Since binary search finishes in logarithmic time, it is possible to implement the above estimation in a live TCP capture.

### 4.6.3 Parameter Selection

We describe how we choose parameters $\Delta$, $w$, and $\gamma$, based on our observations using NDT data from different ISPs.

**Averaging interval:** The value of $\Delta$ has to be large enough so that we construct a low variance rate timeseries, and is defined as follows. We determine $\Delta$ as the *largest* value for which we do not have any zero rate points in $R_r$. In addition, we bound $\Delta$ so that $\Delta_{\min} < \Delta < \Delta_{\max}$. In our implementation, we use $\Delta_{\max} = 250$ms and $\Delta_{\min} = 30$ms based on the typical range of RTTs on the Internet.[11] The above upper bound on $\Delta$ implies that

[11] We could also choose $\Delta_{\min}$ by estimating the connection's round-trip time at the time of the TCP

for a given TCP trace, we may still have zero rate points in $R_r$.

We efficiently determine the value of $\Delta$ using binary search. For cases where the trace is collected at the sender $\mathcal{SND}$, we use $\Delta_{\max} = 500$ms to allow for TCP implementations that use delayed ACKs.

**Window size:** The value of $w$ is chosen based on the estimate of $\Delta$ and the trace duration $\Lambda$, as the number of $\Delta$ time intervals that that fit in a fraction of $\Lambda$: $w = \kappa\Lambda/\Delta$ points for some $\kappa < 0.5$. We choose $\kappa = 0.2$ in our implementation, and round-off $w$ to the nearest odd integer. Note that as $\Delta$ increases, $w$ decreases to accommodate the same window duration.

**Rate ratio:** The value of $\gamma$ is chosen so that we accommodate as many shaping cases as possible. We found that $\gamma = 1.6$ helps in eliminating false positives due to TCP-based rate drops. This value is larger than $\gamma$ in our active detection method; lowering $\gamma$ to improve detection accuracy remains an open question.

**Trace duration:** We can fix the measurement duration $\Lambda$ so that we detect as many ISP shaping configurations as possible, while at the same time keeping the total experiment duration reasonable in case of no-shaping. We see from Figure 16 that the burst duration is at most 48s, except for four tiers out of 36. We choose $\Lambda = 60$s in our implementation.

### 4.6.4 Evaluating Passive Detection

We use traces collected from M-Lab's Network Diagnostic Tool (NDT) [14] from 25 April to 27 April, 2010. NDT performs a TCP bulk-transfer in upstream and downstream directions for 10s and collects per-packet traces at the server side kernel. Incoming NDT clients are redirected to the nearest M-Lab NDT server instance based on latency.

We consider only finished runs of NDT, which have an upstream/downstream run duration of exactly 10s. We analyze traces collected in the client upstream direction, since the shaping may be more likely to start in the limited duration of 10s in the upstream than in the downstream direction (according to ShaperProbe observations).

**Accuracy:** To benchmark the accuracy of our passive method, we collect 452 NDT

handshake.

upstream traces from March 11, 2010 from an M-Lab server in the US. We manually classify each trace into shaping and non-shaping categories based on visual inspection of the rate timeseries. The averaging interval to plot the rate timeseries is based on the passive algorithm for determining $\Delta$. We found a total of 12 shaping cases out of 452 cases. Our method detects 11 of 12 cases as shaping, and all of the remaining 440 non-shaping cases as non-shaping. Note that the number of shaping cases is small (2.65%), since we are using a small experiment duration (10s), and in addition, the onset of shaping can be delayed due to TCP send rate variations.

We have also run the passive method on some of the ShaperProbe traces to verify that the two methods give consistent results.

**Traces:** We now look at a larger set of NDT upstream runs collected on 25-27 April 2010 for which we have ASN to ISP name mapping (we use sibling data from [126]). We have a total of 148,414 upstream runs from 900 ISPs. These traces are collected from *all* M-Lab servers, and correspond to clients from countries including US. The top ISP shaping detections by number of runs are Comcast, Road Runner, Telecom Italia, AT&T, Verizon and Cox.

We found that we see 0-2.1% upstream shaping detections for ISPs in the list, except for Cox, which had 20.5% (615 out of 2,998 runs) detections (Table 5). For example, we saw 2.1% (284 out of 13,536) shaping detections in Comcast. Note that we observed less than 15% shaping detections for AT&T, Road Runner, and Verizon in the ShaperProbe data. This may be because of the experiment duration of 10s - for example, in the case of Comcast (see Table 4(a)), we see that the minimum duration it takes to empty the token bucket among the upstream configurations is 15.2s (with a constant bit-rate stream sent at the link capacity); hence, we would not be able to see shaping in a 10s Comcast upstream trace. The burst duration may be longer with TCP rate variations, and we also require a minimum duration of $k\Delta$s after shaping. The fraction of shaping detections also depends on the distribution of runs among the tiers of the ISPs during the three-day data collection period. Under the hypothesis that all upstream tiers of Comcast exceed the 10s burst duration (or do not have shaping), we can say that the false positive detection rate for the

**Table 5:** Passive shaping detections (upstream): top ISPs by runs.

| ISP | Detections |
|---|---|
| Comcast | 284/13536 (2.1%) |
| Road Runner | 4/8686 (0.04%) |
| Telecom Italia | 0/8330 (0%) |
| AT&T | 46/8110 (0.57%) |
| Verizon | 1/5607 (0.02%) |
| Cox | 615/2998 (20.5%) |

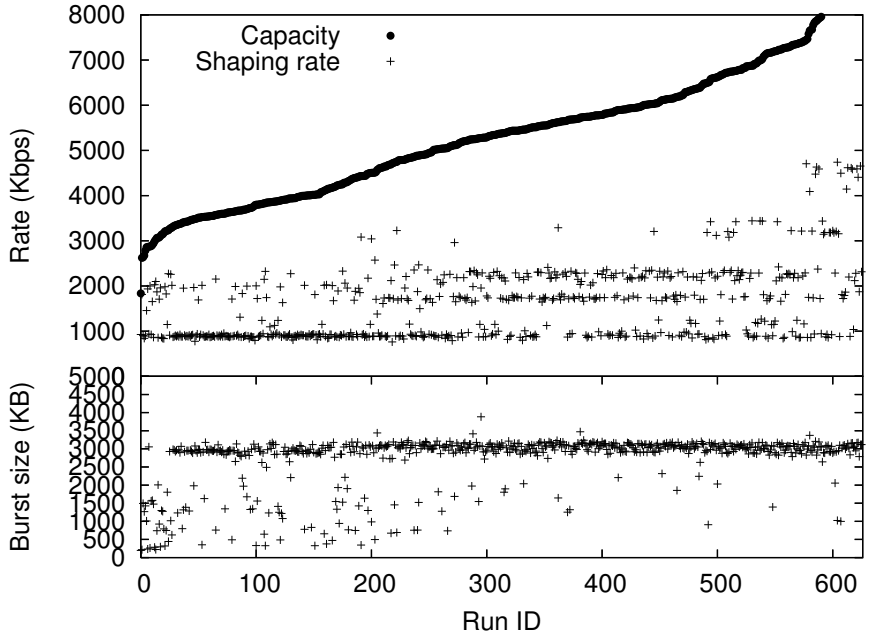



**Figure 33:** Cox: visualizing upstream shaping estimates using passive methods. The modes in this plot are similar to those obtained by active probing methods (Figure 28).

passive method in cable upstream is about 2%.

#### *4.6.4.1 Case Study: Cox*

Cox provides residential [9] and business Internet access using cable and Ethernet access technologies. Cox shapes traffic at different tiers for both residential and business classes. We found that the residential shaping rates and capacities are dependent on the location.

Figure 33 shows shaping configurations of the Cox runs. The plot shows strong shaping rate and burst size modes similar to Figure 28, which is the counterpart using active probing (we show the same capacity ranges for the two plots). We see that some tiers are not sampled in this data as compared to those in ShaperProbe, since the latter is more historical. All

tiers in this plot have a burst duration less than 10s - for example, for all runs with the 1Mbps shaping rate, the average capacity is 4.82Mbps; and with a 3MB token bucket size, the expected burst duration is 6.6s. We found the following advertised tiers on the Cox website (in May 2010) for shaping rates of 1Mbps and 2Mbps: $\{C,\rho\}$ of {1.3, 1}Mbps, {2.5, 1}Mbps, {2.5, 2}Mbps, and {3, 2}Mbps. We see that the range of the estimated capacities for these two shaping rates are higher than the advertised rates. We plan to collect historical NDT traces to investigate this observation as a part of future work.

### 4.6.5 Shaping and TCP Speed Tests

We end this section with a note on TCP-based speed tests. Traffic shapers can impact throughput measurements from "speed tests" that rely on TCP, and typically use a bulk-transfer lasting about $\Lambda = 10$s. In the presence of a traffic shaper with a burst duration of $b$, a TCP-based "speed test" may either: (1) form an estimate of the link capacity if $b \geq \Lambda$, or (2) report a rate between the capacity and shaping rate if $b < \Lambda$. TCP-based speed tests should be aware of traffic shaping (possibly providing the user with a detailed diagnosis of "speed"), and in that case, should be run for a duration $\Lambda$ larger than the typical configurations of burst duration.

## *4.7 Summary*

In this chapter, we presented end-to-end active and passive methods for detection and estimation method of traffic shaping in ISPs. Our evaluation using controlled experiments and in two known ISP deployments shows that the methods have low false positive and false negative detection rates. We presented a first large-scale study of shaping at four large ISPs, and validated some of our observations using ISP advertised tier data. A strong modality of shaping rates and burst sizes suggests that ISPs typically deploy a small set of shaping configurations. We found some shaping detections for which the ISPs do not mention shaping in their service descriptions[12]. Lack of publicly-available information, however, does not necessary imply that these are false detections. We have also validated

[12]ISPs, however, typically mention in their SLAs that "listed capacities may vary".

results of the passive detection method with ShaperProbe data for an ISP.

The ShaperProbe client source are binaries are available at `htt://www.measurementlab.net` and at `http://www.netinfer.net/shaperprobe`. Parts of this work appeared in prior publications [63, 64].

# CHAPTER V

# SPECTRAL PROBING AND INFERRING SHARED LINKS

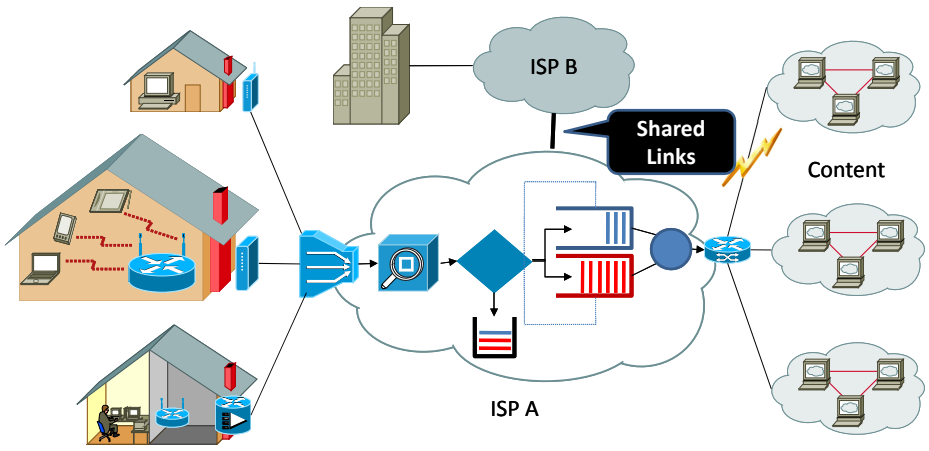



**Figure 34:** Internet model showing shared links between Internet paths. Shared links can be a potential source of performance problems.

## *5.1 Introduction*

Intra-domain and inter-domain networks are a complex mix of highly inter-connected forwarding devices; and these devices may operate in different layers of the protocol stack (L3+, L2 and PHY). The complexity grows as the network evolves with time. A critical part of the network topology in terms of performance is that of *shared links between end-to-end paths*. Shared links can become a potential bottleneck, and a source of performance problems, for one or more paths that traverse those link(s). In general, the actual topology of a network is rarely known even to network operators, especially when an end-to-end path crosses several administrative domains and when it includes devices at different layers. It is often important, for network troubleshooting, failure-risk analysis, QoS provisioning, overlay routing, etc., to know whether two or more network paths share any infrastructure. In this chapter, we design and test methods for efficient end-to-end detection of shared links among two or more Internet paths using a single probing session.

Our work overcomes several limitations of existing tools. The *traceroute* tool is commonly used for topology inference. It is, however, not suitable for detecting sharing between a pair of paths for several reasons: it only detects layer-3 devices, it is often disabled behind firewalls and NATs, and it can fail to detect that two IP segments may be going through the same router, sharing the same queue.[1] Shared congestion detection methods, discussed in Section 2.3, aim to detect whether two or more paths (or TCP flows) share the same *congested queue*. Our methods do not require or assume that the shared queue(s) is congested. In the remainder of this section, we introduce the Spectral Probing framework and some of its applications (our focus is on the shared link detection application).

In analog communication systems, signals and channels, spectral (Fourier) analysis, multiplexing, frequency modulation, and channel interference (or crosstalk) are fundamental concepts. Even though these concepts are also very important in the physical layer of packet networks, they have been traditionally viewed as foreign to the network, transport and application layers of the protocol stack. This chapter proposes a manifestation of the above concepts in IP-based packet networks. Specifically, we introduce a *Spectral Probing (SP)* framework that combines techniques for frequency-based signal generation in packet networks, frequency multiplexing of diverse signal sources, (intentional) crosstalk between network paths, and frequency modulation. Before we present applications of the SP framework, we start by introducing its key underlying ideas.

First, let us consider a network path from host A to host B as a *channel* P(A,B), assuming for now that this path is unique and constant for the time-window of interest. The obvious information-carrying signal is what is contained in the data packets sent on this path. The signal we are interested in, however, is more subtle: it is the *end-to-end queueing delay variations* on P(A,B). Specifically, suppose that we sample the one-way delay variation in P(A,B) using periodic or Poisson measurements. In the absence of any signal-carrying traffic from A to B, these measurements can be viewed as *noise*. It is caused by cross traffic at the queues along the path P(A,B). The measured queueing delay variations can be analyzed in the frequency-domain using standard Fourier analysis techniques to

[1] We show one such case in section 5.5.

characterize the spectral properties of the corresponding timeseries. Prior studies have looked at spectral properties of Internet delays (see related work, Section 2.3).

A new idea, however, is that we can generate information bearing signals on P(A,B) by sending specially crafted traffic that generates desired patterns in the end-to-end delay timeseries. As in analog communication systems, a frequency-based signal is more robust to additive noise than an amplitude-based signal [34]. So, the basic idea is that *we can cause periodic queueing delay increases in P(A,B) by sending periodically, every* $T_p = 1/f_p$ *seconds, a packet train with sufficiently high rate and size.* As explained in detail in the next section, if the transmission rate of the packet train is higher than the available capacity in P(A,B), and the frequency $f_p$ is below a certain threshold, then the end-to-end delay timeseries will acquire a periodic component, a signal in other words, at frequency $f_p$.

A second idea is that we can *multiplex signals* of different frequencies in the same bottleneck queue, if that queue can be modeled as a linear time-invariant channel. Each signal can be generated by a different source at A, or it may be that the same source generates multiple frequencies to synthesize a more complex signal. What is important here is that even though these individual signals can overlap in the time domain, they should not overlap in the frequency domain. Further, as in all bandwidth-limited communication channels, there is a limit on how many signals we can multiplex on the same queue.

A third idea is related to interference, or *crosstalk between different network paths.* In analog communications, crosstalk refers to the (most often, undesirable) leakage of signal power from one channel to a neighboring channel through capacitive/inductive coupling. In the context of IP network paths, coupling can take place when two paths P(A,B) and P(C,D) have a *shared bottleneck.*[2] In that case, queueing due to traffic bursts sent by A in P(A,B) can affect the end-to-end delays in P(C,D). In the SP framework, we exploit this coupling to convey frequency-modulated information from one network path to another.

Spectral Probing has several applications that arise from analogies in the analog communications domain. We anticipate that follow-up research will invent even more. In this work we focus on *detecting sharing* application, illustrated next. We briefly touch upon

[2]This term is defined more precisely in the next section.

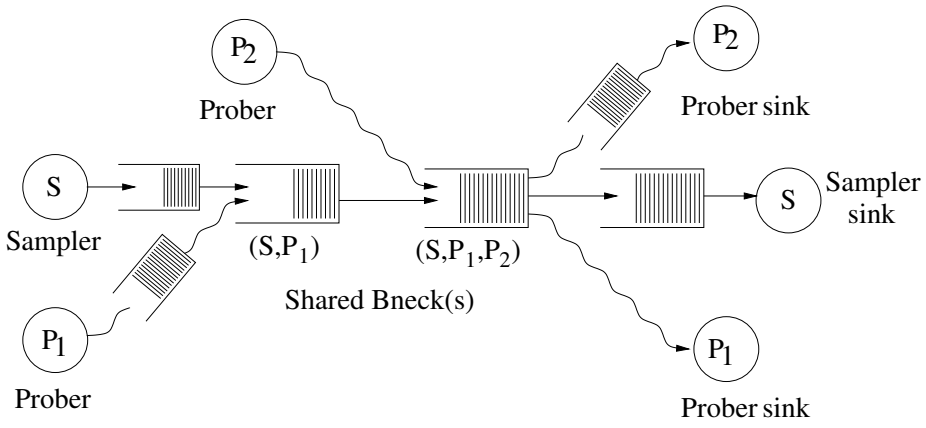



**Figure 35:** The sampler attempts to detect whether it shares a bottleneck queue with each of the two probing paths.

two more applications, *frequency covert channels* and *continuous media modulation*, in the future work chapter (Section 9.2.3). At a high level, a frequency covert channel between two hosts is based on crosstalk, and does not require any form of direct or indirect communication between the hosts. Continuous media modulation potentially reduces the likelihood of queueing delays by using frequency modulation.

The SP framework can be used to *detect shared store-and-forward devices* (links, routers, middleboxes, etc) between two network paths, at layer-2 or above, as long as one of the paths (the "probing" path) can induce queueing delays at a packet queue that it shares with the other path (the "sampling" path). Even though the "detection of shared congestion" problem has received significant attention in the past [44, 72, 74, 75, 107], the proposed approach is significantly different because it does not require that the shared link(s) is/are congested. Instead, the proposed method can detect sharing even when the shared link(s) is/are lightly utilized as long as the sender at the probing path can cause short-term queueing at the shared link(s).

The proposed method can be generalized, using frequency multiplexing, to detect whether each of N probing paths has *a shared bottleneck*, in the previous sense, with the sampling path. Figure 35 illustrates this application in the case of one sampler and two probers.[3] The probing path of $P_1$ has two shared bottlenecks with the sampler. The second bottleneck is

[3]We use the terms "prober" and "probing path" interchangeably. Similarly for "sampler" and "sampling path".

also shared by the probing path of $P_2$.

The proposed method in this chapter can only detect a subset of the possible types of sharing. For instance, we obviously cannot detect sharing at the physical layer. We detect sharing only when one path (the prober) can cause queueing (as in short-lived delay increases; not long-term congestion) in a packet queue of a network device that is shared by another path (the sampler).

**Chapter outline:** In Section 5.2, we describe the SP model and give basic constraints on the parameters of this probing method. In Section 5.3, we describe the signal processing algorithm that the sampler conducts to detect whether it has a shared bottleneck with one or more probers. In Section 5.4, we rely on simulations to evaluate the accuracy of the proposed shared bottleneck detection method, and to examine the impact of certain key parameters. In Section 5.5, we demonstrate that the method works "in the wild" with experimental results from several Internet paths. We conclude in Section 5.7.

## *5.2 Spectral Probing*

This section describes the basic model behind the SP framework. We first explain under which conditions a network queue can be viewed as a linear time-invariant channel. Next, we present basic constraints on the probing frequency, probing intensity, and selection of simultaneous probing frequencies. We are then ready to describe more precisely the shared bottleneck detection problem. Finally, we describe the end-to-end sampling process and give a constraint on the minimum sampling frequency.

### 5.2.1 Linearity approximation

The frequency multiplexing of different signals on the same channel requires that the channel can be modeled as *linear* (i.e., a linear combination of two or more input signals produces an output that is the corresponding linear combination of the individual outputs) and *time-invariant* (i.e., the input/output relation does not vary with time).

A packet queue, however, is *not* a linear system. To see why, consider a single FIFO queue with service rate $C$ bits/sec and buffer size $B$ bits (denoted as {C,B}). The queue size (or backlog) $q(t_k)$ at time $t_k$ can be determined by the following discrete-time equation

for given initial conditions:

$$q(t_{k+1}) = \min\{\max\{q(t_k) + a(t_k) - C\Delta t_k, 0\}, B\} \tag{19}$$

where $a(t_k)$ is the amount of arriving traffic at $t_k$, and $\Delta t_k = t_{k+1} - t_k$. Obviously, the output variable $q(t_{k+1})$ is *not* a linear function of the input $a(t_k)$.

For a different selection of the output variable, however, and under certain conditions and approximations, a packet queue can be modeled as a linear system. Specifically, consider the following function:

$$q_\delta(t_k) = \{q(t_{k+1}) - q(t_k)\}^+ \tag{20}$$

where $\{x\}^+ = x$ if $x > 0$, and zero otherwise. We refer to $q_\delta(t_k)$ as the *backlog increase* function. In the following, we will assume that the arriving traffic is such that the buffer size $B$ rarely causes losses. Note that $q_\delta(t_k)$ is positive only when the amount of arrivals at $t_k$ exceeds what can be serviced in $\Delta t_k$:

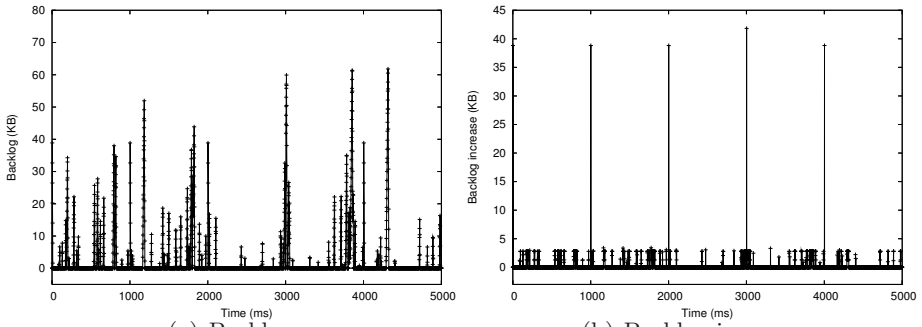
$$q_\delta(t_k) = \{a(t_k) - C\Delta t_k\}^+ \tag{21}$$

If we assume that the time interval $\Delta t_k$ is very small and that the arrivals are bursty (packets arrive instantaneously), we have that $a(t_k)$ is either zero, or it is $a(t_k) \gg C\Delta t_k$. Hence, if one or more packets arrive at $t_k$ the backlog increase function becomes approximately equal to the arrived traffic, otherwise it is zero:

$$q_\delta(t_k) = \{a(t_k)\}^+ \tag{22}$$

So, the backlog increase function for the sum of two or more signals $a_1(t_k) + a_2(t_k) + \ldots$ is the sum of the corresponding backlog increase functions $q_{\delta,1}(t_k) + q_{\delta,2}(t_k) + \ldots$. The queue can be further modeled as time-invariant, as long as its capacity remains constant.

There are three practical implications from the previous analysis. First, the output variable that we should be monitoring is the backlog increase function, not the backlog function. Second, the probing signals should be as bursty as possible, i.e., the probing traffic should arrive at the bottleneck queue with the highest possible rate. In the SP framework, we create a backlog increase signal by sending a sequence of $L$ back-to-back



(a) Backlog (b) Backlog increase

**Figure 36:** Example of a backlog function with the corresponding backlog increase function.

packets (a "packet train"). If the rate with which the packet train arrives at the {C,B} queue is $R_{in}$, we can guarantee a backlog increase as long as $R_{in} > C$; the higher $R_{in}/C$, the better we approximate the previous linearity condition. So, it is important in practice that the probers have high-capacity network interfaces and they are connected to the Internet through high-capacity paths. Third, our signal transmitters should react to packet losses by decreasing their train length, so that the frequency of buffer overflows remains very low.

Figure 36 illustrates the functions $q(t_k)$ and $q_\delta(t_k)$, simulating a single queue (C=50Mbps) with random cross traffic (ON-OFF with exponential burst/idle durations) and four packet trains (L=30 packets, 1500 bytes each) at times 1000, 2000, 3000 and 4000. Note that the backlog function is noisy and it is hard to distinguish the probing signal from the cross traffic spikes. The backlog increase function, on the other hand, acts as a nonlinear filter, setting to zero the decreasing parts of the timeseries and reducing the magnitude of any gradual backlog increases.

### 5.2.2 Probing frequency

The following are constraints on the maximum and minimum probing frequency. Suppose that we want to create a periodic backlog increase signal of frequency $f_p$ at a queue {C,B}. As previously mentioned, we can periodically send, every $T_p = 1/f_p$ seconds, a packet train that consists of L packets. Say that each packet has a size of $s_p$ bits, and so the total size of the packet train is $S = s_p L$.

A first condition is that, even if there is no other traffic at that queue, it will take

$\Delta_p = S/C$ to transmit a packet train. In the presence of cross traffic, $\Delta_p$ can be larger that $S/C$. So, the probing frequency should be much lower than $1/\Delta_p$, or otherwise successive packet trains will overflow the queue:

$$f_p \ll \frac{C}{S} \tag{23}$$

In other words, we can think of a packet queue as a *band-limited channel with a cutoff frequency that is at most $C/S$.*

Another reason to impose an upper bound on the probing frequency is to limit the average traffic rate of the probing signal, which is $f_p S$. Suppose that $R_{max} (\ll C)$ is the maximum average probing rate that we are allowed to generate. Then, the probing frequency should be upper bounded so that the average rate of the probing signal is lower than the maximum allowed rate:

$$f_p < \frac{R_{max}}{S} \tag{24}$$

It is important that the probing frequency is also not too low for two reasons. First, the spectral density of Internet traffic increases as $1/f$ as the frequency f tends to zero [43]. Thus, we should expect significant cross traffic noise in low probing frequencies. In practice, we have observed in many Internet paths that spectral noise increases significantly below the frequency of 1Hz. Figure 37 shows the spectrum of a timeseries of one-way delay variations at an Internet path. Note how the spectrum increases in lower frequencies, especially below approximately 1Hz.

Second, suppose that the duration of our signal is $\Delta_e$. For practical reasons, related to measurement latency or variations in the underlying network conditions, we would like $\Delta_e$ to be as short as possible. For a given $\Delta_e$, the number of signal iterations (number of packet trains) for a probing frequency $f_p$ is $f_p \Delta_e$. Thus, if $f_p$ is too low, we will not have enough signal iterations to accurately detect the signal in a noisy spectrum. In practice, we often set $\Delta_e$=60sec and require at least 50-60 signal iterations. Thus, a probing frequency of around 1Hz seems to be a good lower bound in practice.

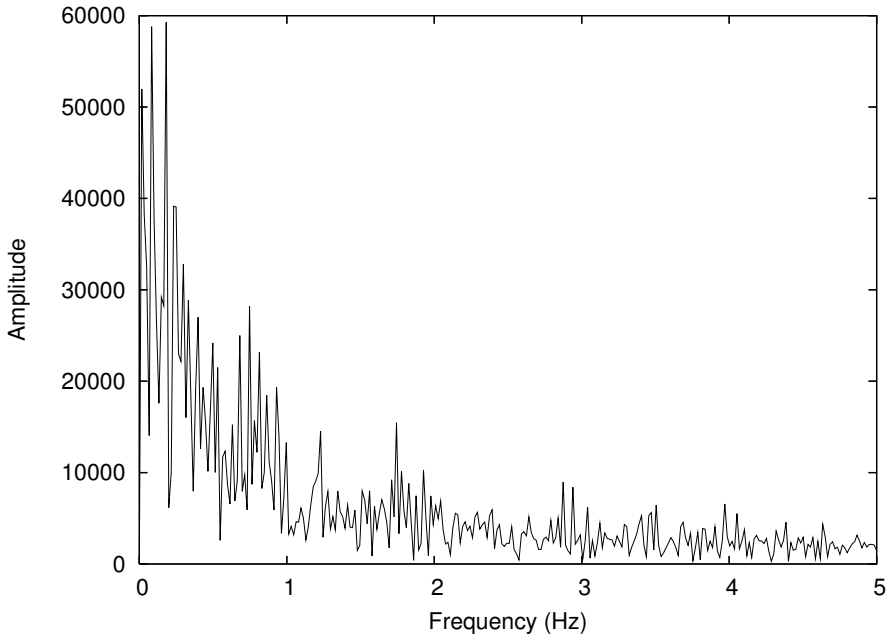



**Figure 37:** The spectrum of the one-way delay variations at an Internet path from procyon.cc.gatech.edu to www.dpls.lib.or.us showing large amplitudes at frequencies below 1Hz.

### 5.2.3 Probing intensity

The transmitted signal power is determined by the size of the packet trains. There are three factors we need to consider here. First, the signal should be stronger than the noise intensity so that the receiver can detect the signal. Suppose that a packet train {S,$R_{in}$} arrives at the queue {C,B} with rate $R_{in} > C$. The minimum (i.e., in the absence of any cross traffic) backlog increase that the train will induce in the queue is $S(1 - C/R_{in})$. So, the packet train will create a queueing delay increase that is at least $\hat{d}_p = \alpha S/C$, where $\alpha = 1 - C/R_{in}$. Suppose that the cross traffic induces queueing delays that are typically less than $\hat{d}_n$ (we define $\hat{d}_n$ more formally in section 5.4). The *Signal-to-Noise Ratio* (SNR) is defined as $\hat{d}_p/\hat{d}_n$. We require that the SNR should be larger than a threshold $\gamma \approx 1$; in other words, we require that the minimum signal-induced delays should be at least as high as the larger delays induced by cross traffic. Then, a lower bound for the train size is:

$$S > \frac{C\,\hat{d}_n}{\alpha} \tag{25}$$

A second consideration is that the packet trains should not be causing packet losses. In our prototype, the probers decrease L when there are packet losses at the receiver (detected

using sequence numbers).

A third constraint is that the signal $\hat{d}_p$ should not be much higher than the queueing delays induced by cross traffic. In other words, the SNR should not be much higher than $\gamma$. This is important both for not being intrusive to cross traffic, but also for "stealth operation" when the SP framework is used to create covert channels of communication.

### 5.2.4 Multiple probing frequencies

Suppose that we want to multiplex several probing signals on the same queue. The basic idea is that each prober $i$ will be sending trains at a frequency $f_p^{(i)}$, creating periodic delay increases of that frequency on the queue. As long as the queue can be modeled as a linear channel, the multiplexing of distinct frequencies will be a reversible process. Here, we examine the constraints for the selection of simultaneous probing frequencies. First, as previously discussed, there is a lower bound (about 1Hz) and an upper bound ($R_{max}/S$) for any probing frequency. Further, any two probing frequencies should differ by at least a *guard band* $g$ so that the receiver can accurately distinguish them. The magnitude of the guard band depends on the exact detection process, and it is discussed in more detail in the next section.

The signal detection process that we propose in the next section does not only rely on the *fundamental frequency* $f_p^{(i)}$ of each probing signal, but it also leverages the first H harmonics of that signal.[4] So, an additional requirement is that the H+1 frequencies of each probing signal (the fundamental frequency and the first H harmonics) should differ by at least one guard band $g$ from the H+1 corresponding frequencies of any other probing signal. Formally, we require that:

$$|m\, f_p^{(i)} - n\, f_p^{(j)}| \geq g \tag{26}$$

for any probing frequencies $i \neq j$ and all $m, n = 1, \ldots H+1$. The appropriate value of $g$ depends on the exact detection process used. In our implementation, a value of g around 0.1-0.2Hz is sufficiently large.

[4]Recall that the harmonics of a periodic signal of frequency f appear at the integer multiples of that frequency.

We determine numerically the maximum number of probing frequencies that can be multiplexed on a queue based on the previous constraint. For instance, for a minimum probing frequency of 1Hz, a maximum probing frequency of 4.76Hz, g=0.167Hz and H=4, we can multiplex five simultaneous probers.

### 5.2.5 The shared bottleneck detection problem

Now that we have described the main ideas behind spectral probing, we can define more precisely the shared bottleneck detection problem. Suppose that the "sampling" path is $P_s$ and the "probing" paths are $P_p(1), \ldots P_p(N)$ with $N \geq 1$. Each probing path may share one or more queues with $P_s$. Note that two or more probing paths may be identical. Also, it may be that one or more probing paths overlap with the sampling path.

We say that a probing path $P_p(i)$ has a *shared bottleneck* (queue) with the sampling path if $P_s$ and $P_p(i)$ share a packet queue and the packet trains transmitted by $P_p(i)$ can cause a backlog increase in that queue. As previously discussed, if the packet trains transmitted by $P_p(i)$ reach a shared queue {C,B} with rate $R_{in} > C$, then that queue is a shared bottleneck. If there is cross traffic in the shared queue, then it is possible that the queue will be a shared bottleneck even if $R_{in} < C$, as long as $R_{in}$ is higher than the queue's available capacity.

In the shared bottleneck detection problem, the sampler aims to infer, relying exclusively on end-to-end information, whether each of the N probing paths has a shared bottleneck with $P_s$. Note that it is possible that there is sharing between a prober and the sampler, but the joint queue(s) is/are not a shared bottleneck. It is also possible that a probing path has more than one shared bottleneck with the sampling path. Also, the shared bottleneck is not necessarily the link with the minimum available bandwidth in that path.

### 5.2.6 Sampling process

The sampler cannot measure the instantaneous backlog at each queue of its path. It can measure, however, the end-to-end delay variations in its path. From those delay variations it aims to detect the presence of periodic increases at specific frequencies, and thus to infer which of the $N$ probing paths have a shared bottleneck with $P_s$.

Specifically, suppose that the sampler sends a periodic packet stream through $P_s$. Let $f_s$

be the frequency with which the sampler sends packets, and $s_s$ be the size of those packets.[5] Say that $d(j)$ is the time difference between the receiver and sender timestamps for the $j$'th sampling packet. Note that clock synchronization between the sampling sender and receiver is not required; and the measurements $d(j)$ will include the clock offset between the two hosts. It is important, however, to compensate for any clock skew (variable clock offset) (as described in prior work [94] for instance). If we only have access at the sender of the sampling path, then we can rely on Round-Trip Time (RTT) measurements using a utility such as *ping*. In that case, however, the reverse channel may affect the accuracy of the method if it introduces significant delay variability.

The receiver of the sampling packets analyzes the following *delay increase* function:

$$d_\delta(j) = \{d(j) - d(j-1)\}^+ \tag{27}$$

Note how the previous function resembles the backlog increase function $q_\delta(k)$ at a single queue. With the delay increase function $\{d_\delta(j)\}$ the sampler aims to detect significant backlog increases at any queue along its path. Instead of searching for a signal at the end-to-end delay variations, which include both positive and negative differences, we aim to only detect *increases* of the measured delays. Delay decreases, on the other hand, are set to zero so that they do not contribute any power in the received spectrum. If we were analyzing, instead, the delay variations $\{d(j)-d(j-1)\}$, there would be significantly higher spectral noise due to all the negative delay differences.

### 5.2.7 Sampling frequency

According to the Nyquist sampling theorem, one would expect that $f_s$ should be higher than twice the largest probing frequency $f_p$. In practice, however, we are working with time-limited signals (recall the earlier discussion about the signal duration $\Delta_e$), while the Nyquist rate assumes infinitely-long signals. Instead, we require that the sampling period is much lower than the transmission latency of a packet train $\{S,R_{in}\}$ at the bottleneck queue $\{C,B\}$ so that the sampling packets can detect the increased queueing delays due to every

[5] In all our simulations and experiments we set $s_p$=1500B using UDP packets.

probing train. The transmission latency of a packet train at {C,B} is at least $\Delta_p$. So, we require that the sampling frequency is much higher, say at least five times, than $1/\Delta_p$,

$$f_s > \frac{5}{\Delta_p} = 5\frac{C}{S} \tag{28}$$

Note that the factor five is empirically chosen (to have a high detection accuracy), based on our Internet experiments and simulations. A higher sampling frequency makes the detection process more accurate but also more intrusive. The packet size of the sampling process can be as small as possible to minimize intrusiveness, and it is typically set to 40B (UDP).

#### 5.2.8 Putting it all together

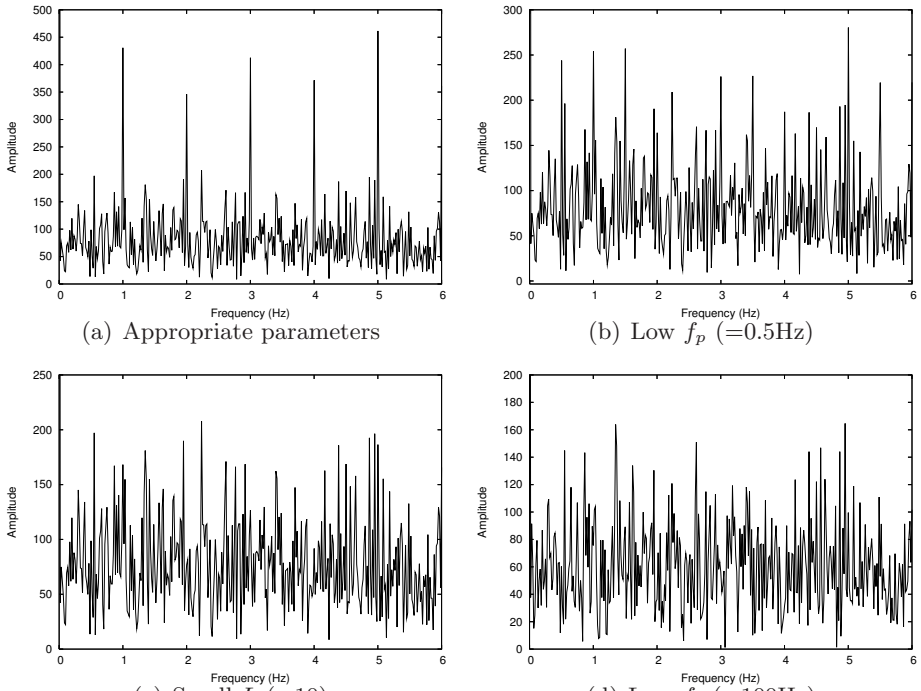
To visually illustrate the significance of the previous parameters, Figure 38 shows the spectrum of four delay increase timeseries (from a simple simulation of a single queue with exponential ON-OFF random traffic). In Figure 38-a we have chosen a probing frequency, train length and sampling frequency based on the previous constrains: $f_p$=1Hz, L=30, and $f_s$=1KHz. Notice the clearly visible spectral spikes at the fundamental frequency 1Hz as well as at its first four harmonics. The next three plots show what happens when one of these important parameters is not sufficiently large. The spectral spikes are not visible, and it would also be hard to detect them with any signal processing algorithm.

### *5.3 Detection Methodology*

In this section, we describe the signal processing algorithm that the sampler follows to detect whether it has a shared bottleneck with one or more probers. The sampler knows the set of $N \geq 1$ probers and the frequency $f_p^{(i)}$ that prober $i$ uses. The goal, then, is to detect which of these $N$ frequencies are present in the sampled signal. The detection scheme should exhibit both high sensitivity (low false negative probability) and high specificity (low false positive probability) for each prober.

#### 5.3.1 Outline

1. The entire experiment is checked in terms of traffic stationarity, path uniqueness and path constancy.




(a) Appropriate parameters (b) Low $f_p$ (=0.5Hz)

(c) Small $L$ (=10) (d) Low $f_s$ (=100Hz)

**Figure 38:** The effect of various key parameters on the probing spectrum.

2. The measured timeseries at the sampler is "conditioned" to deal with lost samples and outliers. Then, we calculate the delay increase timeseries $d_\delta(j)$.

3. We estimate the spectrum of the delay increase timeseries using the Fast-Fourier Transform (FFT) algorithm. The spectrum is then appropriately filtered to reduce the effects of spectral leakage.

4. We infer whether the fundamental frequency, as well as the first H harmonics, of each prober are present in the estimated spectrum. To do so with a given false-positive probability, we also estimate the spectrum of the cross traffic in the sampler path *in the absence of any probing activity.*

5. We combine the H+1 frequency detection outcomes for each prober to infer whether that prober is present. If it is present, we also estimate the false positive probability.

### 5.3.2 Measured timeseries and basic checks

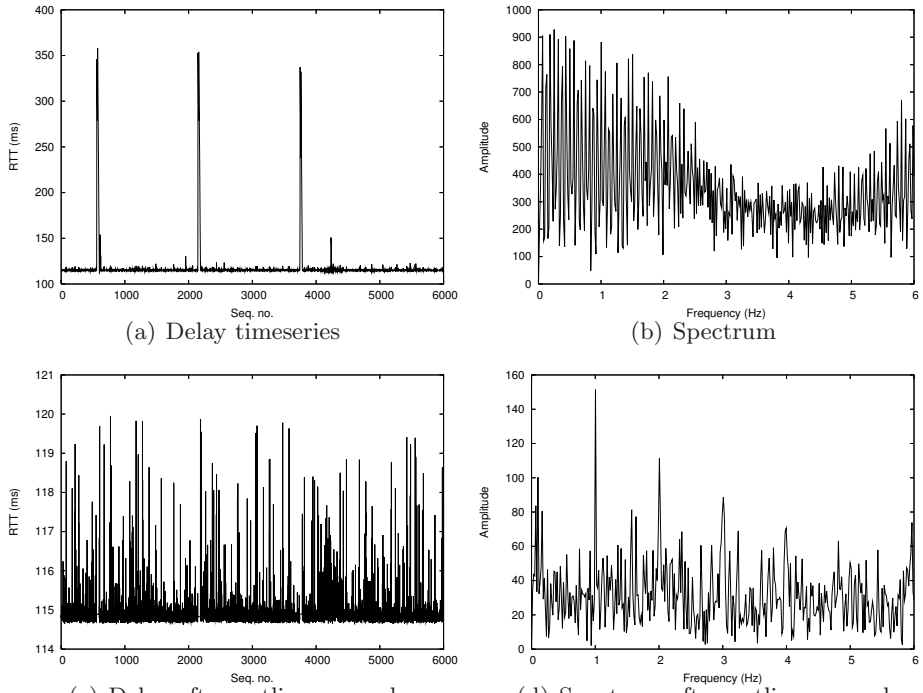
The proposed method involves two timeseries: *preprobing* and *probing*. Both timeseries have the same duration and sampler frequency (and thus the same number of points). The difference is that the former is collected while all probers are idle, while the latter is collected while all probers are active. In the shared bottleneck detection application the probers can communicate with the sampler to coordinate the start of the probing phase. In our simulations and Internet experiments, we set the duration of both timeseries to $\Delta_e$=60sec. The probing timeseries is collected shortly after the preprobing phase is completed.

Each prober, as well as the sampler, use the *Paris-traceroute* utility to examine whether their end-to-end path is unique and stays constant during the measurements [24]. If the underlying routing is not stable, or if it exhibits multipath characteristics even for packets with the same address/protocol/port IP header fields, the shared bottleneck detection problem is not well-defined. We further check whether the received timeseries at the sampler shows clear non-stationarity symptoms, such as major level-shifts. In those cases we abort the measurements. Our Internet experiments have shown that most paths satisfy the previous constraints, but there are few paths in which these constraints are often violated. A 2011 study [23] found per-packet load balancing in less than 3% of Internet paths.

### 5.3.3 Timeseries conditioning and outliers

Because of packet losses, the two timeseries may be missing some samples. The sampler can detect those if the packets carry sequence numbers. Frequently lost samples can affect the spectrum of the received timeseries. In practice, the loss rate in our simulations and Internet experiments is much lower than 1%, and so the effect of lost samples is insignificant. We replace lost samples with the previous successfully received sample.

An important conditioning step is to detect, and somehow remove, outliers from both preprobing and probing timeseries. Outliers can be caused by measurement errors (e.g., large delays introduced by the operating system at the sending or receiving hosts) and/or by sudden and major forwarding delays (not necessarily due to queueing) in forwarding network elements. Outliers can significantly distort the spectrum of the received timeseries,




(a) Delay timeseries (b) Spectrum

(c) Delay after outlier removal (d) Spectrum after outlier removal

**Figure 39:** The significance of outlier removal in spectral analysis. The delay timeseries resulted from ping measurements on the Internet path procyon.cc.gatech.edu to mail-gw.izm.fraunhofer.de.

hiding any periodicities that may exist due to probing. For instance, note how the presence of 3-4 outliers in Figure 39 greatly affects the spectrum of the delay timeseries.

We do not detect outliers based on standard statistical techniques (see prior work [106]). The reason is that those approaches can easily misclassify as outliers the periodic probing delay spikes, especially if the latter are larger than the delays induced by cross traffic. Instead, we develop an outlier detection method that considers the expected number of probing delay spikes.

We start by detecting outliers in the probing timeseries. Suppose initially that a single prober of frequency $f_p$ is present in the probing delay timeseries. During $\Delta_e$ that prober will generate $f_p\Delta_e$ packet trains, each of them causing a delay increase. The basic idea of the method is to estimate an *outlier threshold* $\chi$ so that any delay spike (i.e., a continuous sequence of samples) that exceeds $\chi$ is considered an outlier, *subject to the constraint that the number of detected outliers should be much less than* $f_p\Delta_e$. In other words, $\chi$ is chosen so that it is higher than the magnitude of most probing delay spikes. If $\chi$ was lower than

the magnitude of the probing delay spikes, the number of detected outliers would be more than $f_p\Delta_e$ and we would misclassify all spikes of our probing signal as outliers. In practice, we determine $\chi$ iteratively, starting from the largest delay measurement and reducing $\chi$ in each iteration until the number of detected outliers is more than $h\, f_p\Delta_e$, where $0 < h < 1$. Note that the previous method does not assume that the given prober actually has a shared bottleneck with the sampler; the important point is that, *in case there is such sharing and the corresponding signal is present in the probing timeseries*, we do not misclassify all probing delay spikes as outliers. Second, when multiple probers are active we need to consider the lowest probing frequency because that gives the minimum number of delay spikes we are allowed to detect.

We have found through several Internet experiments, than an appropriate value for $h$ that does not significantly affect true positive detection rate is around 40%. Such a high value of $h$ means that, in the absence of outliers, we may detect a significant fraction of probing spikes as outliers; this is acceptable because, as discussed in the next paragraph, we truncate the amplitude of any detected outliers instead of completely removing them.

After we classify a delay spike as an outlier, we need to modify it somehow so that it does not affect the estimated spectrum. Because it is likely, however, that a fraction of the detected outliers are probing spikes, we do not want to completely remove the corresponding samples from the timeseries. Instead, we "truncate" the spike by replacing each sample of that outlier with the previous sample that is less than the threshold $\chi$. Then, when we construct the delay increase function $d_\delta(j)$, all the successive samples that are equal to $\chi$ give zero difference. So, only the first value of the spike, which is at most $\chi$, will remain in the delay increase timeseries.

After we have detected the threshold $\chi$ from the probing timeseries, we apply the same outlier detection and truncation process on the preprobing timeseries. Finally, we calculate the delay increase function for both timeseries based on Equation 27.

### 5.3.4 Spectrum estimation and leakage reduction

Next, we estimate the amplitude $D_\delta(f)$ of the Discrete Fourier Transform (DFT) of the delay increase timeseries. The basic idea is that if the probing timeseries includes a periodic component due to prober $i$, we will see a significant spectral amplitude at frequency $f_p^{(i)}$ relative to the surrounding frequencies.

To calculate $D_\delta(f)$ we rely on the FFT algorithm, as implemented in Matlab. Recall that if a timeseries consists of $M$ measurements collected with a sampling frequency $f_s$, then the FFT of the timeseries also consists of $M$ frequency points covering the range from 0 Hz (DC component) to $f_s$ with constant spacing $f_s/M$. The spectrum is symmetric around $f_s/2$, and so we only focus on the range $[0, f_s/2)$.

A well-known issue in the analysis of time-limited signals is *spectral leakage*. Because of this effect, even a sinusoid with frequency $f_p$ (say an integer multiple of $f_s/M$) can have some power in frequencies other than $f_p$. A standard approach to reduce the effects of spectral leakage is to multiply the timeseries with a Hamming window before computing the FFT coefficients [30]. The Hamming window smooths the timeseries at its two boundaries, reducing the leakage of spectral power to adjacent frequencies. This is important in the proposed method, where the objective is to detect the presence of a periodic signal of a given frequency comparing the power of that frequency with the power of surrounding frequencies[6]. Figure 40 shows the effects of leakage reduction when we apply a Hamming window with the same duration as the timeseries.

To avoid leakage from high-frequency content that we are not interested in, we also perform low-pass filtering on the received timeseries. The cutoff frequency of the filter is 0.5Hz higher than the H-th harmonic of the maximum probing frequency. The filter that we use is a windowed linear-phase FIR filter of order 200. We also remove the DC component of the spectrum, by subtracting the average from the timeseries.

[6]It should be mentioned that there are more sophisticated methods to estimate the spectrum of a timeseries [96]. Those methods are typically classified as non-parametric (e.g., Welch or multitaper) or parametric (e.g., Yule-Walker or Burg). We found that the DFT-based method is simpler and faster, while other methods are sometimes sensitive to the selection of their parameters.

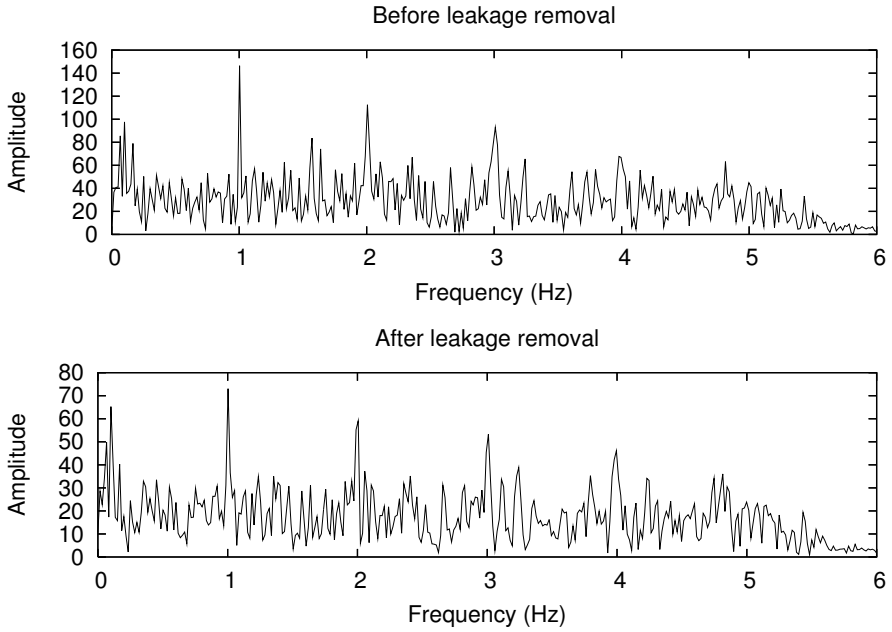



**Figure 40:** Effect of leakage reduction on the spectrum of a probing timeseries from the Internet path procyon.cc.gatech.edu to mail-gw.izm.fraunhofer.de.

### 5.3.5 Signal detection at a single frequency

The next step is to examine the estimated spectrum $D_\delta(f)$ and infer whether a significant periodic signal exists at the fundamental frequency $f_p^{(i)}$ or at the first H harmonics $(h+1)f_p^{(i)}$ $(h = 1, \ldots H)$, for any prober i.

Let us first consider a single frequency $f_p$, assuming it is the fundamental frequency of a certain prober. To examine whether a periodic signal exists at that frequency with a controlled false positive probability, we need to estimate the a priori distribution of $D_\delta(f)$ around $f_p$. We do so using the preprobing timeseries. In that timeseries we know that none of the probers is active, and so any power around $f_p$ will be due to cross traffic noise.

Specifically, let $\tilde{D}_\delta(f)$ be *the spectrum of the preprobing delay increase timeseries.* A naive idea would be to detect the signal if $D_\delta(f_p)$ is much larger than $\tilde{D}_\delta(f_p)$. To make this inference with a given false positive probability, however, we need to consider $\tilde{D}_\delta(f_p)$ as a random variable in an ensemble of preprobing spectra. Let $\Omega(f_p)$ be that random variable. Our aim is to estimate a percentile of $\Omega(f_p)$ such that the false positive probability is, for

instance, at most 20%:

$$\text{Prob}[\Omega(f_p) > \omega_{.2}] = 0.2 \tag{29}$$

where $\omega_{.2}$ is the 20% upper percentile. Then, we can infer that a probing signal exists at $f_p$, with a 20% false positive probability, when the probing spectral power at $f_p$ is higher than $\omega_{.2}$. Later in this section we show how to further reduce the false positive probability by considering several harmonics.

To estimate the percentile $\omega_{.2}$, we consider a frequency band of width 1Hz centered around $f_p$ in the preprobing spectrum. We then estimate the empirical CDF of $\tilde{D}_\delta(f)$ in that frequency band, and use that as an estimate of the CDF of $\Omega(f_p)$. The threshold $\omega_{.2}$ is then calculated as the 20-th upper percentile of that empirical CDF.

To have a more accurate estimate of the signal power $S(f_p)$ at $f_p$ we consider a narrow frequency band of width 0.033Hz centered around $f_p$, instead of considering a single frequency point at $f_p$. The reason is that there may be a mismatch between the exact probing frequency at the sending host and $f_p$. Further, due to operating system jitter, it is possible that the prober does not transmit its packet trains at a constant frequency $f_p$. In practice, we observed that a frequency band of 0.033Hz is sufficient to capture the total power of the probing signal.

Finally, we calculate the SNR at frequency $f_p$ as

$$\text{SNR}(f_p) = \frac{S(f_p)}{\omega_{.2}} \tag{30}$$

If $\text{SNR}(f_p) > 1$, we infer that a signal exists in that frequency with false positive probability 20%. The next step is to combine the inference outcome for the fundamental frequency $f_p$ with the inference for the first H harmonics of that frequency.

### 5.3.6 Signal detection at multiple frequencies

It is possible that a probing signal is hidden in noise at its fundamental frequency $f_p$. Fortunately we can also rely on the harmonics of that frequency, which appear at the integer multiples of $f_p$. Specifically, we consider the first H harmonics of $f_p$. In general, the signal power decreases quickly in higher harmonics and after a certain harmonic the signal

power is practically zero. We examine the effect of H on the accuracy of the method in the next section.

Given a certain H, we apply the previous frequency detection method at the fundamental frequency as well as at the first H harmonics. If H is even, the total number of detection outcomes is H+1 and there is no possibility for ties. Finally, *a signal is detected if the majority of the H+1 detection outcomes were positive.*

Even though the false positive probability for a single frequency is high (20%), the *global false positive probability* is much lower. Suppose that in reality there is no sharing and so there is no probing signal. Then, if we treat each of the H+1 detection outcomes as independent Bernoulli trials with detection probability 20%, a false positive will occur if we get at least H/2+1 individual false positives. For $H$={4, 6, 8} the global false positive probability is {0.027, 0.017, 0.01}, respectively.

## *5.4 Evaluation*

In this section we use simulations to evaluate the accuracy of the proposed shared bottleneck detection method, and to examine the impact of certain key parameters such as the train length L and the number of harmonics H. Simulations allow us to evaluate the method in a controllable and repeatable manner, which would not be possible with Internet experiments. The simulation setup is described in Section 5.8. Unless otherwise specified, the sampling frequency is 1KHz and the minimum probing frequency is 1Hz. To measure the False-Negative (FN) and False-Positive (FP) detection rates, we run each simulation at least 96 times. This gives us a margin of error that is less than 10% with 95% confidence.

### 5.4.1 Static train length

We first examine the FN rate in the case of a single prober that has a shared bottleneck with the sampler, when we use probing trains of length L. Figure 41 shows the FN rate for three values of L. As expected, as the utilization increases the delay variations induced by cross-traffic increase. So, for a fixed probing train length, the higher the utilization is, the lower the SNR becomes. When the SNR is much less than one, the detection method fails to identify the probing "spikes" in the noisy spectrum and we see frequent false negatives.

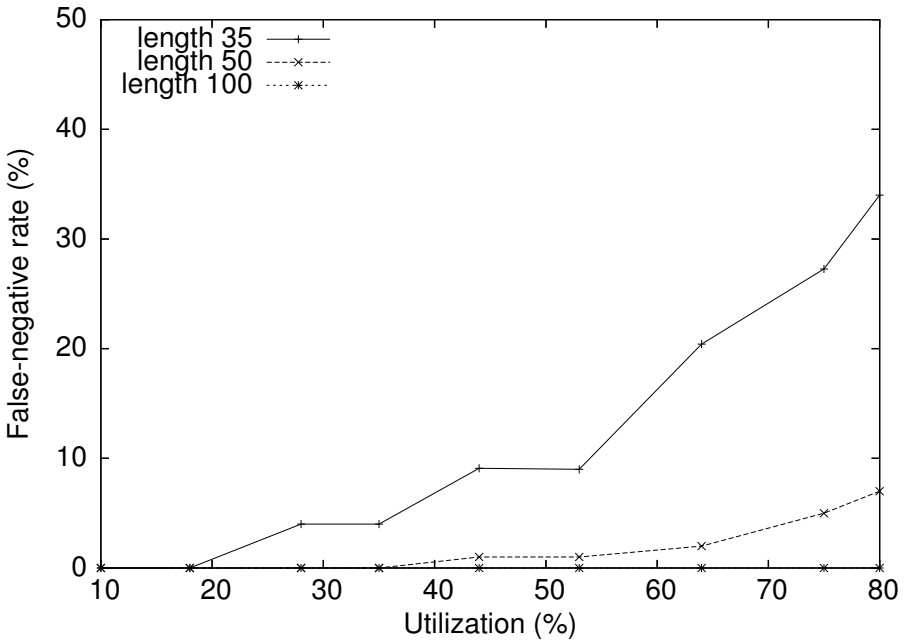



**Figure 41:** False negative rate as utilization increases (static train length).

With L=35 packets, this happens when the utilization becomes larger than 40-50%. As we increase L, we effectively increase the signal power, and thus the SNR. With L=50, the FN rate is below 10% up to a utilization of 80%. With L=100, the FN rate is consistently zero in the entire utilization range.

On the other hand, as we increase the probing train length we also increase the intrusiveness of the tool, in terms of larger delay spikes and higher average probing rate. The previous observations raise the following question: *is it possible to automatically adjust L so that the method is both accurate and non-intrusive?* We examine this question in the next paragraph.

### 5.4.2 Adaptive train length

The basic idea in this variation of the method is to select a packet train length that is large enough to cause queueing delays of about the same magnitude as the 95-th percentile of the queueing delays induced by cross-traffic. In doing so, we expect that the method will be both accurate (because the probing delays will be higher than almost all cross-traffic delays) and non-intrusive (because we do not cause queueing delays that are larger than the large delay spikes caused by cross-traffic).

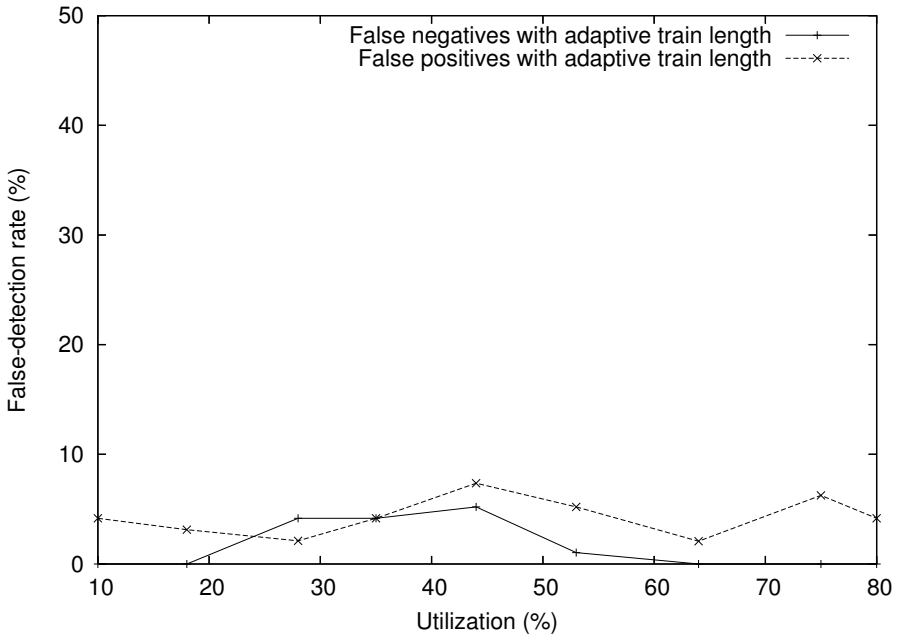



**Figure 42:** False negative and false positive rate as utilization increases (adaptive train length).

To do so, we measure the distribution of delay variations (end-to-end measured delay minus the base-level minimum delay in the path) during the preprobing phase. Then, we use the 95-th percentile $d_{n,.95}$ of that distribution to estimate the desired train length L, as $L = d_{n,.95}C$, where C is a rough estimate of the capacity in the shared bottleneck. A lower bound estimate of C is the minimum of the end-to-end capacity of the sampler and prober paths. These capacities can be measured using one of the existing packet-pair methods, such as *Bprobe*, *Pathrate* or *CapProbe*. We expect that in practice the user (who may be a network manager) will often know the capacity of the potential shared bottleneck between the sampler and a prober (e.g., the access link of the user's campus network).

To not end up with values of L that are too low (when the cross-traffic delay variations are very small) or too high (in heavily loaded or low-capacity paths), we bound L so that it is not less than 35 packets and not higher than 100 packets.[7] In summary, the adaptive train length variation is described by the following equation:

$$L = \max\{35, \min\{100, d_{n,.95}C\}\} \tag{31}$$

[7]The minimum and maximum train lengths were chosen based on data from our simulations and Internet experiments.

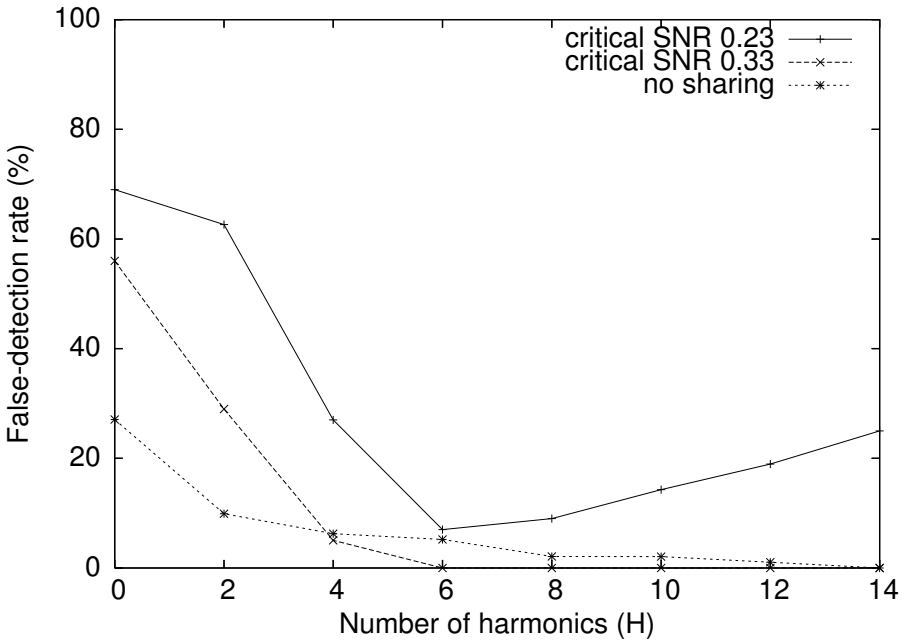



**Figure 43:** Effect of the number of harmonics H on the false detection rate.

Figure 42 shows the FN as well as the FP rates ("no sharing" curve) when we use the above adaptive train length variation. We assume that the user knows the actual capacity of the shared bottleneck. Note that *both false detection rates are persistently below 10%.*

### 5.4.3 Number of harmonics

We next examine the effect of the number of harmonics H in the signal detection process. Recall that the method detects a signal when the majority of the H+1 frequencies (H harmonics and the fundamental frequency) show the existence of a signal in the corresponding spectral band. In the following experiment, we have a single prober that either shares a bottleneck with the sampler (to measure FN rate) or that does not share a bottleneck (to measure FP rate). In the former, we set the train length L to a low value (35 packets) so that there is a clear dependency of the FN rate to the number of harmonics.

Figure 43 shows the FN rate when there is sharing, and the FP rate when there is no sharing. Note that the FP rate drops as the number of harmonics increases. This is expected because as H increases it becomes less likely that the majority of the H+1 frequencies will, randomly, due to cross-traffic noise, happen to have a spectral spike at the corresponding spectral band.

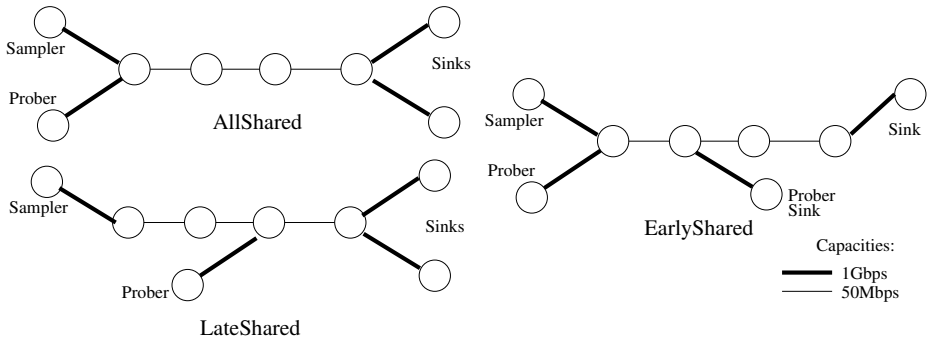



**Figure 44:** Topological variations with multiple bottlenecks.

The FN rate, on the other hand, initially decreases, until H becomes 6, and then it starts increasing again. The reason for the initial decrease is that as we use more frequencies, it becomes less likely that our signal will be masked by noise at the majority of those frequencies. As H increases, however, the signal power (as well as the noise power) decreases. After a certain point there is practically "no signal left" in those higher harmonics, and the resulting detection outcomes become random, causing a slow increase of the FN rate. Through many simulation experiments with various train lengths, we observed that *the optimal value of H actually varies between 4 to 8.*

### 5.4.4 Multiple shared bottlenecks

So far we assumed that the sampler and prober share a single bottleneck. Here we consider several variations of our base topology (see Section 5.8) to examine what happens when the paths share multiple bottlenecks ("All-Sharing"), or when the shared bottleneck appears before ("Early-Sharing") or after ("Late-Sharing") other non-shared bottlenecks. The three topological variations are shown in Figure 44. Each of the three bottlenecks (shared on not shared) are equally loaded with cross-traffic and of the same capacity (50Mbps). In these simulations we use the adaptive train length equation and H=4. The results are shown in Figure 45. Note that *the false detection rates are consistently below 10%, independent of the location of the shared bottleneck or of the number of shared bottlenecks.* It should be mentioned that the FP rate can be further decreased by decreasing the FP threshold for each harmonic (now set to 20%); that may increase the FN rate, however. The FN rate can be further decreased by increasing the train length, making the tool more intrusive though.

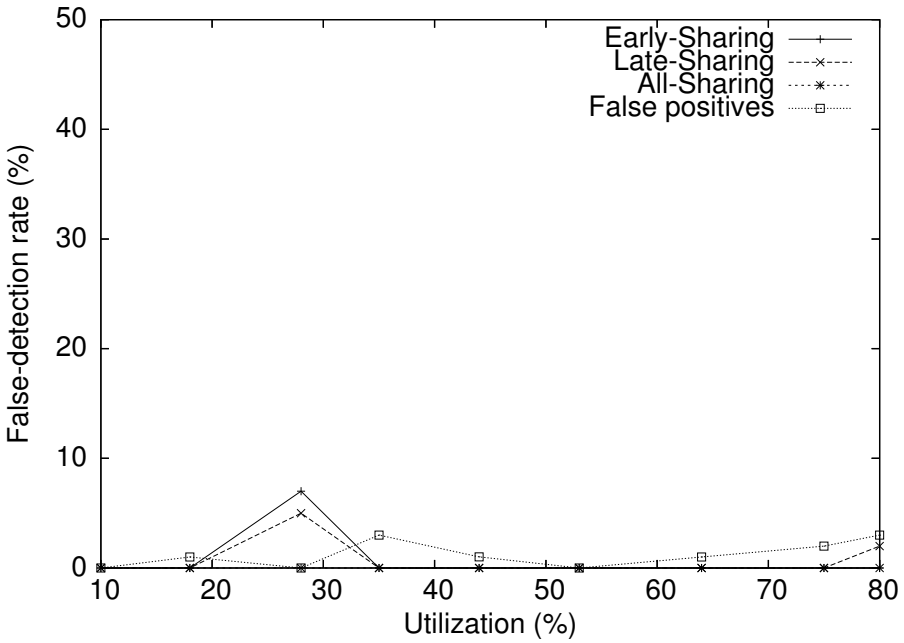



**Figure 45:** FN and FP rate as utilization increases in the three topological variations of Figure 44.

### 5.4.5 Multiplexing probing frequencies

In this subsection, we examine the accuracy of the method when a number of probers are active at the same time. Some of them share the same (single) bottleneck with the sampler, while the rest do not share a bottleneck with the sampler. We measure the FN rate for the former and the FP rate for the latter. Again, we use the adaptive train length equation.

In the first experiment, there are N=11 probers, each of them using H=4 harmonics. Six probers share the sampler's bottleneck, and five do not. Table 6 shows the false detection rates. Note that despite the high multiplexing degree and the large number of allocated frequencies (6 times (4+1)=30 frequencies in the probing spectrum), the method is remarkably accurate in terms of both false positives and negatives.

In the second experiment, we have N=8 probers, each of them using only H=2 harmonics. Four probers share the sampler's bottleneck and four do not. Table 7 shows the false detection rates. Note that the false detection rates are higher, because we are using a very small number of harmonics. On the other hand, the fact that we use a wider guard band (166.7mHz vs. 85.5mHz) does not amortize the negative effect of the low H value.

**Table 6:** 11 probers, 4 harmonics, 85.5mHz guard band.

| Sharing | | No sharing | |
|---|---|---|---|
| Funder-Freq | FN rate | Fundam-Freq | FP rate |
| 1.0Hz | 0% | 2.18Hz | 1.82% |
| 2.7Hz | 0% | 3.1Hz | 3.64% |
| 3.43Hz | 0% | 3.55Hz | 0% |
| 3.7Hz | 0% | 3.85Hz | 0% |
| 4.1Hz | 0.9% | 4.55Hz | 0% |
| 4.7Hz | 0% | | |

**Table 7:** 8 probers, 2 harmonics, 166.7mHz guard band.

| Sharing | | No sharing | |
|---|---|---|---|
| Fund-Freq | FN rate | Fund-Freq | FP rate |
| 1.0Hz | 1% | 3.18Hz | 1% |
| 3.4Hz | 0% | 3.6Hz | 10.1% |
| 3.78Hz | 0% | 4.18Hz | 0% |
| 4.4Hz | 1.03% | 4.6Hz | 1.01% |

In general, it is more important to have enough harmonics than to further isolate a small number of harmonics with a larger guard band.

### *5.5 Internet Experiments*

In this section we present some representative results from Internet experiments. A major issue with such experiments is that we cannot always know the ground truth. The objective of this section is not to conduct a large-scale measurement study, but to present representative examples of sharing between multiple Internet paths, and to show spectral probing in different scenarios in the wild. Nevertheless, in some cases we are certain that there is sharing because we see a clearly visible probing signal in the sampler measurements. In other cases we are confident that there is no sharing because, according to traceroute at least, the two paths go through different ISPs and reach different destinations.

Instead of relying on one-way delay measurements, which would require access at both ends of a path, the following experiments use a *ping*-like utility that we wrote for Linux. The tool, called *SharedBneck*, can run as prober or sampler, sending ICMP ECHO packets in both cases. Because it measures RTTs, it is subject to noise at the reverse-path. Note that

the tool does not require any special timers or other non-standard operating system features. The kernel timestamps are collected using the *SIOCGSTAMP* ioctl call on Linux and they have a resolution of 1 microsecond. The transmission of periodic packet trains is scheduled using the *select* timeout, while the clock interrupt period of all hosts we experimented with is 1ms (allowing a sampling frequency of 1KHz).

Unless stated otherwise, the following experiments use the adaptive train length variation with a sampling frequency of 1KHz and a probing frequency of 1Hz. The capacity of the shared bottleneck is estimated as the minimum of the sampler/prober end-to-end capacities. The prober/sampler sources are located at Georgia Tech's Atlanta campus. We use `procyon.cc.gatech.edu` for the sampler. Unless otherwise stated, we use `foofoo.rnoc.gatech.edu` for the prober, located in a different L2 network from the sampler. The destinations are in various countries and the end-to-end paths cross multiple commercial and academic networks such as Abilene, GEANT2, Qwest and Cogent.

**Detection of sharing when traceroute fails:** In this pair of paths $P_1$ from Georgia Tech (`GT`) to Oracle's data center in Texas (ODC) (destinations: sampler `bigip-wbw-adc.oracle.com`; prober `bigip40-roi-v1.oracle.com`), the prober/sampler traceroute outputs do not show sharing. Actually the traceroutes do not complete, probably because of a firewall or NAT at the destination network, which is the same for both the prober and the sampler. The output of traceroute shows the sampler AS path going through Qwest and Level3, while the prober AS path traverses Cogent and Global Exchange. Figure 46 shows the delay (RTT) increase function for both the preprobing (up to sequence-number 6000) and probing phases. Notice the periodic spikes of the probing signal (delay increase of about 10-15msec) at the right half of the plot. This means that there is certainly sharing between the two paths. Indeed, SharedBneck detects it with high SNR values.

**Effect of probing rate:** This experiment illustrates the importance of a sufficiently high probing rate. In this pair of paths $P_2$ from `GT` to `Insead`'s campus in France (destinations: prober `knowledge.insead.edu`; sampler `faculty.insead.edu`), we do not see a signal when the prober sends trains at a rate of 10Mbps, and SharedBneck reports "no sharing". When we increase the transmission rate at the prober to 100Mbps, there is a clear

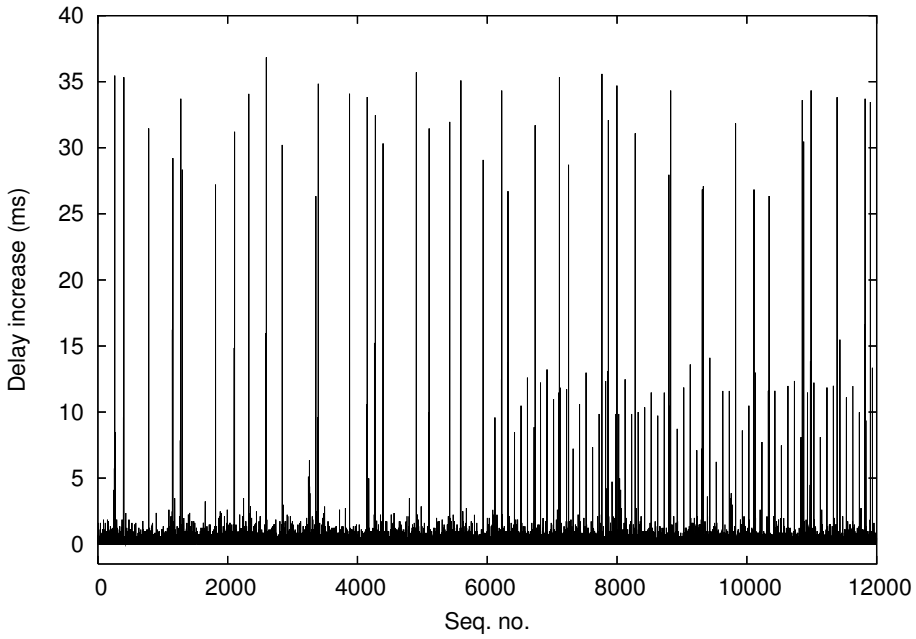



**Figure 46:** Traceroute does not detect sharing but the second half of the timeseries shows clear probing signal: GT to ODC paths.

signal and SharedBneck reports "sharing" with high SNR values. The AS paths traversed Qwest and ALTER.NET ASes. Figure 47 shows the delay increase function timeseries for both the preprobing (first half of the plots) and probing phases with both probing rates.

**Effect of sampling frequency:** This experiment illustrates the importance of a sufficiently high sampling frequency. In this pair of paths $P_3$ from `GT` to University of Cyprus's (`UCY`) CS department (destinations: prober `www2n.cs.ucy.ac.cy`; sampler `pac2.cs.ucy.ac.cy`), we do not see a very clear signal when the sampling frequency is 100Hz, and SharedBneck often gives false positives. When we increase $f_s$ to 1KHz there is a clear signal at the second half of the timeseries (probing phase) and SharedBneck reports "sharing" with high SNR values. The sampler and prober paths traverse the high-capacity Abilene and GEANT2 backbones to reach `UCY`. Figure 48 shows the two delay increase functions.

**SNR variability across harmonics and experiments:** We now present results from four cases of sharing (the previous path pairs $P_1$, $P_2$, $P_3$ and one more path $P_4$ in the `fraunhofer.de` network[8]), as well as a case of no-sharing $P_{ns}$. In the latter, we sample

[8]destinations: prober `pps.izm.fraunhofer.de`; sampler `www.izm.fraunhofer.de`

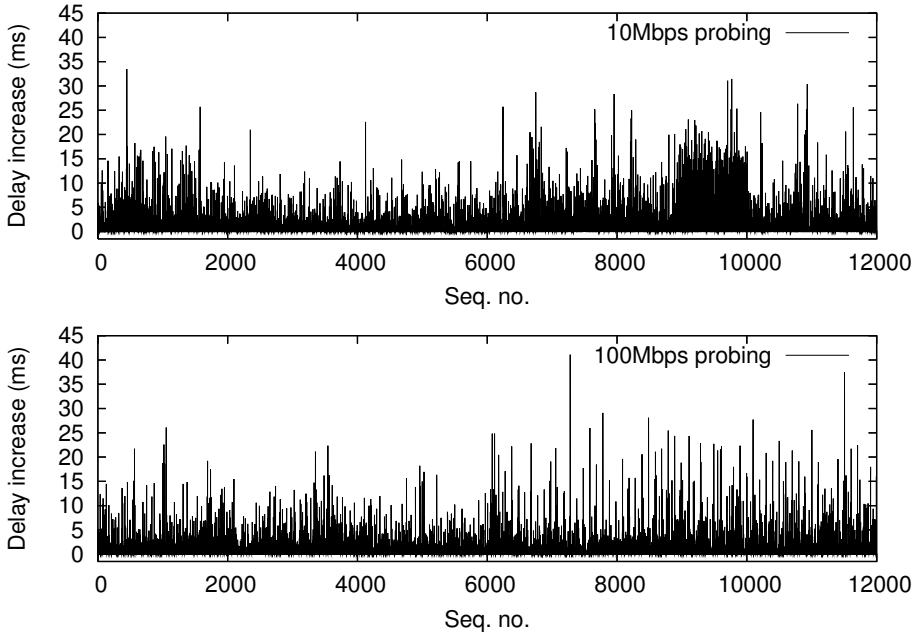



**Figure 47:** A low probing rate may fail to detect sharing: GT to Insead paths.

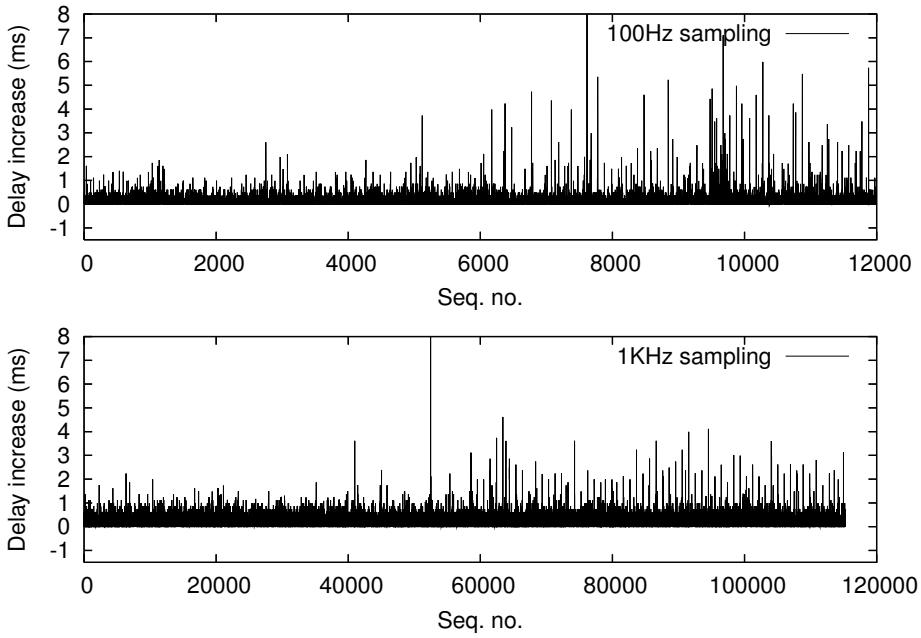



**Figure 48:** A low sampling frequency may fail to detect sharing: GT to UCY paths.

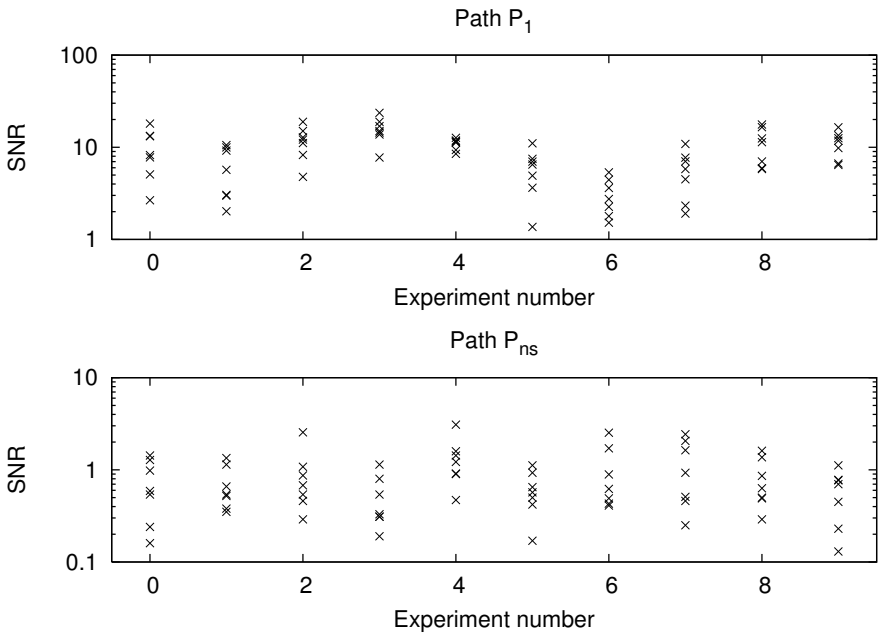



**Figure 49:** SNRs across harmonics and experiments for GT to ODC paths.

without probing on the noisy pair of paths $P_1$ (from `GT` to `ODC`), to make sure that there is no probing signal. We repeated each experiment 10 times, at different times on the same weekday, on multi-processor and single-processor sampler environments, for a total of 50 experiments. The important parameter values for all experiments were: L=35 packets, $f_p$=1Hz, $f_s$=1KHz, H=6 harmonics.

*Out of the 50 experiments, we detected only two false detections; one false negative on $P_3$ and one false positive.* Figure 49 shows the seven (1+H) SNRs for each experiment in two of the path-pairs, $P_1$ and $P_{ns}$. In $P_1$ there is sharing, and this is why most SNR values are much larger than one. Note however the significant variability of the SNR values across experiments, and also across harmonics in the same experiment.

**Experiment with multiple probers:** Here, we perform a three-prober experiment. In addition to the aforementioned sampler and prober (labeled A), we choose the other two probers as `vega.cc.gt.atl.ga.us` (B) and `sirius.cc.gatech.edu` (C). The sinks are chosen in the `fraunhofer.de` network (including paths $P_4$). The key parameter values are: L=35, $f_s$=100Hz, g= 133.3mHz, H=4, while the maximum allowed probing rate is $R_{max}$ =1Mbps. The fundamental frequencies of the three probers are 1Hz, 1.72Hz, and 1.87Hz.

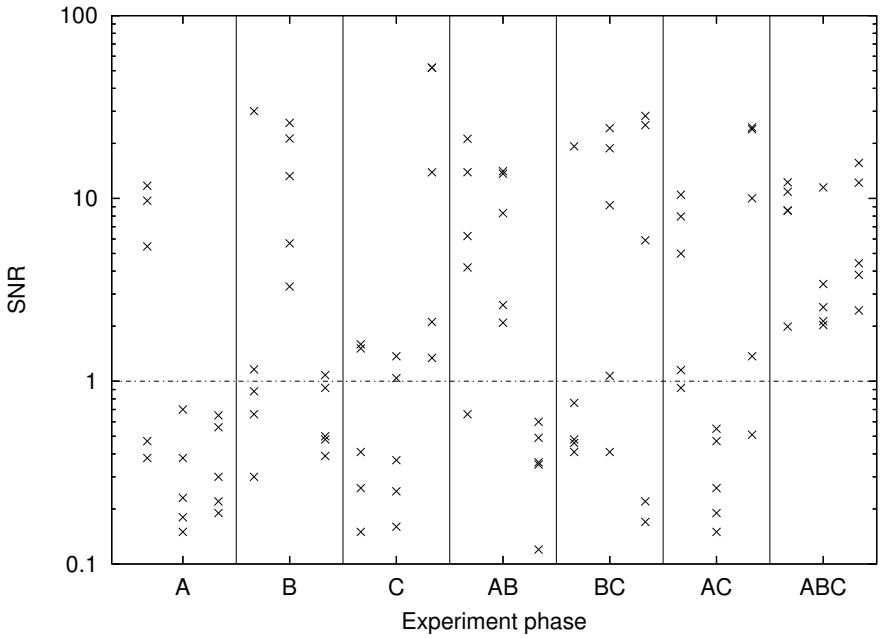



**Figure 50:** Three-prober experiment with seven combinations of active probers: GT to fraunhofer.de paths.

We refer to the three probers as A, B, and C, respectively. The experiment consists of seven phases, each lasting $\Delta_e$=60 seconds. In each phase a different combination of probers is active (see Figure 50). For instance, in the first phase only A is active. *We did not observe any false negatives or false positives for any prober during the experiment.*

Figure 50 shows the resulting five (1+H) SNRs for each prober (in the order A, B, C), in each phase of the experiment. Notice that when a prober is active, its SNR values are mostly (but not always) larger than one. The plot also shows the large variability of SNRs across different harmonics.

### *5.6 Discussion*

The previous two sections have examined the accuracy of the SharedBneck method, focusing on the significance of the sampling and probing frequencies and of the probing train length. In this section, we further discuss various other factors and conditions that can affect the accuracy of SharedBneck.

First, the definition of a shared bottleneck in Section 5.2 requires that the prober is able to cause queueing in the link that it shares with the sampler. If there is a shared link, but

the prober path is such that the probing trains arrive at the shared queue with a rate that is less than that queue's available capacity, SharedBneck will not be able to detect sharing. The practical implication is that probers should be sending their trains at the maximum possible rate and that the hosts that serve as probers should have high-capacity interfaces.

It is also important that the measured paths do not change in terms of the underlying routing and traffic conditions. In our simulations and experiments, we found that simple tests for traffic stationarity, such as major level-shifts, were sufficient, and did not require more elaborate tests for stationarity in the strong or weak mathematical sense. It is important however that, if there is a shared bottleneck, that link should remain in that state during the measurements despite any available capacity fluctuations in the probing and sampling paths. For this reason, we require "operational constancy", a notion described by Zhang et al. in [125], and we examine for violations of this property by detecting level-shifts in the measured timeseries.

The assumption of lossless operation, stated in Section 5.2, is necessary for that model but it is not that important in practice. We have observed that SharedBneck is accurate even when there is noticeable loss rate (say 1%).

The adaptive train length method needs a capacity estimate for the potential shared bottleneck. A lower estimate for that capacity can be obtained with capacity measurements in the sampler and prober paths. In some cases, the SharedBneck user may know the capacity of the potential shared bottleneck when the measured paths are in the network that he/she manages.

Even though we pay attention to the intrusiveness issue, by limiting the probing train length, the method may still be viewed as intrusive. We have not explored whether SharedBneck can cause any performance degradation to other flows, either TCP or jitter-sensitive VoIP flows. We doubt that this would be the case, given the relatively small probing frequency (typically once per second) and the short duration of the probing delay spikes.

The preprobing phase may show that there is significant power in the same probing frequencies that SharedBneck uses. This may happen, for instance, if two different users run SharedBneck at the same time. One option is to abort the measurements after the

preprobing phase. Another approach is to add some randomization in the probing frequencies, so that different users rarely overlap in the frequency domain. A third approach is to choose the probing frequencies so that they do not overlap with the observed frequencies in the preprobing spectrum.

Finally, as any other measurement tool that is more than an idealized model, SharedBneck involves several parameters, such as the SNR threshold $\gamma$, the guardband $g$, the duration of the measurements $\Delta_e$, the outlier-detection parameter $h$, the low-pass filtering parameters, the minimum and maximum probing train length bounds (35 and 100), the sampling packet size, the false-positive probability for a single harmonic (20%), or the accuracy of the capacity estimate $C$. In this chapter, we have investigated the effect of the probing and sampling frequencies ($f_p$ and $f_s$), of the probing train length $L$ and of the number of harmonics $H$. An extensive robustness study for the accuracy of SharedBneck in this multi-dimensional parameter space would be hard. We cannot claim that the parameter values we give in this chapter are "optimal" in any sense. They have been working well, however, in all our experiments, and for this reason we expect that the user would not have to fine-tune the SharedBneck parameters before running the tool in practice.

## *5.7 Summary*

The contributions of this chapter are twofold. First, we introduced the Spectral Probing framework in which fundamental frequency-domain concepts from signal processing and analog communications are applied in active measurements and network tomography. The concepts we explored here relate to frequency modulation of binary information, frequency multiplexing and crosstalk. Spectral Probing has potential for more spectral methods for signal transmission, modulation, and detection in the presence of noise, such as channel coding and spread-spectrum techniques, all in the context of queueing delay variations in IP paths (see the future work section 9.2.3).

Second, we proposed and evaluated one application of the SP framework, namely the detection of shared bottlenecks between two or more paths. This is an important application in practice as it can reveal the presence of shared store-and-forward devices (switches,

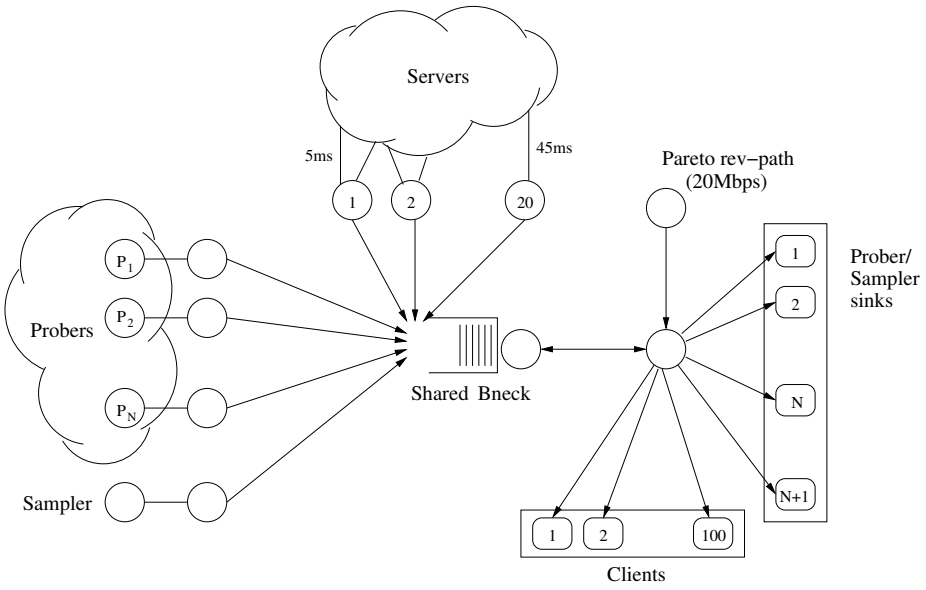



**Figure 51:** Simulation topology.

routers, middleboxes) between end-to-end paths. The proposed method can detect sharing even if the shared device is not congested, as long as the prober(s) can cause short-term queueing at the corresponding shared buffers. Our Internet experiments have shown that the method is effective in discovering shared bottlenecks between paths, even when those links are invisible to traceroute.

The Spectral Probing home page is at `http://www.netinfer.net/spectralprobing`. Parts of this work appeared in a prior publication [65].

## *5.8 Appendix: Simulation Setup*

The NS2 simulation topology is shown in Figure 51. Unless stated otherwise, there is a single shared bottleneck with 50Mbps capacity. The topology includes N prober paths, a sampler path and several cross-traffic paths that are aggregated in the shared bottleneck. Unless stated otherwise, the N prober paths have a shared bottleneck with the sampler.

The capacity of all links that feed into the shared bottleneck is 500Mbps. The access capacity of the probers and the sampler is 1Gbps. The access capacity of the servers is chosen uniformly as either 100Mbps or 1Gbps. The access capacity of the users (clients) is also chosen uniformly as 1, 10 or 100Mbps. The buffer size of the shared bottleneck is set to the bandwidth-delay product of that link (250 packets).

The cross-traffic in the forward path is generated by U=100 users. Each user goes through a cycle of "download" and "think" phases. In the download phase, the user receives a file through a TCP connection from a randomly selected server. We use TCP NewReno with the SACK option enabled. The receive window is set to a large value so that the flows are only limited by the congestion window. The file size distribution is Pareto with mean 80KB and shape parameter 1.5, creating LRD traffic. The duration of the think phase is exponentially distributed. Its duration is controlled to achieve a desired utilization at the shared bottleneck. The RTTs of the TCP connections vary between 30ms to 110ms, due to the heterogeneity in the propagation delays of the server access links.

We also generate some cross-traffic in the reverse path (same direction with the client ACKs) using a packet-level Pareto renewal process. The average rate of that Pareto source is 20Mbps.

# CHAPTER VI

# DIAGNOSING WIDE-AREA PERFORMANCE

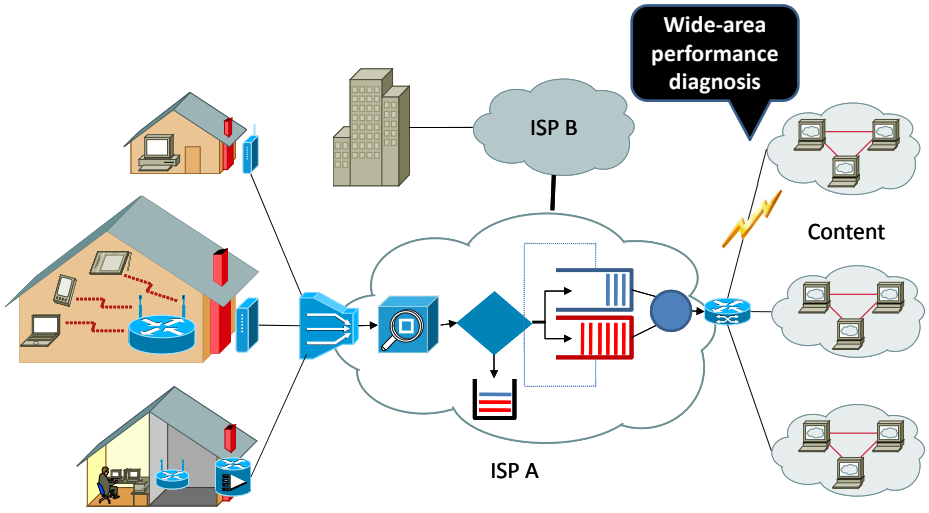



**Figure 52:** Internet model showing a performance problem. Our focus in this chapter is performance problems in the wide-area inter-domain Internet.

## *6.1 Introduction*

Timely monitoring and diagnosis of network health is an important and challenging problem. It requires a distributed monitoring infrastructure with detection and diagnosis algorithms that are real-time and low-overhead. Many networks deploy monitoring nodes at vantage points to record network health. For example, the Internet2 and the US Department of Energy's ESNet networks deploy monitoring nodes that perform full-mesh one-way measurements of delay and packet losses using the *One-way Ping* (OWAMP) tool as a part of the *perfSONAR* software [51]. These measurements are summarized and exported to data repositories for offline analysis by network operators. Measurement services such as Keynote Systems [11] use inter-domain end-to-end active measurements to monitor performance. This chapter plugs a significant missing component in the process of improving network availability and performance.

Pythia is a diagnosis system that works in conjunction with a distributed monitoring infrastructure such as perfSONAR. Suppose that we have monitors labeled $1 \ldots n$. Each monitor $i$ measures one-way delays of the directed path $i \rightarrow j$ for $j \in \{1 \ldots i-1, i+1 \ldots n\}$; we call such a setup as "full measurement mesh". We build two components in this work. First, *detection* observes measurements on the path $i \rightarrow j$ and determines if the path has a *performance problem* at a given point of time, and computes the time window in which the problem persists. Second, *diagnosis* takes as input a performance problem and identifies the *root cause* of the problem. We have designed algorithms for *localization* [123], and we are working on adding the feature to detected problems.

In addition to diagnosis *accuracy*, *scalability* (to hundreds of monitors) and *usefulness*, Pythia has novel contributions. Pythia diagnoses events that occur in *timescales of seconds* (typical operational monitoring deployments look at timescales of hours). Such events can provide insight into performance of short-lived applications such as web browsing. Pythia works in the *wide area, inter-domain* context; this is important, since many sources of data such as SNMP, Netflow, router logs or customer trouble tickets are not available in such scenarios. Pythia is a *near real time* system - it can diagnose network-wide problems as recent as 5-10 minutes - enabling early recovery.

Our contributions are two-fold. First, we design efficient algorithms for detection and diagnosis. The distributed diagnosis model of Pythia also ensures that the system scales to hundreds of monitors. We have a live deployment of Pythia (2012) on the Internet running on 43 paths, and it has diagnosed over 10,000 pathologies a day[1]. A web-based front end displays current pathologies in the monitored network, and summarizes historic data collected across paths to visualize performance trends: `http://pythia.cc.gatech.edu`.

Second, we show a first analysis of performance pathologies in different networks using the proposed detection and diagnosis methods. Our study focuses on research (ESnet and Internet2), inter-domain (PlanetLab) and residential broadband networks (using ShaperProbe data [67]). We study problems and pathologies in different network types.

Our approach to Pythia is based on domain knowledge. We use domain knowledge to

[1]The user can tune the sensitivity of the system towards pathologies.

construct models of typical performance pathologies in networks. We also show that the monitoring nodes can induce performance problem signatures in measurements, and construct models to diagnose such signatures. Further, Pythia is designed to allow the network operator to seamlessly configure new pathologies using knowledge of the network configuration and performance. A domain knowledge-based approach has several advantages over methods that use machine learning. Learning-based methods typically mine dependencies among large sets of data to construct models (see Section 2.4 for a description of prior work). The datasets are typically specific to an administrative domain (an ISP or an enterprise) - such as SNMP, Netflow, syslogs, routing and forwarding tables, and customer trouble tickets. Learning-based methods do not scale to wide area inter-domain settings, where obtaining network and customer data is hard. They are typically offline methods that may require training. On the other hand, learning-based methods can provide detailed diagnosis, since they have access to data. In this work, we show that domain knowledge-based methods can be built for wide area inter-domain diagnosis, and that such approaches can be useful in understanding and diagnosing performance in many network types.

**Chapter organization:** Section 6.2 describes the system. Section 6.3 describes detection. Section 6.4 talks about diagnosing performance and diagnosis accuracy. Section 6.5 describes our live deployment. Section 6.7 is a study of performance problems in networks. Section 6.8 lists ongoing and future work.

## *6.2 System Overview*

The Pythia infrastructure is designed to operate in conjunction with monitoring systems that are based on the following model. A set of $N$ monitors are deployed at vantage points on the Internet so that the paths between them cover a reasonably large set of links in the network between them. The monitors run active probes between each other, potentially covering $N \times (N-1)$ end-to-end paths (in case of a full mesh measurement). A measurement 3-tuple $\{T_{s,p}, S_{s,p}, T_{r,p}\}$ of sender and receiver timestamps $T_{s,p}$ and $T_{r,p}$ and a sequence number $S_{s,p}$ is recorded for each probe $p$. The sequence number is incremented by one at the sender for each probe in the flow. We require loose clock synchronization (error

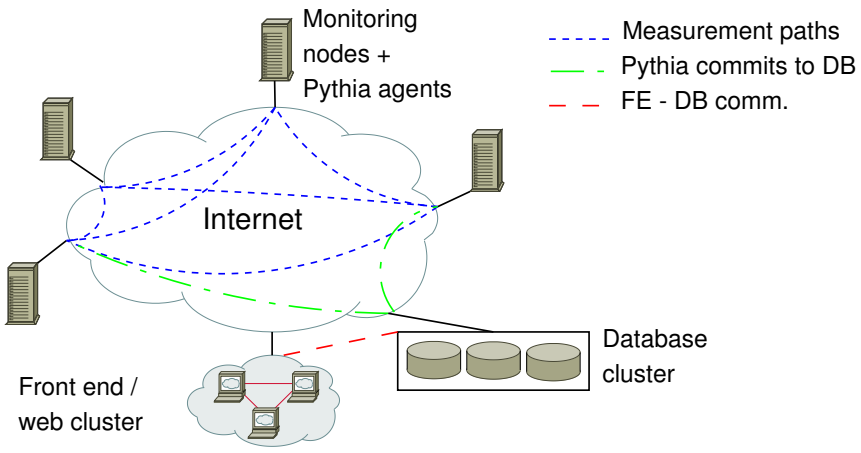



**Figure 53:** System architecture of Pythia. "DB" and "FE" refer to the data repository and front end respectively.

margin of seconds) between the sender and receiver to correlate events between monitors. Monitor $M_i$ collects information about all probes sent to it; a lost packet is either "marked" by $M_i$ as lost after a pre-defined timeout interval, or implicitly marked by a missing sequence number at $M_i$ for that flow. We assume that a suitable *interface* exists on each monitor so that processes can import measurements; the interface could be an API, a local cache, or simply files written to the disk.

Figure 53 shows a monitoring infrastructure and the architecture for Pythia. We run a lightweight *agent* process on each monitor. The agent is designed to consume small amount of CPU and memory resources so that it does not impact measurement accuracy. The agent interfaces with the latest available data to detect and diagnose performance problems, and commits the output to a centralized database repository. The DB is also used for read transactions by the agent to diagnose problems that require measurements from other monitors. A web server interfaces with the DB to render a browser front end.

Figure 54 shows a block diagram of the agent and the interactions between its modules. The agent implements functionality that requires individual packet measurements at its monitor. It consists of five subsystems: pre-processing, the main loop, detection, diagnosis, and the storage. The pre-processing subsystem includes three modules. The *specification to code* module translates operator-specified pathology definitions into code that will be used for diagnosis using a forest of decision trees (as a part of the *diagnosis* module). The

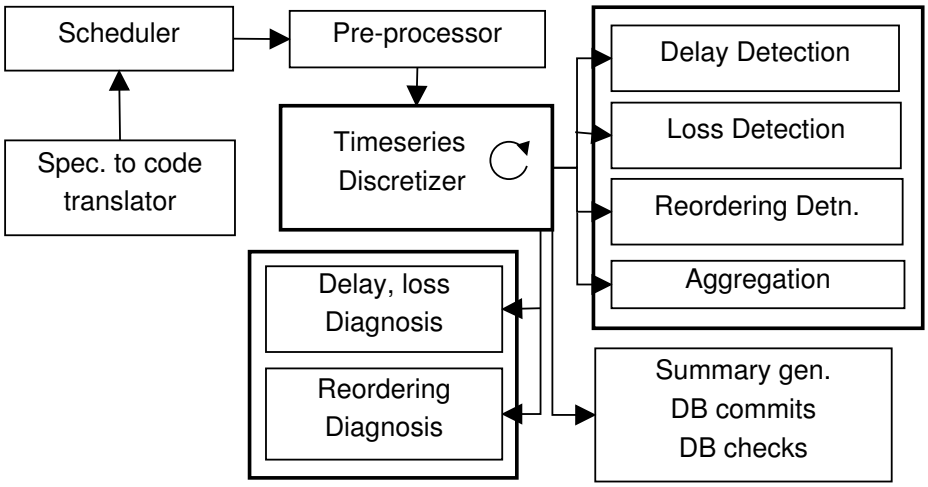



**Figure 54:** Pythia agent: block diagram showing different modules and their interactions.

*scheduler* interfaces with the monitoring system to prepare a queue of jobs for detection and diagnosis (each job is a timeseries of measurements for a path; jobs are prepared in chunks of five minute intervals for each path). The queue is ordered by start time of measurements. The jobs are fed to the *pre-processor* module, which loads the corresponding measurements into in-memory data structures for per-packet delays and packet losses.

The *timeseries discretizer* reads the data structures to divide the timeseries into consecutive, non overlapping, five second *windows*[2]. Each window is fed to the detection subsystem for performance problem detection. The agent implements three detection types: *delay*-, *loss*- and *reordering*-based detection. The *aggregation* module combines detected windows that are nearby in time into a single window and signals that to the discretizer. The discretizer calls the diagnosis subsystem on the aggregated windows. Finally, detection and diagnosis output is committed to the database repository by the discretizer.

### *6.3 Detecting Performance Problems*

The problem of *detection* is the first step in performance diagnosis. We define detection as *identification of significant deviations from baseline end-to-end performance*. Suppose that we have a timeseries of active probing measurements from a sender $\mathcal{S}$ to a receiver $\mathcal{R}$; we assume without loss of generality that the timestamps at $\mathcal{S}$ and $\mathcal{R}$ are taken at user

[2]The interval durations can be adjusted based on the sampling rate for a path. We chose the durations based on the 10Hz sampling used in perfSONAR.

space. Under *normal* conditions, three invariants hold true for a timeseries of end-to-end measurements:

1. The end-to-end delays are *close to* (with some noise margin) the sum of propagation and transmission delays along the path,
2. No packets are lost, and
3. There is no reordering of packets (as received at $\mathcal{R}$).

Invariant (1) assumes that the sampling process does not induce self-queueing effects; we address this in the detection pre-processing stage below.

Based on the invariants, we implement three types of performance problem detection: *delay*, *loss* and *reordering* detection, depending on the baseline invariant that is violated. At a high level, the detection logic divides the timeseries into non-overlapping back-to-back *windows* of 5s duration[3], and *flags* these windows as either "baseline", or as "problem" (i.e., violating one or more invariants). The logic then *aggregates* "problem" windows into detection output.

**Delay detection:** Delay detection identifies *significant deviations* from baseline end-to-end delays. We use the delay invariant condition (1) above; in other words, under normal conditions, the distribution of delay samples in the window will be unimodal with *most* of the density concentrated *close* to the sum of propagation and transmission delays $d_{\min}$ along the path. If there is a deviation from the delay baseline, the delay distribution will either be multimodal with a *low* density around $d_{\min}$, or will be unimodal with a large range of delays under the mode - depending on the "bin width" used.

We use a nonparametric kernel smoothing density estimate [111] to analyze modes in the delay distribution. A smooth density estimate will allow us to locate modes (local maxima), the start and end points of a mode (local minima), and the density inside a mode with a single pass on the density function. We use a Gaussian kernel; it gives us a smooth estimate since it is a continuous function. A kernel density estimator also requires a *bandwidth*

[3]We choose the window duration so that we have a reasonably large sample size in a window.

**Algorithm 1** Delay detection logic.

**Input:** Time series of e2e one-way delays (ordered by send time): $(t_i, d_i)$; *minimum problem delay* threshold $\tau_d$, minimum problem duration threshold $\tau_l$; and *window merge gap* threshold$\tau_m$.

**Output:** Set of disjoint time windows $\mathcal{P}$ in which there was a *problem*.

1. Initialize $\mathcal{P} \leftarrow \varnothing$.
2. **For** each $i$: if packets $i$ and $i+1$ were sent close to each other ($t_{i+1} - t_i < 100$us), set $d_{i+1} \leftarrow d_i$.
3. Divide timeseries into back-to-back windows $\mathcal{W}_j$ of duration 5s such that window $\mathcal{W}_j$ contains delays of all packets sent in $[t_0 + 5j, t_0 + 5j + 5)$.
4. **For** delay sample in window $\mathcal{W}_j$:
   (a) Sample kernel density $\hat{f}_h$ for delays $d$ in the sample range spaced by 100ms: $\hat{f}_h(d) = \frac{1}{n}\sum_i K_h\left(d - d_i\right)$, where $K_h$ is a Gaussian kernel, and bandwidth $h = 0.79\frac{I}{n^{0.2}}$, where $I$ is the inter-quartile range of delays in $\mathcal{W}_j$ and $n =\mid \mathcal{W}_j \mid$.
   (b) Compute the baseline: Traverse $\hat{f}_h$ to find the *smallest-delay local maxima* $\left(d_b, \hat{f}_h(b)\right)$, and the width (delay range) of the "hill" $w_b$ that contains this maxima. When there is no problem, $d_b$ is an estimate of the *baseline* delay. Suppose that the density of delays contained in this "hill" is $\alpha_b$.
   (c) **If** $\alpha_b < 0.4$ **or** $w_b > 2$ms, and if $I > \tau_d$: add window $\mathcal{W}_j$ to $\mathcal{P}$; set $j \rightarrow j+1$.
5. **Aggregate:** If windows $\mathcal{W}_j$ and $\mathcal{W}_k$ are in $\mathcal{P}$, and if they are separated by less than $\tau_m$, delete them and add a single window encompassing $\mathcal{W}_j, \mathcal{W}_{j+1} \ldots \mathcal{W}_k$ to $\mathcal{P}$. Repeat until no merges are possible.
6. **Delete** windows from $\mathcal{P}$ that have a duration less than $\tau_l$.
7. **Return** $\mathcal{P}$.

estimate, which defines the variance of the Gaussian kernel and controls the "smoothness". We use the Silverman's rule of thumb to compute the bandwidth [111]. Algorithm 1 details our algorithm.

The delay detection algorithm works as follows. We pre-process the delay timeseries to: (1) account for delays induced by self-queueing effects by overwriting measured delays of probes sent less than 100us apart with that of the previous probe, and (2) check if the range of delays in the window is larger than an operator-defined delay threshold (5ms by default). The delay detection logic estimates the density function and computes the *lowest*

*mode*. The lowest mode is used as an estimate of the *baseline delay* for the window. The detection module checks the density and width of the lowest mode to flag the window as "normal" or "problem". The aggregation module merges "problem" windows separated by 5s or lower into one "problem" window. The delay detection module also keeps track of the previous window's baseline for diagnosis of longer events.

The approach of detection by inferring a baseline and finding deviations from it has certain limitations. The baseline computation may sometimes be incorrect for *long duration* pathologies (e.g., a congestion event that creates a persistent backlog with low variability). In such cases, after processing a few windows, the baseline estimate would change to the new "level" of delays in the timeseries, although the new windows still include the performance problem. Hence, the new windows will not be diagnosed as a problem (until the delay timeseries start showing a baseline; for example, due to variability in the backlog). In practice, the aggregation module's functionality of merging nearby problem windows may overcome this issue. Our focus in this work, however, is in detecting short-term events.

**Loss detection:** Loss detection identifies *significant deviations from the baseline loss invariant condition* (2). Specifically, loss detection module flags a window as "problem" if the window contains at least one lost packet. Lost packets are identified by missing sequence numbers when the agent reads the complete timeseries into memory. The aggregation module combines "problem" windows close to each other into a single "problem" window.

**Reordering detection:** Reordering detection identifies *significant deviations from baseline reordering invariant condition* (3). The reordering module computes a *reordering metric* $R$ for each 5s window of sequence numbers in received order, $\{n_1 \dots n_k\}$, based on the *RD* metric definition in RFC 5236 [59]. For each received packet $i$, it computes an *expected sequence number* $n_{\text{exp},i}$; $n_{\text{exp}}$ is initialized to the lowest recorded sequence number in the timeseries, and is incremented with every received packet. If a sequence number $i$ is lost, $n_{\text{exp}}$ *skips* the value $i$. A *reorder sum* is computed as (assume without loss of generality that the window starts with $i = 1$):

$$\eta_{sum} = \sum_{i=1}^{k} \mid n_i - n_{\exp,i} \mid$$

The reordering module estimates the number of reordered packets in the window based on mismatch in the two sequence numbers:

$$\eta = \sum_{i=1}^{k} I\left[n_i \neq n_{\exp,i}\right]$$

The reordering metric $R$ for the window is defined as the ratio:

$$R = \begin{cases} (\eta_{sum}/\eta) & \eta > 0 \\ 0 & \eta = 0 \end{cases}$$

Note that $R$ is zero if there was no reordering, and $R$ increases with the *amount* of reordering on the path. Our goal is not to estimate the number of reordered packets, but to get a quantitative measure of the extent of reordering for a window of packets. The reordering detection module flags a window as "problem" if $R \neq 0$ for that window. The aggregation module simply adds $R$ to a "reordering timeseries", which is used for reordering diagnosis.

**Sensitivity:** Pythia provides a knob to the user to configure *sensitivity* of the detection module towards performance problems without defining any thresholds. This functionality reduces the number of events that the system reports to the user, while eliminating relatively *insignificant* events. We use two heuristics to define sensitivity. First, the duration of the event. A less sensitive detection module will only detect longer events. Second, the severity of the problem. This is defined in terms of the fraction of delays much higher than the baseline, the loss rate, or the value of the reordering metric (relative to a preset threshold) during the event.

## *6.4 Diagnosing Performance Problems*

The diagnosis logic takes as input detected performance events and generates a summary of the possible root cause(s) and diagnosis, by analyzing the event's delay, loss and reordering measurements. We use the terms "performance problem" and "performance pathology" interchangeably in this chapter.

**Algorithm 2** Diagnosis specification grammar.

1. `'SYMPTOM' symptom`
2. `'PATHOLOGY' pathology 'DEF' ( symptom | 'NOT' symptom )`
3. `symptom` → `symptom_1 'AND' symptom_2`
4. `symptom` → `symptom_1 'OR' symptom_2`
5. `symptom` → `( symptom )`
6. `'PROCEDURE' symptom func`

We model a performance pathology as *a unique observation on a set of symptoms*. A *symptom* is represented as a boolean value, and an *observation* is a logical expression on one or more symptoms. Pathologies differ either in the set of symptoms on which they are defined, or on the observation. Examples of symptoms include "interquartile of delays exceeds 100ms", "loss probability exceeds 0.5", "non-zero reordering metric", etc. A boolean-valued test $T$ can be defined for each symptom $\mathcal{S}$, given an event timeseries $\mathcal{E}$, such that $T$ returns `true` if the symptom exists in the timeseries: $T(\mathcal{S}) : \mathcal{E} \rightarrow \{0, 1\}$. In the rest of this chapter, we refer to "symptom" to mean a boolean-valued test on the symptom (unless otherwise specified).

The diagnosis module works as follows. Performance pathologies are pre-defined by the network operator in a configuration file as logical expressions of symptoms (Pythia comes with a default configuration of pathologies). The agent fetches the configuration file at bootstrap, and generates code containing the diagnosis logic; this code forms a part of the diagnosis module. The diagnosis module also contains subroutines for boolean-valued symptoms; the subroutines are exposed to the generated diagnosis code. The operator is required to add boolean-valued tests for each *new* symptom used for diagnosis.

### 6.4.1 Configuration Input

The operator-specified configuration file contains (among other things) *diagnosis rules*, which are boolean-valued expressions of one or more symptoms. The operator uses a configuration language supported by Pythia. The language grammar is shown in Algorithm 2. An example of a diagnosis rule for a form of congestion is the following:

```
PATHOLOGY CongestionOverload DEF delay-exist AND high-util AND NOT bursty-delays AND
NOT high-delayRange AND NOT large-triangle AND NOT unipoint-peaks AND NOT delay-levelshift
```

The statement specifies a rule for the pathology `CongestionOverload` using seven symptoms. Note that the grammar allows specification of negations ("`NOT`") and disjunctions ("`OR`"). The keyword `PROCEDURE` specifies subroutines for symptom tests.

The diagnosis module reads the specifications to construct efficient diagnosis logic.

### 6.4.2 Pre-processing

The agent pre-processes the diagnosis rules before generating the diagnosis logic. The agent internally represents the diagnosis rules as a `pathology` $\times$ `symptom` matrix $\mathbf{M}$. Cell $\mathbf{M}(i, j)$ takes the values `true`, `false` and `unused`, according to whether pathology $i$ requires that symptom $j$ should be true, false, or if the rule for pathology $i$ *does not use* symptom $j$ respectively. Matrix representation of the diagnosis rules leads to efficient query operations during construction of the diagnosis logic.

The first step in pre-processing is to check whether any two rows of matrix $\mathbf{M}$ are *identical*. If a conflict is found, the agent aborts (and logs an error message for the operator).

Pythia supports rules that have an `OR` condition (disjunctions) by pre-processing the rules as follows. Note that for a disjunction:

$$a \vee b = (a \wedge (b \vee \neg b)) \vee (b \wedge (a \vee \neg a)) = (a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge b) = a \vee (\neg a \wedge b) \quad (32)$$

Using this logic, for each expression `a OR b` that involves symptoms (or expressions of symptoms) `a` and `b`, we *replace* the expression by two rules:

$$\texttt{a OR b} = \begin{cases} \texttt{a} \\ \\ \texttt{NOT a AND b} \end{cases}$$

We repeat this until all pathology specifications are made of conjunctions alone. Note that this operation will lead to multiple specifications for a pathology.

**Algorithm 3** Diagnosis forest construction.

**Input:** Matrix $\mathbf{M} : \mathcal{P}$ (pathologies) $\times\ \mathcal{S}$ (symptoms).
**Output:** Forest of decision trees covering $\mathcal{P}$.

1. Construct *set of subsets* $\{P_1 \dots P_k\}$ of pathologies ($P_i \subseteq \mathcal{P}$ and $P_i \neq P_j$ for $i \neq j$); such that sets $P_i$ and $P_j$ *do not* use the same symptoms
2. **For** each set $P_i \in \{P_1 \dots P_k\}$:
   (a) Reverse-sort symptom set $\mathcal{S}$ by the frequency of *usage* in $\mathbf{M}$: let the sorted set be $S_1 \dots S_n$
   (b) **Add** root node as $S_1$
   (c) **For** each pathology $p_i \in \mathcal{P}$:
      i. **Create** pathology leaf node $N = p_i$
      ii. **For** each symptom $S_j \in \{S_n \dots S_2\}$:
         A. **Add** parent node $S_j$ to node $N$ with edge $N \rightarrow S_j$ label $\mathbf{M}\,(i, j) \in$ {`true`, `false`, `unused`}
         B. **Set** $N \leftarrow S_j$
      iii. **Add** parent of $S_2$ as root node $S_1$ with edge $S_1 \rightarrow S_2$ label $\mathbf{M}\,(i, 1)$
3. For each tree $i$ constructed from $P_i \in \{P_1 \dots P_k\}$:
   (a) **Node pruning: For** each node $N$:
      i. **merge** child nodes $A$ and $B$ if $N \rightarrow A$ and $N \rightarrow B$ have same edge labels (and $A$ and $B$ are not pathologies).
   (b) **Unused-edge pruning: For** each edge $N \rightarrow A$ with label `unused`:
      i. **move** node $A$ to $N$ **if** $N$ does not have children other than $A$; repeat step 3(b).
4. **Return** a forest of decision trees: one for each pathology set.

### 6.4.3 The Diagnosis Forest

The diagnosis pre-processing step returns diagnosis specifications for pathologies as logical expressions on symptoms. A brute-force approach to diagnose a performance problem is to test the event timeseries against all symptoms, and subsequently, evaluate each pathology by substituting the boolean symptom values in the pathology expression(s). This can be computationally expensive, since each symptom would test for a unique pattern in the timeseries. We want to minimize the number of symptoms we test for while diagnosing an event. An efficient way to do this is to build a decision tree from the pathology specifications.

Existing construction methods such as the *C4.5* and *ID3* algorithms iteratively choose an attribute (symptom in our case) based on the criteria of one that best classifies the instances (pathologies in our case). They generate small trees by pruning and ignoring attributes that are not "significant". There are three reasons we need to design a decision tree construction method from scratch instead of employing existing methods. First, pathologies may use only a small subset of symptoms (in other words, we cannot use `unused` as an attribute value). In addition, not all outcomes of the symptom may be used in diagnosis; for example, Pythia does not include any pathologies which require the "loss exists" symptom to be `false`. Second, a pathology is required to be diagnosed using *all* symptoms in its specification (existing decision tree methods consider the *smallest* set of symptoms). Third, some pathologies may exist *simultaneously* in an end-to-end path, and hence can be diagnosed in parallel; in other words, these pathologies do not use common symptoms. In such cases, we may need to construct disjoint decision trees (a "diagnosis forest") for independent pathologies.

The agent constructs at bootstrap a forest of decision tree(s) for diagnosis using the matrix $\mathbf{M}$. The diagnosis forest is constructed in two steps (full algorithm in Algorithm 3). In the first step, the agent divides the set of pathologies $\mathcal{P}$ into disjoint subsets $\{P_1 \dots P_k\}$, such that: (1) pathologies in each $P_i$ use overlapping symptoms, and (2) sets $P_i$ and $P_j$ $(i \neq j)$ do not use common symptoms. Since no two members of the set $\{P_1 \dots P_k\}$ use overlapping symptoms, we can run tests for $P_i$ and $P_j$ independently.

In the second step, for each $P_i$, we construct a decision tree for diagnosis of members in set $P_i$ using a bottom-up method. The root of the tree is designated as the symptom that is *most frequently* used; the frequency of symptom usage drops as we go towards the leaves. We next construct disjoint linked lists for each pathology $p \in P_i$ from the root down to the child node (with label $p$). We now have a diagnosis tree in which each level corresponds to either a *unique* symptom or a pathology (in case of leaves).

The diagnosis tree construction retains *all* symptoms that are used to diagnose each pathology, while pruning as many unused symptoms as possible. We do two rounds of pruning on each of the constructed trees (illustrated below; the leaves are pathologies, and

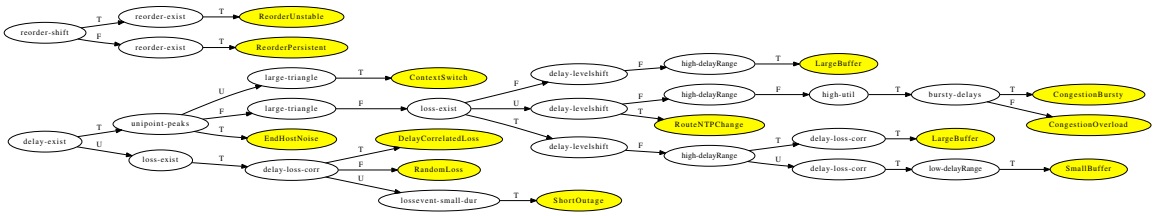
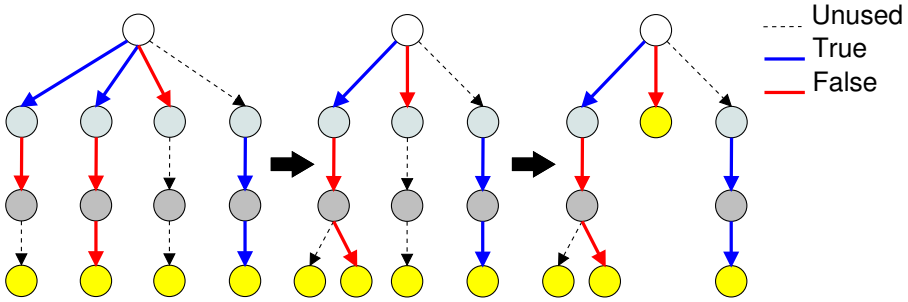



**Figure 55:** Diagnosis forest generated by Pythia based on the default pathology definitions. Shaded nodes represent pathologies, and others are symptom nodes. Edge labels "U", "T" and "F" indicate unused, true and false respectively.

shades show symptoms). First, we *merge*, top down, all edges of each node that have the same labels (since all non-leaf children of a node are the same symptom). This operation will also ensure that each node has *unique outgoing edge labels*. Second, we delete, bottom up, all edges with labels `unused`. More specifically, for each edge $A \rightarrow B$, we delete $A$ and move $B$ upwards if (1) edge $A \rightarrow B$ has label `unused` (i.e., symptom $A$ is unused in diagnosis of the sub-tree of $A$), and (2) $B$ does not have any siblings.




The diagnosis forest currently deployed in Pythia is shown in Figure 55. It contains two trees - one for delay and loss diagnosis, and one for reordering diagnosis.

It could be the case that none of the pathology definitions may match a detected problem signature. In such cases, the problem is tagged by the diagnosis logic as "unknown". In practice, we have observed that less than 10% of problems are tagged as "unknown".

After the agent constructs the forest, it generates code for the diagnosis forest. Each symptom node in the tree corresponds to a procedure call implemented by the agent for that symptom; and each leaf (pathology) node corresponds to the diagnosis output. This code forms an interface for the diagnosis module.

## *6.5 Common Diagnosis Specifications*

We have pre-configured Pythia with specifications for common performance pathologies. Our goal is to provide the network operator with useful diagnosis information, given the limited amount of available data. The design philosophy behind our diagnosis is that domain knowledge helps extract useful information from end-to-end probing data. Our network model is as follows. An end-to-end path consists of a sequence of store-and-forward hops having a limited buffer size. We do not assume that links are work conserving, FIFO, or of a constant capacity (for example, 802.11 wireless links violate these assumptions). We assume that monitoring hosts may induce delays, losses and "noise" in measurements. In this section, we describe common pathologies, and the statistical tests for matching the associated symptoms. The pathologies are identified and defined based on domain knowledge. In addition, the network operator can specify new pathologies (and symptoms) based on knowledge of the network performance and configuration. The tests for symptoms are designed by extracting salient and noise-resilient features of the pathology models.

The symptoms are defined over a detected event. An event is defined by the path, timeseries of send timestamps $T_i$, end-to-end one-way delays $d_i$ and sequence numbers $S_i$, for successfully received probes - where $i$ is a monotonically increasing identifier of each received packet at the receiver. Lost packets can be inferred using sequence numbers $S_i$.

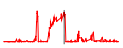
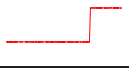
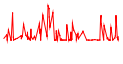
The following are common pathologies that we believe are useful for network diagnosis. These are default pathologies configured in Pythia; symptoms using which pathologies are specified are given in Table 8. Symptoms are configured with a default set of *thresholds*, which can be adjusted by the operator.

### 6.5.1 End-host Pathologies

The first and foremost issue in diagnosis is to eliminate false network-related diagnoses that are caused by the monitors rather than the network.

**End-host effects:** A common artifact of commodity OSes is *context switches*. A busy OS environment may lead to *non-work conserving* behavior in network I/O and thus, significant *wait* periods, either: (1) after a userspace `send()` call till its transmission, or (2) after

**Table 8:** Default symptoms and their boolean-valued tests. The problem event consists of delay samples $\mathcal{D} = \{d_1 \dots d_n\}$, and the estimated baseline delay is $b$ (both in ms). In case of reordering, we work on a reordering metric timeseries $\mathcal{R} = \{R_1 \dots R_l\}$. $I(x)$ is the 0-1 indicator function, and $m(\dots)$ is the sample median. The default thresholds are tuned using empirical observations on perfSONAR data.

| Symptom | Sample | Boolean-valued Test |
|---|---|---|
| High delay utilization |  | More than 50% are *large* delays: $\sum_{i=1}^{n} I(d_i > b+1) > 0.5 \mid \mathcal{D} \mid$ |
| Bursty delays |  | Largest delay hill duration less than 70% of delay hill duration |
| Extreme delay range | | Very small: $\mathcal{D}_{0.95} - \mathcal{D}_{0.05} < 1$ms; very large: $\mathcal{D}_{0.95} - \mathcal{D}_{0.05} > 500$ms |
| Delay-loss correlation |  | For a lost packet $i$: $d_j \leq b+1$ms, for majority of $j \in [i-5, i+5] - \{i\}$ |
| Small loss duration | | All packets between $i$ and $j > i+1$ are lost, and $T_j - T_i > 1$s |
| Delay level shift (LS) |  | Estimate LS: first/last $d_i < b \pm 10$ms; 10% points before/after LS |
| Large triangle |  | Sudden rises in delay over 300ms: $\sum_{i=1}^{n} I(d_{i+1} - d_i > 300) < 0.1 \mid \mathcal{D} \mid$ |
| Single-point peaks |  | Median of neighbors of $i$:$d_i > b+1$ms $\approx$ median of $d_j$:$d_j < b+1$ |
| Reordering shift | | One-proportion test for: $\sum_{i=l/2+1}^{k} I\left(R_i > m(R_1 \dots R_{l/2})\right) = 0.5(l/2)$ |

the network delivers the packet to userspace `recv()` call (and corresponding timestamping). This artifact may induce high delay and/or loss signatures in monitors. Note that context switches may not be unique to end-hosts, and some network devices may exhibit significant delays when the OS resources are busy - though, the likelihood is much higher in end-hosts, since they may be running multiple processes not related to monitoring.

Pythia diagnoses two end-host signatures, both in the delay domain. First, a non-work conserving server inducing "vacation periods" of 100s of milliseconds causes a "large triangle" signature (see Table 8). This signature consists of a *large jump* in delay, followed by a gradual drop. More specifically, if a wait (vacation) period of $W$ is induced while packet $i$ is being served, successive packets will see steadily decreasing wait delays:

$$d_{i+k} = \max\left\{W - \left[T_{i+k} - T_i\right], 0\right\}$$

at the end-host (assuming the other packets do not see vacation periods, and no other sources of delay variation). The duration of this delay triangle is around $W$.

In practice, there may be multiple context switches during this triangle, and the delays may not be monotonically decreasing (due to measurement noise). Our test for a "large triangle" takes into account the salient feature in the signature that is robust to noise: a large increase in delay between consecutive points.

It could be argued that a "vacation period" could be a burst of cross traffic that arrived in sampling duration $\delta$. We choose our threshold for $W$ to avoid matching such cases as follows. Suppose that a burst arrived at a rate $\lambda$ at a link of capacity $C$. If the delay increase was *due to cross traffic*, we have the following condition for the queue backlog: $\left(\frac{\lambda - C}{C}\right)\delta \geq W$; in other words: $\lambda \geq \left(1 + \frac{W}{\delta}\right)C$. In our deployment, $\delta = 100$ms; we can define a threshold for $W$ by choosing an upper bound for the input-output ratio $\lambda/C$. We use a ratio of 4, giving us $W \geq 300$ms in case of a context switch.

Second, context switches in a busy monitor may cause "single-point peaks": waiting periods of 10s of milliseconds, leading to delay *spikes* that are made of a single (or few) point(s) higher than the baseline delay (when the inter-packet gap is typically larger than the context switch duration). Our test captures this feature.

**Unknowns:** End-host signatures are not limited to the above. They can, however, be diagnosed effectively by looking at whether *a significant number of paths originating to/starting from a monitor show a performance problem at the same time*. When an agent tags an event as "Unknown", it checks the Pythia event database for all events with that end-point and with an overlapping time period, and tags all those events as "end-host noise" if a majority of paths were detected as having an "Unknown" or an end-host problem.

**Wireless (802.11) link effects:** An 802.11 link is a common LAN technology in home networks. 802.11 wireless links are non-work conserving, and may induce large "vacation periods" due to channel busy periods, layer-2 backoffs and retransmissions. Further, channel conditions change in small timescales, and the subsequent packets may not see large delays. This behavior leads to a delay signature that *matches* the "large triangle" end-host signature. In this chapter, in the context of ShaperProbe data, we use the term "context switch" to refer to vacation periods in either end-hosts or 802.11 links (if any).

### 6.5.2 Congestion and Buffers

**Network congestion:** We define congestion as a *significant* backlog in one or more queues (due to cross traffic) in the network for an extended period of time (few seconds or longer). Pythia identifies two forms of congestion based on the bottleneck link's queueing dynamics. First, congestion *overload* is a case of a *significant and persistent* queue backlog in one or more links along the path. Overload may be due to a single traffic source or an aggregate of sources with a consistent arrival rate larger than the serving link's capacity. Congestion overload is diagnosed when Pythia observes a high *delay utilization* and when the traffic aggregate is *not bursty* - in other words, there is a contiguous sequence of delays above the baseline that lasts for many seconds.

Second, *bursty* congestion is a case of a significant backlog in one or more network queues, but where the traffic aggregate is bursty (i.e., high variability in the backlog). Specifically, we use the term "bursty" to refer to backlog conditions that are *not persistent*. Bursty congestion is diagnosed when Pythia observes a high delay utilization but the end-to-end delay timeseries shows multiple *hills* such that the longest hill is *relatively short* -

it is shorter than 70% of total hill duration (if the hill is longer, the event is diagnosed as congestion overload). The diagnosis specifications require that any form of congestion is *not* an end-host signature (Figure 55 shows a decision tree representation of this condition).

**Buffering:** A buffer misconfiguration is either an over-provisioned buffer or an under-provisioned buffer. Over-buffering has the potential to induce large delays for other traffic flows, while under-buffering may induce losses (and thus degrade TCP throughput). Pythia diagnoses a path as having a buffer that is either over- or under-provisioned based on two symptoms. First, the delay *range* during the event is either too large or too small. Second, an under-provisioned buffer diagnosis requires that there is *non-random* packet loss during the event (see Section 6.5.3). Note that we do not make any assumption about the buffer management on the path (RED, DropTail, etc.). We choose thresholds for large and small delays as values of end-to-end one-way delays that fall outside the typical delay range in networks (the operator can tune these values based on knowledge of network paths).

### 6.5.3 Loss Events

**Random / delay-correlated loss:** Packet losses can severely impact TCP and application performance. It is useful for the operator to know if the losses that end-to-end flows see on a path are *correlated* with delays. Examples of delay correlated losses include those caused by buffer managers in a router, such as a full DropTail buffer or a RED buffer over the minimum backlog threshold. *Random losses* are events in which packet losses do not show an increase in neighborhood delays. Random losses may be indicative of a potential line card failure, bad fiber or connector, or a duplex mismatch. We define a random loss-based event as the conditional event: $P[\text{ loss } | \text{ delay increase in neighborhood }] = P[\text{ loss }]$, where the *neighborhood* is based on a sufficiently short window of delay samples. We cannot use the definition to diagnose the nature of a loss event, since we may not have sufficient losses during the event to estimate the probabilities. Pythia diagnoses a packet loss as random loss based on the delays in a small neighborhood (on both sides) of the loss. In case of a loss burst, it considers delay samples before and after the burst. If losses are accompanied by delay increases, we diagnose them as "delay correlated loss".

**Short outage:** Network operators are well aware of long outages. Infrequent *short* outages, possibly caused by an impending hardware failure, may be overlooked. A short outage can disrupt existing communication. Pythia diagnoses loss bursts that have a small loss duration (1s up to the event duration) as short outages.

### 6.5.4 Route Changes

A route change is characterized by a *level shift* in the delay timeseries (and possibly, packet loss during the level shift). Routing events could be either long-term route changes, or route flaps. Pythia currently diagnoses long-term route changes. It does so by finding significant changes in the baseline and in the propagation delay. Note that delay level shift events can also be due to clock synchronization at the monitors; such events need to be discarded. Pythia currently reports delay level shifts as "either route change or clock synchronization". We do not support identification of clock synchronization events; this can be done, for example, at a monitor $M$ by correlating delays from all delay timeseries destined to or starting at $M$; if there is a clock synchronization event, *all* timeseries will show a level shift at about the same time.

### 6.5.5 Reordering Events

Reordering may occur due to a network configuration such as per-packet multi-path routing or a routing change; or could be internal to a network device (e.g., switching fabric design in high-speed devices) [28]. Reordering may not be a network pathology, but it can significantly degrade TCP performance. Reordering, if it exists, will be either *persistent and stationary* (e.g., due to a switching fabric or multi-path routing), or *non-stationary* (e.g., routing changes). Pythia diagnoses these two classes of reordering.

The detection logic computes a reordering metric $R$ for each 5s time window of measurements (see Section 6.3). Recall that $R$ is zero if there is no reordering, and increases with the *amount* of reordering on the path.

Pythia diagnoses reordering non-stationarity using a timeseries of the reordering metric $R$; specifically, Pythia looks at the set of $l = 10$ most recent reordering measurements $\mathcal{R} = \{R_1 \dots R_l\}$. No reordering implies that $R_i = 0$ for all $i$. A non-stationarity (or

stationarity) in reordering is diagnosed using the "reordering shift" test.

### 6.5.6 Datasets and Validation

We use four sources of data to validate diagnosis:

- *ESnet:* We have data from the US Department of Energy ESnet network for 12 days using a full mesh of 33 monitors.
- *Internet2:* We have data for 22 days using a full mesh of 9 monitors.
- *ShaperProbe [67]:* We have data from home networks in both directions for about seven months in 2012 - a total of about 58,000 runs. Each run lasts for 10s in each direction.
- *PlanetLab:* We collect data for about half a day in March 2011 in a full mesh of about 70 monitors.

All probing was done with 40 Byte UDP packets, and with an average sampling rate of 10Hz per path. The OWAMP tool uses a Poisson arrival process for probing, while the ShaperProbe and PlanetLab tools maintain a constant packet rate. The ESnet and Internet2 sources of data represent high-capacity wide-area research networks. The ShaperProbe data predominantly comes from home broadband networks. The PlanetLab data covers wide-area commercial and research networks. All monitored networks have wide-area paths.

Our validation methodology is as follows. We first run the detection and diagnosis logic on all of the data to find and categorize events in the data into different "classes" of patterns (where each pattern is a diagnosis type). For each of the four data sources, we choose a uniform random sample of 10 events from each "class"; the total number of events used for ground truth is 382. We shuffle the samples for validation.

The challenge in validation is that we do not have "ground truth" diagnosis for detected events, especially since the traces come from a wide variety of networks and network paths. We hence *generate* ground truth by manually observing the delay and loss timeseries of each event and classifying them into one or more of the diagnosis types (or into an "Unknown" bucket). For each event, we mark as many valid (matching) pathologies as we see fit. This

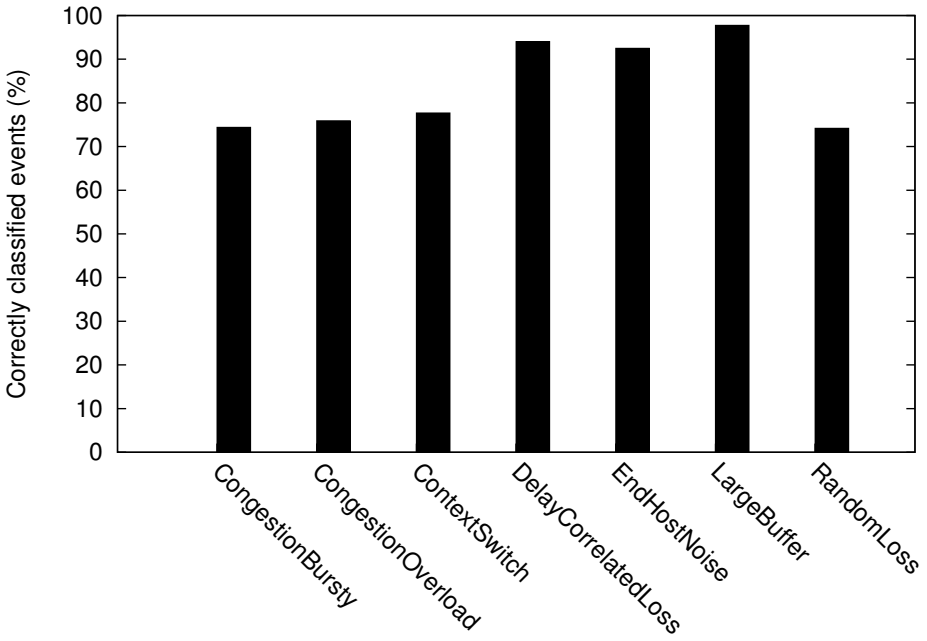



**Figure 56:** Validation of diagnosis logic: classification accuracy of different diagnoses. Datasets include academic, research, commercial and residential broadband networks.

approach has limitations. First, a high range of delays represented in the timeseries (e.g., due to few outliers) may *mask* events that occur over a smaller range. Second, if the event is of large duration (e.g., 100s of seconds long), it may be hard to visualize small timescale behavior (e.g., a small burst of packet losses will show up as a single loss). Third, if there are unsupported features - e.g., the case of short-term delay level shifts - the matching pathology definitions would be wrong, while the ground truth would be correct. Finally, the ground truth generation does not include "cross-path" checks - in other words, looking at all paths originating from/destined to the monitor. An example of such checks is the "EndHostNoise" event (Section 6.5.1).

We compare the diagnosis generated by the agent for each event with the manually classified diagnoses. We ignore events that the user tagged as "Unknown". We did not find any false positive detections. We show two measures of diagnosis accuracy (we use "diagnosis" to imply a set of diagnoses). For each event:

- A diagnosis is correct if *at least one* of the diagnoses exists in the manually classified diagnoses. We find that the diagnosis accuracy across all events is 96%.

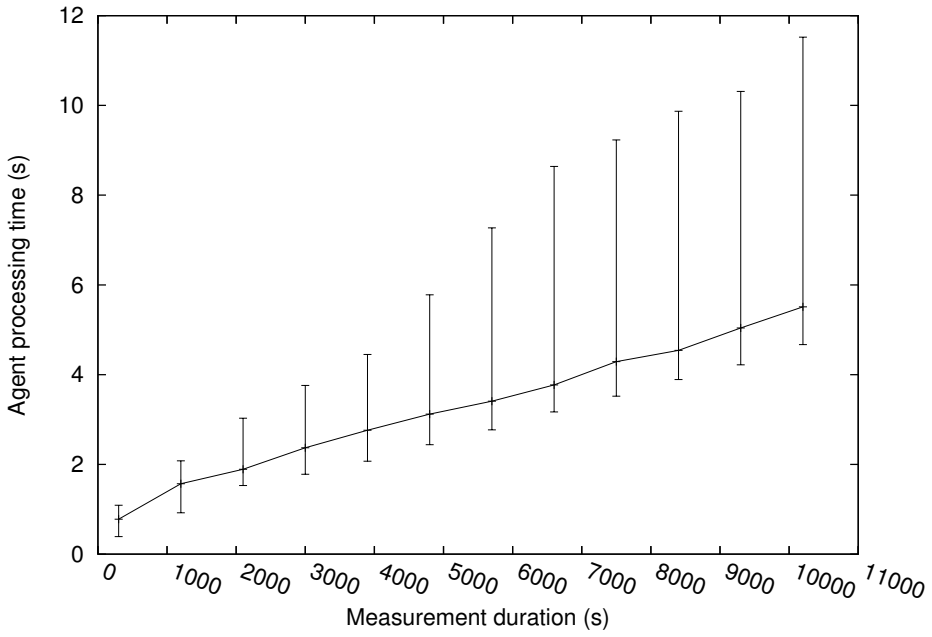



**Figure 57:** Agent run time compared to the duration of input data.

- A diagnosis is correct if *all* of the diagnoses exist in the manually classified diagnoses. We find that the diagnosis accuracy across all events is 74%. There were 1.42 diagnoses per event on average.

Figure 56 shows the validation accuracy breakdown for all diagnosis types; we ignore diagnosis types for which we have less than 35 instances. A diagnosis type for an event is marked as "correct" if it also exists in the ground truth for that event. The "CongestionOverload" and "CongestionBursty" event types include short-term delay level shift events; these are false diagnoses, since Pythia does not support them, and comprise 42% of the total false diagnoses of the two congestion types.

### *6.6 Implementation*

We describe the current (2012) deployment of Pythia in this section. Our prototype has been running since July 2012, and it serves 43 wide-area inter-domain paths at the time of writing. It has diagnosed up to 8,000 events per day so far. The user interface for the live system is at: `http://pythia.cc.gatech.edu`

**Agent:** The scheduler currently supports the perfSONAR toolkit; it can, however, be

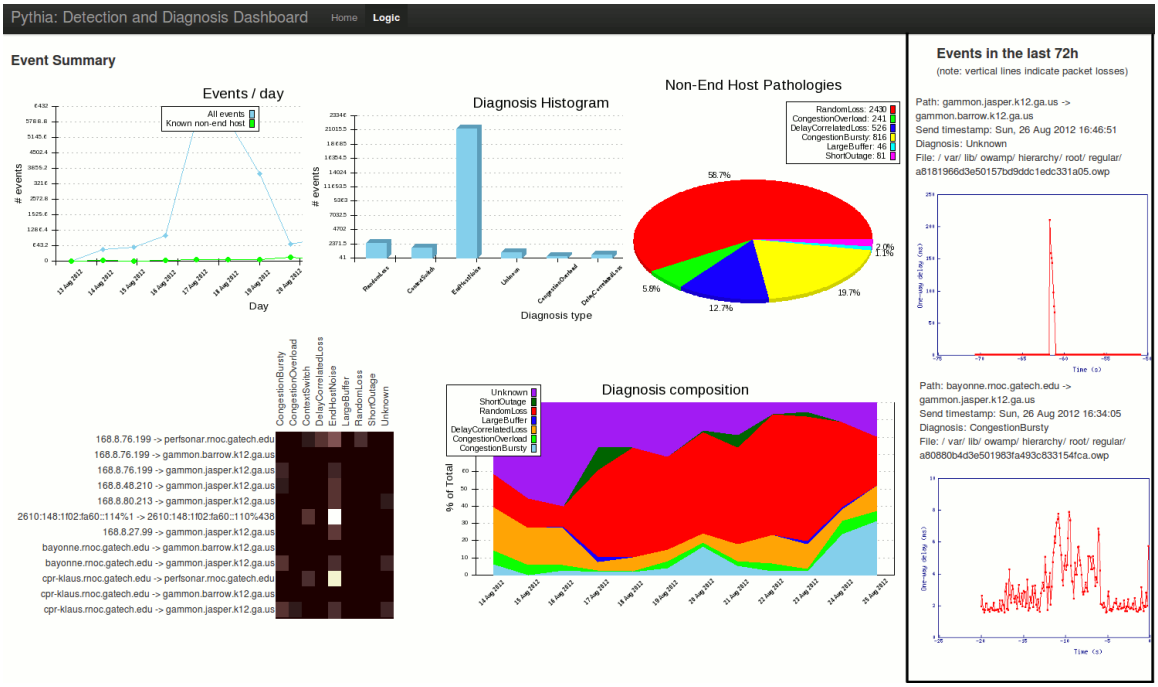



**Figure 58:** The Pythia frontend.

extended to support any monitoring infrastructure that confirms to our monitoring model (Section 6.2). The scheduler does not require any changes to perfSONAR. The agent is a single-threaded process written mainly in Perl, and is about 4,000 lines of code.

The perfSONAR toolkit at a monitor $M$ writes measurement data from as regular files to the disk for each monitored path destined to $M$. The scheduler reads these files in chunks of five minutes and maintains file offsets.

We optimize parts of the agent code for execution time. First, we pre-compile the Gaussian kernel density estimator, which is used extensively by the delay detection module. Second, we minimize disk I/O by reading measurements only once and not writing any temporary files (such as event plots).

The agent takes a small amount of memory, since it reads five minutes of measurement data into memory at a time. Figure 57 shows the agent run time with the duration of input data (median, 5th and 95th percentiles across 100 sample files); we do not include the DB commit latency. The agent takes about two orders of magnitude lesser time to process the data. The agent was run on a multi-core 3GHz Xeon machine.

The agent commits event information to a *MySQL* database. We have observed a nominal amount of CPU usage on the database host at 8,000 commits per day.

**Front end:** The front end is a dynamically generated interface for the operator to the performance problem data; it summarizes the data visually, shows recent events, and allows the operator to specify *filters* using one or more of monitor, path, diagnosis type and time window. It is written in PHP and runs on top of an Apache web server. Figure 58 shows the front end visualizations.

## *6.7 Case Studies of Networks*

In this section, we take a look at performance problems in wide-area research, commercial and residential broadband networks. The reader is referred to Section 6.5.6 for details of data and collection methodology. We compare different aspects of the data below.

### 6.7.1 End-host-related events

Table 9 shows events that are not necessarily related to the network; specifically, it describes the fraction of detected events that were diagnosed as either "EndHostNoise", "ContextSwitch" or "Unknown". We see that the fraction of events that Pythia tags as "Unknown" are typically lower than 10%. The Internet2 and ESnet data come from perfSONAR deployments. The Internet2 data shows a relatively small fraction of end-host-related events compared to ESnet; this is because the ESnet monitors are also used to host other processes such as MySQL data stores (and hence, the OS environment is more likely to be busy), while the Internet2 monitors are resources dedicated to active probing. PlanetLab slices are virtualized instances, and hence are also likely to be busy environments. ShaperProbe data comes from a userspace tool running on commodity OSes in homes. We note that "ContextSwitch" events in ShaperProbe may include events arising from "vacation periods" in either end-hosts or 802.11 access links in homes; we do not differentiate between the two types in this analysis.

In the rest of this section, we focus on events that are neither end-host-related nor "Unknown" events.

**Table 9:** Occurence of non-network events in different datasets. These events are: "E.H.N." - "EndHostNoise", "C.S." - "ContextSwitch", "Unk." - "Unknown".

| Dataset | # events | E.H.N. | C.S. | Unk. |
|---|---|---|---|---|
| *ESnet* | 465,135 | 52% | 43% | 3% |
| *Internet2* | 18,774 | 1% | 3% | 13% |
| *PlanetLab* | 718,459 | 56% | 16% | 1% |
| *ShaperProbe* | 8,790 | 54% | 9% | 9% |

### 6.7.2 Network-related events across network types

For the four datasets, ESnet, Internet2, ShaperProbe and PlanetLab, we found a total of 3727, 15200, 2354 and 185490 network-related events respectively. Figure 59 shows the fraction of each network-related event type across the datasets. We see that events in ESnet and Inernet2 networks have a similar composition. ShaperProbe and PlanetLab have a relatively higher incidence of loss-based events. In particular, PlanetLab has a comparatively high fraction of random loss and short outage events; while ShaperProbe data also includes a significant fraction of congestion events. Note that we do not see a large fraction of "LargeBuffer" events in ShaperProbe data, though the presence of large buffer configurations in home networks is well known [76] - this is because we can sample a "LargeBuffer" event only if cross traffic fills the "large" buffer to a significant extent *during the 10s probing duration.*

### 6.7.3 Residential broadband performance problems

We look at four large residential broadband providers in our dataset: cable Internet providers Comcast, Road Runner and Cox; and DSL provider AT&T. The number of events per ISP per direction ranges from about 250 in the case of AT&T to about 12000 in the case of Comcast (these numbers are a function of the number of runs we received from the ISPs). Figure 60 shows fraction of user runs detected as a performance problem. We select those runs for which we have recorded over 50% of expected number of samples. We see a difference in the number of uplink and downlink detections in the case of cable providers. A plausible explanation is that the DOCSIS uplink is a non-FIFO scheduler, while the downlink is multiplexed with neighborhood homes. Note that the ShaperProbe data has an inherent

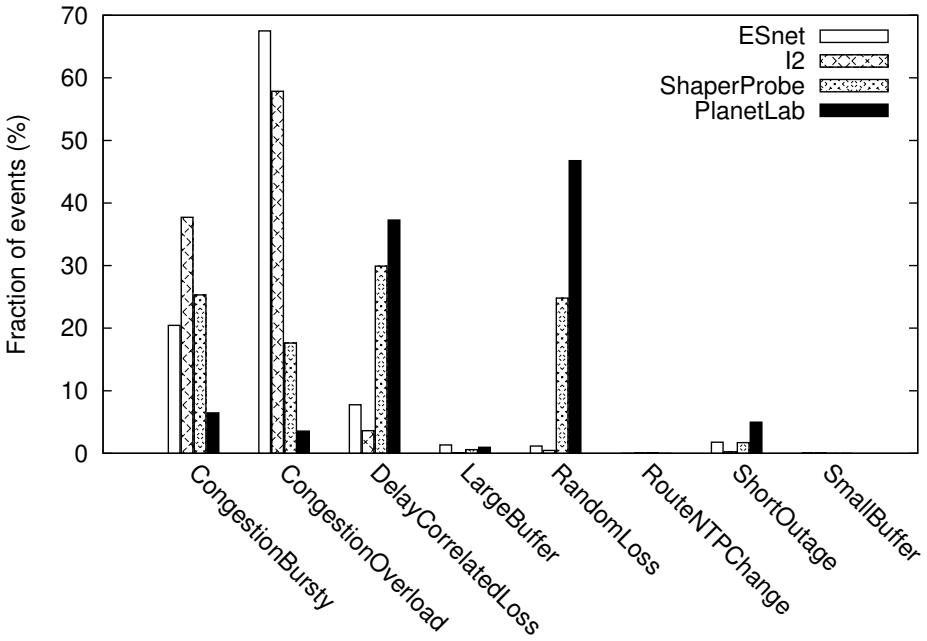



**Figure 59:** Composition of different network-based pathologies among the four datasets (omitting "Unknown", "EndHostNoise" and "ContextSwitch" events).

"bias" - the tool is more likely run by a home user when the user perceives a problem. To cross-check the detection numbers, we looked at the "delay range" - difference in 95th and 5th percentiles of the delay distribution. We found about 59% upstream and 25% downstream Comcast runs had a delay range exceeding 5ms (the default detection threshold for minimum delay range); while, for AT&T, the figures were 45% and 63% respectively.

Figure 61 shows the composition of performance problems across the ISPs. A notable aspect of the plot is that the DSL provider AT&T shows a higher incidence of loss-based events than the cable providers.

We next look at whether cable links show different pathology distributions in the two directions. Figure 62 looks at composition of diagnosis types across all Comcast runs. We do not see a significant difference in the compositions, although there are more problem detections in the cable uplink.

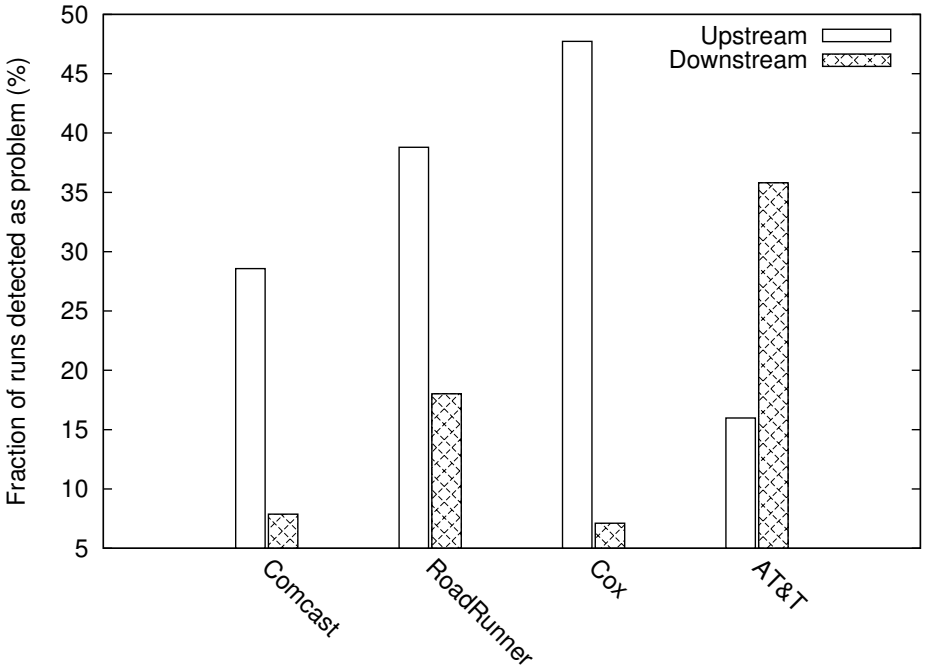



**Figure 60:** Residential broadband networks: fraction of runs detected as pathology in upstream and downstream directions. We look at three cable and a DSL provider.

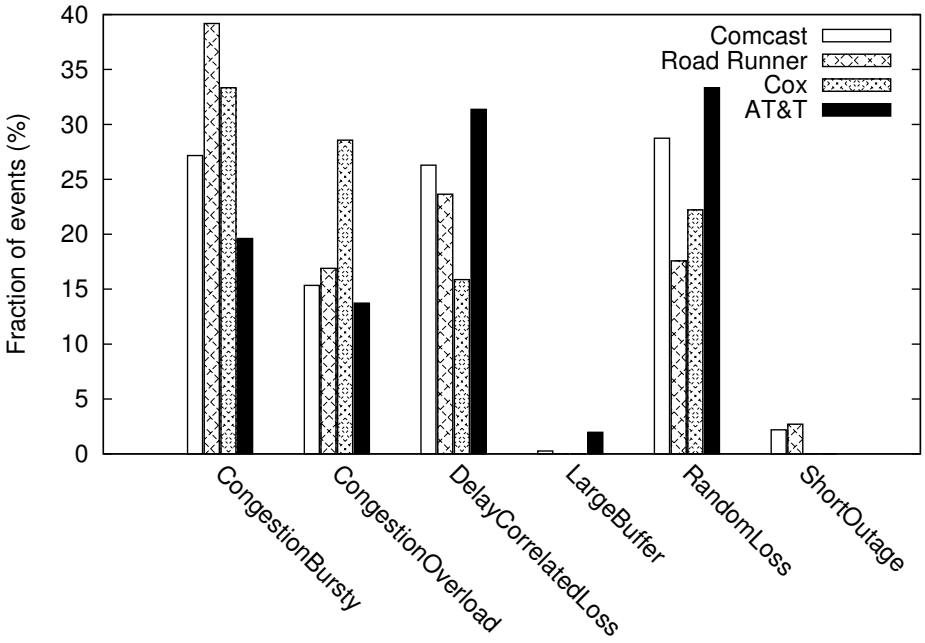



**Figure 61:** Composition of different network-based pathologies among the four residential ISPs (both upstream and downstream).

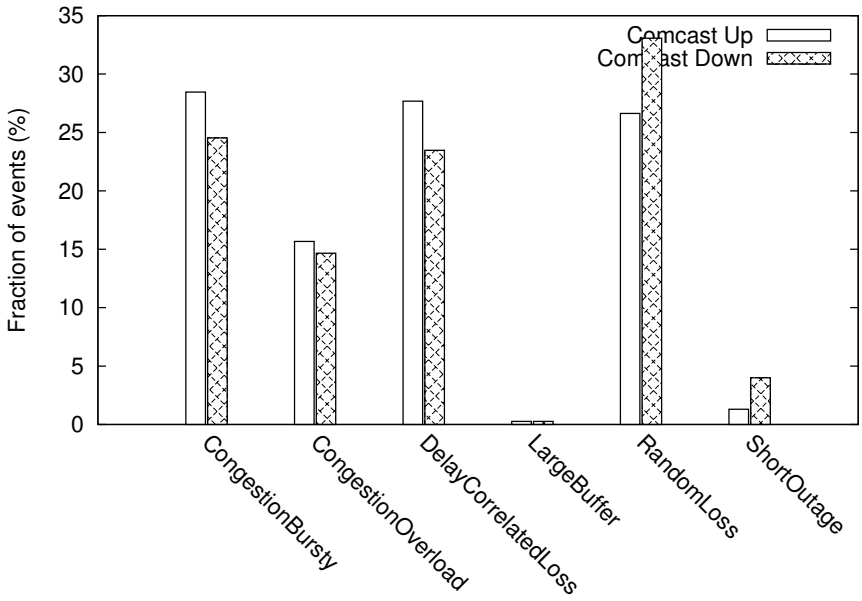



**Figure 62:** Composition of different network-based pathologies in Comcast, as a function of link direction.

### *6.8 Summary*

In this chapter, we have designed and deployed a wide-area, inter-domain, performance problem detection and diagnosis system that works in conjunction with a monitoring infrastructure. Pythia currently works in an inter-domain setting with 43 wide-area inter-domain paths and diagnoses thousands of performance pathologies a day. We have studied the nature of performance problems in three wide-area networks: research, commercial and residential broadband. The live system is at `http://pythia.cc.gatech.edu`, and the Pythia home page is at `http://www.netinfer.net/pythia`.

# CHAPTER VII

# DIAGNOSING WIRELESS PERFORMANCE

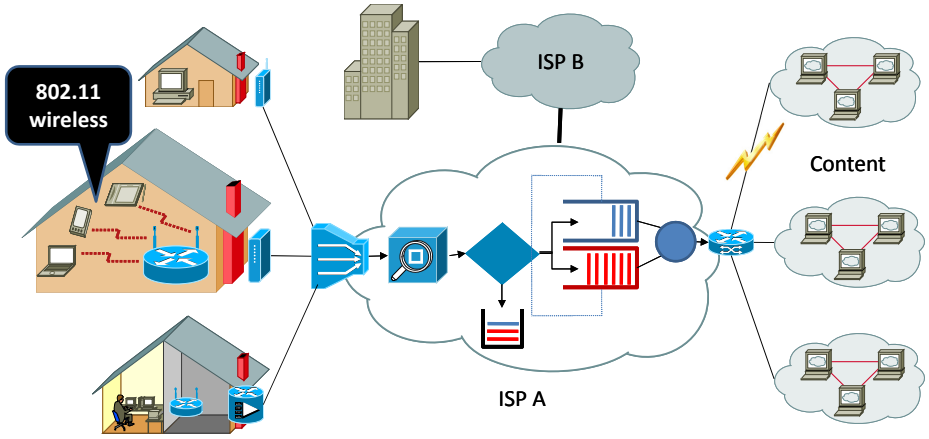



**Figure 63:** Internet model showing a home wireless deployment. Wireless networks are deployed and configured by users and are a source of performance problems.

## *7.1 Introduction*

Most home networks today use an 802.11 Wireless LAN (WLAN) with a single Access Point (AP), typically operating in Distributed Coordination Function (DCF) mode. Home WLANs often suffer from various performance pathologies, such as low signal strength, significant noise, interference from external non-802.11 devices and physical phenomena, various forms of fading, hidden terminals from devices in the same WLAN or in nearby WLANs, or congestion. These pathologies can result in throughput degradation, significant jitter and packet losses. To make things worse, due to the wireless nature of the medium, troubleshooting WLAN performance is hard even for experts, leave alone home users.

User-level probing is a well-established research area in wired networks and it is used in practice to infer various properties and problems in such networks. In the wireless domain, on the other hand, it is still unclear whether user-level probing can be nearly as effective. A main motivation behind this work is to answer the following "intellectual curiosity"

question: *is it possible to diagnose common WLAN performance problems using active probing, without any information from, or modifications to, the 802.11 devices or AP?* The methods presented in this chapter show promising (but preliminary) results, potentially opening a new research thread within the area of wireless networks.

Our objective is to construct a user-level tool for any 802.11 DCF WLAN that can detect: a) low Signal-to-Noise Ratio (SNR), b) Hidden Terminals, or c) congestion. The methodology we propose, referred to as *WLAN-probe*, is a simple, easy-to-use, client-server application that eliminates the need for vendor-specific network card (NIC), driver, AP, monitoring devices, or network modifications. It is also portable across platforms, since it only requires a user-level socket library (e.g., Berkeley sockets, POSIX, or Winsock APIs).

There are several reasons for a user-level probing tool:

**Usability:** We want to build a diagnostic tool that would not require the user to install a specific NIC, AP, or modify the kernel (or administrative privileges). The user would just run a *single instance* of the WLAN-probe client at the wireless link that appears problematic, and a userspace daemon at a wired host connected to the AP.

**Hardware-agnostic:** Most wireless cards today export some form of signal strength; for example, the Received Signal Strength Indicator (RSSI). RSSI implementations are vendor-specific and they are not uniform across NICs. A user-level approach avoids the need to calibrate NIC statistics across devices and drivers on different OSes.[1]

**Software-agnostic:** A user-level approach eliminates the need to write and maintain a hardware-compatibility layer for different OSes that would expose NIC statistics at user-space. An example of that approach is WRAPI [17], designed to work on Windows XP with NICs supporting NDIS 5.1 drivers.

State-of-the-art diagnosis tools either require (or modify) vendor-specific drivers and NICs, or require special monitors at the home network.

WLAN-probe is based on two fundamental effects: a) the fact that low-SNR and hidden

[1]We require that the probing host's NIC is 802.11abg-compliant and does not implement proprietary extensions.

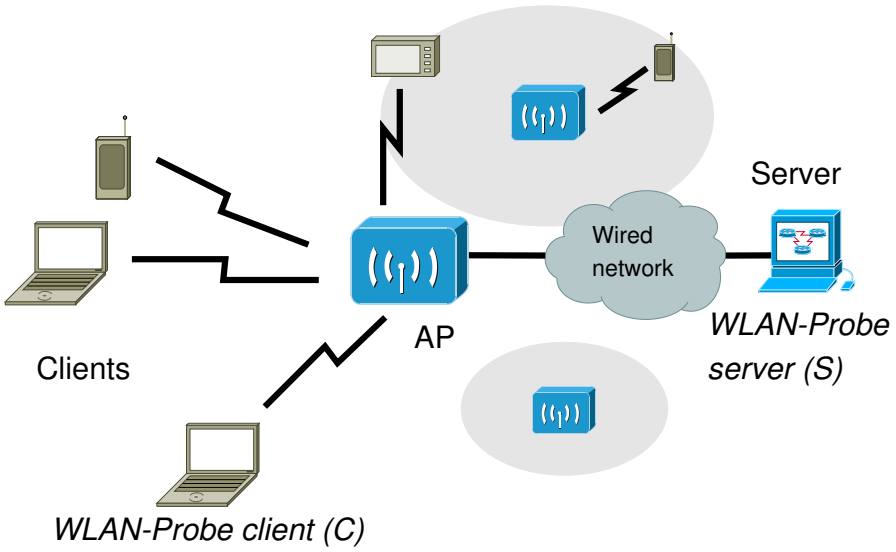



**Figure 64:** System architecture for wireless diagnosis. We use a single wireless end point for probing, and a wired machine connected to the access point.

terminals cause a dependency between packet size and retransmission probability, while congestion does not do so, and b) the fact that low-SNR conditions differ significantly from hidden terminals in the delay or loss temporal correlations they create. However, measuring layer-2 retransmissions and delays is not feasible without information from the link layer. In this chapter we present the basic ideas and algorithms for user-level inference of link layer effects, with a limited testbed evaluation.

We consider the following architecture, which is typical for most home WLANs (see Figure 64). A single 802.11 AP is used to interconnect a number of wireless devices; we do not make any assumptions about the exact type of the 802.11 devices or AP. A user who is experiencing performance problems at her wireless station $C$ runs the WLAN-probe client at $C$. We assume that another computer, used as our WLAN-probe measurement server $S$ is connected to the AP through an Ethernet connection. This is not difficult in practice given that most APs provide an Ethernet port, as long as the user has at least two computers at home. The key requirement for the server $S$ and its connection to the WLAN AP is that it should not introduce significant jitter (say more than 1-3msec). The server $S$ allows us to probe the WLAN channel without demanding *ping-like* replies from the AP and without distorting the forward-path measurements with reverse-path responses. The

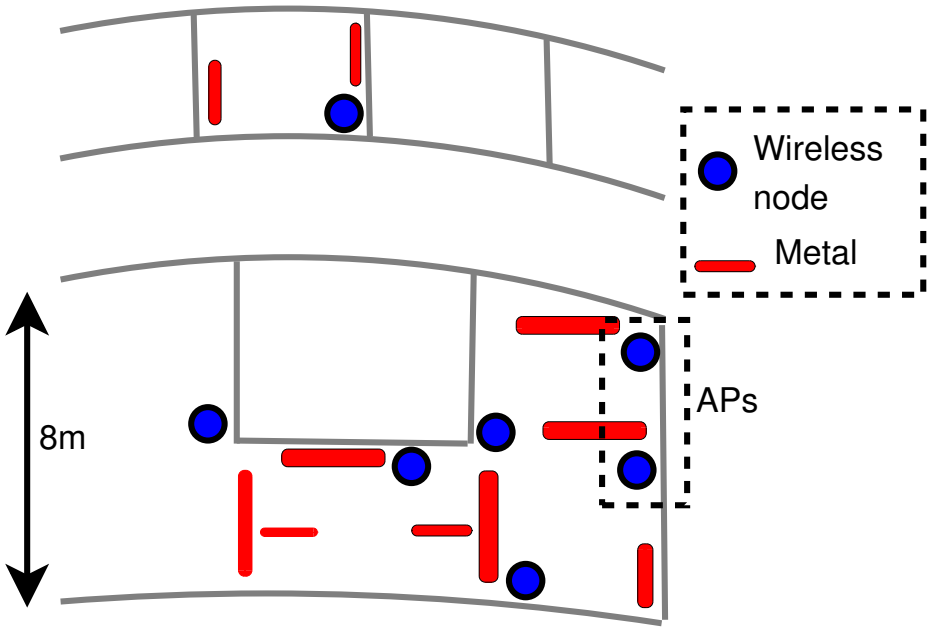



**Figure 65:** Testbed layout showing AP locations for some of the wireless experiments.

measurements can be conducted either from $C$ to $S$ or from $S$ to $C$ to allow diagnosis of both channel directions; we focus on the former. Note that some APs or terminals that are not a part of our WLAN may be nearby (e.g., in other home networks) creating hidden terminals and/or interference, while the user has no control over these networks.

We have conducted all experiments in this chapter using a testbed that consists of 802.11g Soekris `net4826` nodes with mini-PCI interfaces. The mini-PCI interfaces host either an Atheros chipset or an Intel 2915ABG chipset, with the MadWiFi and ipw2200 drivers respectively (on the Linux 2.6.21 kernel). The MadWiFi driver allows us to choose between four rate adaptation modules. We disable the optional MadWiFi features referred to as *fast frames* and *bursting* because they are specific to MadWiFi's *Super-G* implementation and they can interfere with the proposed rate inference process. The testbed is housed in the College of Computing at Georgia Tech, and the geography is shown in Figure 65.

## *7.2 Wireless Access Delay*

The proposed diagnostics are based on a certain component of a probing packet's One-Way Delay (OWD), referred to as *wireless access delay* or simply *access delay*. Intuitively, this term captures the following delay components that a packet encounters at an 802.11 link:

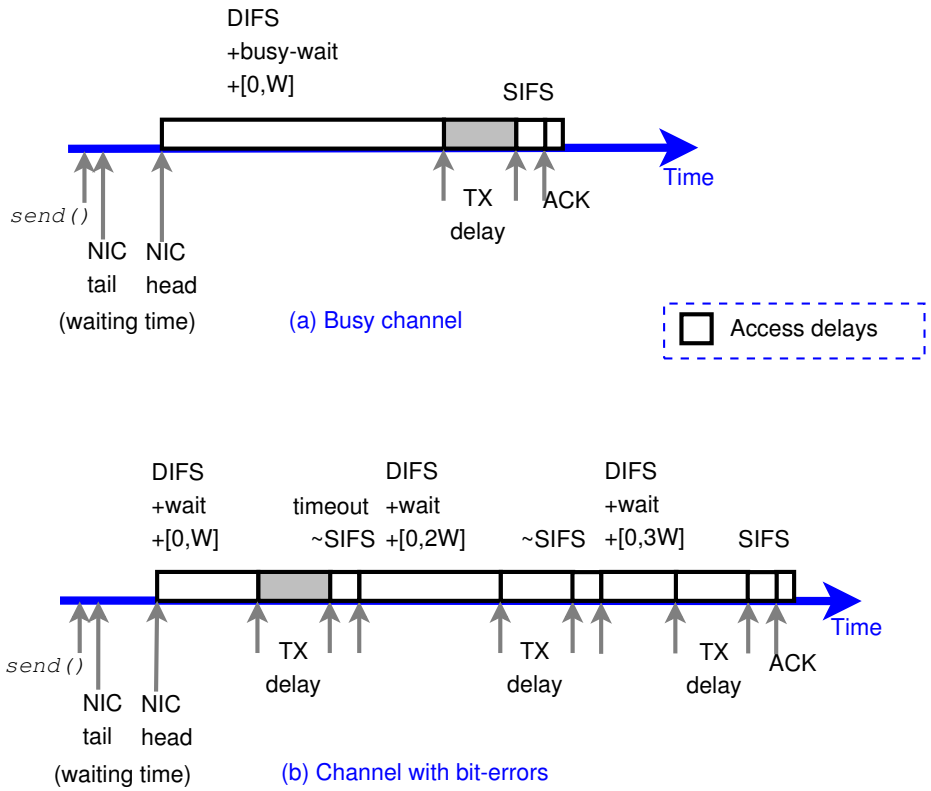



**Figure 66:** 802.11 model used in WLAN-probe: timeline of an 802.11 packet transmission in two pathological channel conditions. The figure also shows *access delay* components.

a) waiting for the channel to become available, b) a (variable) backoff window before its transmission, c) the transmission delay of potential retransmissions, and d) certain constant delays (DIFS, SIFS, transmission of ACKs, etc). The access delay does *not* include the potential queueing delay at the sender due to the transmission of earlier packets, as well as the latency for the first transmission of the packet. The access delay captures important properties of the link layer delays which allow us to distinguish between pathologies; further, we can estimate it with user-level measurements.

Before we define the wireless access delay more precisely, let us group the various components of the OWD $d_i$ of a packet $i$ from $C$ to $S$ (see Figure 64) into four delay components. We assume that the link between the AP and $S$ does not cause queueing delays. The four delay components are illustrated in Figure 66. First, packet $i$ may have to *wait* at the sender NIC's transmission queue for the successful transmission of packet $i-1$ - this is due to the FCFS nature of that queue and it does not depend on the 802.11 protocol. If the

time-distance ("gap") between the arrival of the two packets at the sender's queue is $g_i$, packet $i$ will have to wait for $w_i$ before it is available for transmission at the head of that queue, where $w_i$ is estimated as:

$$w_i = \max\{d_{i-1} - g_i, 0\} \tag{33}$$

We can estimate $w_i$ only if packet $i-1$ has *not* been lost - otherwise we cannot estimate the access delay for packet $i$. The second delay component is the *first* (and potentially last) *transmission delay* of packet $i$. In 802.11, packets may be retransmitted several times and each transmission can be at a different layer-2 rate in general. The ratio $s_i/r_{i,1}$ represents the first transmission's delay, where $s_i$ is the size of the packet (including the 802.11 header and the frame-check sequence) and $r_{i,1}$ is the layer-2 rate of the first transmission; we focus on the estimation of $r_{i,1}$ in the next section. The third delay component $c$ includes various *constant latencies* during the first transmission of a packet; without going into the details (which are available in longer descriptions of the 802.11 standard), these latencies include various DIFS/SIFS segments, the preamble and PLCP header [2], and the layer-2 ACK transmission delay (which is always at the same rate). Finally, there is a *variable delay* component $\beta_i$. When the packet is transmitted only once, $\beta_i$ consists of the waiting time ("busy-wait") for the 802.11 channel to become available as well as a random backoff window (uniformly distributed in a certain number of time *slots*). If the packet has to be transmitted more than once, $\beta_i$ also includes *all the additional delays* because of subsequent retransmission latencies, busy-wait, backoff times and constant latencies. We define the wireless access delay $a_i$ as

$$a_i = c + \beta_i \tag{34}$$

and so it can be estimated from the OWD as

$$a_i = d_i - w_i - \frac{s_i}{r_{i,1}} \tag{35}$$

where $w_i$ is derived from Equation 33.

[2]The preamble can be either long, short or absent, and is assumed to not change in the short probing duration.

Another way to think about the wireless access delay is as follows. Suppose that we compare the OWD of a packet that traverses an 802.11 link with the OWD of an equal-sized packet that goes through a work-conserving FCFS queue with constant service rate $r$ (e.g., a DSL or a switched Ethernet port). The OWD of the latter would include the sender waiting time $w_i$ and the transmission latency $s_i/r$. In that case the term $a_i$ would only consist of the queueing delay due to cross traffic that arrived at the link before packet $i$. In the case of 802.11, the link is *not* work-conserving (packets may need to wait even if the channel is available), the transmission rate can change across packets, and there may be retransmissions of the same packet. Thus, the wireless access delay captures not only the delays due to cross traffic, but also all the additional delays due to the idiosyncrasies of the wireless channel and the 802.11 protocol. A significant increase in the access delay of a packet implies either long busy-waiting times due to cross traffic, or problematic wireless channel conditions due to low SNR, interference etc. In the following sections we examine the information that can be extracted from either temporal correlations in the access delay, or from the dependencies between access delay and packet size. It should be noted that the *access delay* can have additional applications in other wireless network inference problems (such as available bandwidth estimation).

### 7.2.1 Diagnosis tree and probing structure

Having defined the key metric in the proposed method, we now present an overview of the WLAN-probe diagnosis tree that allows us to distinguish between pathologies (see Figure 67). We start by analyzing each packet train separately, and use a novel dispersion-based method to infer the per-packet layer-2 transmission rate, when possible (Section 7.3). Based on the inferred rates, we can estimate the wireless access delay for each packet. We then examine whether the access delays increase with the packet size (Section 7.4). When this is *not* the case, the WLAN pathology is diagnosed as congestion. On the other hand, when the access delays increase with the packet size, the observed pathology is due to low SNR or hidden terminals. We distinguish between these two pathologies based on temporal correlation properties of packets that either encountered very large access delays or that

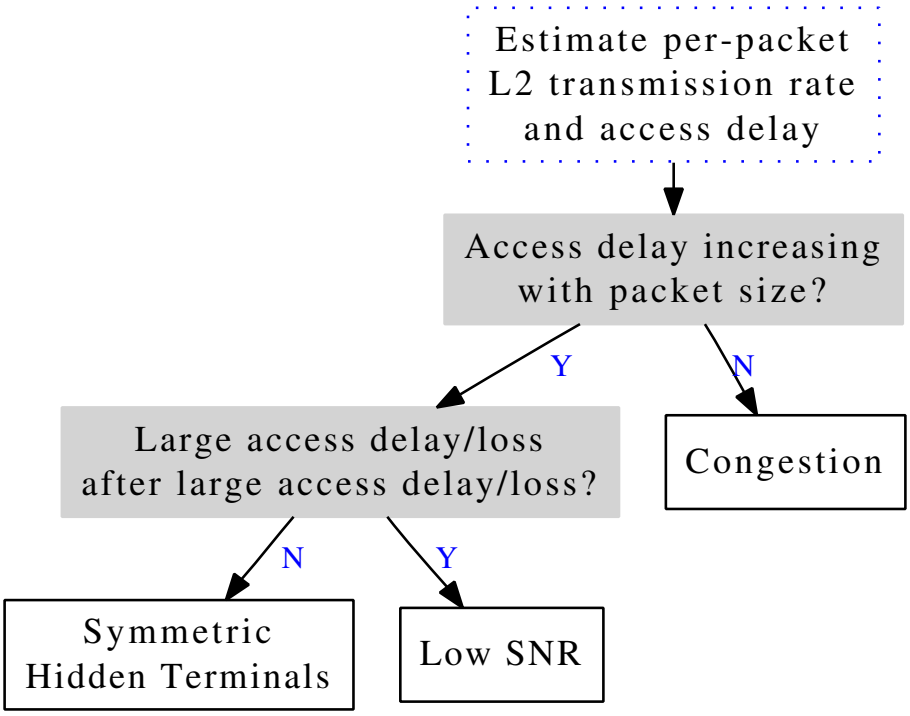



**Figure 67:** WLAN-probe decision tree.

were lost at layer-3 (Section 7.5).

To conduct the diagnosis tests, we need to probe the WLAN channel with multiple packet trains and with packets of different sizes. Each train provides a unique "sample" - we need multiple samples to make any statistical inference. Each train consists of several back-to-back packets of different sizes. The packets have to be transmitted back-to-back so that we can use dispersion-based rate inference methods, and they have to be of different sizes so that we can examine the presence of an increasing trend between access delay and size. Specifically, the probing phase consists of 100 back-to-back UDP packet trains. These packet trains are sent from the WLAN-probe client $C$ to the WLAN-probe server $S$. The packets are timestamped at $C$ and $S$ so that we can measure their *relative* One-Way Delay (OWD) variations. The two hosts do not need to have synchronized clocks, and we compensate for clock skew during each train by subtracting the minimum OWD in that train. The send/receive timestamps are obtained at user-level. There is an idle time of one second between successive packet trains. Each train consists of 50 packets of different sizes. About 10% of the packets, randomly chosen, are of the minimum-possible size (8-bytes for

a sequence number and a send-timestamp, together with the UDP/IP headers) and they are referred to as *tiny-probes* - they play a special role in transmission rate inference (see Section 7.3). The size of the remaining packets is uniformly selected from the set of values $\{8 + 200 \times k, k = 1 \dots 7\}$ bytes.

### *7.3 Transmission Rate Inference*

The computation of the wireless access delay requires the estimation of the rate $r_{i,1}$ for the *first transmission* of each probing packet. Even though capacity estimation using packet-pair dispersion techniques in wired networks has been studied extensively [48, 69], the accuracy of those methods in the wireless context has been repeatedly questioned [80, 100]. There are three reasons that capacity estimation is much harder in the wireless context and in 802.11 WLANs in particular. First, different packets can be transmitted at different rates (i.e., time-varying capacity). Second, the channel is not work-conserving, i.e., there may be idle times even though one or more terminals have packets to send. Third, potential layer-2 retransmissions increase the *dispersion* between packet pairs, leading to underestimation errors. On the other hand, there are two positive factors in the problem of 802.11 transmission rate inference. First, there are only few standardized transmission rates, and so instead of estimating an arbitrary value we can select one out eight possible rates. Second, most (but not all) 802.11 rate adaptation modules show strong temporal correlations in the transmission rate of back-to-back packets. In the following, we propose a transmission rate inference method for 802.11 WLANs. Even though the basic idea of the method is based on packet-pair probing, the method is novel because it addresses the previous three challenges, exploiting these two positive factors.

**Approach:** Recall that WLAN-probe sends many packet trains from $C$ to $S$, and each train consists of 50 back-to-back probing packets (i.e., 49 packet-pairs). Consider the packets $i-1$ and $i$ for a certain train; we aim to estimate the rate $r_{i,1}$ for the first transmission of packet $i$ given the "dispersion" (or interarrival) $\Delta_i$ between the two packets at the receiver $S$. Of course this is possible only when neither of these two packets is lost (at layer-3). Further, we require that packet $i$ is *not* a "tiny-probe".

Let us first assume that packet $i$ was transmitted only once. In the case of 802.11, and under the assumption of no retransmissions of packet $i$, the dispersion can be written as:

$$\Delta_i = \frac{s_i}{r_{i,1}} + c + \beta_i \tag{36}$$

using the notation of the previous section. To estimate $r_{i,1}$, we first need to subtract from $\Delta_i$ the constant latency term $c$ and the variable delay term $\beta_i$ which captures the waiting time for the channel to become available and a uniformly random backoff period. The sum of these two terms $c+\beta_i$ is estimated using the tiny-probes; recall that their IP-layer size is only 8 bytes and so their transmission latency is small compared to the transmission latency for the rest of the probing packets. On the other hand, the tiny-probes still experience the same constant latency $c$ as larger packets, and their variable-delay $\beta$ follows the same distribution with that of larger probing packets (because the channel waiting time, or the backoff time, do not depend on the size of the transmitted packet). So, considering only those packet-pairs in which the second packet is a tiny-probe, we measure the *median dispersion* $\Delta_{\text{tiny}}$. This median is used as a rough estimate of the sum $c+\beta_i$, when packet $i$ is *not* a tiny-probe. [3] We then estimate the transmission rate $r_{i,1}$ as:

$$r_{i,1} = \frac{s_i}{\Delta_i - \Delta_{\text{tiny}}} \tag{37}$$

If the $i$'th packet was retransmitted one or more times, the dispersion $\Delta_i$ will be larger than $s_i/r_{i,1}+c+\beta_i$ and the rate will be underestimated. A first check is to examine whether the estimated $r_{i,1}$ is significantly smaller than the lowest possible 802.11 transmission rate (1Mbps). In that case, we reject the estimate $r_{i,1}$ and *flag* that packet. Of course it is possible that some remaining packets have been retransmitted, but without being flagged at this point. We also flag all tiny-probes, as well as any packet $i$ if packet $i-1$ was lost.

The next step is to map each remaining estimate $r_{i,1}$ to the nearest standardized 802.11 transmission rate $\hat{r}_{i,1}$. For instance, if $r_{i,1}$=10.5Mbps, the nearest 802.11 rate is 11Mbps. (note that this transmission rate applies to the 802.11 frame and so $s_i$ has to include the layer-2 headers).

[3] This estimate is revised in the last stage of the algorithm, after we have obtained a first estimate for the transmission rate during a train. We then estimate the transmission latency of each tiny-probe and subtract it from its measured dispersion.

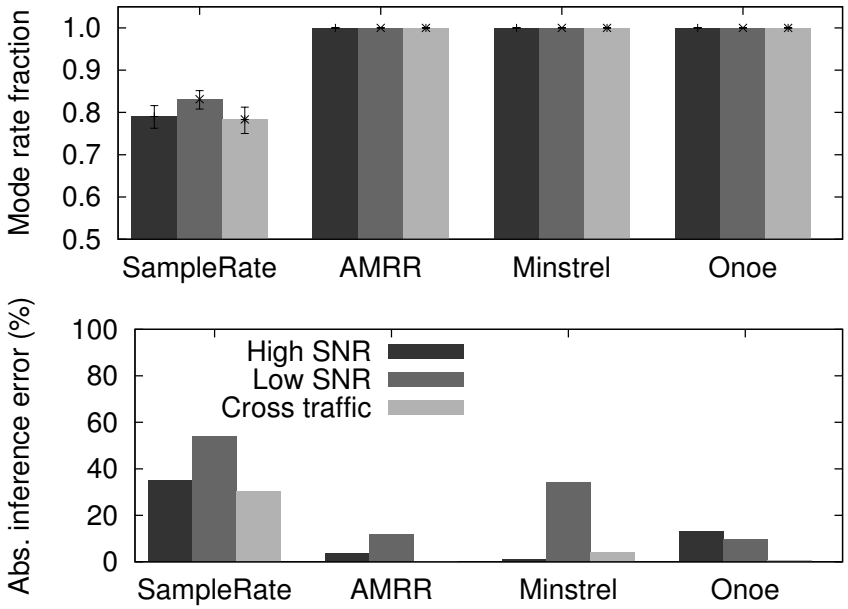



**Figure 68:** Rate inference: strong temporal correlations between the transmission rate of packets in the same train (top) and rate inference accuracy (bottom). Low-SNR conditions are created by separating $C$ and its AP by several meters; congestion is caused by a UDP bulk-transfer over a second network that is in-range.

We also exploit the temporal correlations between the transmission rate of successive packets (within the same train) to improve the existing estimates and to produce an estimate for all flagged packets. We have experimented with the four rate adaptation modules available in the MadWiFi driver used with the Atheros chipset (SampleRate, AMRR, Onoe and Minstrel). Figure 68 (top) shows the fraction of probing packets in a train that were transmitted at the most common transmission rate during that train, under three different channel conditions. These results were obtained from 100 experiments with 50-packet trains; we also show the Wilcoxon 95% confidence interval in each case. Note that all rate adaptation modules exhibit strong temporal correlations, while three of them (AMRR, Minstrel and Onoe) seem to use a single rate for all packets during a train (each train lasts for 5-250msec, depending on the transmission rate).

Based on the previous strong temporal correlations, we compute the *mode* $\tilde{r}$ (most common value) of the discrete $\hat{r}_{i,1}$ estimates. If the mode includes less than a fraction (30%) of the measurements, we reject that packet train as *too noisy*. Otherwise, we replace

every estimate $\hat{r}_{i,1}$, and the estimate for every flagged packet, with $\tilde{r}$. If most trains show weak modes (i.e., a mode with less than 30% of the measurements), we abort the diagnosis process because the underlying rate adaptation module does not seem to exhibit strong temporal correlations between successive packets. In our experiments, this is sometimes the case with the SampleRate MadWiFi module. In the rest of this work, we only use the SampleRate adaptation module (which is also the default in MadWiFi) because we want to examine whether the proposed diagnostics work reliably even under considerable rate estimation errors.

**Evaluation:** Figure 68 (bottom) shows the accuracy of the proposed rate estimation method under three quite different channel conditions. In particular, we show the *average of the absolute relative error* across all probing packets for which we know the ground truth transmission rate. The "ground truth" for each packet was obtained using an AirPcap monitor, positioned close to the sender, that captured most (but not all) probing packets. We detect the first transmission for each packet using the "Retry" flag in the 802.11 header. We see that the inference error is low in most cases; the SampleRate module gives a relatively higher error.

### *7.4 Detecting Size-dependent Pathologies*

The first "branching point" in the decision tree of Figure 67 is to examine *whether access delays increase with the size of probing packets.* Recall that each probing train consists of packets with eight distinct sizes. The pathologies in an 802.11 WLAN can be grouped in two categories: a) pathologies that are more likely to increase the access delay of larger packets, because of increased waiting at the sender or increased retransmission likelihood, and b) pathologies that increase the access delay of all packets with the same likelihood, independent of size. We refer to the former as *size-dependent pathologies* and the latter as *size-independent.*

The first category includes a broad class of problems such as bit errors due to noise, fading, interference, low transmission signal strength, or hidden terminals. In the simplest (but unrealistic) case of independent bit errors, the probability that a frame of size $s$ bits

will be received with bit errors when the bit-error rate is $p$ is $1-(1-p)^s$, which increases sharply with $s$. Of course, in practice bit errors are not independent and 802.11 frame transmissions are *partially* protected with FEC and rate adaptation techniques. We expect however that when the previously mentioned pathologies are severe enough to cause performance problems, larger packets have a higher probability of being retransmitted, causing an increasing trend between access delay and packet size.

The size-independent class includes pathologies that can also cause large access delays, due to increased waiting at the sender or retransmissions, but where the magnitude of the access delay is independent of the packet size. The best instance in this class is WLAN congestion. It is important, however, that the traffic that causes congestion is generated by WLAN terminals that can "carrier-sense" each other (otherwise we have hidden-terminals). In the case of congestion, the access delays will be larger than the case when there is no congestion (packets have to wait more for the channel to become available) but the access delays would not depend on the packet size.

**Approach:** We distinguish between the two pathology classes using statistical trend detection in the relation between access delay and packet size. Figure 69 shows the inferred access delays from experiments with 100 packet trains. In the first experiment (left), the client $C$ and the AP are separated by a large distance of 5-6m, so that $C$'s bulk-transfer throughput drops to about 1Mbps. In the second experiment, we attempt to saturate the WLAN with UDP traffic that originates from another terminal. All terminals and APs can carrier-sense each other (we test this based on throughput comparisons when one or more nodes are active). We use 802.11g channel 6 and SampleRate in both experiments.

The access delays in the case of low signal strength increase with the packet size, while this is not true in the case of congestion. A more thorough analysis of these measurements reveals that *not all* access delays increase with the packet size, under low signal strength. Instead, *the increasing trend is clearly observed among those packets that have the larger access delays for each probing size.* This is not surprising: the packets with the larger access delays among the set of packets of a certain size, are typically those that are retransmitted, and the retransmission probability increases with the packet size under size-dependent

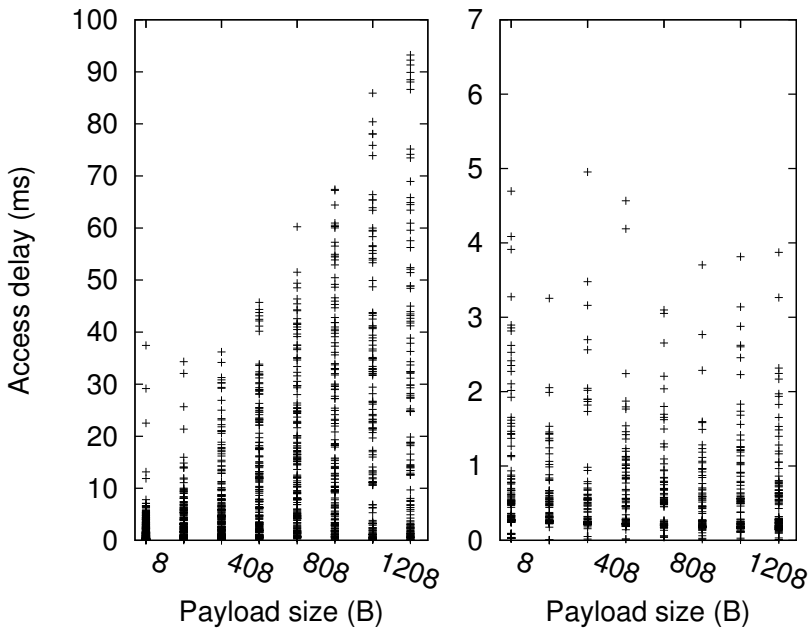



**Figure 69:** Distinguishing bit error-prone channels (low signal strength and hidden terminal cases) from congestion: effect of packet size on access delays (using SampleRate module).

pathologies. For this reason, instead of examining the average or the median access delay for each packet size, we consider instead the 95-th percentile $\tilde{a}_{95p}(s)$ of the access delays for each packet size $s$.

The trend detection is performed using the nonparametric Kendall one-sided hypothesis test [54]. The null hypothesis is that there is no trend in the bivariate sample $\{s, \tilde{a}_{95p}(s)\}$ for $s = \{8 + k \times 200, k = 1 \ldots 7\}$ (bytes), while the alternate hypothesis is that there is an increasing trend.

**Evaluation:** For the experiments of Figure 69 the test strongly rejects the null hypothesis under low signal strength with a p-value of 0 (the p-value is less than 0.01 across all MadWiFi rate modules), while the p-value in the case of congestion is 0.81 (0.7-1.0 across all MadWiFi rate modules). We have repeated similar experiments with all other MadWiFi rate adaptation modules and under different signal strengths and congestion levels. The p-values in all experiments show a clear difference between size-dependent and size-independent pathologies, as long as the received signal strength is less than about 8-10dBm. For higher signal strengths, the user-level throughput is more than 5Mbps, and so it is questionable whether there is a pathology that needs to be diagnosed in the first place.

### *7.5 Low SNR and Hidden Terminals*

After the detection of a size-dependent pathology, WLAN-probe attempts to distinguish between *low-SNR conditions* and *Symmetric Hidden Terminals*. The former represents a wide range of problems (low signal strength, interference from non-802.11 devices, significant fading, and others) - a common characteristic is that they are all caused by *exogenous factors* that affect the wireless channel independent of the presence of traffic in the channel. Symmetric hidden terminals represent the case that at least two 802.11 senders (from the same or different WLANs) cannot carrier-sense each other and when they both transmit at the same time neither sender's traffic is correctly received. Symmetric hidden terminals do not represent an exogenous pathology because the problem disappears if all but one of the colliding senders backoff. The case of asymmetric hidden terminals (or one-node hidden terminals), where one sender's transmissions are corrupted while the conflicting sender's transmissions are correctly received, is no different than the exogenous factors we consider and WLAN-probe will diagnose them as low-SNR.

**Approach:** To distinguish between low-SNR and symmetric hidden terminals, we look at the channel *responsiveness* to probing packets. Under a symmetric hidden terminal condition, a hidden terminal backs off in response to probing traffic; however, under a low-SNR condition, the signal strength is likely to remain low even after introducing probing packets.

We first introduce some additional terminology of events that probing packets may see. A probing packet may be *lost at layer-3* (denoted by L3), after a number of unsuccessful retransmissions at layer-2. A probing packet may see an *outlier delay* (OD), if its access delay is significantly higher than the typical access delay in that probing experiment - we classify a packet as OD if its access delay is larger than the sample median plus three standard deviations (the sample includes all measured access delays in that probing experiment - across all trains). Finally, a probing packet may see a *large delay* (LD) if its access delay is higher than the *typical* access delay in that probing experiment - we classify a packet as LD if its access delay is higher than the 90-th percentile of the empirical distribution of access delays (after we have excluded OD packets). Note that the access delays of OD packets are

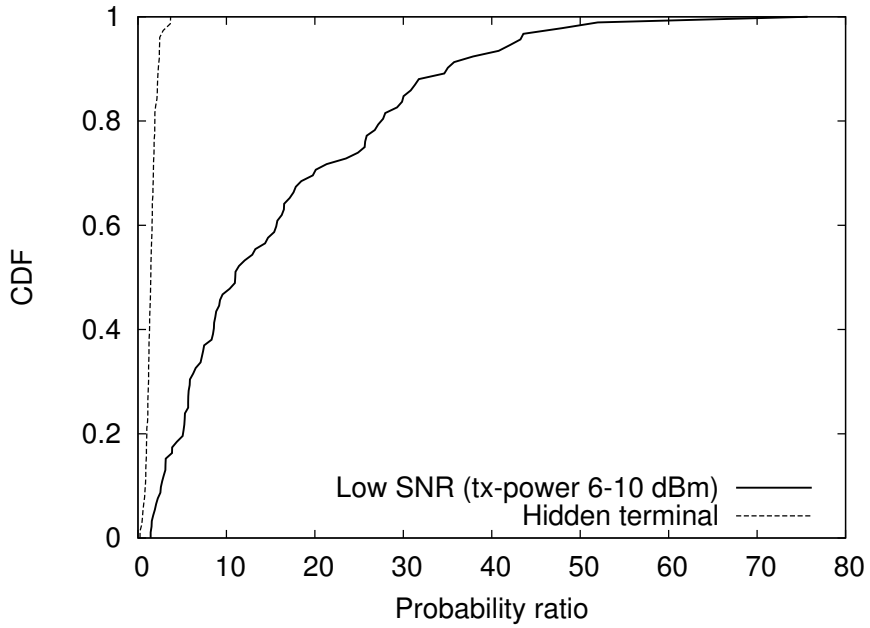



**Figure 70:** Probability ratio ($p_c/p_u$) to distinguish between low-SNR and symmetric hidden terminal conditions. The ratio quantifies temporal correlation in access delays.

typically much larger than the access delays of LD packets.

The probing and diagnosis process works as follows. The probing packets in this WLAN-probe session are of the largest possible size that will not be fragmented. The reason is that larger packets are more likely to collide with other transmissions in the case of symmetric hidden terminals. We then identify all OD or L3 packets in the probing trains of the experiment, and estimate the unconditional probability $p_u$ that either event takes place:

$$p_u = \text{Prob}\left[\text{OD} \vee \text{L3}\right] \tag{38}$$

We then focus on the *successor* of an OD or L3 event, i.e., the probing packet that follows an OD or L3 packet. Under low-SNR scenarios we expect that the channel conditions exhibit strong temporal correlations, and so if a packet $i$ experiences an OD or L3 event, its successor packet $i+1$ (denote by *successor(i)*) will see a large delay (LD) or layer-3 loss (L3) event with high probability.

On the other hand, if packet $i$ experiences an outlier delay (OD) or a loss (L3) event due to a symmetric hidden terminal, the colliding senders will backoff for a random time period and it is less likely that the successor packet will be LD or L3. To capture the

previous temporal correlations between an OD or L3 packet and its successor, we consider the conditional probability of a successor LD or L3 event:

$$p_c = \text{Prob}\left[\text{successor}(i) : \text{LD} \vee \text{L3} \mid i : \text{OD} \vee \text{L3}\right] \tag{39}$$

The detection method focuses on the ratio $p_c/p_u$ of the conditional and unconditional probabilities. If there is a strong temporal correlation between a probing packet that experiences an OD or L3 event and its successor, this probability ratio will be much larger than one. We expect this to be the case under low-SNR conditions. Otherwise, under a symmetric hidden terminal condition, the previous temporal correlation is much weaker and the probability ratio will be closer to one.

**Evaluation:** Figure 70 shows the distribution of the probability ratio $p_c/p_u$ for 100 low-SNR and 90 symmetric hidden terminal experiments in our testbed. We create low-SNR conditions by reducing the transmission power of the WLAN-probe client $C$ to 6-10dBm; the access point is about 3m away. We create hidden terminal conditions using two different networks on 802.11g channel 6, such that the two senders cannot carrier-sense each other. When only one sender is active, the throughput in the corresponding network is higher than 10Mbps - when both senders are always backlogged, the throughput drops to less than 1Mbps. The probability ratio is always less than 5 under symmetric hidden terminals, while it is higher than 5 in 80% of the experiments under low-SNR conditions. A probability ratio threshold between 3-5 should be sufficient to diagnose almost all symmetric hidden terminal conditions accurately. Under low-SNR conditions, however, we should expect some diagnosis errors: in 10-20% of the cases, WLAN-probe will diagnose a low-SNR condition as symmetric hidden terminal.

## *7.6 Summary*

In this chapter, we proposed a home WLAN diagnosis process that only requires user-level active probing, and presented some preliminary but promising results that show the feasibility of such diagnostics. A design consideration for our methods is *usability*: we do not require administrative privileges, any form of support from the wireless card/driver/AP,

or sensor nodes at vantage points in the home. We showed the accuracy of the methods using a testbed that emulates home wireless conditions.

The WLAN-probe home page is at `http://www.netinfer.net/wlan-probe`. Parts of this work appeared in a prior publication [66].

# CHAPTER VIII

# LESSONS

In this thesis, we have designed inference tools for different types of networks and network elements. We distill some of the lessons we learned in this chapter.

## *8.1 Impossibility Conditions*

Inference methods should be aware of, and test for, conditions under which they would not be able to accurately detect, estimate or diagnose accurately the property in question. In particular, methods that are based on domain knowledge models should check for conditions that violate assumptions made by the model. For example, available bandwidth estimation tools assume that the end-to-end path is comprised of work conserving, fixed capacity, FIFO schedulers; 802.11 wireless and cable upstream links, both of which are not uncommon in residential access networks, violate these assumptions. When an impossibility condition is triggered, the inference method should abort operation (and if required, notify the user with a diagnostic message).

We tackle impossibility conditions that arise from models; for example: In DiffProbe, we include a "BLP" probing period to test for our assumption that the normal traffic queue(s) have a backlog in the discriminating link (Section 3.3). We also check whether there are sufficient (based on the statistical test used) number of losses to test for impossibility in loss discrimination detection. If we did not test for these conditions, DiffProbe may output false negatives when detecting discrimination. In Spectral Probing, we test for routing stability before probing for shared links (Section 5.3).

We note that some assumptions may be valid with a high likelihood, and it may not be necessary to check violations of such assumptions to maintain high inference accuracy. For example, our model for ShaperProbe assumes that the bottleneck link and the shaping module are both located close to the home (and hence, the server used should not affect the inference). Our model also assumes that the token bucket is full when the tool starts

probing. These assumptions can be checked by repeating the measurements using different servers and at different points of time (and cross-checking the inference). In practice, we have not seen a high variability in results across user runs across time of day and across servers used. Checking violations of assumptions may also lower the usability and non-intrusiveness of inference, as it may need additional probing and processing time.

## *8.2 End-host Accounting*

End-hosts can add significant *noise* to measurements, due to, for example, busy operating system environments. This statement is particularly true in case of long-term continuous sampling methods (for instance the perfSONAR monitoring toolkit uses 10 Hz sampling on each path). Alert monitoring systems and diagnosis engines should be aware of end-to-end delay measurement patterns induced by the monitors themselves rather than the network. Passive detection and estimation methods should also account for potential end-host-based measurement artifacts. End-host effects are not likely to affect very short-term probing (that lasts for less than the operating system context switch duration) - for example, probing patterns such as short-duration packet trains (order of tens of packets).

We show end-host-induced patterns in one-way delay measurements in Pythia, in particular the "large triangle" and the "single point peaks" signatures (Section 6.5.1). The "large triangle" signature can induce hundreds of milliseconds of waiting time during probing. We also find that end-host signatures are not limited to the two patterns, and we describe methods to identify other end-host effects by cross checking pathologies across measurement paths at a monitor. When implementing ShaperProbe, we craft low-CPU probing structures (sequence of small packet trains) to prevent end-host-induced latencies when probing in the duration of minutes (Section 4.3.2).

## *8.3 Ground Truth*

Ground truth is hard to obtain when evaluating end-to-end inference methods. This is particularly true when the end-to-end path traverses administrative domains that the tool designer does not have access to. In this thesis, we take the following approach towards ground truth. First, we use simulations and emulations (where possible) to establish ground

truth and test our methods. For example, to evaluate DiffProbe, we setup wide-area emulation of software-based discriminatory scheduling and buffer management with cross traffic; and we used simulations to test conditions with significant cross traffic flows. To set up ground truth for wireless diagnosis (WLAN-probe), we used additional nodes in monitor mode in our testbed. Note that simulations and emulations are not always feasible - for example, it is hard to set up wide-area inter-domain pathologies such as router line card failures or fiber issues. When the inference methods are based on domain knowledge, an approach is to employ *manual inspection* of visualizations (e.g., timeseries plots) to "generate" ground truth based on our knowledge; we use this approach to test accuracy of our pattern matching methods in Pythia, and to validate wide-area shared link detection in spectral probing (the existence of periodicity in delays during probing, and the absence just before probing, indicates a high likelihood of shared links).

Second, we test the inference methods *in the wild*, by deploying the method as a public service. A powerful method to look at accuracy at this stage is to use *crowdsourcing*. In other words, we look at inference output and measurements across a large set of users to draw statistically significant conclusions from the data. We can infer accuracy of the tool by computing confidence intervals based on this data. We have employed this approach in our ShaperProbe work.

### *8.4 Wired vs. Wireless*

There has been significant work on active probing in end-to-end wired paths; for example, to estimate the end-to-end available bandwidth or the narrow link capacity. These techniques, however, are based on models that are specific to wired paths. For example, many models assume FIFO scheduling, work conserving, and constant service rate links along the path. Many of these assumptions do not hold for paths that include wireless links such as 802.11 and mobile links. Techniques for end-to-end paths that include wireless (or any non-FIFO, non-work conserving, variable service rate) hops will need to re-design the path model before considering reusing techniques from wired networks.

Let us consider a specific example from this thesis. In our work on WLAN-probe, we

defined a delay model for 802.11 links that included variable delays, retransmissions and constant delays in layer-2 (as well as delays due to self queueing in the end host). We adopted the packet pair technique from wired paths to this model to estimate wireless link capacity by further considering the behavior of rate adaptation algorithms (Section 7.3). We also discarded measurements from probing trains which violated our model. We expect that such models will need to be defined for mobile data links (such as HSPA, EDGE, GSM, CDMA, LTE etc.) for estimation or diagnosis methods.

Last but not the least, active probing methods should consider to what extent they may be perturbing the wireless channel when they introduce packets into the network. In particular, sending packets may change the client rate adaptation algorithm's behavior and may cause random backoffs in hidden terminals (if any).

## *8.5 Wireless Inference Limitations*

Our experience in designing diagnosis methods for 802.11 wireless has revealed two limitations. First, rate adaptation algorithms may make it hard to design delay-based inference methods (unless otherwise the inference tool has access to per-frame transmission rate). This is because the transmission delay of a frame may be of the order of, or higher than, delays due to the 802.11 standard. In our research, we observed that many rate adaptation algorithms exhibit *temporal constancy* in the modulation rate (Section 7.3). An adversarial rate adaptation module can modulation to affect inference accuracy.

Second, it may be hard to detect pathological *asymmetric hidden terminal* conditions using a single wireless end point without prior information about transmission patterns of the hidden terminal. This is because the terminal whose transmissions are always successful will not be seeing bit errors, and hence will not undergo random backoffs and retransmissions due to the other terminal.

## *8.6 Non-standardized Implementations*

Non-standardized implementations are vendor-specific features that are added to (potentially) aid performance. Examples include the rate adaptation algorithms in 802.11 network

cards, and the *fast frames* and *bursting* features in the MadWiFi 802.11g driver. Non-standardized implementations may affect inference accuracy if they significantly change packet forwarding behavior. For instance, the MadWiFi features mentioned above can impact capacity estimates that are based on packet pairs, since the implementation, when turned on, sends bursts of 802.11 frames in a single 802.11 transaction (instead of waiting for channel idle state for each frame). Domain knowledge-based methods should be aware of behavior induced by non-standardized implementations. This is a potential limitation of domain knowledge-based methods, since non-standardized implementations can be propreitary.

## *8.7 Endnote: Designing New Inference Tools*

We end this chapter with an overview of our approach towards writing new end-to-end inference tools. We have applied this approach to inference in heterogeneous wired paths as well as wireless networks, in local area and inter-domain settings; and we believe that it will be a useful approach for many other inference scenarios. Our approach is as follows. First, we design a model for an end-to-end path that incorporates domain knowledge of the different types of network elements and cross traffic that can be present on the path (or a network of paths); the model will list assumptions made, and it will not include possible network properties that will not affect the inference. Using this model, we design packet probing structure, the measurements that will be taken for each packet, and the statistical methods for inference and diagnosis.

Second, we implement the inference methods in a user-level end-to-end tool/system. In doing so, we consider user-level constraints such as accuracy (or granularity) of packet timestamping and packet timing. Our probing should be non-intrusive: it should not affect the end-host or network environments adversely (e.g., by context switches or by impacting cross traffic). The implementation should account for noise in measurements (e.g., using some form of filtering). We the look at accuracy and validation, which we have discussed earlier in this chapter.

# CHAPTER IX

# CONCLUSION AND OPEN PROBLEMS

## *9.1 Thesis Conclusion*

In this dissertation, we presented a number of techniques to discover Internet path properties and to diagnose Internet performance. In particular, we have shown that *end-to-end methods can be designed based on domain knowledge; and without requiring special hardware or software support, or data from the network.* The methods span single administrative domain (e.g., homes and ISPs) to multiple administrative domains (the wide area, inter-domain Internet); and from wired paths to paths with wireless links.

We have performed a large-scale study of Internet properties and performance using some of our inference methods. First, we showed evidence of traffic shaping deployments in residential home ISPs. Second, we did not find any evidence of traffic discrminination against Skype and Vonage traffic in several residential ISPs. Third, we studied the composition of different performance pathologies in residential, commercial and academic networks.

## *9.2 Future Work*

There is a rich set of open problems in inference. For example, inference problems in emerging technologies. Such measurement systems could create a platform to understand Internet evolution and contribute open-access data that allows further research.

### 9.2.1 Ground truth and validation

Ground truth of network configuration or performance pathologies is hard to obtain, particularly in the inter-domain Internet. In this thesis, we have used several approaches (emulations, simulations, small-scale Internet experiments as well as large-scale crowdsourcing) to look at accuracy of our methods. We have validated our observations using publicly available data from ISPs. Our focus has been on designing inference methods that enable further understanding of Internet performance, and to study Internet performance using some of

the methods. A large-scale deployment of the other tools (DiffProbe, spectral probing and WLAN-probe) will give us valuable data about the Internet, application performance and user experience. This could allow better validation when combined with operational data from ISPs and networks, and activity in 802.11 wireless networks.

### 9.2.2 Schedulers and buffer managers

In Chapter 3 we designed a tool, DiffProbe, to determine the existence of discriminatory scheduling and buffer management in ISPs. A large-scale measurement study of ISP discrimination practices using DiffProbe is an open project that can help both residential Internet users and policy makers understand broadband performance.

We propose two extensions to DiffProbe. First, to detect more than two classes of service, additional packet scheduling and buffer management mechanisms, and also to quantitatively characterize the parameters of some mechanisms (e.g., to infer the WRED parameters or the WFQ weights). Second, to generate of additional pairs of $A$ and $P$ flows for applications such as BitTorrent and for IPTV applications such as Veoh and Hulu.

A work that complements detection and estimation of discriminatory buffer managers is the problem of detecting the type of a non-discriminatory buffer manager. Specifically, we propose the problem of detecting the presence of Random Early Detect (RED) buffer managers in an end-to-end path. We have done some preliminary work towards a solution. The basic idea is that DropTail (DT) queues show a *temporal correlation* in losses, while RED queues will show a relatively lower correlation structure (assuming a single lossy link in the path). This method is robust to the presence of shared buffers in network devices.

### 9.2.3 Spectral probing applications

In Chapter 5 we presented the Spectral Probing (SP) framework, and used it in the shared link detection problem. We briefly discuss two applications of spectral probing. First, we can use SP to create *covert channels* between hosts that are not allowed to exchange any data traffic or that are under surveillance. Denote a path between end hosts A and B as P(A,B). Consider paths P(A,B) and P(C,D) and suppose that they have one or more shared bottleneck queues. A can generate delay signals of different frequencies, which can

be detected at the delay measurements from C to D. Note that A does not send any traffic to C or D. D detects A's signal simply from the spectral characteristics of the delay variations in path P(C,D). We have verified with Internet experiments that this application is feasible in practice.

Another application is based on the notion of *frequency modulation* in analog communications. Recall that this type of modulation translates the spectrum of a *base signal* in the frequency domain, so that the modulated signal is more robustly transmitted (and multiplexed) on a given channel. Similarly, the base signal in our context can be any periodic packet stream, such as a voice or video stream. Such applications typically generate one packet every few tens of milliseconds. That frequency, however, may not be ideal for a given IP path. Queueing delay variations (jitter), because of other traffic between A and B or because of delay crosstalk from other paths, can be detrimental to the performance of a voice or video stream. One possibility is to modify the video transmission rate (at the source itself or at an application gateway at the edge of the source network) to a frequency in which path P(A,B) experiences less jitter (spectral noise). Frequency increase can be achieved by segmenting video packets into smaller packets, while a frequency decrease can be achieved by aggregating multiple consecutive video packets.

### 9.2.4 Inter-domain performance

In Chapter 6 we designed an inter-domain performance diagnosis system called Pythia. There are open questions and features that can be incorporated into Pythia. First, the addition of *localization* functionality to diagnosed events. Second, augmenting "Unknown" diagnoses with a *similarity* measure over a pre-specified space of features. Similarity will compare the event against a representative set of diagnoses to find the *most similar diagnosis* with respect to the known pathologies. Third, scaling Pythia to a large number of monitors; this may require tuning the the database tier to scale vertically.

### 9.2.5 Home wireless performance

In Chapter 7 we designed a diagnosis tool, WLAN-probe, for root cause diagnosis of home 802.11 wireless pathologies. There are several possible extensions of WLAN-probe. First,

since WLAN-probe is initiated by the user, it assumes that there is a performance pathology, and that the pathology is located in the wireless network. It is possible that there is no WLAN pathology - this requires a method that can distinguish between normal (baseline) operation and the wireless pathologies. Second, some preliminary work shows that we can detect certain non-802.11 interference sources, such as microwave ovens; recent efforts have used passive methods to detect non-802.11 interference [104]. Third, there is scope for improvements in the rate inference method and testing these methods with additional commodity rate adaptation mechanisms. Finally, a larger-scale evaluation of the WLAN-probe diagnostic accuracy with actual home WLAN deployments, and a large-scale study using WLAN-probe will shed light on wireless performance pathologies in homes.

### 9.2.6 Inference in emerging technologies

Many tools have been written to understand the performance of the mobile Internet (3G, 4G, WiMAX, 802.11, etc.); specifically, these tools estimate user-level factors such as TCP throughput and round-trip time. These metrics do not explain, however, the MAC-layer behavior of the mobile device that likely caused the bad performance. We propose to design user-level methods and build usable tools to understand MAC-layer characteristics of mobile devices. The methods would answer layer-2 performance questions, for example: (1) is the user too far from the base station? (In other words, is there a low signal strength?), (2) are there MAC-level retransmissions (if the MAC layer supports it) due to interference or a high noise level?, (3) what could be the cause of the interference?, and (3) How do such problems affect the energy consumption on the mobile device? Note that a part of this research is to infer if an end-to-end performance problem is due to the mobile link.

There has been recent interest in root-cause analysis of performance problems in cloud infrastructures and data center environments. The current methods discover dependencies in data from different knowledge sources inside the cloud, enabling operators to debug performance. A missing focus is *user-centric*: what can we tell the user of a service in the cloud or a data center about the end-to-end service performance (in particular, if the root cause is within the cloud)? Designing such methods presents many challenges: (1) the user

does not have access to service or network data from the cloud, (2) we cannot introduce significant probing traffic to the data center, and (3) the user may not have historic data from the cloud, data center, or the service. The methods could utilize performance diagnosis data from other users across the Internet - allowing the possibility of a performance diagnosis as a user service.

### 9.2.7 Improving application performance using inference

User quality of experience can be impacted by factors such as *application performance*, and *energy usage* in mobile devices. The performance of distributed applications and systems can be improved using network inference. In prior work, we investigated how estimates of link utilization can improve performance of DNS-based ingress load balancing in multihomed content providers [68]. Traffic shaping estimates could help bulk-transfer applications such as Bittorrent improve throughput of content transfers (ShaperProbe is currently a plug-in to the Vuze client).

We propose two pieces of work in this area. First, the possibility of collating inference information from the methods in this thesis to understand network performance - we propose a user-level library or API (call it *NetInferLib*) that does active probing and passive measurements to infer network elements in the path. Such an API provides a quick interface to applications, end-users, and service providers. NetInferLib reduces the probing overhead by using efficient probing methods and combining probing structures that may be common across inference methods. NetInferLib allows each application to specify what properties to infer and the granularity, frequency, and type of inference (e.g., "is value of property $M$ greater than a threshold $\alpha_M$?"). Second, we propose to design methods for (and understand the extent of and limitations to) application performance improvement using inference knowledge from methods presented in this thesis.

## REFERENCES


[1] *AT&T FastAccess Business DSL Plans (May 12, 2010).* http://smallbusiness.bellsouth.com/internet_dsl_services.html.

[2] *AT&T FastAccess DSL Plans (May 12, 2010).* http://www.bellsouth.com/consumer/inetsrvcs/inetsrvcs_compare.html?src=lftnav.

[3] *Cisco Systems: Configuring Weighted Random Early Detection (2010).* http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfwred_ps1835_TSD_Products_Configuration_Guide_Chapter.html.

[4] *Cisco Systems: Congestion Avoidance (2010).* http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconav.html.

[5] *Comcast Business Class Internet (May 12, 2010).* http://business.comcast.com/internet/details.aspx.

[6] *Comcast High Speed Internet FAQ: PowerBoost (July 2009).* http://www.comcast.com/Customers/Faq/FaqCategory.ashx?CatId=377.

[7] *Comcast High-Speed Internet (residential; May 12 2010).* http://www.comcast.com/Corporate/Learn/HighSpeedInternet/speedcomparison.html.

[8] "Comparing Traffic Policing and Traffic Shaping for Bandwidth Limiting," *Cisco Systems: Document ID: 19645.*

[9] *Cox: Residential Internet (May 12, 2010).* http://intercept.cox.com/dispatch/3416707741429259002/intercept.cox?lob=residential&s=pf.

[10] *Data from M-Lab Tools (May 2011).* http://www.measurementlab.net/data.

[11] *Internet performance monitoring, Keynote Systems.* http://www.keynote.com.

[12] *Linux Advanced Routing and Traffic Control.* http://lartc.org.

[13] *Mediacom: Hish-speed Internet (May 12, 2010).* http://www.mediacomcable.com/internet_online.html.

[14] *Network Diagnostic Tool (M-Lab).* http://www.measurementlab.net/measurement-lab-tools\#ndt.

[15] *Road Runner cable: central Texas (May 12, 2010).* http://www.timewarnercable.com/centraltx/learn/hso/roadrunner/speedpricing.html.

[16] *ShaperProbe (M-Lab).* http://www.measurementlab.net/measurement-lab-tools#tool5.

[17] "WRAPI: API for Real-time Monitoring and Control of an 802.11 Wireless LAN." http://sysnet.ucsd.edu/pawn/wrapi, 2002.

[18] "AirMagnet WiFi Analyzer." http://www.airmagnet.com, 2010.

[19] "Aruba Networks: RFProtect Spectrum Analyzer." http://www.arubanetworks.com, 2010.

[20] "MaxMind GeoLite City Database (Free)." http://dev.maxmind.com/geoip/geolite, 2011.

[21] Aggarwal, B., Bhagwan, R., Das, T., Eswaran, S., Padmanabhan, V., and Voelker, G., "NetPrints: Diagnosing home network misconfigurations using shared knowledge," in *USENIX NSDI*, 2009.

[22] Ahmed, N., Ismail, U., Keshav, S., and Papagiannaki, K., "Online estimation of RF interference," in *ACM CoNEXT*, 2008.

[23] Augustin, B., Friedman, T., and Teixeira, R., "Measuring Multipath Routing in the Internet," *IEEE/ACM ToN*, vol. 19, pp. 830–840, June 2011.

[24] Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., and Teixeira, R., "Avoiding traceroute anomalies with Paris Traceroute," in *ACM SIGCOMM IMC*, 2006.

[25] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D., and Zhang, M., "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *ACM SIGCOMM CCR*, vol. 37, pp. 13–24, 2007.

[26] Barford, P., Kline, J., Plonka, D., and Ron, A., "A signal analysis of network traffic anomalies," in *ACM SIGCOMM IMW*, 2002.

[27] Bauer, S., Clark, D., and Lehr, W., "PowerBoost," in *ACM SIGCOMM HomeNets workshop*, 2011.

[28] Bennett, J. C. R., Partridge, C., and Shectman, N., "Packet reordering is not pathological network behavior," *IEEE/ACM ToN*, vol. 7, pp. 789–798, Dec. 1999.

[29] Biczók, G., Young, W., and Kuzmanovic, A., "Monitoring Network Bias," in *ACM SIGCOMM 2008 (poster)*.

[30] Bloomfield, P., *Fourier Analysis of Time Series: An Introduction*. Wiley-Interscience, 2nd ed., 2000.

[31] Bonfiglio, D., Mellia, M., Meo, M., Rossi, D., and Tofanelli, P., "Revealing Skype traffic: when randomness plays with you," in *ACM SIGCOMM*, 2007.

[32] Broido, A., Nemeth, E., and kc claffy, "Spectroscopy of private DNS update sources," in *IEEE Workshop on Internet Applications*, 2003.

[33] Cai, K., Blackstock, M., Feeley, M., and Krasic, C., "Non-intrusive, dynamic interference detection for 802.11 networks," in *ACM SIGCOMM IMC*, 2009.

[34] Carlson, A. B., Crilly, P., and Rutledge, J., *Communication Systems (4th edition)*. McGraw-Hill, 2004.

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

[36] Chandra, R., Padmanabhan, V., and Zhang, M., "WiFiProfiler: Cooperative diagnosis in wireless LANs," in *ACM Mobisys*, 2006.

[37] Chen, A., Cao, J., and Bu, T., "Network tomography: Identifiability and Fourier domain estimation," in *IEEE INFOCOM*, 2007.

[38] Cheng, C.-M., Kung, H., and Tan, K.-S., "Use of spectral analysis in defense against DoS attacks," in *IEEE GLOBECOM*, 2002.

[39] Cheng, Y., Afanasyev, M., Verkaik, P., Benko, P., Chiang, J., Snoeren, A., Savage, S., and Voelker, G., "Automating cross-layer diagnosis of enterprise wireless networks," *ACM SIGCOMM CCR*, vol. 37, no. 4, pp. 25–36, 2007.

[40] Cheng, Y., Bellardo, J., Benko, P., Snoeren, A., Voelker, G., and Savage, S., "Jigsaw: Solving the puzzle of enterprise 802.11 analysis," *ACM SIGCOMM CCR*, vol. 36, no. 4, pp. 39–50, 2006.

[41] Claypool, M., Kinicki, R., Li, M., Nichols, J., and Wu, H., "Inferring queue sizes in access networks by active measurement," *PAM*, 2004.

[42] Cousins, D., Partridge, C., Bongiovanni, K., Jackson, A., Krishnan, R., Saxena, T., and Strayer, W., "Understanding Encrypted Networks Through Signal and Systems Analysis of Traffic Timing," in *IEEE Aerospace Conference*, 2003.

[43] Csabai, I., "1/f Noise in Computer Network Traffic," *Journal of Physics A*, vol. A27, no. L417-421, 1994.

[44] Cui, W., Machiraju, S., Katz, R. H., and Stoica, I., "SCONE: A tool to estimate shared congestion among Internet paths," *UC Berkeley Technical Report UCB/CSD-04-1320*.

[45] Dischinger, M., Haeberlen, A., Gummadi, K., and Saroiu, S., "Characterizing residential broadband networks," in *ACM SIGCOMM IMC*, 2007.

[46] Dischinger, M., Marcon, M., Guha, S., Gummadi, K., Mahajan, R., and Saroiu, S., "Glasnost: Enabling End Users to Detect Traffic Differentiation," in *USENIX NSDI*, 2010.

[47] Dischinger, M., Mislove, A., Haeberlen, A., and Gummadi, K., "Detecting Bittorrent blocking," in *ACM SIGCOMM IMC*, 2008.

[48] Dovrolis, C., Ramanathan, P., and Moore, D., "Packet-dispersion techniques and a capacity-estimation methodology," *IEEE/ACM ToN*, vol. 12, no. 6, pp. 963–977, 2004.

[49] Feather, F., Siewiorek, D., and Maxion, R., "Fault detection in an Ethernet network using anomaly signature matching," in *ACM SIGCOMM CCR*, 1993.

[50] Giustiniano, D., Malone, D., Leith, D., and Papagiannaki, K., "Measuring transmission opportunities in 802.11 links," *IEEE/ACM ToN*, no. 99, p. 1, 2010.

[51] Hanemann, A., Boote, J., Boyd, E., Durand, J., Kudarimoti, L., Łapacz, R., Swany, D., Trocha, S., and Zurawski, J., "PerfSONAR: A service oriented architecture for multi-domain network monitoring," *Service-Oriented Computing*, 2005.

[52] He, X., Papadopoulos, C., Heidemann, J., and Hussain, A., "Spectral characteristics of saturated links," Tech. Rep. USC/CS-TR-2004-827, USC, June 2004.

[53] He, X., Papadopoulos, C., Heidemann, J., Mitra, U., Riaz, U., and Hussain, A., "Spectral analysis of bottleneck traffic," Tech. Rep. USC/CS-TR-2005-853, USC, June 2005.

[54] Hollander, M. and Wolfe, D., "Nonparametric Statistical Methods," 1999.

[55] Honan, M., *Inside Net Neutrality: Is your ISP filtering content? (2008).* `http://www.macworld.com/article/132075/2008/02/netneutrality1.html`.

[56] Huang, P., Feldmann, A., and Willinger, W., "A non-intrusive, wavelet-based approach to detecting network performance problems," in *ACM SIGCOMM IMW*, 2001.

[57] Huang, Y., Feamster, N., Lakhina, A., and Xu, J., "Diagnosing network disruptions with network-wide analysis," in *ACM SIGMETRICS PER*, vol. 35, pp. 61–72, 2007.

[58] Hussain, A., Heidemann, J., and Papadopoulos, C., "A framework for classifying denial of service attacks," in *ACM SIGCOMM*, 2003.

[59] Jayasumana, A., Piratla, N., Banka, T., Bare, A., and Whitner, R., "Improved Packet Reordering Metrics (RFC 5236)," June 2008.

[60] Kaminsky, D., "Black Ops of TCP/IP 2011," in *Black Hat*, 2011.

[61] Kandula, S., Mahajan, R., Verkaik, P., Agarwal, S., Padhye, J., and Bahl, P., "Detailed diagnosis in enterprise networks," in *ACM SIGCOMM CCR*, vol. 39, pp. 243–254, 2009.

[62] Kanuparthy, P. and Dovrolis, C., "DiffProbe: Detecting ISP Service Discrimination," in *IEEE INFOCOM*, 2010.

[63] Kanuparthy, P. and Dovrolis, C., "End-to-end Detection of ISP Traffic Shaping using Active and Passive Methods," tech. rep., Georgia Tech. `http://www.netinfer.net/shaperprobe-TR.pdf`, 2011.

[64] Kanuparthy, P. and Dovrolis, C., "ShaperProbe: end-to-end detection of ISP traffic shaping using active methods," in *ACM SIGCOMM IMC*, 2011.

[65] Kanuparthy, P., Dovrolis, C., and Ammar, M., "Spectral probing, crosstalk and frequency multiplexing in Internet paths," in *ACM SIGCOMM IMC*, 2008.

[66] Kanuparthy, P., Dovrolis, C., Papagiannaki, K., Seshan, S., and Steenkiste, P., "Can user-level probing detect and diagnose common home-WLAN pathologies?," *ACM SIGCOMM CCR*, vol. 42, no. 1, pp. 7–15, 2012.

[67] Kanuparthy, P. and Dovrolis, C., "ShaperProbe: end-to-end detection of ISP traffic shaping using active methods," in *ACM SIGCOMM IMC*, 2011.

[68] Kanuparthy, P., Matthews, W., and Dovrolis, C., "DNS-based Ingress Load Balancing: An Experimental Evaluation," *CoRR*, vol. abs/1205.0820, 2012.

[69] Kapoor, R., Chen, L., Lao, L., Gerla, M., and Sanadidi, M., "CapProbe: A simple and accurate capacity estimation technique," in *ACM SIGCOMM*, 2004.

[70] Karagiannis, T., Papagiannaki, K., and Faloutsos, M., "BLINC: multilevel traffic classification in the dark," in *ACM SIGCOMM*, 2005.

[71] Kashyap, A., Ganguly, S., and Das, S., "A measurement-based approach to modeling link capacity in 802.11-based wireless networks," in *ACM MOBICOM*, 2007.

[72] Katabi, D. and Blake, C., "Inferring congestion sharing and path characteristics from packet interarrival times," Tech. Rep. MIT-LCS-TR-828, MIT, 2002.

[73] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K., "Internet traffic classification demystified: myths, caveats, and the best practices," in *ACM CoNEXT*, 2008.

[74] Kim, M. S., Kim, T., Shin, Y. J., Lam, S. S., and Powers, E. J., "Scalable clustering of Internet paths by shared congestion," *IEEE INFOCOM*, 2006.

[75] Kim, M. S., Kim, T., Shin, Y., Lam, S. S., and Powers, E. J., "A wavelet-based approach to detect shared congestion," in *ACM SIGCOMM*, 2004.

[76] Kreibich, C., Weaver, N., Nechaev, B., and Paxson, V., "Netalyzr: illuminating the edge network," in *ACM SIGCOMM IMC*, 2010.

[77] Kuzmanovic, A. and Knightly, E., "Measuring service in multi-class networks," in *IEEE INFOCOM*, 2001.

[78] Lakhina, A., Crovella, M., and Diot, C., "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM CCR*, vol. 34, pp. 219–230, 2004.

[79] Lakhina, A., Crovella, M., and Diot, C., "Mining anomalies using traffic feature distributions," in *ACM SIGCOMM CCR*, vol. 35, pp. 217–228, 2005.

[80] Lakshminarayanan, K., Padmanabhan, V. N., and Padhye, J., "Bandwidth estimation in broadband access networks," in *ACM SIGCOMM IMC*, 2004.

[81] Lakshminarayanan, K. and Padmanabhan, V., "Some findings on the network performance of broadband hosts," in *ACM SIGCOMM IMC*, 2003.

[82] Lakshminarayanan, K., Sapra, S., Seshan, S., and Steenkiste, P., "RFDump: An architecture for monitoring the wireless ether," in *ACM CoNEXT*, 2009.

[83] Lee, J., Lee, S., Kim, W., Jo, D., Kwon, T., and Choi, Y., "RSS-based carrier sensing and interference estimation in 802.11 wireless networks," in *IEEE SECON*, 2007.

[84] Li, S. and Pruneski, J., "The Linearity of Low Frequency Traffic Flow: An Intrinsic I/O Property in Queueing Systems," *IEEE/ACM TON*, vol. 5, pp. 429–443, June 1997.

[85] Lu, G., Chen, Y., Birrer, S., Bustamante, F., Cheung, C., and Li, X., "End-to-end inference of router packet forwarding priority," in *IEEE INFOCOM*, 2007.

[86] Luo, X. and Chang, R., "Novel approaches to end-to-end packet reordering measurement," in *ACM SIGCOMM IMC*, 2005.

[87] Mahajan, R., Rodrig, M., Wetherall, D., and Zahorjan, J., "Analyzing the MAC-level behavior of wireless networks in the wild," *ACM SIGCOMM CCR*, vol. 36, no. 4, pp. 75–86, 2006.

[88] Mahajan, R., Spring, N., Wetherall, D., and Anderson, T., "User-level Internet path diagnosis," in *ACM SIGOPS OSR*, vol. 37, pp. 106–119, 2003.

[89] Mahajan, R., Zhang, M., Poole, L., and Pai, V., "Uncovering performance differences among backbone ISPs with Netdiff," in *USENIX NSDI 2008*.

[90] Mahimkar, A., Yates, J., Zhang, Y., Shaikh, A., Wang, J., Ge, Z., and Ee, C., "Troubleshooting chronic conditions in large IP networks," in *ACM CoNEXT*, 2008.

[91] Mahimkar, A., Ge, Z., Shaikh, A., Wang, J., Yates, J., Zhang, Y., and Zhao, Q., "Towards automated performance diagnosis in a large IPTV network," in *ACM SIGCOMM CCR*, vol. 39, pp. 231–242, 2009.

[92] Mahimkar, A., Song, H., Ge, Z., Shaikh, A., Wang, J., Yates, J., Zhang, Y., and Emmons, J., "Detecting the performance impact of upgrades in large operational networks," in *ACM SIGCOMM CCR*, vol. 40, pp. 303–314, 2010.

[93] Mathis, M., Heffner, J., O'Neil, P., and Siemsen, P., "Pathdiag: automated TCP diagnosis," *PAM*, 2008.

[94] Moon, S. B., Skelly, P., and Towsley, D., "Estimation and removal of clock skew from network delay measurements," in *IEEE INFOCOM*, 1999.

[95] Niculescu, D., "Interference map for 802.11 networks," in *ACM SIGCOMM IMC*, 2007.

[96] Oppenheim, A. V. and Schafer, R. W., *Discrete-Time Signal Processing*. Prentice Hall, 1989.

[97] Padhye, J., Agarwal, S., Padmanabhan, V., Qiu, L., Rao, A., and Zill, B., "Estimation of link interference in static multi-hop wireless networks," in *ACM SIGCOMM IMC*, 2005.

[98] Partridge, C. and Cousins, D., "Systems and methods for creating covert channels using packet frequencies." United States Patent Application 20030091064, May 2003.

[99] Poese, I., Uhlig, S., Kaafar, M., Donnet, B., and Gueye, B., "IP geolocation databases: Unreliable?," *ACM SIGCOMM CCR*, vol. 41, no. 2, pp. 53–56, 2011.

[100] Portoles-Comeras, M., Cabellos-Aparicio, A., Mangues-Bafalluy, J., Banchs, A., and Domingo-Pascual, J., "Impact of transient CSMA/CA access delays on active bandwidth measurements," in *ACM SIGCOMM IMC*, 2009.

[101] Qiu, L., Zhang, Y., Wang, F., Han, M., and Mahajan, R., "A general model of wireless interference," in *ACM MOBICOM*, 2007.

[102] Qiu, T., Ge, Z., Pei, D., Wang, J., and Xu, J., "What happened in my network: mining network events from router syslogs," in *ACM SIGCOMM IMC*, 2010.

[103] Rabbat, M., Coates, M., and Nowak, R. D., "Multiple-source Internet tomography," *IEEE JSAC*, vol. 24, no. 12, pp. 2221–2234, 2006.

[104] Rayanchu, S., Patro, A., and Banerjee, S., "Airshark: detecting non-WiFi RF devices using commodity WiFi hardware," in *ACM SIGCOMM IMC*, 2011.

[105] Reis, C., Mahajan, R., Rodrig, M., Wetherall, D., and Zahorjan, J., "Measurement-based models of delivery and interference in static wireless networks," *ACM SIGCOMM*, 2006.

[106] R.S.Tsay, "Outliers, Level Shifts, and Variance Changes in Time Series," *Journal of Forecasting*, 1988.

[107] Rubenstein, D., Kurose, J., and Towsley, D., "Detecting shared congestion of flows via end-to-end measurement," *IEEE/ACM TON*, vol. 10, no. 3, pp. 381–395, 2002.

[108] Shalunov, S. and Carlson, R., "Detecting duplex mismatch on Ethernet," *PAM*, 2005.

[109] Sheskin, D., *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.

[110] Sheth, A., Doerr, C., Grunwald, D., Han, R., and Sicker, D., "MOJO: A distributed physical layer anomaly detection system for 802.11 WLANs," in *ACM Mobisys*, 2006.

[111] Silverman, B., *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability, Taylor & Francis, 1986.

[112] Sundaresan, S., de Donato, W., Feamster, N., Teixeira, R., Crawford, S., and Pescapè, A., "Broadband Internet performance: a view from the gateway," in *ACM SIGCOMM*, 2011.

[113] Tariq, M., Dhamdhere, A., Dovrolis, C., and Ammar, M., "Poisson versus periodic path probing (or, does PASTA matter?)," in *ACM SIGCOMM IMC*, 2005.

[114] Tariq, M., Motiwala, M., Feamster, N., and Ammar, M., "Detecting network neutrality violations with causal inference," in *ACM CoNEXT*, 2009.

[115] Turner, D., Levchenko, K., Snoeren, A., and Savage, S., "California fault lines: understanding the causes and impact of network failures," in *ACM SIGCOMM CCR*, vol. 40, pp. 315–326, 2010.

[116] Varghese, G., *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005.

[117] Vutukuru, M., Jamieson, K., and Balakrishnan, H., "Harnessing exposed terminals in wireless networks," in *USENIX NSDI*, 2008.

[118] Wang, T., Srivatsa, M., Agrawal, D., and Liu, L., "Learning, indexing, and diagnosing network faults," in *ACM SIGKDD KDD*, 2009.

[119] Wang, T., Srivatsa, M., Agrawal, D., and Liu, L., "Spatio-temporal patterns in network events," in *ACM CoNEXT*, 2010.

[120] Weinsberg, U., Soule, A., and Massoulie, L., "Inferring traffic shaping and policy parameters using end host measurements," in *IEEE INFOCOM Mini-conference*, 2011.

[121] Wright, C., Coull, S., and Monrose, F., "Traffic morphing: An efficient defense against statistical traffic analysis," in *IEEE NDSS*, 2009.

[122] Yan, H., Breslau, L., Ge, Z., Massey, D., Pei, D., and Yates, J., "G-RCA: a generic root cause analysis platform for service quality management in large IP networks," in *ACM CoNEXT*, 2010.

[123] Zarifzadeh, S., Madhwaraj, G., and Dovrolis, C., "Range tomography: Combining the practicality of boolean tomography with the resolution of analogue tomography," in *ACM SIGCOMM IMC*, 2012.

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

[125] Zhang, Y., Duffield, N., Paxson, V., and Shenker, S., "On the Constancy of Internet Path Properties," in *ACM SIGCOMM IMW*, 2001.

[126] Zhang, Y., Oliveira, R., Zhang, H., and Zhang, L., "Quantifying the Pitfalls of Traceroute in AS Connectivity Inference," in *PAM*, 2010.

[127] Zhang, Y., Mao, Z. M., and Zhang, M., "Detecting traffic differentiation in backbone ISPs with NetPolice," in *ACM SIGCOMM IMC*, 2009.

# VITA

Partha Kanuparthy enjoys building things. His research interests include Internet measurements and diagnosis, and large-scale distributed systems. He joined the Georgia Institute of Technology in Fall 2006 and worked on his doctoral dissertation under the able guidance of Dr. Constantine Dovrolis. He won a Best Research Award from the Federal Communications Commission for his work on tools to enable Internet transparency. He is an author of the ShaperProbe and Pathload2 tools. He has worked as a research intern at Intel Labs and at Microsoft Research. He received an undergraduate degree in Mechanical Engineering from the Indian Institute of Technology Kharagpur.