Making Broadband Access Networks Transparent to Researchers, Developers, and Users

Marcel Dischinger

Dissertation

zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Eingereicht Saarbrücken, 10. Juni 2010

Dekan:	Prof. Dr. Holger Hermanns
Vorsitzender des Promotionsausschusses:	Prof. Dr. Holger Hermanns
Erstgutachter:	Prof. Dr. Peter Druschel
Zweitgutachter:	Krishna P. Gummadi, PhD
Drittgutachter:	Stefan Saroiu, PhD
Tag des Kolloquiums:	27. Oktober 2010

Abstract

Broadband networks are used by hundreds of millions of users to connect to the Internet today. However, most ISPs are hesitant to reveal details about their network deployments, and as a result the characteristics of broadband networks are often not known to users, developers, and researchers. In this thesis, we make progress towards mitigating this lack of transparency in broadband access networks in two ways.

First, using novel measurement tools we performed the first large-scale study of the characteristics of broadband networks. We found that broadband networks have very different characteristics than academic networks. We also developed Glasnost, a system that enables users to test their Internet access links for traffic differentiation. Glasnost has been used by more than 350,000 users worldwide and allowed us to study ISPs' traffic management practices. We found that ISPs increasingly throttle or even block traffic from popular applications such as BitTorrent.

Second, we developed two new approaches to enable realistic evaluation of networked systems in broadband networks. We developed Monarch, a tool that enables researchers to study and compare the performance of new and existing transport protocols at large scale in broadband environments. Furthermore, we designed SatelliteLab, a novel testbed that can easily add arbitrary end nodes, including broadband nodes and even smartphones, to existing testbeds like PlanetLab.

Kurzdarstellung

Breitbandanschlüsse werden heute von hunderten Millionen Nutzern als Internetzugang verwendet. Jedoch geben die meisten ISPs nur ungern über Details ihrer Netze Auskunft und infolgedessen sind Nutzern, Anwendungsentwicklern und Forschern oft deren Eigenheiten nicht bekannt. Ziel dieser Dissertation ist es daher Breitbandnetze transparenter zu machen.

Mit Hilfe neuartiger Messwerkzeuge konnte ich die erste groß angelegte Studie über die Besonderheiten von Breitbandnetzen durchführen. Dabei stellte sich heraus, dass Breitbandnetze und Forschungsnetze sehr unterschiedlich sind. Mit Glasnost habe ich ein System entwickelt, das mehr als 350.000 Nutzern weltweit ermöglichte ihren Internetanschluss auf den Einsatz von Verkehrsmanagement zu testen. Ich konnte dabei zeigen, dass ISPs zunehmend BitTorrent Verkehr drosseln oder gar blockieren.

Meine Studien zeigten darüberhinaus, dass existierende Verfahren zum Testen von Internetsystemen nicht die typischen Eigenschaften von Breitbandnetzen berücksichtigen. Ich ging dieses Problem auf zwei Arten an: Zum einen entwickelte ich Monarch, ein Werkzeug mit dem das Verhalten von Transport-Protokollen über eine große Anzahl von Breitbandanschlüssen untersucht und verglichen werden kann. Zum anderen habe ich SatelliteLab entworfen, eine neuartige Testumgebung, die, anders als zuvor, beliebige Internetknoten, einschließlich Breitbandknoten und sogar Handys, in bestehende Testumgebungen wie PlanetLab einbinden kann.

Acknowledgements & Danksagungen

First and foremost, I am grateful to my advisor Krishna P. Gummadi for his help, advice, and mentoring. I am deeply indebted to him for showing me how to pick interesting and challenging research topics, how to do successful research, and how to communicate research results effectively.

I would like to thank Michael Backes, Peter Druschel, Rodrigo Rodrigues, and Stefan Saroiu for their service on my thesis committee. I would also like to thank Rose Hoberman for her valuable feedback on this thesis and her patience to explain the English language to me.

I am grateful for the close collaboration with Andreas Haeberlen. I will always remember the countless nights we worked together to get experiments done or to polish a paper for submission. The work I am presenting in this thesis would not have been possible without my other collaborators: Ivan Beschastnikh, Saikat Guha, Ratul Mahajan, Massimiliano Marcon, Alan Mislove, and Stefan Saroiu.

Last but not least thanks to all my friends and colleagues at MPI-SWS, who created a challenging and inspiring working environment and who made my time at MPI-SWS a great experience.

Dank auch meinem besten Freund Tobias für seine Unterstützung und speziell für seine hilfreichen Kommentare zu Teilen dieser Dissertation.

Ich danke Ute für ihre Unterstützung, Geduld und Liebe, und ganz besonders für ihr Verständnis, dass meine Forschungen manchmal wichtiger waren als unsere gemeinsame Zeit. Danke, dass du für mich da bist.

Ganz besonders möchte ich meiner Familie danken, neben meinem Bruder Felix besonders meinen Eltern, die mich unablässig unterstützt, mir über so manches Tief hinweggeholfen und immer an mich geglaubt haben.

Contents

List of Figures

List of Tables

1 Introduction

The degree of heterogeneity of today's Internet is much higher than before. While it started in the 1960s as a network of academic and military computers with a few hundred users in the USA, it has since grown to a global network interconnecting nodes from academic institutions, governmental agencies, corporations, content providers, and private individuals. Today, the Internet is an important part of the daily life of hundreds of millions of people worldwide and is composed of a mix of different networks. These networks have highly heterogeneous bandwidth speeds ranging from high-bandwidth research networks to lower-bandwidth cellular networks. Their levels of oversubscription are also very different. While corporate networks often have low levels of oversubscription or even dedicated network connections for business offices, residential networks often sign up hundreds of customers over a single link connected to the residential ISP.

The network applications running on the Internet today also have very different characteristics. The number of different workloads the Internet supports is indeed impressive. Besides the traditional content served via webpages in the World Wide Web (WWW), the Internet is also used to distribute software, to share files among users, to play online multiplayer games, to download music and videos, to stream audio or video live or on-demand (e.g., web radio or Internet TV), it is used for video chats, and to place phone calls (e.g., voice-over-IP). Increasingly even TV and landline phone calls are transmitted over the Internet. These workloads can impose rather distinct requirements on the network they run on. While some applications, such as online multiplayer games or Internet telephony, are latency-sensitive, others, such as large media or software downloads, are less latency-sensitive, but instead have a high bandwidth demand and thus require networks that provide high bandwidth links.

The network types that comprise the Internet can have very different characteristics. The differences may lie in low-level flow characteristics, such as bandwidth, latencies, and loss rates. However, networks' characteristics could also vary due to higher-level policies for traffic management, such as traffic shaping.

Network characterization is very important to application and system designers because network properties often influence the performance of Internet systems. Network studies often guide designers to help them make their systems more suitable to the characteristics of the underlying network. For example, the asymmetric nature of network speeds in residential broadband networks have made P2P system designers adopt "file swarming", a file transfer technique in which one peer downloads from multiple other peers simultaneously. Developers can make incorrect assumptions about the networks' characteristics that can lead to systems that function well when run in a testbed environment, but fail to work properly when deployed on the Internet. For example, a recent scheme for providing network coordinates was found to have much higher error rates when deployed on the Internet than when run on PlanetLab [LGS07]. As our studies show, these rates likely stemmed from an incorrect assumption about how latencies vary on residential access networks. Consequently, system evaluations should be performed over the network a system

is supposed to run on. Simulations must use an accurate model of the network — which in return requires to study the network's characteristics in depth. And testbeds should offer a similar networking environment than what is expected in a real deployment scenario.

There is a large body of literature that studied the characteristics of different network types in the Internet. However, most of these studies focused on academic network or the Internet backbone [Bol93, Pax97, Pax99, GP02, ICM+02], despite the widespread deployment of residential broadband networks and their importance to emerging applications. Broadband networks remain relatively unexplored by the academic community with only a few small-scale studies [LP03, CKL+04], and evaluation of applications in broadband networks relies mostly on simplistic models in simulations [SKMM05, ns2]. But at the same time, broadband networks are known to have very different characteristics from academic networks [BGP04, PHM06].

The main reason for this situation is that most academic institutions and research laboratories do not access the Internet over broadband. This lack of access makes it hard to study broadband networks in depth and at scale. And even state-of-the-art Internet testbeds, such as PlanetLab [Pla] and RON [ABKM03], which are often used for measurement studies and for running realistic evaluation experiments, are comprised of mostly academic nodes and have only a handful of broadband nodes. For example, as of May 2010 PlanetLab has only 5 nodes that connect to the Internet over broadband out of a total of 1088 testbed nodes.

1.1 Broadband Internet Access

Residential broadband networks such as Digital Subscriber Line (DSL) and cable are increasingly becoming popular. In 2009, more than 271 million people used these networks worldwide [OEC09], and this number is expected to rise to 636 million by 2014 [Win09]. In Germany alone, more than 66% of all Internet users connect to the Internet via residential broadband networks [Ini09]. The percentage are similar in other western countries like the USA [Nie04]. Many governments are adopting policies to promote ubiquitous broadband access [Fed09, Uni01, Fed06]. In these policies broadband access is often defined as an Internet connection for home users that offers at least 1 Mbps bandwidth (compared to dial-up connections with only a few tens of Kbps bandwidth). These policies may result in even more people using broadband networks to access the Internet in the future.

Residential broadband networks provide the critical "last mile" access to the Internet infrastructure. It is widely thought that the bottlenecks in the performance of the Internet lie in its access networks [ASS03]. Thus, the reliability and the performance of Internet applications — including voice-over-IP (VoIP), video on demand (VoD), online games, and peer-to-peer (P2P) content delivery systems — depend crucially on the characteristics of broadband access networks.

Today, there are mostly two broadband access network technologies that dominate the way people access the Internet: cable and DSL. In the remainder of this section, we present a brief description of their architectures and point out differences from other access networks, such as corporate and academic networks. Finally, we discuss other existing and emerging broadband technologies.

1.1.1 Cable

Cable networks use the cable television infrastructure to connect home users to the Internet. In these networks, a master headend connects to several regional headends using fiber-optic cables. Each regional headend serves a set of customers (up to 2,000 homes). A single coaxial cable, carrying both television and data signals, links these customers to their headend.

DOCSIS [Cab06] is the most common specification defining the interface requirements of cable modems. In DOCSIS, each cable modem (CM) exchanges data with a cable modem termination system (CMTS) located in a regional headend. In the downstream direction, the CMTS broadcasts data to all cable modems that are connected to it. The cable modems filter all received data and forward only the bytes destined for their customer's host. In the upstream direction, the access channel is time-slotted — a cable modem must first reserve a time slot and wait until the CMTS grants the reservation. When the time slot has been granted, the cable modem can transmit data upstream. Figure 1.1 illustrates a typical setup of a cable access network.

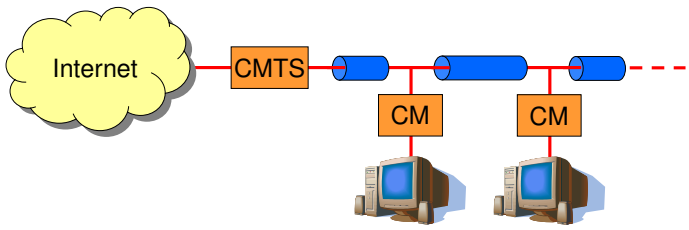


Figure 1.1: A typical setup of a cable access network. On the user side, a cable modem (CM) connects the user to the cable modem termination system (CMTS).

There are several important differences between cable and other access networks. First, cable links typically have asymmetric bandwidths: their downstream bandwidth is much higher than their upstream bandwidth. Second, customers cannot use the full raw capacity of their cable links. Instead, cable operators shape users' traffic preventing them from consuming more bandwidth than their contract stipulates. Although cable networks currently allow raw data rates of up to 40 Mbps, the contracts of individual customers specify much lower rates, typically between 128 Kbps and 20 Mbps. Further, some ISPs over-subscribe their cable access networks. In this case, the level of service experienced by customers can vary depending on the amount of competing network traffic.

Finally, cable modems can concatenate multiple upstream packets into a single transmission, which results in short bursts at high data rates. Thus, the upstream latencies can fluctuate heavily, depending on the allocation policy, and the amount of signaling and packet concatenation used by the CMTS.

1.1.2 DSL

DSL access networks use existing telephone wiring to connect home users to the Internet [Cis03]. Unlike cable customers, DSL customers do not share their access link. Each customer's DSL modem uses a dedicated point-to-point connection to exchange data with

a Digital Subscriber Line Access Multiplexer (DSLAM). The connection carries both data and telephone signals, which are encoded in different frequencies. On the customer side, a splitter separates the two signals and forwards the data signal to the DSL modem. Figure 1.2 illustrates a typical setup of a DSL access network.

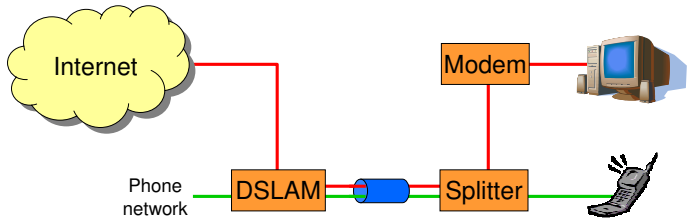


Figure 1.2: A typical setup of a DSL access network. Telephone and data signals are multiplexed on the same line by the DSLAM. The signal is demultiplexed again by the splitter in a user's home and the data signal is forwarded to the DSL modem.

Today, typical DSL deployments offer bandwidth rates between 128 Kbps and 16 Mbps. With VDSL (Very high bitrate DSL), customers can get speeds of up to 100 Mbps.

There are two important differences between DSL networks and other access networks. First, like cable networks, DSL networks often have asymmetric bandwidths; their downstream bandwidth is higher than their upstream bandwidth. Second, the maximum data transmission rate falls with increasing distance from the DSLAM. To boost the data rates, DSL relies on advanced signal processing and error correction algorithms, which can lead to high packet propagation delays. Consequently, the properties of DSL access links vary depending on the length of the wiring or the quality of the wiring between a modem and its DSLAM — the longer the distance between the modem and the DSLAM, the lower the signal quality. Decreased signal quality requires more extensive signal error correction, ultimately leading to higher latencies and also lower available bandwidth. Hence, the bandwidths, packet latencies, and loss rates can vary from link to link.

1.1.3 Other Broadband Access Technologies

DSL and cable technologies are the dominant broadband technologies used today by the majority of home users. And while our focus is on DSL and cable access networks, there are other existing and emerging broadband technologies to connect people with the Internet at high speeds that make the Internet even more heterogeneous. We now list other important broadband technologies for the sake of completeness.

Fiber-To-The-Home (FTTH) replaces the traditional copper wires of the last mile in DSL with optical fiber cables. This allows for much higher speeds, typically ranging from several tens of Mbps to 1 Gbps. Deployments of this technology are often limited to densely-populated areas as they require laying new cables out which is rather expensive.

WiMax (Worldwide Interoperability for Microwave Access) [The09c] is a wireless alternative to DSL and cable on the last mile. It provides similar speeds to DSL and is deployed mostly in regions (such as rural areas) with no access to broadband access networks like DSL and cable.

Satellite is also often used in areas where no other broadband access networks are available. It allows high speed Internet connections with typically a few Mbps downstream and less than one Mbps upstream speeds per user. Furthermore, satellite connections usually suffer from high signal latencies as the network packets have to be sent to orbiting satellites and back.

Mobile broadband technologies, such as EDGE, EVDO, or UMTS, play an increasing role in the way people connect to the Internet. These technologies are also often referred to as third generation (3G) wireless communications technologies [ITU]. Mobile broadband technologies are cell-based, with users in one cell sharing the bandwidth of the underlying medium with each other.

Mobile technologies can suffer from radio interference, a property specific to wireless networks. The level of radio interference can become very high when many users share the wireless medium. For example, mobile networks often suffer from low per-user bandwidths in densely populated areas such as urban cities. As a result, many Internet Service Providers (ISPs) heavily manage user traffic, e.g., by introducing performance enhancing proxies that can significantly lower the bandwidth needs of web applications, or by introducing traffic shaping and blocking of bandwidth-intensive applications such as filesharing applications. While EDGE is rather low-speed with theoretical speeds up to 1.3 Mbps in the downlink and 0.653 Mbps in the uplink, 3G-networks, such as EVDO [CDM] and UMTS [3GP08a], support up to 7 Mbps per user. Future mobile broadband technologies like Long Term Evolution (LTE) [3GP08b] promise speeds of up to 50 Mbps per user.

1.2 Network Transparency

Network operators are usually hesitant to reveal details about their network deployments because they consider these details as business secrets. This has made most networks lack the transparency needed by Internet system designers. While it might make sense for ISPs to conceal certain details about their deployments that usually have no impact on application performance (e.g., the exact network topology), it is desirable to publish information about network details that might affect their customers (e.g., latencies or traffic management policies).

ISPs often establish service level agreements (SLAs) with their business customers. SLAs specify the quality of service an ISP has to deliver to a business customer, such as guarantees about their levels of bandwidth, latency, and network reliability. However, there are no such precise SLAs for residential customers. Instead, contracts between users and ISPs tend to be imprecise and cover only a very limited set of properties. For example, most broadband contracts only specify an upper bound on the available bandwidth, e.g., "up to 6 Mbps", while the actual bandwidth can be much lower. Further, these contracts usually do not mention other performance details, such as latencies, loss rates, or connection reliability. Also, traffic management policies are not described, although typically the ISP grants itself the right to deploy any policy they choose without further notice.

As a result, residential broadband networks remain remarkably opaque, despite the fact that millions of users access the Internet using broadband and many popular Internet applications run in this environment. While the protocols and network technologies are well-known, the network configuration and the traffic characteristics in broadband networks are not well studied. But knowing about these networks' characteristics is crucial to under-

stand and predict the behavior of networked systems in this environment, as broadband networks are known to be different from other, well-studied networks [PHM06, BGP04].

Transparent networks have many advantages over opaque networks. Transparency allows users to make an informed choice about their ISP. For example, in recent years broadband networks have been subject to traffic management practices that can significantly affect the performance of distributed applications popular with users, such as BitTorrent [Top07, Can08b]. However, as ISPs are hesitant to reveal their practices publicly, they leave their customers in the dark about the potential impact of these practices on their Internet experience. But broadband users appear to be very interested in their Internet access links, especially if applications do not perform as expected. For example, there are popular Internet forums [Bee99] where users post their experience with their ISPs and discuss problems they have with their Internet connections. With transparency, users can avoid signing up with ISPs that throttle their favorite applications, and they can diagnose what is causing unexpected application behavior.

Details about broadband deployments are also important for designing new systems or optimizing and adapting systems to make them perform well over these networks. A recent example of how transparency can influence system design beneficially is P4P [XYK+08], a project run jointly by an ISP, a BitTorrent software vendor, and researchers. The project seeks to keep BitTorrent traffic local in order to increase users' download speeds while reducing bandwidth costs for the ISP.

Finally, telecommunication regulators need transparency to monitor ISPs and hold them accountable. In today's opaque network deployments, regulators rely on information provided by ISPs themselves for their investigations [Com08b, Can08b]. Obviously, this does not allow regulators to monitor ISPs independently and enables ISPs to hide details about their network deployments from regulators. For example, it only came to the attention of the US telecommunication regulator (FCC) that Comcast blocked BitTorrent traffic after network measurements performed by a civil rights group revealed this practice [Top07, The08].

To improve the current situation with rather opaque network deployments, researchers have sought to make networks more transparent without the direct support of network operators. Using network measurements, network characteristics can be studied independently and simulation models can be derived for more realistic system evaluations. However, to date, there are only a few, small-scale studies that characterized residential broadband networks [LP03, CKL+04]. Evaluation experiments rely on either simulations, which use simplistic and thus often not accurate models, or testbeds that only include a handful of broadband nodes.

The goal of this thesis is to make broadband access networks more transparent to users, researchers, and developers. We achieved this by building novel tools that allow the study of broadband networks at scale for the first time. Furthermore, we developed tools and systems that allow researchers and developers to evaluate their new system designs in broadband networks at large scale.

1.3 Contributions

We make the following contributions towards increasing transparency in broadband access networks.

1. **The first large-scale study of the characteristics of broadband access networks of major ISPs in Europe and North America**. To perform this study, we developed a novel measurement methodology that requires only minimal cooperation from remote hosts to study important characteristics (such as bandwidth, latency, and loss rates) of a large number of access links. Our study reveals important differences between broadband and academic networks regarding bandwidth, latency, and loss rates. For example, we are the first to point out that many broadband hosts have surprisingly long queues deployed that can significantly impact the performance of latency-sensitive applications such as VoIP and VoD.

2. **The first large-scale study of the prevalence of traffic differentiation in broadband access networks.** Our Glasnost system is an easy-to-use tool that enables even lay users to test their broadband access links for traffic differentiation, such as blocking or throttling of application traffic. Using data collected by Glasnost, our study reveals that blocking of BitTorrent transfers was widely used until the end of 2008, and that ISPs today increasingly deploy throttling-based traffic management practices instead. Our findings were covered in the public media and even attracted the attention of telecommunication regulators.

3. **A novel methodology to study transport protocols at large scale over the Internet.** Building on our measurement methodology for studying network characteristics, we present Monarch, a tool that emulates transport protocol flows to a large number of hosts on the Internet without requiring direct access to them. Monarch enables researchers to study the behavior of transport protocols in the wild in heterogeneous settings.

4. **A novel testbed design that allows adding end user nodes easily to existing Internet testbeds.** While today's testbeds mostly consist of well-connected academic nodes, SatelliteLab allows creating heterogeneous testbeds, including broadband and mobile nodes that cannot be added to today's testbeds such as PlanetLab.

Some of the material in this thesis was previously published in a series of conference papers [HDGS06, DHGS07, DHB+08, DMHG08, DMG+10].

1.4 Structure of this Thesis

The rest of this thesis is divided into two parts. In Part I, we introduce novel methodologies for studying various aspects of residential broadband networks. Based on these methodologies, we performed large-scale studies of residential broadband networks.

In Chapter 2, we outline the challenges in studying broadband networks and discuss related work. In Chapter 3, we describe a novel methodology for studying broadband networks with only minimal cooperation from the measured broadband hosts. We used this methodology to perform the first large-scale study of the characteristics of residential broadband networks. Next, Chapter 4 presents Glasnost,which allowed hundreds of thousands of users to test their broadband links for traffic differentiation. The chapter outlines the design of Glasnost and uses the collected data to investigate the prevalence of traffic shaping in the Internet.

In Part II, we focus on tools that allow researchers and developers to evaluate their applications and systems in broadband networks.

In Chapter 5, we discuss the need for the evaluation of emerging systems and applications over the Internet. Discussing related work we point out that state-of-the-art network simulators and testbeds do not represent the heterogeneity of the Internet as they are mostly represent academic and high-bandwidth networks and do not consider nodes from broadband access networks. Chapter 6 presents Monarch, a tool to emulate transport protocol flows over the Internet at large. And Chapter 7 presents SatelliteLab, a new testbed design that allows constructing heterogeneous testbeds easily, particularly including broadband nodes.

Finally, in Chapter 8 we conclude and discuss possible directions for future work.

Part I

Characterizing Broadband Access Networks

2 Background and Related Work

More than 271 million people worldwide use broadband access networks to connect to the Internet. But despite these networks' widespread usage, they remain relatively unexplored by the academic community. Understanding the characteristics of broadband networks is important for three reasons. First, studying broadband access networks allows researchers to model the underlying network and thus create realistic simulation environments, which can be used to test new applications and systems before deploying them in the wild. Second, measurements allow researchers and users to detect network anomalies, which might affect the operation of the network or of the applications running over it. Previously, such measurements revealed flaws in widely used protocols [AEO03]. Third, measurements make network deployments more transparent. For instance, measurements have revealed recently that many major ISPs deploy equipment that restricts the bandwidth for popular Internet applications [DMHG08].

While there are a large number of measurement tools available, they are often not suitable to measure residential broadband networks as most of these tools require access to the broadband nodes. But researchers often have only limited access to broadband environments and, as a result, previous studies were restricted to a few tens of broadband nodes.

In this thesis, we present two approaches to study the characteristics of broadband networks at large scale. First, we developed a novel measurement tool that allows researchers to study properties of broadband networks, such as bandwidth, queue lengths, and loss rates, without the requirement to have access to many end hosts. Second, we present Glasnost, a system that allowed hundreds of thousands of end users to check their access links for traffic differentiation. Glasnost enabled us to study the prevalence of BitTorrent blocking and throttling in broadband networks.

In the next sections, we give an overview of existing measurement methodologies and discuss why most of these techniques are not suitable to study broadband access networks at large scale. We also present previous network measurement studies including small-scale studies of broadband access networks.

2.1 Measurement Methodologies and Tools

Measuring the Internet has been a long tradition in networking research. Since the early days of the Internet, researchers have been interested in the characteristics of the Internet. Traditionally, measurement studies have examined the bandwidths, latencies, and loss rates of Internet paths. Additionally, recent studies have investigated the presence and configurations of network policies found on Internet paths, such as traffic shaping and traffic prioritization policies.

Researchers have devised measurement tools that use either active probes or passive observation of ongoing network traffic to measure the Internet. To infer the network characteristics from the measurements, the collected measurement data has to be analyzed

carefully as confounding factors (such as noise, the used TCP congestion control algorithm, and the local configuration of protocol parameters) might have affected the data, potentially resulting in misleading interpretations of the data [Pax04]. While most of today's Internet measurement tools are used by researchers, there are easy-to-use tools that are built for end users to measure characteristics they are interested in. We will present some popular examples later in this section.

2.1.1 Active and Passive Measurement Techniques

Many measurement methodologies have been developed to study the characteristics of networks. They are typically divided into two classes: *active measurement techniques* and *passive measurement techniques*. In the following, we introduce each of these classes of techniques and present several examples.

Active measurement techniques

Active measurement techniques send probes among Internet hosts. These probes are used to infer the characteristics of network paths. They often require control over the Internet hosts used to send and receive probe traffic.

Pathload [JD03] estimates the bandwidth available to TCP flows by sending periodic streams of measurement traffic. Its inference is based on the insight that one-way delays of measurement probes increase if they are sent at a rate higher than the available bandwidth. Claypool et al. [CKL+04] presented a methodology for estimating queue sizes in broadband networks. ICMP ping probes are used to measure the latency to a nearby host. Then a large download is started that is supposed to fill up the queue at the bottleneck link, which is usually the broadband link. Once the queue is filled up, the latency to the nearby host is measured again. The increase in latency compared to the first measurement is used to estimate the queue size at the bottleneck link.

Some active measurement tools use packet-pair or packet-train techniques to estimate the network path capacity with only a small number of probes. Pathrate [DRM04] sends many packet pairs back-to-back and measures the packet dispersion at the destination host. This technique estimates the capacity of the traversed network path and works even in the presence of cross traffic. Probegap [LPP04] is able to measure a link's capacity even in the presence of links with multiple distinct rates, as, e.g., caused by traffic shaping equipment such as token buckets. Sending packet trains, it measures the one-way delay of the probes and detects gaps that correspond to idle periods, which corresponds to periods of full link capacity. The one-way delays measured during these idle periods are then used to estimate the path's capacity.

NetPolice [ZMZ09] (previously named NVLens [ZMZ08]) compares the aggregate loss rates of different flows to infer the presence of "network neutrality violations". Their focus is on inferring backbone ISPs' traffic differentiation policies. Probes that carry contents of application messages are sent between pairs of measurement points; to control which path segment is measured the TTL field of the IP header is adjusted.

Finally, the DIMES project [SS05] uses volunteer-contributed hosts to run "traceroute" measurements that map the connectivity of edge networks. DIMES requires end user hosts to run arbitrary code (similar to the popular SETI@home software [ACK+02]). It was deployed more than four years ago and has about 8,000 users.

Most of the active measurement tools presented so far require access to both end points to measure the path characteristics between them. Such a methodology restricts the scale of a measurement study and cannot be used for networks researchers typically lack access to, such as broadband networks. To overcome this limitation, researchers have come up with new methodologies that leverage existing protocols in unanticipated ways while requiring access to only one end point to perform measurements that were previously intractable.

Sting [Sav99] manipulates the TCP protocol to measure packet loss. In principle, Sting uses TCP's loss detection to derive the underlying loss rate. To get a significant statistical sample of probes, Sting manipulates TCP to send packets that overlap in payload, thus sending many more packets than a usual TCP implementation would send while still conforming to the TCP standard. Sting only requires the remote host to run a TCP-based service (e.g., a webserver) it can connect to. Using similar techniques, SProbe [SGG02] sends packet pairs of TCP SYN packets to measure bottleneck bandwidth to uncooperative Internet hosts. This approach is problematic today as the TCP SYN probes look like a port scan and might trigger intrusion detection alarms.

T-BIT [PF01, MAF05] exploits the TCP protocol to characterize Web servers' TCP behavior, such as which congestion control algorithm is used and whether the remote server supports selective acknowledgments. Its packet probes are indistinguishable from normal webpage requests and work with any web server.

King [GSG02] estimates the path latencies between two arbitrary end points on the Internet without having access to any of them. The measurement methodology is based on the observation that most end hosts are located close to their DNS server. King uses recursive DNS queries to measure latencies between two arbitrary DNS servers which can then be used to estimate the path latencies of two end points located close to these DNS servers. King assumes that the path characteristics between two (typically well-connected) DNS servers are similar to those between two arbitrary end hosts. However, this assumption does not hold for broadband nodes as we will show in Chapter 3.

TCP Sidecar [SS06] is a technique to map the topology of a network. It tries to overcome the shortcomings of the popular `traceroute` tool that is often blocked by firewalls and that is unable to map nodes behind Network Address Translation equipment (NAT). TCP Sidecar injects duplicates of TCP packet into passing TCP streams. The injected packets are TTL-limited and use the IP record route option to map the network. To measure a node, TCP Sidecar must be placed in its communication path. However, this is very challenging and requires the support of ISPs or content providers to measure a large number of nodes.

All of these tools are successful in measuring a large number of nodes on the Internet, but they also raise usage concerns — probes are sent to remote hosts which did not consent to the measurements. Such measurements can interfere with normal user traffic and might even impose costs on the measured users if a per-byte payment model is in place. For all these tools, precautions have to be adopted that mitigate these concerns, e.g., by minimizing the interference and other side-effects for users.

Passive measurement techniques

Instead of sending probes to measure a network, passive measurement techniques monitor and analyze the ongoing network traffic to infer characteristics and properties of the

network or the applications running on top of it. Passive measurement methodologies are less intrusive than active measurements: they use only real, pre-existing traffic for inference and do not generate any artificial traffic that might influence the network's behavior or raise security concerns. At the same time, they allow measuring hosts even without access to them as long as their traffic passes the passive measurement monitor. Although passive measurement techniques have many strengths, in practice they have three major limitations.

First, passive measurement techniques rely on pre-existing network traffic to infer network characteristics. Thus, the tools have to be placed in network locations that allow monitoring sufficient traffic that crossed the link of interest. This can be very challenging as it requires support from ISPs or content providers, which often restricts passive measurement studies to a small number of network links researchers have access to.

Second, the performance of an Internet flow can be affected by many confounding factors. One such factor is the used software platform, such as the operating system (and especially its TCP/IP networking stack) and its configuration. Another factor is the characteristics of the observed flows, as they can have significant differences in packet sizes and burstiness. And finally, transient noise caused by background traffic has to be considered. In most cases when using passive measurement tools it is not possible to avoid these factors. Thus, in order to draw accurate inferences, passive measurement tools must account for many confounding factors, which is a complicated task that can lead to inaccurate results. Active measurement tools on the other hand can avoid most confounding factors as the measured traffic is under the control of the researcher and thus, also repeatable.

Third, privacy is a concern with passive measurements [OSG07]. Since real traffic is monitored and recorded, it may contain private data, including passwords and data that identifies the sender (e.g., the sender and receiver IP addresses). Hence, techniques to anonymize such data must be carefully applied [PAPL06, FXAM04].

In practice, these limitations make it hard to design passive measurement methodologies to study network characteristics accurately and at large scale. Nevertheless, passive measurement techniques have been used in a large number of measurement work in recent years. Here, we list some popular examples. Sen et al. [SW02] monitored peer-to-peer traffic from a single ISP to learn about the dynamics of this traffic type, such as traffic patterns and traffic volume. Cleary et al. [CM01] presented a methodology to estimate bandwidth from passive measurement traces using sampling techniques. Eriksson et al. [EBN08] developed an algorithm that extracts a network's topology from network traces without actively probing the network. The algorithm works by analyzing the TTL hop count of IP packets.

2.1.2 Measurement Tools for End Users

The measurement tools presented so far were developed and used by researchers to study network characteristics. To use these tools and understand their results, users are required to have networking experience. Thus, these tools are not meant to be run by ordinary users. Nevertheless, also end users are interested in particular characteristics (such as available bandwidth or end-to-end latencies) of the networks they use. Consequently, there are a number of network measurement tools that are used by end users or that were built for end users to measure characteristics they are interested in. Designing and deploying measurement tools for end users has one major benefit: it allows researchers to

collect measurement data from end user networks, which are typically not accessible to researchers at large scale.

The currently existing measurement tools for end users can be grouped in three major categories. A first class of network measurement tools measures a particular network property. For example, the `ping` tool, which comes with most operating systems, is widely used to check the availability of a remote host or to measure the path latency to a remote host. Internet speed tests (e.g., [Ook06]) are probably among the most popular measurement tools used by end users. These tests measure the end-to-end throughput of a network path by running large TCP downloads and uploads.

A second class of network measurement tools for end users is formed by network diagnosis tools. Netalyzr [Int09] is a web-based tool that focuses on the detection of networking problems. It detects, for instance, manipulation of web content by a HTTP proxy in the path or blocking of traffic on some prominent ports. NPAD [MHR08] allows end users to diagnoses performance problems with TCP downloads from remote sites. NPAD can identify problems that are caused by failures in the last-mile network, or common misconfigurations on a client's host.

Finally, a third class of network measurement tools detects traffic shaping. Our own Glasnost tool (cf. Chapter 4) falls into this category. The Electronic Frontier Foundation's "Test Your ISP" project [Ele08] offers instructions for tracing a BitTorrent transfer and checking for forged packets. However, this method requires access to two hosts in different ISPs and involves the use of tools like Wireshark, which is beyond the capabilities of most end users. DiffProbe [KD10] is a probing method that checks for traffic differentiation based on active queue management. In particular, this methodology detects whether RED or weighted fair queueing is used to manipulate flow performance. For its measurements DiffProbe uses pairs of flows that share similar characteristics, but differ in their application payloads or the port they are sent on. Difference in these flows' performance is attributed to differentiation. Using statistical methods DiffProbe can distinguish different active queue management techniques. NANO [TMFA09] uses causal inference to detect the presence of traffic "performance degradation". NANO relies on a vast amount of passively collected traces from many users to infer if traversing a particular ISP leads to poor performance for certain kinds of traffic. NANO does not provide immediate per-user results, but aggregates results across users after enough data was collected. At this point, NANO is only available for the Linux operating system, which limits its adoption by end users.

While all of the tools presented in this section measure network characteristics that are relevant to end users, not all of these tools are equally popular. In fact, the popularity of a tool seems to be related to how simple it is to run measurements. The most popular tools typically make it very easy to set up measurements (e.g., they run measurements from the user's host to an already set-up measurement server), and they mostly do not require users to install software (e.g., by running measurements from a user's web browser). Additionally, the results these tools output are easy to understand, even for novice users. We used these insights while designing our Glasnost system, which is now a very popular tool with hundreds of thousands of users.

2.2 Measurement Platforms and Studies

Many researchers use Internet testbeds, such as PlanetLab [Pla], RON [ABKM03], and NIMI [PAM02] to conduct measurement studies. These testbeds are designed explicitly for use by researchers and usually do not allow end users to measure their own links. Recently, Measurement Lab [Mea] was started as a platform for Internet measurement tools for end users. As of May 2010, Measurement Lab provides a number of generic measurement tools that characterize certain features of the Internet path they measure (e.g., latencies and bandwidth) and tools that detect traffic shaping in the Internet.

Most measurement studies focus on the characteristics of academic and corporate networks or the core of the Internet. For example, Paxson [Pax97] studied network packet dynamics among a fixed set of Internet hosts located primarily in academic institutions. In later work, Paxson [Pax99] measured end-to-end packet dynamics by transferring 100 KBytes among 35 Internet sites. These probe streams were used to investigate the prevalence of packet reordering in the Internet, and also to measure bottleneck bandwidth and path latencies. Bellardo et al. [BS02] quantified the amount of packet reordering in the Internet by probing a number of popular hosts on the Internet. Ianneccone et al. [ICM+02] analyzed the prevalence and impact of link failures in an ISP backbone.

Several other studies have measured the network paths connecting the PlanetLab testbed, examining characteristics such as bandwidth [LSB+05] and node connectivity [BGP04]. However, Pucha et al. [PHM06] found that the network paths of today's testbeds are not representative for the Internet, as they are mostly comprised of academic nodes and lack nodes in corporate and broadband environments, which can have vastly different characteristics. In fact, our study in Chapter 3 supports this finding as we show that broadband networks have inherently different characteristics than academic networks.

2.2.1 Broadband Network Studies

Only few studies have investigated broadband access networks, and rigorous measurement data that characterize these networks at scale are lacking. Claypool et al. [CKL+04] performed a measurement study of access networks' queue sizes using 47 volunteering broadband hosts. They found that the median queue size was 350 ms in DSL networks and 150 ms in cable networks, and they showed in simulation that large queue sizes are detrimental to network traffic from interactive applications. Similarly, Jehaes et al. [JVC+03] observed a large increase in round-trip delays over saturated broadband links. Their experiments were limited to one DSL and one cable link. Lakshminarayanan and Padmanabhan [LP03] performed measurements from 25 broadband hosts to examine several application-level metrics such as TCP throughput and latency to different Internet hosts. In later work, Lakshminarayanan et al. [LPP04] outlined pitfalls in measuring link capacities of cable and DSL networks by using existing bandwidth estimation tools. In particular, the accuracy of these tools is greatly influenced by the rate regulation schemes used in cable and DSL networks. Three passive measurement studies have examined the traffic generated by residential customers in Japan [CFEK06], France [SCUB07], and Germany [MFPA09]. Each of the studies was limited to one ISP per country.

Beverly et al. [BBB07] use participants of filesharing applications to test for port blocking in edge networks. Their tool pretends to share popular files and waits for clients to

connect to them on a particular port. If there is port blocking, clients' connection attempts will fail. They found port blocking to be relatively common in the Internet.

While these studies are a first step towards a rigorous characterization of residential broadband networks, all of them (but Beverly's study on port blocking) remain small in scale.

In the next two chapters, we present the first large-scale studies of broadband access networks using two different approaches.

In Chapter 3, we present a novel active measurement methodology that can measure broadband networks remotely and without cooperation from end hosts connected to the broadband links. It allowed us to perform a large-scale measurement study of 11 major broadband ISPs in North America and Europe, focusing on characteristics such as bandwidth, latency, and loss rates.

In Chapter 4, we study the properties of broadband networks using a different approach. We built Glasnost, a system that allows end users to detect traffic differentiation, such as blocking or throttling of application traffic, in their broadband links. Glasnost has been used by hundreds of thousands of users worldwide. Using the data collected by Glasnost we were able to study the prevalence of traffic differentiation in broadband networks at large scale.

3 Characterizing Residential Broadband Networks

In the absence of systematic studies, knowledge about residential broadband networks is based on anecdotal evidence, hearsay, and marketing buzzwords. Although broadband networks are known to have very different characteristics from academic networks [PHM06, BGP04], there have been no large-scale studies quantifying these differences. As a result, researchers today are left to second-guess how well protocols or systems evaluated in academic networks would work in the commercial Internet, and in particular in broadband networks with their hundreds of millions of home users.

As discussed in the previous chapter, most existing measurement tools are not suitable for studying broadband access networks at scale. One major reason for this is that these tools typically require the user to run a program on both endpoints of a path. This requirement makes these tools be precise because they can separate upstream and downstream effects; however, to achieve large scale, we needed to work with hosts that were not under our control.

Hence, we developed a novel measurement methodology that allowed us to study broadband access networks with minimal end host cooperation. We use TCP acknowledgment and ICMP echo request probes to measure broadband end hosts. We leverage that Internet hosts are mandated by the corresponding protocols to respond to these probes. As we will show in this chapter, our methodology allows easy and accurate characterization of broadband link properties.

In this chapter, we present the first large-scale measurement study examining 1,894 broadband hosts from 11 major commercial cable and DSL providers in North America and Europe. The goal of our study was to perform a rigorous characterization of an extensive set of properties of broadband links. For this, we measured link bandwidths, latencies, and loss rates. We also characterized the properties of broadband queues, including queue sizes and packet drop policies. Finally, we examined a physical property specific to the cable transmission medium: the time-slotted access policy of the upstream channel. We measured the effects of this access policy on latency and jitter. Because broadband access links are asymmetric, we measured the properties of the upstream and downstream directions separately. Our analysis was driven by three questions:

1. What are the typical bandwidth, latency, and loss characteristics of residential broadband links?

2. How do the characteristics of broadband networks differ from those of academic or corporate networks?

3. What are the implications of broadband-network properties for future protocol and system designers?

Our study reveals important ways in which cable and DSL networks differ from the conventional wisdom about the Internet, accumulated from prior studies of academic networks. For example, many cable links show high variation in link bandwidths over short timescales. Packet transmissions over cable suffer high jitter as a result of cable's time-slotted access policy. DSL links show large last-hop delays and considerable deployment of active queue management policies such as random early detection (RED). Both cable and DSL ISPs use traffic shaping and deploy massive queues that can delay packets for several hundred milliseconds.

Our findings have important implications for emerging protocols and systems. For instance, the high packet jitter in cable links can affect transport protocols that rely on round-trip time (RTT) measurements to detect congestion, such as TCP Vegas [BP95] and PCP [ACKZ06]. Further, the large queue sizes found in cable and DSL ISPs can be detrimental to latency-sensitive applications such as VoIP when they are used concurrently with bandwidth-intensive applications, such as BitTorrent.

3.1 Measuring Network Characteristics with Minimal Cooperation

For our measurements to be generally applicable, the study needed to be performed at large scale. Previous studies of broadband networks [LP03, CKL+04, LPP04] used measurement tools that required cooperation from the remote broadband hosts. Such a methodology restricts the scale of the measurement study. Instead, we developed a different methodology for conducting large-scale detailed broadband measurements. Our approach requires minimal cooperation from the remote hosts, allowing our measurements to scale to thousands of broadband links.

Remote hosts need to cooperate only in two simple ways. First, they must respond to ICMP echo request packets with ICMP echo responses. Second, they must send TCP reset (RST) packets when they receive TCP acknowledgments (ACK) that do not belong to an open TCP connection. Both responses are mandated by the corresponding protocols [Pos81, Uni81].

Our technique is simple: we probe the broadband link with packet trains at different rates, using packets of various types and sizes. We use the responses received to infer a broad range of characteristics, both downstream and upstream. This approach requires support from only one endpoint of an Internet path, but obtaining accurate measurements is more challenging than with tools that require support from both endpoints or with tools that have been explicitly designed to measure one specific property [LPP04, JD03, DRM04].

3.1.1 Probe Trains to Measure Broadband Links

We used five types of probe packet trains to measure a broadband link. These trains differ with respect to three properties: the protocol packets used, the size of the probes, and the rate at which the trains were sent.

We refer to our high-rate probe trains as *floods*, and we refer to our low-rate probe trains as *trickles*. All floods were sent at 10 Mbps to saturate the broadband links. Consequently, packet floods measure the network under congestion. By contrast, all packet trickles were

sent at a rate of a few tens of Kbps, so they characterize the broadband network under normal operational conditions.

We limited the packet floods to at most 10 seconds, whereas we allowed trickles to last from several hours to several days. To capture diurnal variations in network properties, we repeated the floods every half hour for one week.

- **Asymmetric large-TCP flood.** We sent large (1,488-byte[1]) TCP ACK packets, and the remote host responded with small (∼40-byte) TCP RST packets. The ACK packets saturated the downstream links and router queues, but the responses, being smaller and fewer, did not saturate the upstream links or queues.
- **Symmetric large-ICMP flood.** We sent large (1,488-byte) ICMP echo request packets, and the remote host responded with ICMP echo response packets of the same size. This packet train saturated the links and router queues in both downstream and upstream directions.
- **Symmetric small-TCP flood.** We sent small (40-byte to 100-byte) TCP ACK packets, and the remote host responded with small (∼40-byte) TCP RST packets. Like the symmetric large-probe flood, this packet train saturated the network in both downstream and upstream directions but with much smaller packets.
- **Symmetric large-ICMP trickle.** We sent large (1,488-byte) ICMP echo request packets spaced at large intervals randomly chosen between 10 ms and 30 ms, and the remote host responded with ICMP echo response packets of the same size. Unlike the above probe trains, this packet train did not saturate the downstream or upstream links.
- **Symmetric small-TCP trickle.** We sent small (40-byte) TCP ACK packets spaced at large random intervals between 10 ms and 30 ms, and the remote host responded with small (∼40-byte) TCP RST packets. This packet train did not saturate the downstream or upstream links.

3.1.2 Measured Broadband Link Properties

Our measurements rely on one assumption: the broadband access link is the only bottleneck along the Internet path between our measurement hosts and the remote broadband hosts. We validate this assumption in the next section. This section describes how we measure the properties of the broadband links based on this assumption.

Link bandwidth

To estimate the allocated downstream bandwidth, we calculate the fraction of answered probes in the large-TCP flood, which saturates the downstream link only. For example, we estimate the downstream bandwidth of a link to be 6 Mbps when 60% of packets in our 10 Mbps large-TCP flood are answered. We use the same technique to estimate upstream bandwidths from the symmetric large-ICMP flood. The behavior of the large-ICMP flood

[1] We used 1488-byte probes because some DSL links running PPPoE or PPPoA have an maximum transmission unit of fewer than 1,500 bytes.

is driven by the bandwidth of the slower link, which for cable and DSL is the upstream link.

Our techniques yield incorrect estimates in the presence of cross-traffic. To identify and eliminate all measurement probes affected by cross-traffic, we use the IP identifier (IPID) field in the IP headers of the response packets. Many Internet hosts increment the IPID field by a fixed number (typically one) for every new packet they create. We can use this fact to detect when the broadband host sent additional packets at the time of our measurements, thus excluding measurements that were affected by cross-traffic.

Packet latencies and jitter

We characterize three types of packet delays and their variation (jitter) for each link: *queueing delay*, *propagation delay*, and *transmission delay*. Intuitively, queueing delay is the time a packet spends queued behind other packets in the router, transmission delay is the time taken by the router to send the first packet in the queue over the wire (thus it depends on the rate of the link), and propagation delay is the time taken by the packet to be received on the other end of the wire (thus it depends on the link length and the propagation speed of the used medium).

We estimate the maximum possible queueing delays (or queue lengths) by calculating the variation in RTTs of packets in our floods. To determine downstream queue lengths, we calculate the difference between the 95th percentile highest RTTs[2] and the minimum RTTs of packets in the large-TCP flood. Remember that the large-TCP flood overflows only the downstream router queues. A similar calculation for the large-ICMP flood, which overflows queues in both directions, estimates the sum of downstream and upstream queue lengths. We subtract the downstream queue length from this estimate to obtain the length of the upstream queue.

To study broadband link propagation delays, we estimate their last-hop delays. We computed last-hop delay as the difference between the latencies of small-TCP trickle probes to the broadband host and to its last-hop router. By comparing the last-hop delays for different packet sizes, we are able to infer the transmission delays in broadband links.

Packet loss

We estimate typical packet loss rates in broadband networks by calculating the fraction of lost packets in the small-TCP trickle. To detect packet loss due to queue management policies, such as random early detection (RED), we examine how the loss rate varies with the latencies of the packets.

3.1.3 Validating our Assumptions

Next, we discuss seven important concerns about our methodology:

1. While protocol standards require a response to each of our probes, NATs and firewalls can block incoming probe packets. *How many Internet hosts respond to our probes?*

[2]This filters out spikes and jitter to obtain a more reliable estimate of the maximum RTT.

2. To be accurate, our probes must traverse the entire Internet path reaching the broadband host and not be answered by an intermediate router. *Do our measurements accurately reflect the properties of broadband access links?*

3. We assumed that the broadband links are the bottlenecks in the measured Internet paths. *How often are broadband links the bottlenecks along the measured Internet paths?*

4. We assumed broadband hosts respond to all probes without any delay. In practice, end hosts could drop or rate limit their responses. *How often do broadband hosts delay or drop response packets?*

5. Our probes can be interpreted as port scans or attacks. *What are the best practices we adopted?*

6. Our methodology requires us to flood end hosts with different probe packets at 10 Mbps. *Why is it necessary to flood end hosts with probes to characterize broadband links?*

7. Measuring network characteristics with minimal cooperation from an end host is more challenging than with tools that require full cooperation. *What are the limitations of our methodology?*

How many Internet hosts respond to our probes?

In theory, our methodology should enable us to measure all end hosts on the Internet since the protocols require a response to each of our probes. In practice, however, many hosts are either offline or behind NATs and firewalls that block or rate-limit incoming probe packets.

We conducted a simple experiment to estimate the fraction of Internet hosts that can be measured with our methodology. We sent probes to three types of hosts: end hosts in commercial broadband ISPs; end hosts in academic and research networks; and Internet routers. We selected end hosts in broadband and academic networks from a 2001 trace of peers participating in the Gnutella file-sharing system [SGG02]. We used DNS names to select hosts belonging to major DSL and cable ISPs and university domains in North America and Europe. For example, we classified a host as a BellSouth DSL host if its DNS name is of the form `adsl-*.bellsouth.net`. We discovered Internet routers by running `traceroute` to the end hosts in broadband and academic networks.

Table 3.1 presents our results. We probed 1,000 hosts in each of the three host categories. Overall, more than 6% of the broadband hosts, 5% of the academic hosts, and 69% of the routers responded to both probe types (ICMP echos and TCP ACKs). While this may seem like a small percentage, there are millions of hosts in the Internet, and it should be easy to find thousands of suitable hosts for a measurement study.

We believe that the primary reason for the large difference in the response rates between routers and other end hosts is the low availability of the end hosts. Unlike routers, many end hosts are often offline and disconnected from the Internet. To estimate the effect of host unavailability, we probed the set of end hosts that responded to our probes for a second time after a few weeks. Only 67% of the hosts responded again, suggesting the high impact of end host unavailability. Moreover, our end hosts were selected from a trace

	Broadband	Academic	Router
TCP ACK	7.2%	13.4%	69.6%
ICMP Echo	25.0%	8.9%	89.3%
Both	6.6%	5.3%	69.4%

Table 3.1: Fraction of Internet hosts responding to our probes. We selected a sample set of 1000 hosts from each of three different categories of hosts: hosts in commercial broadband ISPs, hosts in academic and research environments, and Internet routers.

collected five years earlier. In contrast, the router list was generated from traceroutes conducted only a few weeks before this experiment.

Our results suggest that our methodology can be used to perform a large-scale study on the characteristics of residential broadband networks.

Do our measurements reflect accurately the properties of broadband access links?

We ran controlled experiments using five broadband hosts (two cable and three DSL) under our control located in North America and Europe. These experiments were performed on a small scale because they required end host cooperation. Although we hoped to recruit more volunteers, the effort required to set up our experiments made it difficult to convince users to perform them: our experiments require root access and manual changes to the users' firewalls.

First, we checked whether the probe packets were being sent over the broadband link or whether they were being answered by a router in the middle of the network. We found that in all cases the probes were being responded to by the NAT-enabled modems in the customers' premises. By configuring the modems to forward any arriving probe packets to end hosts, we were able to receive the probes at our end hosts. Note that the probes must cross the broadband link to reach the modems.

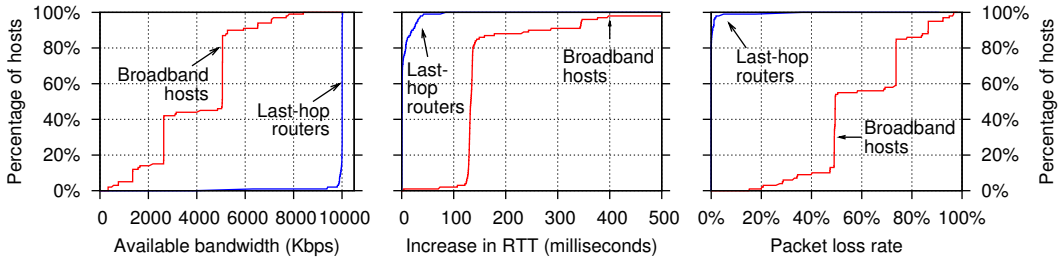
Second, we checked whether the NAT-enabled modems affected the measurements by delaying or rate-limiting their responses. We gathered two traces for each link: one when the modem responded to the probes, and another when the modem forwarded all probes to the broadband hosts. We configured the broadband hosts to respond to the probes without any delay (less than 100 μs) or rate-limiting. We compared the two traces with respect to latencies and losses of probes and responses. The two traces matched closely in all cases, suggesting that the modems do not adversely affect our measurements.

Finally, we verified the accuracy of our bandwidth and queue length measurements. We compared the measured bandwidths of the access links with the rate speeds advertised by their ISPs. We found that these bandwidths matched very closely – the average difference in downstream bandwidths was less than 3%. To validate our queue length estimates, we used our access to the end hosts to measure the upstream and downstream queue lengths separately and accurately. The measurements matched the estimated queue lengths very well. The close match suggests that both our bandwidth and queue length measurements are accurate.

How often are broadband links the bottlenecks along the measured Internet paths?

Our methodology assumes that the broadband link is the bottleneck on the Internet path measured. When the probes are sent from well-connected hosts, the broadband links are likely to be the bottlenecks in these paths. To validate this assumption, we sent a large-TCP flood probe train from a well-connected host in an academic network to the broadband host and another train to its last-hop router. We used `traceroute` to discover these routers. Comparing these two probe trains revealed that the broadband links are in fact the bottlenecks.

Figure 3.1 compares the available bandwidth, the RTT increases, and the packet loss rate of the two traces for 1,173 randomly selected broadband hosts. Most paths to the last-hop routers achieved the full 10 Mbps throughput, experiencing almost no losses or RTT fluctuations. By contrast, the paths including the broadband link had much lower throughput, considerable RTT increases, and high packet loss. This suggests that these variations are caused by the last hop, i.e., the broadband link.



(a) Bandwidth **(b)** Latency **(c)** Loss

Figure 3.1: The broadband link is the bottleneck. Comparison of the paths to the residential broadband hosts and their corresponding last-hop routers. The former include the broadband link, while the latter do not. The two sets of paths have very different characteristics, which validates our assumption that broadband links are the bottlenecks along the Internet paths to broadband hosts.

How often do broadband hosts delay or drop response packets?

Our methodology assumes broadband hosts respond to probes without any delay. However, several factors could delay or even prevent hosts from responding to some or all of our probes. For example, a firewall may block certain types of probes, such as ICMP echo requests. Some routers add a delay between the arrival of a probe and the departure of the response [GP02]. Also, a host with limited processing power might delay or drop packets arriving at high rates.

In our analysis, we removed all hosts that did not respond to our probes. We also removed the broadband hosts that rate-limited their probe responses. We identified such hosts by checking for large loss episodes occurring periodically.

We performed the following experiment to check whether our probe trains were too aggressive for the processing power of some hosts. We sent probe trains at 10 Mbps but

with varying packet sizes. Although all trains consumed the same bandwidth, their packet sending rates were different. We checked whether hosts experienced higher losses at faster sending rates. A higher loss rate suggests that an end host cannot process packets at fast rates. We checked how losses varied with packet sending rates for all broadband hosts in our study. The loss rates remained constant for over 99% of the hosts in our study, suggesting that the end hosts have sufficient processing power to handle our probing rates.

What are the best practices we adopted?

Performing active measurements on the Internet raises important usage concerns. Although it is difficult to address and eliminate all such concerns, we adopted a set of precautions to mitigate these concerns. We restricted our high rate probe trains to no more than 10 seconds each. We also embedded a custom message in each of our probe packets which described the experiment and included a contact email address. To date, we have not received any complaints.

Another cause for concern was that users with a per-byte payment model end up paying for our unsolicited traffic. To mitigate this concern, we only measured hosts in ISPs that offer flat-rate payment plans, and we limited the total amount of data sent to any single broadband host over our entire study.

Why is it necessary to flood end hosts with probes to characterize broadband links?

Our methodology allows us to measure three different major characteristics of broadband links: link bandwidth, packet latency, and packet loss. While we use only low-bandwidth probe trains to measure packet loss, we use high-bandwidth probe floods at 10 Mbps to measure link bandwidth and packet latency.

We admit that there are more efficient ways to measure link bandwidths. For example, Croce et al. [CEUB08, CEUB09] refined our methodology to measure ADSL link bandwidths with two orders of magnitude less measurement traffic. However, to date we are not aware of a methodology that allows us to study packet latencies, including the router queue sizes deployed in broadband networks, without flooding the end host. To limit the absolute amount of traffic we generate for our measurements, we re-use the latency measurements to also infer link bandwidths.

What are the limitations of our methodology?

Our approach requires support from only one endpoint of an Internet path to study broadband networks at large scale. Compared to tools that require support from both endpoints, this makes it more challenging to obtain accurate measurements. The reason is that our approach does not allow measuring the characteristics of the upstream and downstream link independently. Instead, we infer characteristics for both directions by exploiting the fact that broadband networks have asymmetric links. In practice, having access to only one endpoint can cause our measurements to be less accurate. However, as pointed out before, we found that our results on link bandwidth, latencies, and loss rates closely matched the results obtained from measurements of each direction individually.

While our technique builds on the fact that broadband networks have asymmetric links, it works even if links are symmetric, i.e., when upstream and downstream capacities are the same. It is inaccurate only when the upstream bandwidth exceeds the downstream

bandwidth. The residential ISPs we measured in this paper do not offer such configurations.

3.2 The Characteristics of Big DSL and Cable ISPs

In this section, we analyze the data gathered from sending probe packet trains to a large number of residential broadband hosts in 11 major ISPs (see Table 3.2). We examine three important characteristics of broadband networks, namely link bandwidths, packet latencies, and packet loss. Analyzing these properties is important because they affect the performance of protocols and systems running over broadband.

We conducted our measurement study in April and May 2007. Each probe train was sent from well-connected hosts located in four academic networks (Figure 3.2). The academic networks used are dispersed geographically – three in North America (in the south, northwest, and northeast) and one in Europe.

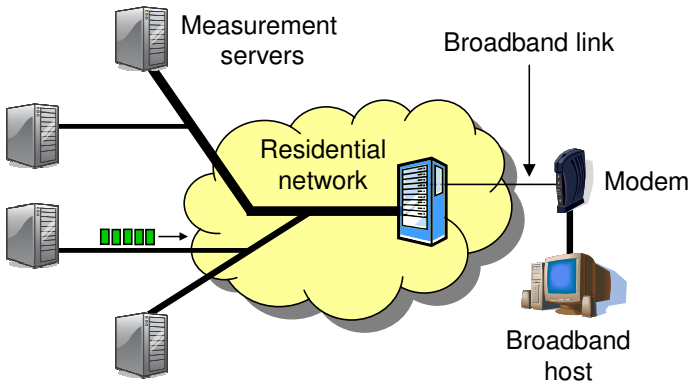


Figure 3.2: Measurement setup.

Two important limitations affect our measurement study. First, we studied only major cable and DSL ISPs in North America and Europe. Our conclusions are unlikely to generalize to high-speed fiber-based broadband ISPs, such as those in Japan or South Korea [CFEK06], or to mobile broadband technologies, such as UMTS or EVDO. Second, we removed all hosts that did not respond to our probes or that were rate-limited, which could introduce some unknown bias.

3.2.1 Selecting Residential Broadband Hosts

We measured the link characteristics of 1,894 broadband hosts from 11 major cable and DSL providers in North America and Europe. We selected these hosts by probing hosts measured in a previous study of Napster and Gnutella [SGG02]. We identified IP address ranges of popular residential ISPs from IP-to-DNS mappings (e.g., BellSouth's DNS names are `adsl-*.bellsouth.net`), and we scanned for IP addresses responding to our probes.

Table 3.2 summarizes high-level information about the ISPs we measured. Our study includes five out of the top ten largest broadband ISPs in the U.S.[3] [Gol08a], the largest cable provider in Canada [Gol08b], the second-largest cable provider in the Netherlands [Alb08], and the largest DSL provider in the U.K. [Tel07]. From each ISP, we chose approximately 100 hosts randomly and measured them.

Table 3.2 also shows the bandwidths advertised by ISPs on their webpages in May 2007. Although a range of speeds is available, all advertised bandwidths are lower than 10 Mbps. We took advantage of this property by using 10 Mbps probe streams for measuring these broadband links and their routers.

	DSL						***Cable***				
	Ameritech	*BellSouth*	*BT Broadband*	*PacBell*	*Qwest*	*SWBell*	*Charter*	*Chello*	*Comcast*	*Road Runner*	*Rogers*
Company	AT&T	AT&T	BT Group	AT&T	Qwest	AT&T	Charter Comm.	UPC	Comcast	TimeWarner	Rogers
Region	S+SW USA	SE USA	UK	S+SW USA	W USA	S+SW USA	USA	Netherlands	USA	USA	Canada
Hosts measured	113	155	173	158	97	397	114	120	118	301	148
Offered BWs (bps)	768K, 1.5M, 3M, 6M	768K, 1.5M, 3M, 6M	2-8M	768K, 1.5M, 3M, 6M	256K, 1.5M, 7M	768K, 1.5M, 3M, 6M	3M, 5M, 10M	384K, 1.5M, 3M, 6M, 8M	6M, 8M	5M, 8M	128K, 1M, 5M, 6M

Table 3.2: Measured hosts in our 2007 study. We measured 1,894 broadband hosts from 11 major commercial cable and DSL providers in Europe and North America.

3.2.2 Allocated Link Bandwidth

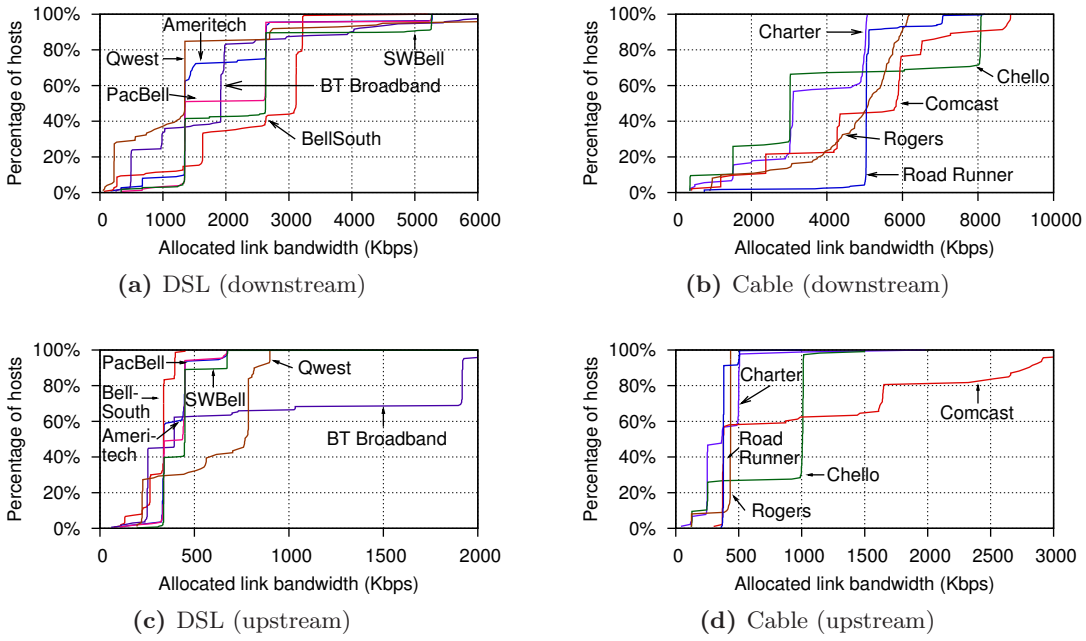
Allocated link bandwidth refers to the bandwidth reserved by a provider for a single broadband user. In cable networks, allocated link bandwidth is the portion of the shared link's capacity assigned to an individual user, whereas in DSL networks it is the ISP's cap on a user's traffic rate. Characterizing allocated link bandwidths in broadband networks helps to predict the maximum throughput any transport protocol (such as TCP Reno or TCP Vegas) or application (such as BitTorrent) can achieve. As described in Section 3.1.2, our probe streams measured allocated bandwidths by saturating the broadband links.

What are the allocated link bandwidths?

Figures 3.3a and 3.3b show the cumulative distributions of downstream link bandwidths for the different DSL and cable ISPs. For many ISPs, the distributions jump sharply at distinct bandwidth levels, such as 256 Kbps, 384 Kbps, 512 Kbps, and 1 Mbps. Only two cable ISPs (Rogers in Canada and Comcast in the United States) allocate bandwidths distributed along a continuous spectrum.

By comparing these measured allocated bandwidths to the advertised link speeds from Table 3.2, we can confirm some commonly held opinions. We find that most DSL ISPs have bandwidth rates corresponding to those advertised. By contrast, major cable ISPs,

[3] During the recent consolidation of the U.S. telecom industry, many large ISPs merged with each other. Four of the eleven ISPs we measured are owned today by AT&T, a single company. However, our measurements show that their networks have very different characteristics. For the purposes of this study, we treat them as independent ISPs.



(a) DSL (downstream) **(b)** Cable (downstream)

(c) DSL (upstream) **(d)** Cable (upstream)

Figure 3.3: Allocated downstream and upstream link bandwidths. Most ISPs offer upstream bandwidths of 500 Kbps or less, even when the downstream bandwidths exceed 5 Mbps.

such as Comcast and Rogers, show rates different from those advertised (both higher and lower). We assume that this discrepancy is due to the nature of the two technologies: cable is a shared medium, whereas DSL is not. Our data also shows that many cable ISPs have significantly higher downstream bandwidths than DSL.

Figures 3.3c and 3.3d show the cumulative distributions of upstream link bandwidths. Upstream bandwidths are strikingly different from downstream bandwidths — with the exception of a few ISPs, most upstream bandwidths are lower than 500 Kbps, even when their downstream bandwidths exceed 5 Mbps. To examine this difference, we plotted the ratio of downstream to upstream link bandwidths in Figure 3.4. Most DSL hosts have much smaller ratios than cable hosts, because compared to cable, DSL hosts have

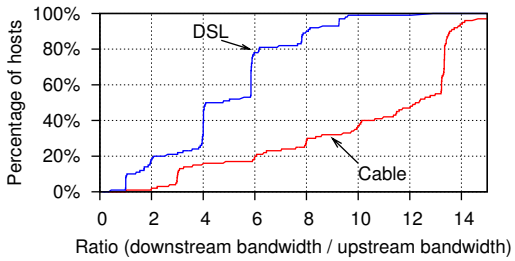


Figure 3.4: The ratio of downstream to upstream link bandwidths. The gap between downstream and upstream bandwidths is much wider for cable networks than for DSL networks.

lower downstream but similar upstream bandwidths. For over half of the cable hosts, the downstream bandwidths exceed upstream bandwidths by a factor of more than 10.

The highly asymmetric nature of bandwidths does not align well with the requirements of emerging peer-to-peer systems [Coh01, CGN+04], whose workloads tend to be symmetric. Despite all the excitement surrounding user-driven content generation and distribution, residential networks continue to be predominantly optimized for client-server workloads.

How stable are the allocated link bandwidths in the short term?

Next, we study the short-term and long-term stability of link bandwidths. Understanding the stability of link properties is useful for designing network protocols that can quickly adapt to changing link conditions.

We examined the stability of the allocated link bandwidths over the 10 second duration of our packet floods. For this, we divided the 10 seconds into 100 ms intervals (the RTT of a typical Internet path), we estimated the bandwidth within each interval, and we compared the different estimates across intervals. Figure 3.5 shows how bandwidths for a PacBell link (DSL) and a Rogers link (cable) vary over time. While the PacBell link shows stable bandwidth, the Rogers link weaves above and below its average bandwidth of 3 Mbps.

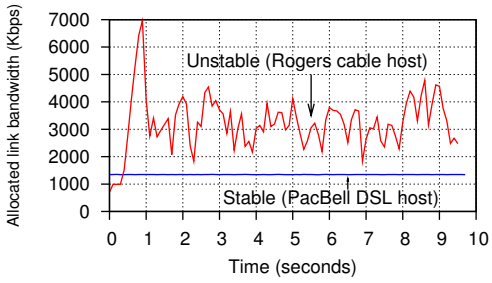
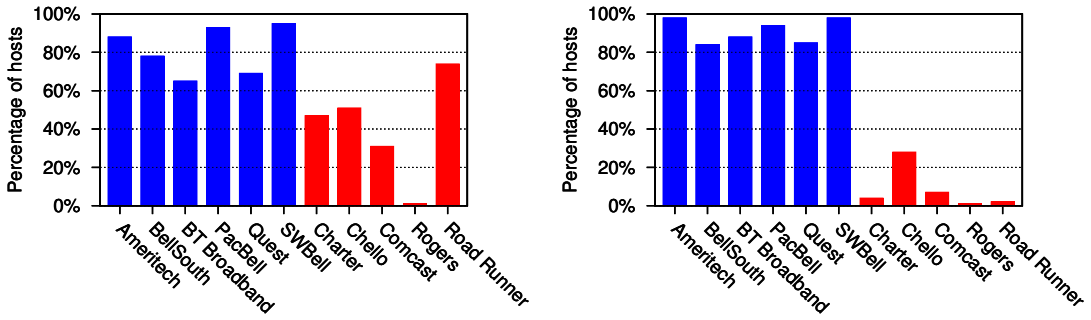


Figure 3.5: Stable and unstable link bandwidths. The allocated link bandwidth is stable for the PacBell DSL host. For the Rogers cable host, the access link bandwidth varies greatly over time.

Figure 3.6a shows the fraction of DSL and cable links that exhibit stable bandwidths in the downstream direction. We classify a link as stable if at least 90% of the 100 ms intervals show a bandwidth estimate within 10% of the average bandwidth. Although most DSL ISPs show stable link bandwidths, we found that most cable ISPs have bandwidths that vary significantly even within the short 10 s duration of our probes. We also found that upstream bandwidths have unstable short-term characteristics (see Figure 3.6b). The instability is even more striking for cable ISPs.

While we do not know the exact reason for these short-term bandwidth instabilities, there are a number of possible explanations for them. Variations in bandwidth can be due to cross-traffic that competes with our probe floods. This cross-traffic can occur on the last mile, within an ISP's network, or somewhere else on the Internet path between our



(a) Downstream **(b)** Upstream

Figure 3.6: Fraction of hosts with "stable" downstream link bandwidths. Most DSL links show stable bandwidths, whereas most cable links do not. Cable downstream bandwidths tend to be more stable than upstream bandwidths, likely an artifact of TDMA in the upstream.

measurement servers and the broadband host measured. While cross-traffic can explain some of the variations we observed, it cannot explain the significant difference between cable and DSL ISPs, as we expect the effect of cross-traffic to be evenly distributed across all the nodes of our measurements. Instead, we believe that this difference is rooted in the nature of the technologies used. Cable links are shared amongst users and links are likely to be oversubscribed; thus the achieved bandwidth can vary with the current network load. This is especially true for the upstream direction where Time Division Multiple Access (TDMA) is used to share the bandwidth amongst users.

This large short-term variation in cable bandwidths poses new challenges to transport protocol designers. Traditionally, transport protocols have been developed to achieve stable throughput and to avoid reacting to short-term events (on timescales less than one RTT) [FHPW00]. However, when running in a cable network environment, protocols need to adjust quickly to rapidly changing link conditions. Slow reacting protocols might not achieve good throughput in cable networks.

How stable are the allocated link bandwidths in the long term?

We now turn our focus to the long-term diurnal stability of link bandwidths. We took measurements of the upstream and downstream bandwidths every half an hour for one week, selecting 70 random hosts from each ISP[4]. Figure 3.7 shows the diurnal variation in bandwidths for one DSL ISP (BT Broadband) and one cable ISP (Rogers). Each curve shows the bandwidth variation averaged across all measured links within one ISP. To account for links with different bandwidths, we normalize each link's bandwidth by using the maximum measured bandwidth of that link during the entire measurement period.

We found that most ISPs have high long-term stability. As the curve for BT Broadband illustrates (Figure 3.7), bandwidths usually do not vary with the time of the day. By contrast, a small number of ISPs, such as Rogers, show a clear diurnal trend in link

[4]To minimize DHCP effects, we discarded any host that went offline (i.e., did not respond to probes) during this period. We also excluded measurements taken when we detected cross-traffic.

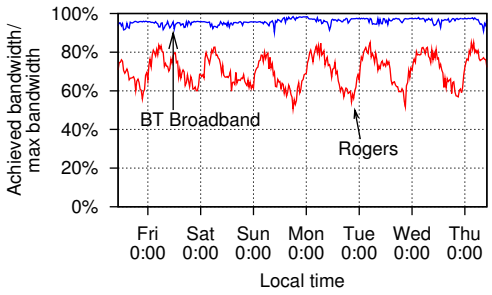


Figure 3.7: Long-term link bandwidth stability. Whereas BT Broadband has stable bandwidths over time, Rogers's allocated link bandwidths show diurnal patterns.

bandwidths. Rogers's end hosts see significantly lower bandwidths (almost a 25% reduction) in the evening (typically between 4pm and 7pm) than in the early morning (typically between 1am and 5am). In the upstream direction, we find stable bandwidths (not shown) for all ISPs, including Rogers. These findings seem to contradict the popular belief that competing traffic affects the bandwidths of broadband hosts. For most ISPs, we found little evidence that competing traffic affects link bandwidths during the day.

Is there evidence of traffic shaping?

Traffic shaping is likely to be one of the factors leading to the bandwidth instability encountered in broadband networks. Some ISPs allow an initial burst of bandwidth that is often many times greater than the advertised bandwidth. For example, Comcast's PowerBoost feature [Com07] increases the customer's allocated bandwidth for a short time — typically for transfers up to 5 or 10 MBytes —, reducing the download times of relatively small files, such as MP3s. Other ISPs throttle the bandwidth allocated to long running transfers to discourage heavy hitters from consuming a disproportionate share of the bandwidth.

Because our probe floods were limited to 10 seconds, we could only detect the traffic shaping associated with short-duration flows. To do this, we performed the following experiment. We used our packet streams to compute the allocated link bandwidth of each 100 ms interval. To detect the presence of traffic shaping, we checked for a consistent and significant drop in bandwidth after some initial period. Figure 3.8 shows an example link from Ameritech DSL, whose bandwidth drops from 2.5 Mbps to 1.5 Mbps (its long-term rate) after the first second.

We found similar downstream traffic shaping techniques used by three ISPs: Ameritech, Comcast, and Chello. 11% of the Ameritech links, 26% of the Comcast links, and 67% of the Chello links provide an initial burst of bandwidth to speed up short transfers. The burst rates are typically more than 1 Mbps above the long-term bandwidth. However, in many cases, we were unable to quantify precisely the burst rates because they exceeded the rate of our probe train. In the upstream direction, we found no evidence of traffic shaping or bandwidth throttling of our probe stream. The short duration of our probe trains (10 seconds) could have prevented us from detecting upstream traffic shaping.

Recently, there have been numerous reports of ISPs throttling or even blocking traffic of certain applications, such as BitTorrent, to mitigate network congestion [Can08a, Top07].

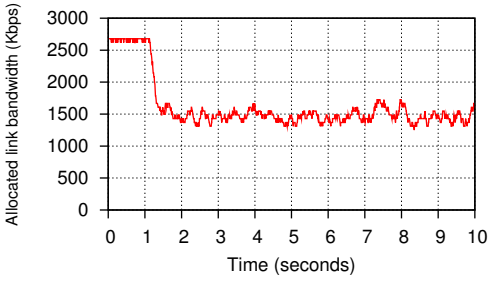


Figure 3.8: Traffic shaping in broadband networks. The downstream bandwidth of this link is initially 2.5 Mbps, but it drops to 1.5 Mbps after one second.

The measurement methodology we presented here cannot identify such instances. Instead, we discuss this matter in detail in Chapter 4.

Since history-based bandwidth prediction is a popular technique used in several transport protocols [BP95, FHPW00, ACKZ06] and content distribution systems [CRSZ01], the presence of traffic shaping in broadband networks has implications for applications and transport protocols. Although our traffic shaping analysis is preliminary, it suggests that using past bandwidth estimates to predict future bandwidth conditions might not work well over broadband links.

3.2.3 Packet Latencies

We analyzed each of the three components of packet latencies: propagation delays, transmission delays, and queueing delays.

Do broadband links have large propagation delays?

A link's propagation delay is the time elapsed between sending a bit at one end and receiving it at the other end. On one hand, broadband propagation delays could be short because the links themselves are short. On the other hand, sophisticated signal processing and error correction algorithms could increase broadband propagation delays.

Our methodology does not allow us to directly measure the propagation delay of a broadband access link. Instead, we estimated the round-trip delay of the last-hop of the path between our measurement hosts and the broadband hosts. This last-hop delay roughly approximates the sum of downstream and upstream broadband propagation delays.

To estimate this delay, we sent small-TCP trickle probes to both the broadband host and its last-hop router. The trickle consisted of several hundred widely spaced small probes and their responses. We calculated the last-hop RTT by subtracting the minimum RTT to the last-hop router from the minimum RTT to the broadband host. We used the minimum RTT estimates to avoid transient jitter as a result of queueing at intermediate routers.

Figure 3.9 shows the RTTs for a representative DSL host and its last hop router. While the RTT to the router stays constant at 53 ms for all probe sizes, the minimum RTT to the DSL host varies from 102 ms for 100-byte probes to 152 ms for 1,488-byte probes. This RTT increase is likely due the low bandwidth of this DSL link. Sending larger packets takes

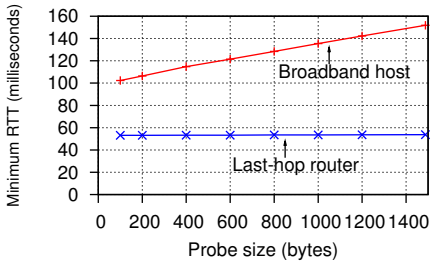
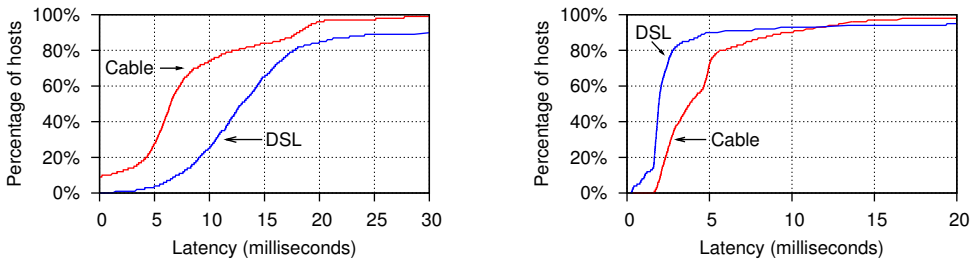


Figure 3.9: Minimum RTT packets of different size for a representative DSL host and its last hop router. Subtracting the minimum RTTs for both targets results in the last-hop RTT.

more time than sending smaller packets. The difference in RTT of router and broadband host is the last-hop RTT.

Figure 3.10a shows our results for last-hop RTTs for cable and DSL networks. DSL hosts exhibit considerably higher propagation delays than cable hosts. 75% of all DSL hosts have last-hop delays larger than 10 ms, while 15% have propagation delays larger than 20 ms. In comparison, typical US coast-to-coast RTTs between academic hosts are around 50 ms. The large propagation delays for DSL hosts are surprising because many last-hop routers are located in the same city as their end hosts[5].

Figure 3.10b shows the jitter in our latency measurements. We used the RTTs of the small-TCP trickle to estimate the jitter of the broadband link. We calculated jitter by subtracting the 10th percentile RTT from the 90th percentile highest RTT. Compared to cable, DSL links have higher last-hop delays but lower jitter. We believe that the characteristics of the upstream cable links are responsible for these differences. We examine this hypothesis next.



(a) Last-hop delay **(b)** Last-hop jitter

Figure 3.10: Last-hop delay and jitter in cable and DSL networks. DSL shows higher last-hop RTTs than cable, while cable exhibits higher jitter than DSL.

[5]We inferred the locations of hosts and routers from their DNS names as suggested in [SMWA04].

How do cable's time-slotted policies affect transmission delays?

Transmission delay refers to the time elapsed between a router starting to transmit a packet and ending its transmission. It is usually calculated by dividing the packet length by the link bandwidth. However, cable links use a reservation policy to transmit packets in the upstream direction. To send a packet upstream, a cable modem first requests a time-slot from the cable headend. This policy can cause additional delays to a packet's transmission. After receiving a request grant, the modem sends the accumulated packets in a bursty transmission at high-speed. We examined the effects of such transmission policies under both low and high network loads.

First, we studied transmission delays under low network loads. We used the large-ICMP trickle to calculate the last-hop delays, similar to the experiment conducted in the previous section. We compared these last-hop large-packet delays to the last-hop small-packet delays measured in the earlier experiment. The differences in the last-hop delays between large (1,488-byte) and small (100-byte) packets are mostly due to the additional transmission delays incurred by sending larger packets.

Figure 3.11 shows the difference in transmission delays between large and small packets for cable and DSL hosts. We found that the transmission delays for DSL are large, on the same order of magnitude as their propagation delays, shown in Figure 3.10a. By contrast, the transmission delays for cable are surprisingly low: 99% of hosts show an increase of less than 1 ms to send an extra 1,388 bytes.

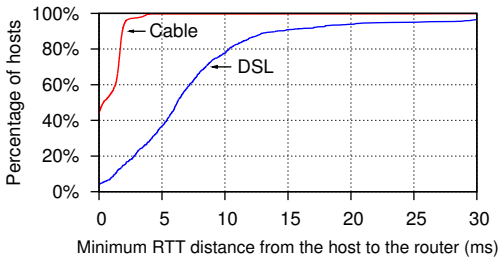


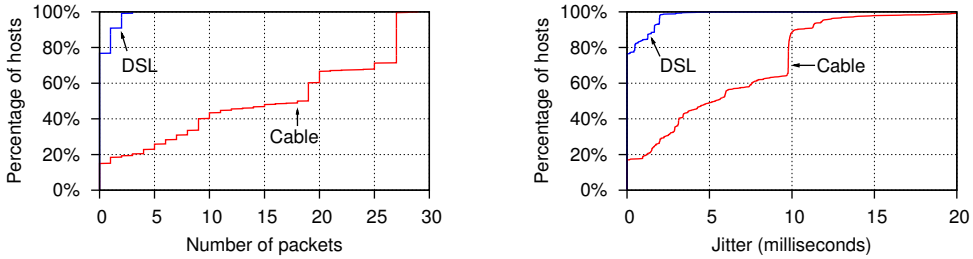
Figure 3.11: Difference in transmission delays between large and small packets. DSL shows longer transmission delays than cable.

We believe that the time-slotted nature of cable links is responsible for these short transmission delays. All our probes, both large and small, experience similar waiting times for a time slot. Once a slot has been granted, packets are transmitted at the full link speed (10.24 Mbps according to the DOCSIS 1.0 specification [Cab06]). This matches our data very well; our measured transmission delays correspond to an upstream link speed of about 11 Mbps.

Next, we examined transmission delays under high network load. In this case, packets have to wait longer to reserve a time slot. When the reservation is granted, multiple waiting packets can be concatenated and sent in a single burst. Although concatenation reduces the overhead of scheduling many small packets, such as TCP ACKs, it introduces a systematic jitter, which we refer to as the *concatenation jitter*.

We used the small-TCP flood to examine the effects of concatenation because it saturates the upstream link with a large number of small packets, which are well suited for concatenation. We clustered probe responses received in very close succession (separated by less than 100 μs) into a single bursty transmission, and we calculated the number of packets in the largest cluster. Because there is no known concatenation feature for DSL, we expected these links to show only minimal burst sizes.

Figure 3.12a shows the extent of packet concatenation in DSL and cable ISPs. As expected, DSL links show only very short bursts, whereas 50% of cable links concatenate 19 packets or more in a single burst. We used the link's speed and the number of packets in a burst to estimate a lower bound on the amount of concatenation jitter when the link is saturated. Figure 3.12b shows the results. Whereas the mean concatenation jitter for cable networks is about 5 ms, many links experience 10 ms or more of jitter due to concatenation.



(a) Maximum number of packets per burst

(b) Lower bound estimate of concatenation jitter

Figure 3.12: Cable links show high RTT variation. In addition to a high level of basic jitter, cable modems can send small packets in a single burst and thus cause additional jitter.

In cable networks, the concatenation jitter under high network load can be higher than the end-to-end jitter over the entire path under normal load (shown in Figure 3.10a). The presence of high jitter in cable networks has important consequences for protocols such as TCP Vegas [BP95] and PCP [ACKZ06] which interpret changes in RTT as a sign of incipient congestion. High jitter could cause these protocols to enter congestion avoidance too early, leading to poor performance.

How large are broadband queueing delays?

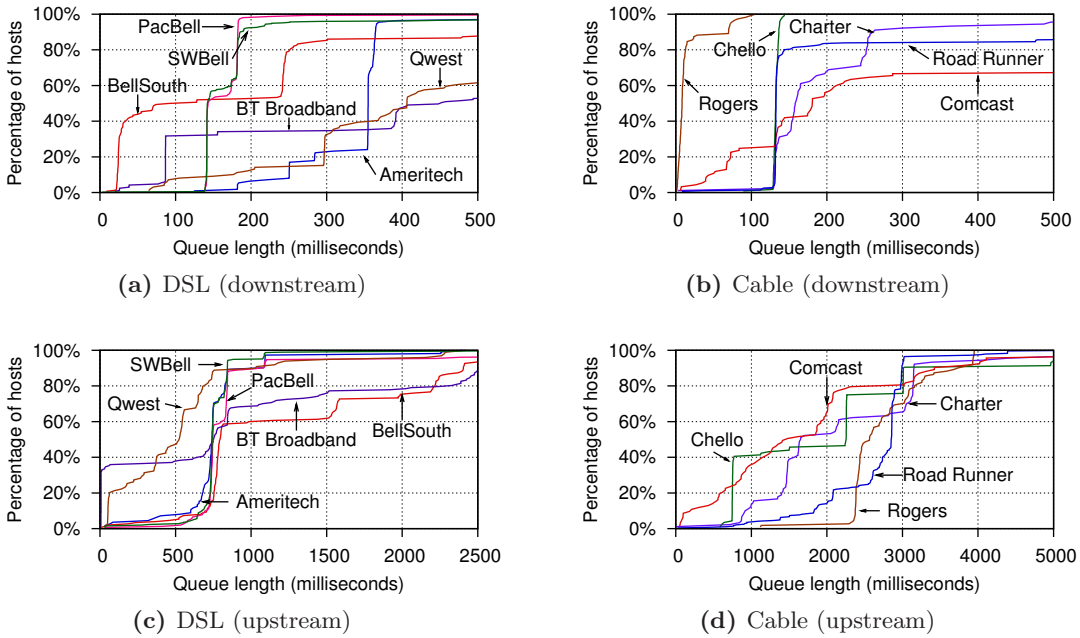
Sizing router queues is a popular area of research (e.g., [AKM04, BGG$^+$08]). A common rule of thumb (attributed to [VS94]) suggests that router queues' lengths should be equal to the RTT of an average flow through the link. Larger queues lead to needlessly high queueing delays in the network. We investigated how well this conventional wisdom holds in broadband environments.

We measured queue lengths in milliseconds by calculating the RTT variation of our probe streams' packets. To estimate downstream queue lengths, we used large-TCP flood probe trains, which saturate the downstream but not the upstream link. We calculated the

difference between the minimum RTT and the 95th percentile highest RTT. To estimate upstream queue lengths, we first measured the difference between the minimum RTT and the 95th percentile highest RTTs of large-ICMP flood probe trains. This difference corresponds to the sum of downstream and upstream queue lengths. We then subtracted the estimate of the downstream queue length to obtain the length of the upstream queue.

Figures 3.13a and 3.13b show the cumulative distributions of downstream queue lengths for different cable and DSL providers. Across most cable ISPs and two DSL ISPs (PacBell and SWBell), the curves show a sharp rise at 130 ms. This value is consistent with that recommended by the ITU G.114 standard for maximum end-to-end latency in a network running interactive traffic: 150 ms. Nevertheless, these queue lengths are significantly higher than a typical flow's average delay, which ranges between 50 ms and 75 ms within North America or Europe. By contrast, we observed queueing delays of up to 2 seconds for a significant number of Comcast and Qwest hosts and up to 6 seconds for some BT Broadband hosts (not shown). Our findings show diverse queue configurations for broadband links, with most hosts exhibiting queue lengths significantly higher than 130 ms.

Figures 3.13c and 3.13d show the cumulative distributions of upstream queue lengths for the different cable and DSL providers. Compared with downstream queues, the lengths of upstream queues are very large. Most DSL links exhibit queues of 600 ms or higher, and many cable links allow their upstream queues to grow to several seconds. Although some of the upstream queues' build-up results from the low upstream link bandwidths, the excessive lengths will negatively affect interactive traffic like VoIP when users simultaneously upload content, such as when using BitTorrent.



(a) DSL (downstream) **(b)** Cable (downstream)

(c) DSL (upstream) **(d)** Cable (upstream)

Figure 3.13: Downstream and upstream queue length in milliseconds. Some downstream queue lengths follow the recommendation for voice calls (150 ms), but most are significantly longer. The upstream queue length can be massive, especially for cable links.

3.2.4 Packet Loss

In this section, we characterize packet loss in residential broadband networks. We compare our results with the loss rates typically found in academic networks. Our tools cannot measure the access links' loss rates. Instead, we examined the packet loss rates of the Internet paths between our well-connected measurement hosts and the broadband hosts. Because the broadband access links are part of these Internet paths, our measured loss rates provide an upper bound on the broadband links' loss rates.

Do broadband links see high packet loss?

We used the small-TCP trickle probe trains to calculate the loss rates along the round-trip paths to remote broadband hosts. We sent widely spaced trickle probes at a very low rate for a week, and we measured the fraction of probes for which the broadband hosts did not respond. This experiment measures the loss rate under normal operating conditions of the network. The measured loss rate includes losses on both the upstream and the downstream paths. Note that the loss rate we measured might differ from the loss rate that would be experienced by application traffic (e.g., TCP flows) that saturates broadband links.

Figure 3.14 presents our results. We found that both cable and DSL have remarkably low packet loss rates. The loss rate is below 1% for more than 95% of all DSL and cable paths. Overall, we found that the packet loss rates for broadband access networks are similar to those observed in academic network environments [DCGN03, Pax97].

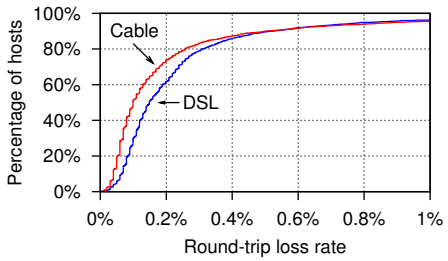


Figure 3.14: Observed round-trip loss rate for residential broadband paths. DSL and cable paths show similar loss rates. 95% of all DSL and cable hosts have loss rates of less than 1%.

We also examined how loss rates varied over the course of the week. Figure 3.15 shows our measurements for two typical providers: a DSL ISP (Ameritech) and a cable ISP (Chello). The horizontal axis shows the local time for the ISPs. The loss rates shown along the vertical axis are averaged over intervals of 120 minutes. We found that loss rates exhibit diurnal patterns with occasional spikes. Both ISPs follow similar diurnal patterns, showing lower loss rates in the early morning than in the evening.

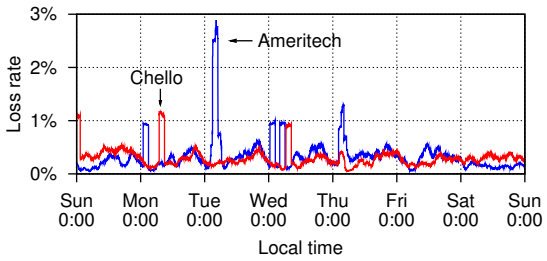


Figure 3.15: Packet loss over time. The loss rate is generally low and shows heavy diurnal variations with intermittent spikes. Note that this graph includes both upstream and downstream losses; the time axis shows local time (EDT for Ameritech and CEST for Chello).

Do ISPs use active queue management?

When packets are sent very quickly, they begin to fill up queues, and the routers must eventually drop some of the packets. The most common queue management policy is tail-drop, i.e., all packets arriving after the queue is full are discarded. More active queue management policies, such as RED [FJ93], proactively drop packets using probabilistic schemes when the queue starts to fill up but before the queue is full. Active queue management has been extensively studied, but relatively little is known about the extent to which it is deployed in practice.

We performed the following experiment to infer whether the broadband ISPs are using active queue management policies. We used the small-TCP flood to overflow both downstream and upstream links, and we used IPIDs to distinguish between losses occurring upstream and those occurring downstream [MSWA03]. For each successfully received response, we recorded the RTT, and we calculated the average loss rate over a sliding window of 40 packets. We examined the correlation between the loss rates and the corresponding RTTs. On the basis of this correlation, we can infer whether routers use tail-drop or more active queue management policies. A tail-drop policy will result in a steep increase in loss rate when the queue is full (i.e., for a large RTT value); if an active queue management policy such as RED is used, then the loss rate will increase proportionally to the RTT after a certain threshold.

Figure 3.16 shows how the loss rates increase with the RTT for two broadband hosts, one in PacBell and one in SWBell. For the PacBell host, the loss rate increases steeply around an RTT of 850 ms, which suggests that a tail-drop queue is used. The loss rate for the SWBell host shows a different trend; after 500 ms, it increases almost linearly with the RTT before stabilizing at around 85%. This behavior matches the description of the RED active queue management policy.

To quantify the extent of RED deployment in broadband networks, we tested whether the increases in RTT and loss rates are strongly correlated. If the correlation coefficient is high (≥ 0.9) beyond a threshold loss rate of 5%, we conclude that the link may be using RED as its drop policy. We did not calculate the correlation coefficient for low loss rates (below 5%) because these losses might be sporadic and not representative of the broadband router's queue policy.

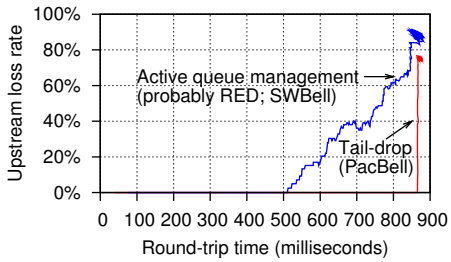


Figure 3.16: Tail-drop and active queue management. When a tail-drop queue overflows, the loss rate increases sharply. If the loss rate increases proportionally to the queue length after a threshold, then this suggests that active queue management (probably RED) is being used.

We found that 26.2% of the DSL hosts show a RED-style drop policy on their upstream queues. The three providers owned by AT&T (Ameritech, BellSouth, and PacBell) exhibit deployment rates between 50.3% and 60.5%, whereas all other DSL providers' deployment rates are below 23.0%. The partial deployment of RED-style policies within ISPs could be due to heterogeneity in the ISPs' equipment. We did not detect RED in any of the cable ISPs measured.

3.2.5 Summary

We have presented an in-depth characterization of the properties of residential broadband networks. Our analysis reveals important ways in which these networks differ from academic networks, and it quantifies these differences. We summarize our key findings below.

Allocated link bandwidths. Our results show that downstream bandwidths exceed upstream bandwidths by more than a factor of 10 for some ISPs. In contrast to popular belief, for most ISPs, the measured bandwidths matched well with the advertised rates at all times of day, and we found little evidence of competing traffic affecting their links. Although link bandwidths remain stable over the long term, they show high variation in the short term, especially for cable hosts. For some ISPs, link bandwidths change abruptly as a result of traffic shaping.

Packet latencies. Many DSL hosts show large ($\geq$ 10 ms) last-hop propagation delays. Cable hosts suffer higher jitter than DSL hosts as a result of time-slotted packet transmission policies on their upstream links. Packet concatenation on the upstream links can add another 5−10 ms of jitter in cable links.

All ISPs deploy queues that are several times larger than their bandwidth-delay products. Whereas downstream queues can delay packets by more than 100 ms, the upstream queueing delays can exceed several hundreds of milliseconds and, at times, a few seconds. Internet paths with such large queueing delays are very uncommon in academic or corporate networks.

Packet loss. Both DSL and cable ISPs exhibit surprisingly low packet loss. In fact, their loss rates are comparable to those in academic network environments. We also found that many DSL hosts use active queue management policies (e.g., RED) when dropping packets.

3.3 Implications

We believe that our observations about broadband networks' characteristics can help researchers to understand how well existing protocols and systems work in the commercial Internet. Our findings offer useful insights for the designers of future applications. To illustrate this, we briefly discuss the potential implications of our measurements for three popular Internet-scale systems.

Transport Control Protocols. Our bandwidth and latency findings have several implications for transport protocol designs. For example, protocols such as TCP Vegas [BP95] and PCP [ACKZ06] use RTT measurements to detect incipient congestion. In the presence of the high jitter found in our measurements, this mechanism might trigger congestion avoidance too early. Bandwidth-probing techniques, such as packet-pair [Kes91], could return incorrect results in the presence of traffic shaping or packet concatenation. This could be detrimental to transport protocols that rely on probing to adjust their transfer rates, such as PCP.

Network coordinate and location systems. Many IP-to-geolocation mapping tools [WSS07, GZCF06] use latency measurements to determine a host's location. The large propagation delays and high jitter found in broadband networks are likely to seriously interfere with the accuracy of these systems.

Similarly, network coordinate systems [DCKM04, NZ02] use latency estimates to assign a set of coordinates to their participating hosts. A recent study [LGS07] found that network coordinate systems do not perform well when deployed in BitTorrent networks, because RTTs between nodes can vary by up to four orders of magnitude. Our measurements explain and provide insights into these findings: BitTorrent networks typically include many residential links, which have very large RTT variations as a result of their long queues. BitTorrent traffic compounds these variations because it tends to fill up the queues.

Interactive and latency-sensitive applications. Recently, the popularity of VoIP and online games has grown considerably. Our data shows that latency-sensitive applications will be negatively affected by the broadband links' large queueing delays. Because queueing delays increase in the presence of competing traffic, these time-sensitive applications are likely to experience degraded service when they are used concurrently with bandwidth-intense applications, such as BitTorrent.

Differences to academic networks Our results also show that academic networks and broadband networks have different characteristics. Academic networks are often modeled using links with a constant bitrate and a simple tail-drop queue. Our findings suggest that this simple model may not be appropriate for broadband networks, which appear to have

advanced features such as traffic shapers, RED queues, or packet concatenation. Moreover, even if the single-queue model is used as a first approximation, it must be parametrized differently. For example, a well-known rule of thumb states that queue capacities should be set to one bandwidth-delay product. This rule does not hold in broadband networks where queue sizes appear to be much higher.

This difference also suggests that the widespread practice of evaluating new applications and distributed systems only in academic and corporate networks may be flawed. Such an evaluation strategy may not be able to predict how they will perform in broadband networks. But as many Internet applications are targeted at normal end users who increasingly use broadband networks to connect to the Internet, it becomes increasingly important to evaluate these systems also in broadband environments.

However, evaluating new systems in broadband environments raises similar problems as studying the characteristics of broadband networks: researchers and developers lack access to these networks for the evaluation of their systems at large scale. In Chapter 6 and Chapter 7 we present two approaches to tackle this problem.

In the next chapter, we will focus on a particular characteristic of residential broadband networks that has received a lot of attention recently: ISPs traffic management practices in broadband networks.

4 Glasnost: Detecting Traffic Differentiation

A confluence of technical, business, and political interests has made "network neutrality" a hot button issue [Mar06, Sto07]. The debate revolves around whether and to what extent ISPs, who own and operate data networks, should be allowed to differentiate one class of traffic from another. Many ISPs want to restrict bandwidth-hungry applications that can hurt other applications in the network. Some also want to control applications such as VoIP that reduce ISPs' ability to profit from competing services of their own. In contrast, many content providers are against traffic differentiation because it gives the ISPs arbitrary control over the quality of service experienced by users. In parallel, regulatory bodies and politicians are trying to devise policies that balance these competing concerns [The07, The08, Fed05].

As this debate rages, ordinary Internet users are often in the dark, even though they are the ones most directly affected. The information sources available to users today, such as media reports, blogs, and statements made by ISPs, are imprecise and sometimes incorrect. As a result, traffic differentiation frequently occurs without users' knowledge. However, when ISPs traffic management practices come to light, user outrage forces regulatory bodies to conduct public hearings on prevalent practices [The07, The08].

This situation led us to build and deploy a system, called Glasnost, that enables users to detect if they are subject to traffic differentiation. We make no judgment about whether traffic differentiation should be permitted by regulatory policy. Rather, our motivation is to make any differentiation along their paths *transparent* to users.

While other recent research efforts also aim to detect traffic differentiation [TMFA09, ZMZ08, KD10], Glasnost is unique in its focus on users. Instead of a broad characterization of differentiation in the Internet, our goal is to let individual users determine if they experience differentiation and quantify its impact at the time they use our system.

Our focus is on enabling individuals who are not technically savvy. This creates design constraints that are typically not present in other measurement systems. First, the bar to using the system must be low. For instance, it is undesirable to require the installation of special software on client machines, especially if such software needs privileged access. This constraint hinders our ability to collect high-fidelity data (e.g., packet traces) or to finely control packet transmissions. We must limit ourselves to coarse-grained data obtained through unprivileged client operations. Second, the results for an individual user must be accurate and simple to interpret. For example, we cannot return results that rely on inferences derived from data aggregated across users. Third, the system must evolve with ISP practices. Users would stop trusting the system if they strongly suspect the presence of differentiation but Glasnost is unable to detect it.

We designed Glasnost to satisfy these constraints. The result is a system that is effective and easy to use. A user can detect differentiation by simply pointing her browser to a Web page. The browser downloads and runs a Java applet which exchanges traffic with our measurement server. The client-server nature of our architecture helps to avoid many of the operational issues with network measurements, such as traversing NATs and firewalls,

or raising alarms in network intrusion detection systems. The traffic exchange is designed to accurately and quickly detect any differentiation. We also build a simple flow emulation tool that simplifies the incorporation of tests to detect new differentiation techniques that emerge in the Internet.

The diversity of ISP practices makes it challenging to detect traffic differentiation reliably. For instance, an ISP might employ differentiation only at specific times (e.g., in the evenings), or only under high loads, or only for flows that send too much traffic. These factors led us to design an on-demand system. Each time a user uses Glasnost, she performs an *individual* test that detects the presence of traffic differentiation for her Internet connection *at the time of the test*. This approach provides a user with a more reliable answer than what could be obtained by extrapolating the results from other testing times or other users.

Glasnost has been operational for over a year, enabling users to detect BitTorrent differentiation. During this time, more than 350,000 users from over 5,800 ISPs worldwide have used the system. Several individuals and corporations volunteered to host Glasnost measurement servers on their own infrastructure in order to allow operations on an even larger scale. We believe that our design principles have directly contributed to the success of Glasnost. To our knowledge, Glasnost is the first tool to offer highly specific and reliable traffic differentiation detection to a large number of end users.

In addition to the design and evaluation of Glasnost, in this chapter we also present a detailed analysis of BitTorrent differentiation in the Internet. Our study showed that traffic differentiation is already in widespread use in broadband access networks. We found that about 10% of our users experience differentiation of BitTorrent traffic. Also, we studied ISPs' BitTorrent differentiation policies in detail over a period of two months using data from the Glasnost tests. We found, for instance, that it is more common for ISPs to differentiate against file uploads than downloads and to differentiate throughout the day rather than only during peak hours.

4.1 Background

In this section, we provide background on the network neutrality debate and discuss different types of traffic differentiation.

4.1.1 Network Neutrality

The term "network neutrality" means that ISPs, i.e., network operators, should handle all Internet traffic in the same, neutral way. This forbids ISPs from degrading network performance for particular applications and services, or from allowing service providers to pay a fee to receive preferential service and thus better performance than their competitors. While the recent network neutrality debate seems to suggest that traffic management is something new that needs regulation, Crowcroft [Cro07] points out that the Internet has always had mechanisms intended to differentiate network traffic. For instance, policies used in inter-domain routing typically differentiate traffic based on the source and destination IP address. Also, firewalls to block unwanted traffic and proxies to load-balance traffic are widely used. Such mechanisms are beneficial in many scenarios, and have thus been in use since the early days of the Internet.

Today, traffic management policies are often driven by business interests. However, ISPs tend to not reveal their exact policies or their motivation for implementing them in the first place. As a result ISPs' policies and their incentives to deploy traffic management are often not known to the public. Nevertheless, there are three commonly mentioned motivations for deploying traffic management policies in ISPs' networks.

One motivation is network congestion. In broadband networks, there is anecdotal evidence that ISPs tend to over-subscribe the network links connecting their customers to the rest of the Internet, thus cutting costs. This usually works well and provides all users with good bandwidth performance, since only a few users use their Internet connection extensively. However, with an increasing number of users consuming large amounts of content over the Internet, including large software downloads, video-on-demand services, and file-sharing applications, networks can become congested [Can08a]. Thus, traffic management can help to distribute the available bandwidth fairly amongst consumers, avoiding an overload which would result in decreased performance for everyone.

Another motivation for deploying traffic management is that bandwidth costs continuously increase with the increase in end user bandwidth usage. For example, BitTorrent [Coh08] is a popular P2P file-sharing protocol that accounts for a large fraction of the data bytes sent over the Internet [SM09]. The resulting increase in Internet traffic is raising the cost of transit for ISPs, many of which are selling flat-rate plans with unlimited Internet access to their customers. Hence, it is not surprising that an ISP would implement strategies to reduce the amount of BitTorrent traffic generated by its customers.

Finally, differentiating traffic can help to improve service quality. For example, VoIP services are very sensitive to latency variations. ISPs can configure their network equipment to handle VoIP traffic preferentially to ensure it is not affected by other traffic, thus guaranteeing good voice quality.

Recent reports of ISPs blocking or shaping traffic of particular popular applications [Top07] have sparked an intense and wide-ranging policy debate on acceptable ISP traffic management practices and network neutrality between ISPs, consumer advocacy groups, website operators, and government agencies. As a result, network neutrality has recently moved into the focus of legislative and regulatory bodies.

In 2005 the FCC, the telecommunication regulator of the USA, published a policy statement on network neutrality [Fed05]. This policy lists four principles for an open Internet consumers should be entitled to: (1) access to all legal Internet content, (2) the ability to use any service and run any applications they want, (3) the ability to connect any devices to the network, and (4) fair competition among ISPs, application providers, service providers, and content providers. Similarly, the European Commission issued a statement that they will monitor the implementation of national policies as well as the market and technological developments regarding network neutrality in Europe. Based on the observed situation, the European Commission will decide whether the situation requires regulatory interventions to ensure network neutrality [The09b]. And in 2009, the US Congress passed the Recovery and Reinvestment Act [The09a], which announces that the US government will make substantial investments in ISPs' broadband infrastructures, with the stipulation that ISPs receiving funds must follow the FCC's network neutrality policy.

4.1.2 Traffic Differentiation

Traffic differentiation refers to an ISP treating the packets of one flow differently than those of another flow. Note that our definition of differentiation does not include traffic shaping that affects all flows of a user, e.g., restrictions that affect users after they exceed their network usage quota. Based on information published by ISPs, researchers, and equipment vendors [ipo09, DMHG08, Can08b], we characterize traffic differentiation along two dimensions.

1. Identifying specific flows. To treat flows differently, ISPs must distinguish the packets of one flow from those of other flows. Flows can be distinguished using two techniques:

(a) **Source or destination identification.** The address information carried in the IP header is used to determine how an ISP treats a flow. For example, universities routinely rate-limit traffic to and from their student dorms.

(b) **Deep packet inspection (DPI).** DPI is used to identify the application that generated a flow. For example, P2P applications such as BitTorrent are identified by scanning the packet payload for specific P2P protocol messages. Also, the transport protocol header can be inspected to identify application flows based on typical port numbers or other transport protocol identifiers.

2. Manipulating flows. There are a number of standard techniques to shape flows listed below. Thereby, flow manipulation can target different properties of a flow, such as throughput or packet latencies. Remember that not all flows are shaped — only the flows identified previously as belonging to a particular application or host.

(a) **Blocking.** A flow is terminated either by blocking its packets or by injecting a connection termination message (e.g., sending a TCP FIN or TCP RST packet).

(b) **Deprioritizing.** Routers can use multiple priority queues when forwarding packets. ISPs can use this mechanism to assign differentiated flows to lower priority queues, to increase packet latencies of particular application flows, and to limit the throughput of certain classes.

(c) **Packet dropping.** Another possibility is to drop a flow's packets using a drop rate that is either fixed or variable.

(d) **Modifying TCP advertised window size.** ISPs can lower the advertised window size of a TCP flow, prompting a sender to slow down.

(e) **Application-level mechanisms.** ISPs can control an application's behavior by modifying its protocol messages. For example, transparent proxies [Vel08] can redirect HTTP or P2P flows to alternate content servers.

What kinds of traffic differentiation does Glasnost detect?

Our current implementation of Glasnost detects traffic differentiation that is based on deep packet inspection. For residential access ISPs, which are our primary focus, this is much more common than differentiation based on IP addresses [Can08b, Com08b].

Instead of inferring differentiation by finding evidence of particular manipulation mechanisms, Glasnost detects the presence of differentiation based on its impact on application performance. The reason for this is that detecting particular differentiation mechanisms reliably is hard as not all of them are directly observable. Also, there often can be multiple explanations for observed performance degradation. For example, a high loss rate might be due to an ISP dropping packets, or due to network congestion. An exception to this is blocking, which is directly observable from endpoints since packets (e.g., TCP FIN or TCP RST packets) are injected into flows. The presence of these injected packets allows for rather accurate detection. In Section 4.5 we describe in detail how we detect traffic differentiation in general, and blocking in particular.

4.2 Design Challenges and Requirements

Building a system that lets ordinary Internet users determine if they are affected by of traffic differentiation poses several challenges.

The focus on end users and the nature of our measurement places certain requirements on our design that are typically not present in other measurement systems. We distill these requirements into three design challenges. These challenges dictate that the system must be simple enough for any Internet user to use, its inferences must be robust and simple to interpret, and it must keep pace with network policies as they evolve. While our focus is traffic differentiation, the design challenges we identified for Glasnost are more general and apply to many measurement systems that want to attract a large number of users.

4.2.1 Challenge #1: Low Barrier of Use

A measurement system that wants to attract a large number of users must have a low barrier of use. Although this principle appears obvious at first glance, adherence to it from the start is the key to a system's success.

To keep the barrier of use low, we identified four design requirements: First, because most users are not technically savvy, we must make the interface simple and intuitive. Second, we cannot require users to install new software or perform OS administrative tasks. Many network measurement techniques require installing drivers (e.g., the WinPcap library for Windows) or running privileged code (e.g., raw sockets) on users' machines. Such code can provide detailed, low-level data (e.g., packet traces) that simplifies the measurement task. But, in our experience, users are often unwilling to use systems with such requirements. For example, one of our earlier attempts required users to run code with administrator privileges on their machines and to leave a port open in their firewalls and NATs. These obstacles greatly limited adoption; we attracted fewer than fifty users. Third, because many users have little patience, the system must complete its measurements quickly. Fourth, to incentivize users to use the system in the first place, the system should display per-user results immediately after completing the measurements.

4.2.2 Challenge #2: Measurement Accountability

As researchers, we are used to interpreting complicated measurement results, through the prism of the experiments' methodology. Our training and experience prepares us for this

task. For instance, consider the results of an experiment that infers path capacities in the Internet. As the measurement method can be affected by transient noise, the researchers will know that the answer computed along an individual path cannot be trusted, while the answers can be aggregated to provide an overall view. An ordinary user interested in the capacity of her own path might not be in a position to make that distinction.

In the case of detecting traffic differentiation accurate and clear test results are even more critical due to the controversial nature of traffic management in the Internet: people are still debating whether it is legal for an ISP to employ traffic management. If people were to falsely interpret results as their ISP performing traffic differentiation when in fact it is not, the system would quickly lose credibility. In fact, in the past there have been instances when some widely publicized studies have mistakenly accused ISPs of using policies they never deployed [BMGS08, Sny08].

Thus, keeping measurements accountable must have high priority. To achieve this, we identified three design requirements: First, the test to detect differentiation should be direct and any factors that add uncertainty should be removed if possible while designing the test. The performance of an Internet flow can be affected by many confounding factors. This includes which operating system is used, especially its TCP/IP networking stack, and, of course, its configuration. Additionally, directly using application clients against the same endpoint is problematic. The short-term throughput of such "natural" flows can differ because of differences in packet sizes and burstiness. Finally, we have to consider transient noise, e.g., as caused by background traffic.

When using passive measurement tools it is not possible to avoid these factors most of the time. Thus, these tools are required to account for a large number of confounding factors in their inference that is a complicated task, often leading to inaccurate results.

Active measurement tools on the other hand can avoid most confounding factors. Running all measurements from the same host removes factors like OS and networking stack and having full control over the traffic that is sent to measure the performance of a link eases the analysis. In this case, the only remaining confounding factor is transient noise that can be dealt with using simple techniques like repeating measurements multiple times.

Second, because not all uncertainty can be removed from the inference, the result presented to the user must be conservative, with a near-zero false positive rate. In the context of traffic differentiation, a false positive means that the system falsely claims that the user is experiencing traffic differentiation when in fact it is not. Minimizing false positives is challenging because it comes with a price — an increase in the false negatives rate. This trade-off is inherent. Third, we must be prepared to provide the data and the evidence behind our inferences when requested.

4.2.3 Challenge #3: Easy to Evolve

To be useful over time, a system that wants to detect traffic differentiation must be able to evolve along with policies in the network since ISPs continuously adjust their traffic management policies. For example, in 2008, Comcast blocked BitTorrent uploads for some of its customers [Com08b]. Several months later, they started replacing this practice with less severe forms of differentiation [Com08a]. In fact, our recent measurements indicate that BitTorrent traffic blocking is very rare today while it was widespread in 2008. In consequence, in an evolving environment a system with a fixed set of capabilities will have

a limited shelf life and will become less relevant with time. This makes the case for a system that provides a way to evolve with the network.

ISPs might target new applications in the future or change their traffic shaping mechanisms, which might manipulate flows in new ways. A detection system should be extensible, i.e., it should be easy to add new tests to detect traffic differentiation of popular new applications or to use new techniques in order to detect differentiation based on new shaping techniques. Also, ISPs could start to whitelist traffic from measurement servers if they do not like the transparency the system provides to their users, thus trying to evade detection. A successful system must be aware of this problem and must find ways to minimize whitelisting.

4.3 The Glasnost System

We now present the design of Glasnost based on the requirements outlined above.

4.3.1 System Architecture

Glasnost is based on a client-server architecture. Clients connect to a Glasnost server to download and run various tests. Each test measures the path between the client and the server by generating flows that carry application-level data. This data is carefully constructed to detect traffic differentiation along the path.

Figure 4.1 presents a step-by-step description of how clients measure their Internet paths. A client first contacts a central webpage at `http://broadband.mpi-sws.org/transparency/glasnost.php` that redirects to a Glasnost measurement server. This dy-

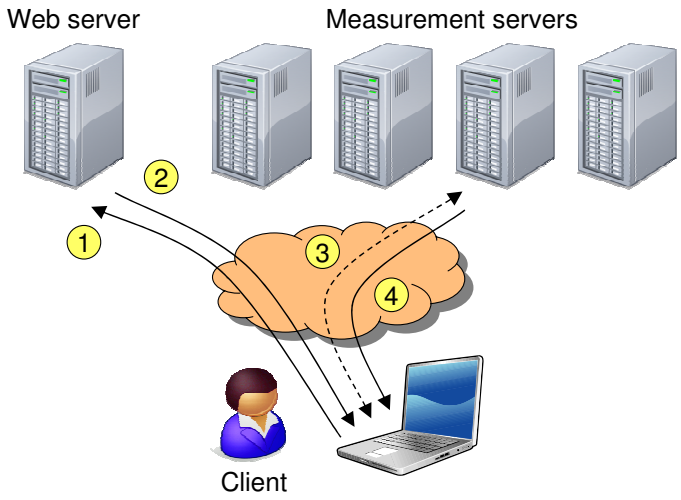


Figure 4.1: Overview of the Glasnost system. (1) The client contacts the Glasnost webpage. (2) The webpage returns the address of a measurement server. (3) The client connects to the measurement server and loads a Java applet. The applet then starts to emulate a sequence of flows. (4) After the test is done, the collected data is analyzed and a results page is displayed to the client.

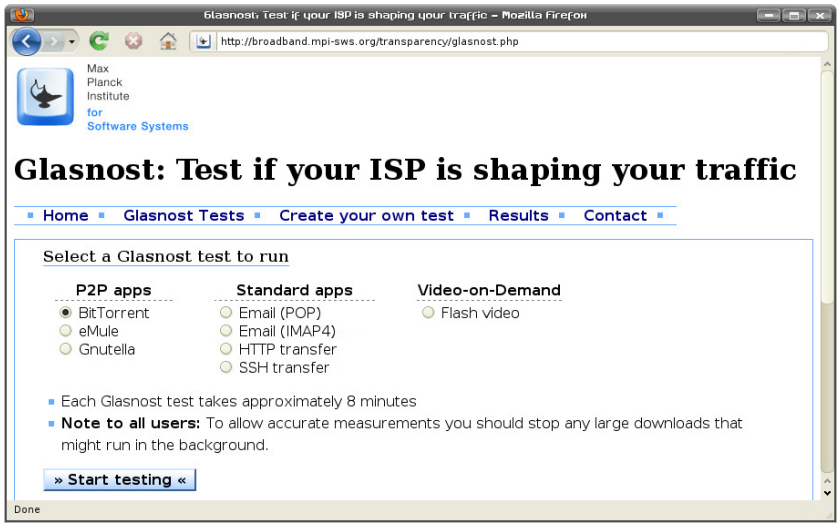


Figure 4.2: The Glasnost web interface.

namic redirection enables load balancing across measurement servers and makes it easy to incorporate new servers by adding them to the redirection list.

After the client is redirected, the measurement server presents a simple interface to the user. As shown in Figure 4.2, the user first chooses from a list of Glasnost tests and then clicks the "Start testing" button. The client's browser downloads a Java applet that starts exchanging packets with the server. Glasnost tests typically complete within 8 minutes, ensuring a timely response to end users.

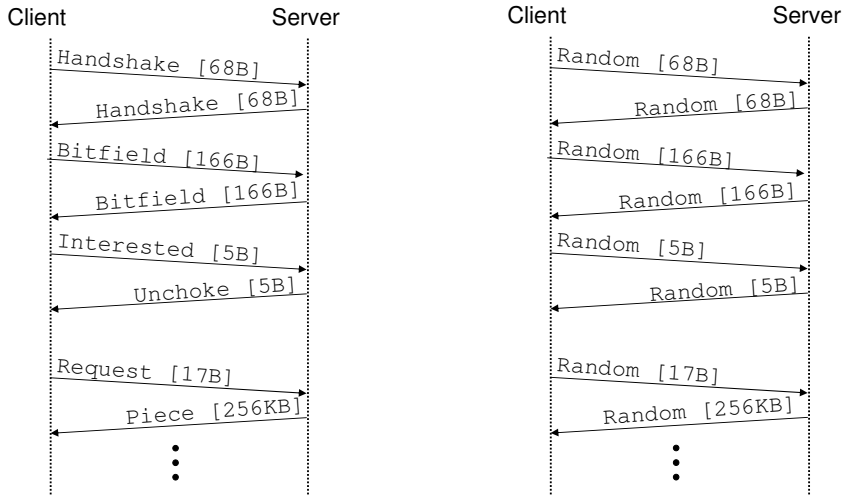
Glasnost tests have a low barrier of use; any end user with a standard web browser and a (often pre-installed) Java plugin can test their Internet connection in a few minutes time without installing any additional software. Also, the tests require no user interaction as all tests run fully automatically.

We elaborate on the Glasnost measurement tests next.

4.4 Emulating Application Traffic

Glasnost tests are designed to detect whether ISPs along the path between a client and a server are treating flows belonging to two different applications differently. The key primitive behind the Glasnost measurement tests is the *emulation* of a pair of flows that are identical except in one respect that we suspect triggers differentiation along the path. Comparing the performance of these flows helps determine if differentiation is indeed present.

Figure 4.3 shows two flows designed to detect whether differentiation based on BitTorrent protocol content is present along a path. The exchange on the left is the first flow. The client opens a TCP connection to the measurement server and starts exchanging packets that implement the BitTorrent protocol: the packet payloads carry BitTorrent protocol headers and content. The exchange on the right is the second flow (i.e., the reference flow). The client opens another TCP connection and performs the same packet exchange, but the packets contain random bytes instead of BitTorrent headers or data. An



(a) BitTorrent flow **(b)** Reference flow

Figure 4.3: A flow pair used in Glasnost tests. The flows are identical except for packet payloads, which enables us to detects differentiation that targets flows with BitTorrent content.

ISP that differentiates against BitTorrent based on protocol messages would rate-limit or block only the first flow. Thus, significant differences in the flows' performance is likely to be caused by the differences in their payloads and lets us detect whether differentiation is present along the path. Transient noise can also lead to differences in flows' performance; we describe in the next section how we handle noise.

During the test, the measurement server records a packet-level trace of all emulated flows and the client applet records ancillary information including exceptions caused by network errors. Once the test ends, the client uploads the recorded information to the server. The server analyzes this information together with the packet-level traces collected on the server-side and shows the findings to the client. The collected packet-level trace and the uploaded data enable us to reproduce our inference at any time.

Glasnost's emulation methodology leads to measurement robustness. As Figure 4.3 shows, application-level data is the only difference between the two emulated flows. The two flows traverse the same network path and have the same network-level characteristics, such as port numbers, packet sizes, etc. In contrast, other techniques such as passive measurement have many factors that differ across measured flows. Correctly accounting for all such differences is more challenging.

Another benefit of active measurement is the ability to carefully control the measurement test. For example, we can repeat flows with different payloads or port numbers. This ability allows Glasnost to precisely identify the specific factors that trigger differentiation.

In the next section, we describe our measurement test in more detail and how we make it robust to transient noise. In Section 4.6 we describe how we use a trace replay based tool for constructing measurement tests in order to ensure that the system is easy to evolve.

4.5 Detecting Traffic Differentiation

Glasnost currently detects two types of traffic differentiation: blocking and throttling. Blocking means that a flow is terminated by injecting special protocol messages, such as TCP RST packets. Throttling on the other hand means that a flow received less throughput than a reference flow. As throughput measurements can be affected by transient noise, this inference is more involved and requires careful analysis of the measurement data to avoid false positives.

As described earlier, Glasnost emulates a pair of flows and determines the presence of traffic differentiation by comparing their performance. Our hypothesis was that the identification can be based on three flow characteristics: the TCP port number of the flow, the specific application protocol messages in the flow, and the direction of the flow. Thus, Glasnost tests vary the following three parameters:

- **TCP port:** Half of the flows use the default port of the tested application. The others use a port not associated with a specific application or protocol. We call this a *neutral port*.
- **Direction:** Half of the flows transfer content downstream (from the server to the user's host), while the others transfer content upstream (from the user's host to the server).
- **Protocol:** Half of the flows contain real application messages. The others contain messages of the same size and in the same order, but filled with random bytes.

This results in eight possible combinations.

Glasnost examines the results for each direction separately. If all tests using one port (e.g., the default application port) measure significantly lower performance than tests on another port regardless of whether application data or random data was sent, Glasnost reports *port-based traffic differentiation* in the tested direction. Similarly, if all the tests in one direction that use application messages (regardless of the port on which the test runs) get significantly lower performance than the tests with random data, Glasnost reports *application-based traffic differentiation* in that direction.

We describe how we infer of blocking and throttling in the following section.

4.5.1 Blocking of Application Traffic

Glasnost can detect whether middleboxes in the network are inserting forged RST packets to interrupt application flows. Therefore, Glasnost analyzes the server trace along with any Java exceptions seen by the user-side applet for each flow. A flow is considered to have been torn down by a forged RST packet only when *all* of the following three conditions hold:

1. **An IOException with a specific set of messages is seen by our applet.** This indicates that an error was observed with the TCP connection on the user side. Glasnost looks for the messages "Connection reset by peer" or "An existing connection was forcibly closed by the remote host" in the IOException, which indicate that the host has received a RST packet[1].

[1] Unfortunately, parsing the exception message is the only way to distinguish the reset of a TCP connection from other causes in Java.

2. **The server's packet trace contains at least one incoming RST packet.** This RST packet caused the connection to be torn down at the server.

3. **The server's packet trace contains no outgoing RST packets before a FIN or RST packet is received.** Once the server receives a FIN or RST packet, the connection is torn down. Thus, any subsequent data packets received on the connection will be naturally responded to with RSTs.

The presence of all three conditions strongly indicates that a forged RST caused the flow to be torn down. The first two conditions indicate that a RST was received at both the server and the user's host. While we cannot say for sure that the user's host received a RST packet (as we do not have a packet-level trace from the host), we look for IOExceptions with messages that are caused by the receipt of a RST packet. The third condition indicates that the server did not initiate the connection tear-down (in other words, it received either a FIN or a RST before it sent any RSTs). Thus, Glasnost detects forged RSTs by looking for flows (1) which were torn down by a RST received at the user's host and/or server and (2) which contain no RSTs sent by the user's host or the server before the connection was torn down.

Other application traffic blocking detection tools

There have been other approaches to detect BitTorrent blocking in the Internet. To detect blocking of BitTorrent transfers, the network monitor plugin for the popular Vuze BitTorrent client [Sny08] reports the number of aborted connections. Since the plugin does not correlate observations from both endpoints of an aborted flow, it cannot reliably determine whether the transfer termination is due to blocking or another reason. In contrast, our methodology is more reliable as it employs active measurements to detect BitTorrent blocking.

The Electronic Frontier Foundation's "Test Your ISP" project [Ele08] offers instructions for tracing a BitTorrent transfer and checking for forged packets. This method requires access to two hosts in different ISPs and involves the use of tools like Wireshark, which is beyond the capabilities of most end users. Glasnost, on the other hand, was designed to allow even lay users to test their access links for blocking of application traffic.

4.5.2 Throttling of Application Traffic

When comparing the performance of a pair of flows, we must ensure that their difference is indeed due to the differences in their content and not due to some changes in the test environment. Our measurement tests are constructed in a way that eliminates all major confounding factors except one – transient noise due to interference from cross-traffic (background traffic) along the measurement path. Here we discuss techniques to robustly detect traffic throttling in the face of transient noise.

The primary challenge in this task stems from the fact that the noise can vary at small time-scales. Thus, two flows can be affected differently even if run back-to-back. For example, in one egregious case, we found that the throughput of two back-to-back flows differed by a factor of three even though the flows were identical. A simplistic detection method will mistakenly detect differentiation in this case. It might appear that the differential impact of noise could be reduced by running the flows simultaneously. But

we find that such a setup is even worse than running measurements back-to-back because the test traffic itself can overload the path.

Our basic strategy for robust detection is to run each flow type multiple times. We use the variance in the performance of the flows of the same type to identify paths that are too noisy to enable reliable detection. For the remaining paths, we can confidently detect differentiation by comparing the flows of different types. We first describe how we apply this strategy when tests are run long enough that we do not have to worry about having too little data. As we found that many users are too impatient to run long tests, we adapted our strategy to tests that run for a shorter duration. Shorter tests decrease the robustness of our detection technique to transient noise; we account for this during the data analysis.

We describe our method using throughput as the measure of flow performance, since it is of primary interest to many applications and is the target of many ISPs looking to reduce their network load. Because of TCP dynamics, throughput is directly affected by any differentiation that impacts flow latency or loss. Note that our method can be extended to other measures of performance such as jitter.

Filtering tests affected by noise

To detect the level of transient noise, we repeat the runs of the two flow types multiple times back-to-back. Unlike active ISP differentiation, transient noise does not discriminate based on flow content; it would not affect multiple runs of the same flow type and can be detected by comparing the performance of the repeated flows.

To understand transient noise patterns and the extent to which they affect flow throughput, we configured our Glasnost deployment to run a BitTorrent flow and a reference flow with random bytes, five times each. The runs of the two flow types were interspersed and each flow lasted for 60 seconds to allow sufficient time for TCP to achieve stable throughput. Over a period of a month, we collected measurements of 3,705 residential broadband hosts, 2,871 in the upstream and 834 in the downstream direction.

We compared the throughput obtained by the five runs of each flow type with each other. Our analysis of the maximum, median, and minimum throughput reveals the four distinct patterns shown in Figure 4.4, corresponding to four different cross-traffic levels.

1. **Consistently low cross-traffic:** all throughput measurements belonging to the same flow type fall within a narrow range (i.e., the minimum is close to maximum).

2. **Mostly low but occasionally high cross-traffic:** a majority of throughput measurements are clustered around the maximum but a few points are farther away (i.e., the maximum and the minimum are far but the median is close to the maximum).

3. **Highly variable cross-traffic:** the throughput measurements are scattered over a wide range (i.e., the maximum and the minimum are far apart and the median is far from both).

4. **Mostly high but occasionally low cross-traffic:** a majority of throughput measurements are clustered around the minimum but a few measurements are farther away (i.e., the maximum and the minimum are far apart but the median is close to the minimum).

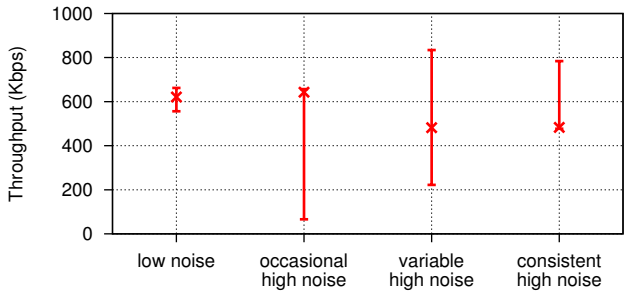


Figure 4.4: The four classes of noise we distinguish in our analysis. The graph shows minimum, median, and maximum throughput for each class.

Our categorization of the level of cross-traffic in each case is based on two key observations about the nature and impact of cross-traffic. First, cross-traffic only lowers throughput and never improves it. Thus, when a majority of throughput measurements are close to the minimum but far apart from the maximum (as in category 4 above), it is more likely that the noise-free minimum throughput is closer to the maximum than to the minimum.

Second, cross-traffic is unlikely to be consistently high over a long period of time. In theory, measurements in category 1 above could be explained by consistently high cross-traffic. But, this would require the cross-traffic to remain high and consistent (without changing) over the duration of the entire experiment, which is ten minutes. We believe that this is unlikely.

Based on this study, for robust detection of differentiation, we discard all tests where a majority of flows are affected by high noise (i.e., categories 3 and 4). For these tests, we cannot determine whether the difference in throughput is caused by differentiation or transient noise. We analyze only the remaining tests, for which a majority of runs experience low noise (i.e., categories 1 and 2).

To help determine which tests belong to the predominantly low noise category, we plot the difference between maximum and median throughput as a percentage of maximum throughput in Figure 4.5. We found that for a large majority of tests (85.2% of upstream tests and 75.7% of downstream tests) the median throughput is within 20% of the maximum. The difference between median and maximum throughput is considerably larger

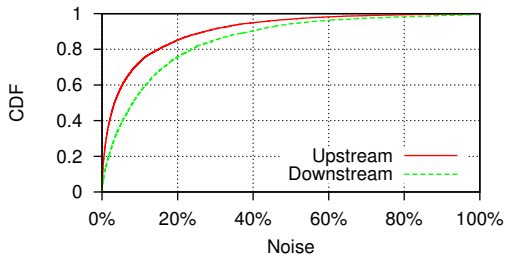


Figure 4.5: Noise in our 3,705 sample dataset. 85.2% of upstream flows and 75.7% of downstream flows have less than 20% false positives.

for the remaining flows. We thus use the 20% difference between median and maximum throughputs as a threshold to discard tests that are significantly affected by noise. Next, we describe how we detect traffic differentiation within the remaining tests.

Detecting differentiation in low-noise tests

To detect traffic differentiation among tests that are identified as low noise, we compare the maximum throughput of each flow type. Our decision to use the maximum is based on the observations that (a) in low-noise cases, most measurements lie close to the maximum throughput and (b) because noise tends to lower throughput, the maximum throughput is a good approximation for what the flows would achieve without cross-traffic.

We infer that the two flow types are being treated differently if the maximum throughput of one differs from that of the other by more than a threshold δ. Selecting a good δ involves a tricky trade-off. With high values, we cannot detect differentiation unless the impact on throughput is high. For instance, with δ=50%, we would only detection differentiation that halves the flow throughput. Thus, high values raises the false negative rate. On the other hand, with low values of δ (say 5%), we risk false positives, i.e., declaring that ISPs are employing traffic differentiation while they actually do not.

To understand how the false positive rate varies with δ, we selected 302 test runs from users using ISPs that we know do not differentiate. Figure 4.6 plots the percentage of tests that are falsely marked as being differentiated for different threshold values. The plot shows an interesting trend: the false positive rate drops steeply until δ reaches 20%. Beyond this threshold, there are still a handful of hosts (0.58%) that pass our noise tests but are falsely marked as differentiated. To avoid any false positives, we would need to raise the threshold to 40%, which increases the false negative rate.

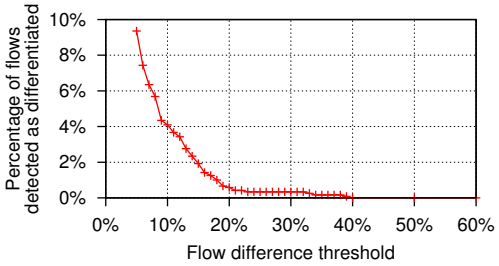


Figure 4.6: Evaluating the throughput difference threshold. Thresholds smaller than 20% tend to produce a significant number of false positives.

We thus set δ to 20%. With this value we maintain a low false positive rate (under 0.6%), but we fail to detect differentiation that reduces a flow's throughput by less than 20%. We consider this an acceptable trade-off.

Other application traffic throttling detection tools

The recent public interest in network neutrality and traffic management policies of ISPs has inspired other work in detecting traffic differentiation in the Internet.

NANO [TMFA09] uses causal inference to infer the presence of traffic performance degradation. NANO relies on a vast amount of passively collected traces from many users to infer if traversing a particular ISP leads to poor performance for certain kinds of traffic. In contrast, Glasnost uses active measurement and a simple head-to-head comparison of two flows to be able to quickly tell a user if they face traffic differentiation, without relying on other users.

NetPolice [ZMZ09] (previously named NVLens [ZMZ08]) compares the aggregate loss rates of different flows to infer the presence of "network neutrality violations" in backbone ISPs. Their methodology allows them to reason about which ISP along the network path is responsible for differentiation. To perform measurements, they require control over only a single end-point and employ TTL tricks to probe backbone networks at large scale. For detection, they compare the loss rates of two different application flows with each other. Compared to Glasnost, this methodology appears to be less robust as differences in loss rates could also be caused by diverse characteristics of the measurement flows or by side-effects of the single-sided measurement technique, such as rate-limiting of probe responses.

Finally, DiffProbe [KD10] is a probing method to detect whether traffic differentiation based on active queue management (AQM) (i.e., RED or weighted fair queueing) is deployed in the network path. Like Glasnost, this technique detects traffic differentiation by comparing pairs of flows. DiffProbe complements Glasnost as it allows the detection of latency-based differentiation as well as the identification of which AQM technique is being used. Note that Glasnost is also able to detect differentiation based on AQM if it affects application throughput.

4.5.3 User Impatience with Long Tests

As described above, we configured Glasnost to run a pair of one-minute-long flows five times, resulting in a total test time of 10 minutes. The tests we originally deployed was 20 minutes long as it also detected whether the differentiation was based on port number or payload. While this test configuration enabled us to detect differentiation with high confidence, we noticed a practical problem: a considerable fraction of users were aborting the tests before completion.

Figure 4.7 shows how long users keep their Glasnost test running. The plot for 20 minute long tests shows an alarming decline in the percentage of users as the test progresses. Only

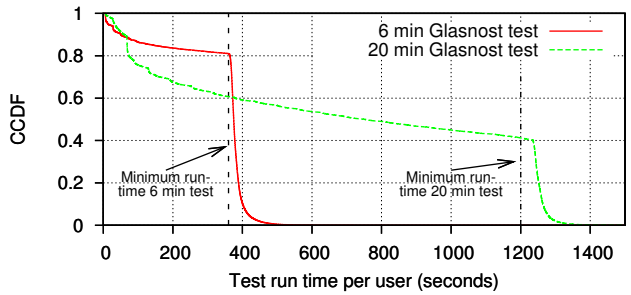


Figure 4.7: Duration users run the Glasnost test. The longer the test runs the more users abort it before the test has finished.

40% of the users stay till the end and nearly 50% aborted their tests within the first 10 minutes. The sudden drop near the 20 minute point corresponds to successfully completed tests.

Our results show that users are impatient. Most are not willing to use tests that take more than a few minutes. To confirm this, we reconfigured Glasnost to use shorter-duration tests. We reduced the number of times we repeat each flow type to two (from five), and we decreased the duration of each flow to 20 seconds (from 60 seconds), which is still sufficient for TCP to exit slow-start and achieve stable throughput. We bundled the tests for both upstream and downstream directions, and the resulting test takes 5.33 or roughly 6 minutes.

Figure 4.7 also shows how long users keep the 6 minute Glasnost test running. More than 80% of the users stay till the end, confirming that shorter tests on the order of a few minutes are more effective at retaining users.

Detecting differentiation with short tests

Short duration tests gather fewer measurements than longer ones, which poses an additional problem for detecting differentiation robustly. To estimate the impact of this reduction in data on detection accuracy, we considered data from the longer tests for which we had a result, i.e., for which we knew whether or not the ISP was differentiating. We pruned the data to include only what would be gathered by the short test and ran our analysis on the pruned data. We compared the results from this shorter test data with those obtained before.

We found that nearly 25% of the long tests were discarded as too noisy after pruning, although we were able to successfully analyze them before. We estimated the false positive rate (i.e., when the long test found no traffic differentiation but the short test did) to be 2.8% and the false negative rate (i.e., when the long test found traffic differentiation but the short test did not) to be 0.9%.

We also found that we can achieve a four-fold reduction in the false positive rate, to 0.7% (which is comparable to the false positive rate of long tests), by raising the δ threshold from 20% to 50%. While this increases the false negative rate to 1.7%, we consider it an acceptable trade-off.

4.5.4 Limitations

We now discuss two limitations of Glasnost and how we address them in our system design.

Glasnost cannot determine at which point along the path traffic differentiation is applied, thus which ISP is responsible for shaping traffic. This can be a problem since a typical Internet path between a host and our measurement servers is likely to cross multiple ISPs. It is the subject of ongoing work to develop techniques to pinpoint the location of the traffic management equipment using network tomography.

Glasnost's centralized architecture makes it possible for ISPs to avoid detection by whitelisting the Glasnost servers. This is unlikely to have affected the data we present in this chapter, but it may become a problem now that Glasnost is more widely known. To minimize the impact of ISPs whitelisting Glasnost servers, we made our server code publicly available. Anyone can setup Glasnost on a well-provisioned server that can then be used by users. Making our code publicly available allowed other Glasnost servers to

appear on the Internet, making it hard for ISPs to evade detection. We also deployed Glasnost on a diverse set of servers distributed across many locations. For hosting Glasnost, we use Measurement Lab (M-Lab), an open platform for researchers to deploy Internet measurement tools. As of this writing, M-Lab provides 46 servers at 12 sites in the USA and Europe and is still growing.

However, deploying Glasnost on an increasing number of servers is not foolproof; a determined ISP may choose to stay up-to-date with the list of Glasnost servers. However, we believe that not many ISPs would be willing to invest significant effort in evading detection. As much as an ISP would like to conceal its traffic management practices from the public, denying those practices or blatantly attempting to hide them is risky. For example, Comcast blocked BitTorrent transfers [Com08b] while proclaiming innocence to the public and the FCC. When its practices were eventually revealed, it was fined by the FCC and the highly critical media coverage damaged its reputation.

4.6 Facilitating New Test Construction

Manually implementing Glasnost tests for a new application is a laborious and error prone task. It requires detailed knowledge of the application's protocols and their popular implementations. This creates a high barrier for new test construction, making it difficult to evolve Glasnost tests to keep pace with the evolution of ISPs' network policies.

To make it easy to construct new traffic differentiation tests, we built a tool called `trace-emulate` that automates most of the test construction process. In this section, we describe how `trace-emulate` works and how Glasnost users can use it to create new tests for their chosen applications. We also present a validation of the accuracy of tests constructed by `trace-emulate` using the open source DPI engine of a commercial traffic shaper [ipo09].

The `trace-emulate` tool is an adaptation of existing trace replay methods. Existing TCP replay tools were already capable of generating new flows from packet-level traces, either maintaining similar network-level properties, such as latency and bandwidth [CHC+04], or even using the same (and not necessarily known) application-level protocol [CPC+08]. For Glasnost, however, we are interested in simply replaying traces of flows and not to automatically generate new flows. Thus, a simpler approach suffices.

Our `trace-emulate` tool automatically generates a new Glasnost test from the packet-level trace of an application. It extracts the essential characteristics of the application flows. These include packet sizes and packet payloads, the order of packets with protocol messages, and the inter-packet timing.

The test configuration that `trace-emulate` outputs is then used by the Glasnost Java applet to run the test. When run against the server, the applet exchanges two flows. The first flow has the same characteristics as the original trace. For example, assume that in the original trace the client performed the following operations: (1) sent packet A, (2) received packet B, (3) sent packet C after t seconds. These operations occur in the same order and relative times in the generate flow. In some cases, simultaneously preserving packet ordering and inter-packet spacing is impossible. Such cases arise when an endpoint is waiting to receive a packet that gets delayed in the network. We make the endpoint (client or the server) wait until the packet is received before continuing the emulation, even though it increases the inter-packet spacing. Our decision to preserve ordering at the

expense of spacing is motivated by the observation that ISPs often use the sequence of protocol messages to identify applications, rather than their relative timing. The second flow exchanged by the applet is a reference flow with the same characteristics as the first flow but that uses different payloads and/or ports. By default, packet payloads and port numbers in the reference flows are set randomly. However, the user uploading the trace can set the ports to specific values, e.g., the application's default port.

Our tests validated that the emulated flows generated by `trace-emulate` have the same packet sizes, payloads, and ordering as the original trace. We omit detailed results. In the next section, we validate that the tests generated by `trace-emulate` accurately capture the essential flow properties that are used by traffic management equipment to identify application traffic.

4.6.1 Validating Tests Generated by `trace-emulate`

Next we validate that our `trace-emulate` tool indeed captures the essential characteristics of a given application flow that an ISP might use to identify that application in practice. It is important to note that while ISPs might, in theory, use arbitrarily complex mechanisms, in practice they are limited to using mechanisms that can scale to at least multiple Gbps. We are therefore interested in validating `trace-emulate` against *practical* detection mechanisms used by ISPs.

As one might imagine, ISPs use traffic-differentiation solutions from third-party vendors (e.g. Sandvine, BlueCoat, and Arbor Networks) since not every ISP is in a position to build such a system from scratch. Fortunately, pressure from privacy watchdogs compelled one of these vendors – Ipoque – to release the code it uses to inspect user traffic [ipo09]; Ipoque did not release code that operates on traffic encrypted by the user. Thus, for the first time, the research community was given access to actual production code used by ISPs to detect the application that a user is running.

Having access to a commercial DPI engine does not only allow us to validate our `trace-emulate` tool with tremendous accuracy, it makes it extremely easy to do so. By inspecting the code we can determine what applications the Ipoque DPI engine can detect. We run the application and check whether the Ipoque DPI engine detects the application from the packet flow. We then use our `trace-emulate` tool to generate a Glasnost test for that application. We run the test and check whether the result is the same. If the Ipoque DPI engine detects our emulated trace as the target application that we are attempting to emulate, then `trace-emulate` has successfully captured all the essential characteristics for that application necessary for it to be detected by a commercial traffic differentiation solution.

Ipoque's detector can identify traffic from more than 90 widely-used applications broadly classified as P2P, video streaming, instant messaging, online gaming, and other applications (email, web, etc.). It took us less than 2 hours to generate Glasnost tests for 10 representative applications in all 5 of the above categories. We generated tests for eMule, Gnutella, and BitTorrent (all P2P); YouTube (streaming video); World of Warcraft (online game); IRC (instant messaging); and HTTP, FTP, and IMAP. For eMule and Gnutella, Ipoque's detector separately identifies their control and data connections; consequently, we used `trace-emulate` to generate the corresponding two tests. That we were able to generate all tests in a matter of hours is a testament to the simplicity of `trace-emulate`.

In every single case, Ipoque's DPI engine identified the test generated by `trace-emulate` as the target application. To the extent that the Ipoque DPI engine is representative of the engines of other similar vendors, we can confidently claim that `trace-emulate` captures the essential flow characteristics for applications that do not encrypt their traffic. However, we readily admit that without knowledge of how Ipoque detects applications from encrypted traffic, we cannot make any claims in that regard.

To convince ourselves that the Ipoque result holds in the real-world, we further validate `trace-emulate` against Kabel Deutschland, the biggest cable ISP in Germany. Kabel Deutschland targets P2P filesharing applications between 6pm and midnight [Röt08]; their choice of vendor for traffic shaping equipment is unknown, although it is rumored to be Sandvine. In any event, since we know their policy, validating `trace-emulate` is straightforward. We ran tests we generated for BitTorrent, eMule, Gnutella, HTTP, IMAP, and SSH from a Kabel Deutschland user, and saw whether Glasnost detected traffic differentiation.

Glasnost detected traffic differentiation for *each* of the P2P applications, and *none* of the non-P2P applications. In fact, by running the tests in both directions (downstream and upstream) and using different ports (default application port, random port), we were able to refine the policy published by Kabel Deutschland. Table 4.1 shows that P2P traffic is differentiated regardless of the port number used (i.e., based on the packet content). Next, we ran the HTTP, IMAP, and SSH tests on the ports typically used by the three P2P applications and found the flows achieved substantially lower throughput. Running the same tests on random ports resulted in normal throughput. This is precisely what one might expect if Kabel Deutschland additionally uses port-based detection, which naturally has false-positives. Regardless of whether Kabel Deutschland sought to omit mention of side effects of their differentiation policy or we have identified a misconfiguration, our finding demonstrates the value of network transparency tools such as Glasnost.

Application	Port-based	Content-based
BitTorrent	6881, down	down
eMule data	4662, down	down
Gnutella control	6346, down+up	down+up
Gnutella data	6346, down+up	down
HTTP	no	no
IMAP	no	no
SSH	no	no

Table 4.1: Using new Glasnost tests on a host connected via Kabel Deutschland. We identified instances of port-based and content-based traffic differentiation in the downstream (down) and upstream (up) direction.

4.6.2 Allowing Users to Contribute Glasnost Test

It is not feasible for us to create Glasnost tests for each of the large number of applications and possible traffic differentiation policies that are of interest to users. Hence, we decided to allow users to use our `trace-emulate` tool to create their own Glasnost tests. To create

a new test, users need to capture a packet trace of their target application using `tcpdump` and then use `trace-emulate` to create a new Glasnost test from the trace. These new tests can be uploaded to our measurement servers using the Glasnost webpage. Our interface for creating new tests is targeted not at lay users, but at advanced users who have some familiarity with capturing network traces.

We have deployed this interface only recently, and we do not yet have a lot of experience with it. However, we asked a handful of our colleagues, who are doctoral students not associated with our project, to use the interface to create new Glasnost tests: they were able to create new tests quite easily.

Allowing users to upload and run their own traffic differentiation tests using the Glasnost infrastructure raises a number of security concerns.

1. **Can uploaded user tests overload the measurement servers, maybe even affecting other tests running in parallel?** The Glasnost measurement servers only allow a limited number of users to run tests in parallel from the same measurement server. We conservatively limit the number of concurrent tests to ten per server which ensures that there are enough resources available to support all tests. We chose this low number of parallel tests to make sure that the Internet connection of the server does not become a bottleneck[2].

2. **Can the Glasnost service be abused to attack other Internet servers?** Glasnost is written to ensure that it only runs transfers between a measurement server and the host that runs the Java applet. It is not possible to configure a measurement server so that it sends traffic to arbitrary Internet hosts.

3. **Can users upload tests that claim to test for certain application differentiation, but in fact do not, thus producing wrong test results?** While we allow users to upload and share new tests, we cannot guarantee that these tests emulate realistic and meaningful application traffic. However, on the Glasnost frontpage we provide a number of tests that were reviewed and verified by us to ensure they test for differentiation of the application they claim to test for.

4.7 Large-scale Study of Traffic Differentiation in Broadband Access Networks

We now present the first large-scale study of traffic differentiation in broadband access networks using Glasnost.

4.7.1 Deployment of Glasnost

We deployed Glasnost publicly on the Internet on March 18th, 2008 and it has been operational ever since. Initially, Glasnost was deployed on eight local servers at our institute. Over the last year, the number of servers has grown to eighteen as other users and corporations have volunteered to host Glasnost measurement servers. Eleven servers are in Europe, and the rest are in the USA (three on the west coast and four on the east coast). We anticipate that the number of measurements servers will grow in the future.

[2]The Glasnost measurement servers are currently connected with at least a 1 Gbps link to the Internet.

In the beginning, we developed our system and refined its techniques as we chose to focus on single application. We picked BitTorrent because it is a very popular P2P file-sharing protocol that is widely suspected of being manipulated by ISPs [Azu09]. However, our differentiation detection techniques are not specific to BitTorrent and can be applied to other applications as well. While we recently deployed tests for other applications, an overwhelming majority of the data we have gathered is from BitTorrent tests. Thus, most of the discussion here is limited to results about BitTorrent differentiation.

Measurement Lab

In fall 2008, Google, PlanetLab, the New America Foundation, and academic researchers started M-Lab, an open platform for researchers to deploy Internet measurement tools [Mea]. M-Lab went public in January 2009. As of March 2010, M-Lab provides more than 45 servers in 15 sites in the USA, Europe, and Australia. The servers are currently provided and maintained by Google and PlanetLab.

M-Lab's founding was motivated by the ongoing network neutrality debate and also because of the success and findings of Glasnost. Consequently, Glasnost was one of the first tools to be deployed on M-Lab. M-Lab enables us to distribute the Glasnost deployment over a large set of locations and servers making it easier to serve a large number of users worldwide. Furthermore, M-Lab archives all collected measurement data and makes them available to researchers and other interested entities.

Details of deployed tests

We present results for four BitTorrent tests deployed on Glasnost. These tests detect port- and content-based differentiation in the upstream as well as the downstream direction. Each test involves emulating BitTorrent flows followed by reference flows. For detecting content-based differentiation, we replace BitTorrent packet payloads with random bytes in corresponding reference flow, while keeping other aspects identical. For detecting port-based differentiation, only the port of the reference flow is switched from a well-known BitTorrent port (e.g., 6881) to a neutral port that is not associated with any particular application (e.g., 10009). We emulate flows in both upstream and downstream directions to check for manipulation of both BitTorrent uploads and downloads. As described in Section 4.5, we configured Glasnost to offer a 6 minute long test to users, with each flow running for 20 seconds. Also, each flow is repeated once.

Usage

Between March 18th, 2008 and September 21st, 2009, a total of 368,815 users from 5,846 ISPs used Glasnost to test for traffic differentiation. We believe that our large user base is a direct result of our focus on lowering the barrier of use such that even lay users can use our system.

Figure 4.8 shows that the users came from across the world: North America (38%), Europe (36%), South America (11%), Asia (12%), Oceania (3%), and Africa (<1%). Table 4.2 lists the top 20 access ISPs to which our users belonged. A large fraction of our end users came from some of the largest residential ISPs in their respective countries, such as Comcast in the US, Bell Canada in Canada, or BT in the UK.

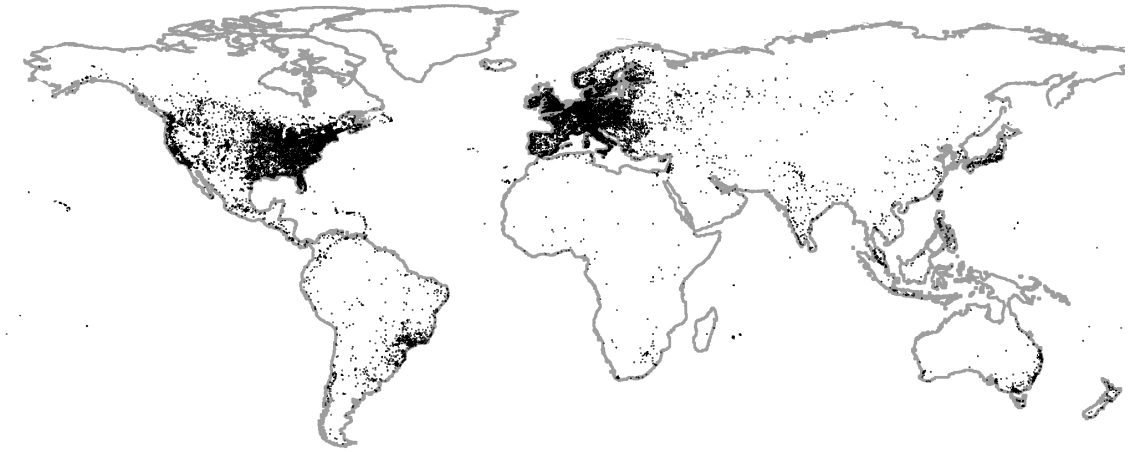


Figure 4.8: Location of Glasnost users.

ISP	Loc.	Tests	ISP	Loc.	Tests
Comcast	US	29,464	BT	UK	5,192
RoadRunner	US	16,257	Chunghwa Telec.	TW	5,084
AT&T	US	10,884	Shaw	CA	4,933
UPC	NL	8,871	Brasil Telec.	BR	4,862
Verizon	US	7,611	Rogers	CA	4,499
Cox	US	4,194	Telefonica	BR	4,408
Net Virtua	BR	7,207	Telefonica	ES	4,229
Telecom Italia	IT	6,955	NTL	UK	3,852
Charter	US	3,634	Vivo	BR	3,723
Bell Canada	CA	5,233	GVT	BR	3,723

Table 4.2: The names of the 20 ISPs from which most of our users connected to the Internet with.

Figure 4.9 shows the distribution of the users per week who ran Glasnost tests during each of the last 18 months. There are two notable jumps in the number of users. The first was in May 2008, when the system was covered in several prominent blogs and popular news media, which led to more users becoming aware of it. The second was in January 2009, when Google adopted Glasnost for its M-Lab platform. In the period between the two jumps there were about 2,500 users per week. The number of users jumped to 40,000 in the week immediately following the M-Lab deployment and has come down to 3,000 since. The overall impact of Google's adoption is that we had almost as many users in the first two months of 2009 as the entire 10 months before that.

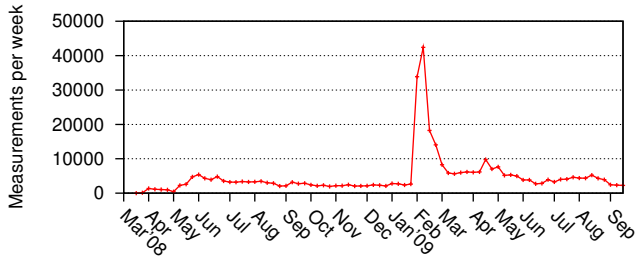


Figure 4.9: Number of users that uses Glasnost per week during our year-long deployment. The jump in January 2009 corresponds to Google's adoption of Glasnost as part of its Measurement Lab effort.

User feedback

Since our system became operational, we have received more than one hundred emails from users scattered across four continents. The feedback was overwhelmingly positive and reveals two pieces of information. First, we find evidence of false negatives in our results. Around 6% of our emailers were skeptical when Glasnost did not discover traffic differentiation. They were convinced that their ISP differentiates, sometimes based on information their ISP publishes. If these users are right, their cases confirm that our decision to minimize the false positive rate comes at the cost of false negatives. While we continue to investigate ways to reduce the false negative rate, we are pleased to report that no user has complained about the presence of a false positive.

Second, a number of emails requested that Glasnost test for other P2P applications such as eMule as well as non-P2P applications such as FTP, SSH, and HTTP. The constant stream of such requests was what motivated us to open the Glasnost platform and allow users to contribute new Glasnost tests. We also added tests for the most-requested applications to our Glasnost deployment.

4.7.2 Aggregate data analysis

We now turn our attention to understanding the policies of individual access ISPs. For this analysis, we map users to their access ISPs[3] and assume that the access ISP is responsible for any observed differentiation. While it is possible that a transit ISP along the path is responsible, differentiation is a much more common practice among access ISPs [Can08b, Com08b, Röt08].

Our Glasnost deployment was so popular that we had hundreds of users from some of the largest ISPs worldwide. Aggregating results from all the users belonging to an ISP can provide an understanding of the extent to which the ISP differentiates traffic. Such ISP-wide perspectives are especially useful for policy makers and government regulators responsible for monitoring ISP behavior. Further, end users can compare the state of differentiation across different ISPs to make a more informed choice when selecting their ISP.

Note that in this section we use the terms "tests", "IP addresses", and "users" interchangeably. There were very few IP addresses from which we saw repeat tests and a vast

[3]We map IPs to ISPs using whois information from the Regional Internet Registries.

majority of tests corresponded to unique IP addresses. The same end user might be associated with different IP addresses during the course of our study. By overlooking this, we may be over-counting the number of unique end users.

Sanitizing traces

In total, Glasnost runs a sequence of 16 flows (all 8 possible combinations of TCP port, direction, and protocol, repeating each flow once) between the user's host and the server. However, some hosts abort the test early or experience problems when running the applet. Therefore, we only considers a test result in our analysis when the following two conditions hold:

- **All 16 flows were tested and produced a result.** Test runs which do not contain results for all 16 flows are not considered in the results below. This can be caused by the user closing her web browser or browsing to another site, or by a crash of the applet.
- **All 4 flows sending random data on a neutral port were able to send some data.** Flows where at least one of the sanity check flows had no data packet ACKed (in the case of a download) or received (in the case of an upload) are discarded. This indicates that the applet was unable to contact our measurement server, which could be caused by misconfigured NATs, firewalls, or Java applet security policies.

When either of these conditions is not met, we discard the test result from our dataset.

In the following, we present results from our analysis of the measurement data collected by Glasnost for two popular types of traffic differentiation: blocking and traffic shaping of BitTorrent traffic.

4.7.3 Blocking in Broadband Networks

We used the data collected during our deployment to characterize BitTorrent blocking in the Internet. Unless otherwise stated, we limit most of our analysis in this section to tests conducted in the period from March 18th to July 25th, 2008. We found that ISPs changed their differentiation behavior as a result of the publication of our results on BitTorrent blocking and the public interest in this matter from users but also regulators. In fact, as we will show later in this section, we found most ISPs have stopped blocking BitTorrent starting in late 2008. Hence, we decided to base our analysis on the properties of BitTorrent blocking on the time period we found a large number of users being affected by it.

From March 18th to July 25th, 2008, our Glasnost servers collected a total of 47,318 result sets from end users connected to 1,987 ISPs worldwide. 146 result sets did not contain results for all 16 flows, and a further 17 failed to send data during at least one of the sanity-check flows.

Some users ran our test multiple times. To avoid biasing our results, for each IP address we considered only the first result set that passes the two conditions above, and we ignored all other result sets for that IP address. After removing the duplicate tests, we were left with 41,109 result sets.

We found evidence of BitTorrent blocking in 3,353 (8.2%) of the 41,109 result sets. We now take a closer look at the hosts that observed blocking.

Where are the blocked hosts located?

First, we examined the countries in which hosts observed BitTorrent blocking.

Table 4.3 lists all countries where we found BitTorrent blocking for at least one host. Our results indicate widespread BitTorrent blocking only for the USA and for Singapore. Interestingly, even within these countries, we observed blocking only for hosts belonging to a few ISPs.

Next, we looked at the ISPs whose hosts were affected by BitTorrent blocking. Overall, we found that hosts of 47 ISPs experienced blocking; the ISPs are listed in Table 4.3, along with the number of hosts we tested from each ISP and the number of hosts whose BitTorrent flows were blocked. The results show that not all hosts of these ISPs are affected by blocking.

We do not have enough data to determine why only some (but not all) hosts of an ISP are subjected to blocking, but there are several possible explanations. For example, the middleboxes that block BitTorrent transfers might not be deployed on all of an ISP's network paths, or blocking might depend on the current load of the network. Also, some ISPs might allow BitTorrent traffic up to a certain threshold and apply the blocking to the "heavy hitters" only.

Country	ISP	Meas.	Block.	Country	ISP	Meas.	Block.
Australia	AARNet	2	1	UK	Tiscali	354	2
Belgium	MAC Telecom	5	1	USA	Comcast	4397	2574
Brasil	Brasil Telecom	54	1		Cox	1004	508
	PaeTec Comm.	9	1		RoadRunner	2086	50
Canada	RISQ	7	1		Cablevision	646	1
	Westman Comm.	4	3		Suddenlink	123	4
China	China Telecom	49	2		Mediacom	120	17
Finland	Joensuun Elli	1	1		Clearwire	34	9
Germany	Uni Göttingen	1	1		Midcontinent	21	13
Greece	OTEnet	122	8		General Comm.	13	5
Hungary	DataNet	17	1		Pavlov Media	11	2
India	SonicWall	1	1		PaeTec	9	1
Ireland	IBIS	9	1		PrairieWave	4	2
Jamaica	Terrenap	1	1		UC Riverside	4	1
Kuwait	Wataniya Tel.	5	4		Journey Comm.	3	1
Malaysia	Tel. Malaysia	336	12		NHCTC	2	1
	Maxis Comm.	9	2		Bergen.org	1	1
New Zealand	TelstraClear	22	1		DHL Systems	1	1
Saudi Arabia	SaudiNet	8	1		Moric.org	1	1
Singapore	StarHub	156	101		PSC	1	1
South Korea	Korea Telecom	12	5		Shaw Group	1	1
Spain	Telefonica	602	1		WSIPC	1	1
Taiwan	TANet	214	2				
	Cheng Kung U.	11	2				
	APOL	10	1				

Table 4.3: The number of hosts with BitTorrent blocking grouped by country and ISP.

How do ISPs identify BitTorrent flows?

Next, we wanted to understand what flow properties ISPs were using to detect and block BitTorrent flows. We calculated how many of the 3,353 result sets contained evidence of blocking based each of the three flow characteristics Glasnost varies.

- **TCP port:** We found that only 530 (15.8%) of the result sets showed evidence of blocking based on BitTorrent ports, regardless of whether or not the flows actually contained BitTorrent messages. Thus, blocking of TCP connections based only on well-known BitTorrent ports seems to exist, but does not appear to be widespread.
- **Direction:** We found that 3,335 (99.5%) of the result sets contained evidence of blocking in the upstream direction, but only 71 (2.1%) of them contained evidence of blocking in the downstream direction. Thus, ISPs seem to be blocking primarily BitTorrent uploads and are rarely interfering with BitTorrent downloads.
- **Protocol:** Finally, we found that 3,293 (98.2%) of the result sets contained evidence of blocking based on BitTorrent messages. Thus, ISPs appear to be using DPI to block BitTorrent flows regardless of the port they are using.

In summary, the BitTorrent blocking we observed seems to be focused primarily on BitTorrent uploads, and it appears to affect flows using the BitTorrent protocol regardless of whether or not they are using a well-known BitTorrent port.

Case study: Comcast. Our analysis found that most ISPs identify BitTorrent flows based on protocol messages. Presumably, the ISPs are using DPI to monitor the protocol messages exchanged and to decide whether a flow should be blocked. To understand the precise protocol messages that trigger blocking, we ran a controlled experiment using a Comcast host in Seattle, WA, to which we had access. In this experiment, we emulated BitTorrent transfers just as Glasnost does, but we varied more aspects of the flows. For example, we obfuscated BitTorrent protocol messages by flipping bits, we left out some of the messages, and we changed the number of advertised pieces in the `bitfield` message to emulate different sharing scenarios, e.g., both peers having some but not all pieces of the file.

We found that, on this particular access link, BitTorrent uploads were blocked if and only if all of the following conditions hold:

- the server sent a valid BitTorrent `handshake` message,
- the Comcast host sent a valid `bitfield` message, and
- the Comcast host's `bitfield` message indicated that it had all pieces.

In other words, uploads of a file were blocked only when the Comcast host had finished downloading the file and was uploading it altruistically. However, the uploads were not blocked when the Comcast host was still missing some of the pieces of the file and thus appeared to be interested in downloading. From this experiment, we conclude that the middleboxes which tear down BitTorrent connections maintain some per-flow state and inspect the packet payload for specific protocol messages.

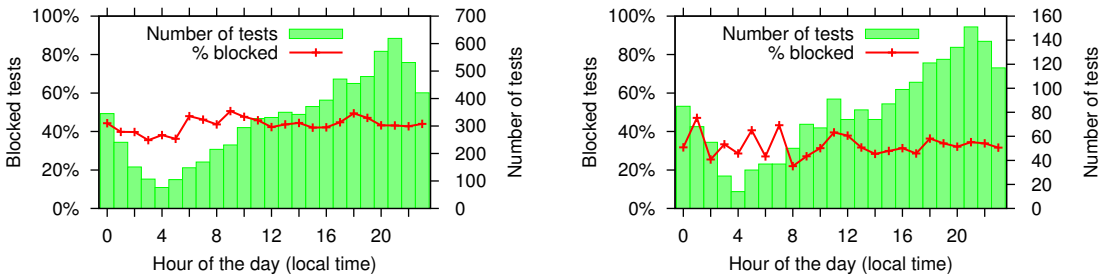
Note that this case study only applies to Comcast. Unfortunately, we did not have access to hosts connected to other ISPs and were therefore unable run the same controlled experiment for them.

When do ISPs block BitTorrent flows?

ISPs that have admitted to blocking BitTorrent flows claim that they do so only during the hours of peak load, i.e., when their networks are congested. The data we collected with Glasnost enables us to check whether blocking occurs continuously throughout the day or is limited to just a few hours of the day. For each hour of the day, we calculated the percentage of result sets that contained evidence of blocking. For each result set, we inferred the location of the tester and then computed the local time[4] when the test had been performed. We then grouped together measurements from the same hour. Here we present data for Comcast and Cox because these are the two ISPs for which we had the most data points.

Figure 4.10 shows our results. While the number of measurements per hour shows a diurnal pattern with more measurements in the evening than in the early morning, the fraction of blocked tests shows no clear trend. We observed blocking for a significant fraction of the tests throughout the day. Figure 4.11 groups the result sets by day of the week instead. Again, there is no clear trend; we observed a significant fraction of blocked hosts on all days of the week. Finally, we used a Comcast host under our control in Seattle, WA, to run Glasnost at 30-minute intervals for an entire week. We found that BitTorrent flows were constantly blocked during the entire week.

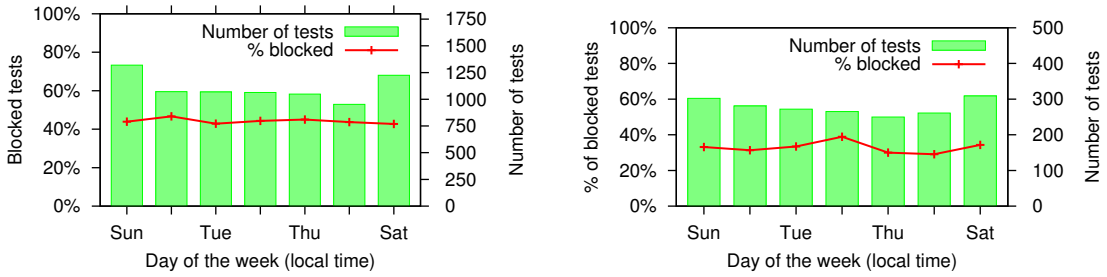
In conclusion, our data suggests that BitTorrent flows are being blocked independent of the time of the day or the day of the week.



(a) Comcast (USA) **(b)** Cox (USA)

Figure 4.10: Result sets grouped by the hour of the day for Comcast and Cox. BitTorrent flows were blocked at all times of the day.

[4]We used an IP-to-geolocation tool to infer the timezone of each tester.



(a) Comcast (USA) **(b)** Cox (USA)

Figure 4.11: Result sets grouped by the day of the week for Comcast and Cox. Blocking occurred on every day of the week.

At what stage are flows blocked?

Finally, we took a closer look at the Glasnost packet traces to see at which stage of the BitTorrent protocol the blocking occurred. The RST packets can be injected at different points in a transfer, that is, at different stages of the BitTorrent protocol shown in Figure 4.3a. To perform this analysis, we used the data reported by our user-side applet about the last message it sent before the connection was torn down.

In total, we identified four different places in the protocol at which connections were blocked. We found a very strong correlation in behavior across ISPs, and we observed mostly consistent behavior for hosts of the same ISP.

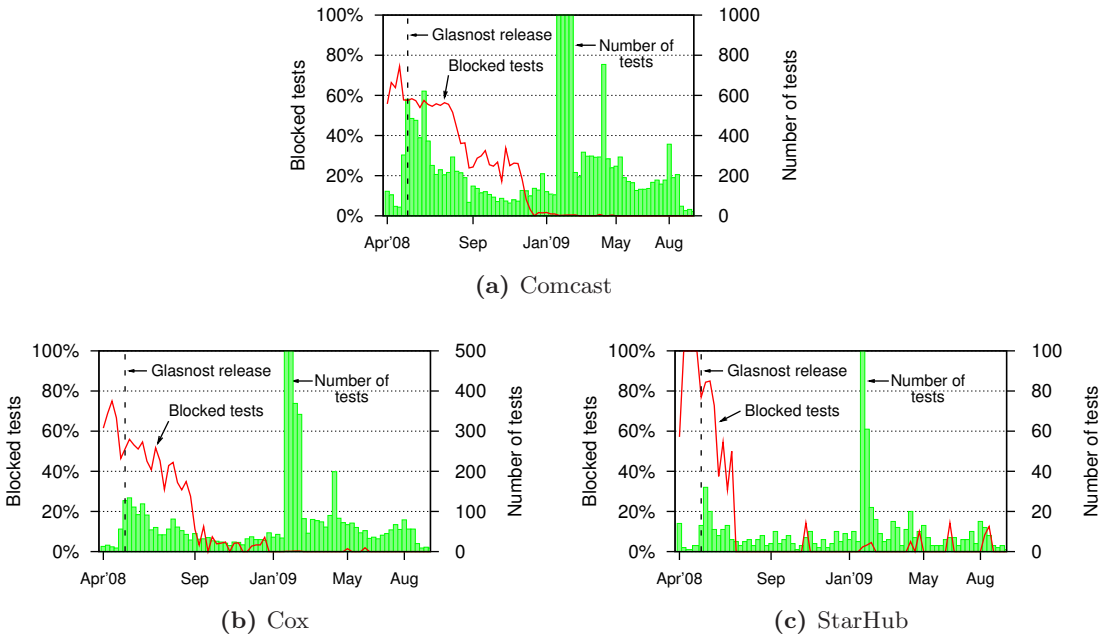
- **After the `handshake` message:** For Telekom Malaysia and Brasil Telecom we observed that the connection with BitTorrent messages was torn down immediately after the `handshake` message was sent by the leecher (i.e., the host that wants to download).
- **After the `bitfield` message:** For StarHub, RoadRunner OTEnet, and most other ISPs we observed connection tear-down for connections with BitTorrent messages after the leecher sent the `bitfield` message.
- **After the `interested` message:** For most Comcast and Cox hosts, we observed that the connections with BitTorrent messages were torn down after the `interested` message was sent by the leecher.
- **Later in the transfer:** Finally, for Comcast, Cox, and Mediacom, we observed that connections with random data on BitTorrent ports were occasionally torn down later in the transfer. However, we were unable to determine a common pattern for the exact point where the connection was torn down.

While the types of blocking can sometimes vary even between hosts of the same ISP, we found that the basic characteristics of blocking were mostly consistent across hosts and even across some of the ISPs. Because of this, we suspect that many ISPs are using similar equipment for traffic identification and reset injection, e.g., the specialized hardware sold by Sandvine. However, it is possible that these boxes are configured differently in different locations or at different times of the day.

Do ISPs change their traffic management policies?

ISPs' traffic management policies are not static. For instance, ISPs can decide to target new applications or change their technique for shaping flows. Glasnost allows us to study whether ISPs have changed their policies over the time of our deployment. Here, we focus on three ISPs, Comcast, Cox, and StarHub, as we found that a large fraction of their customers suffer from BitTorrent blocking.

Figure 4.12 shows the number of tests performed per week between March 2008 and September 2009. The number of tests varies a lot with occasional usage spikes, as for example in January 2009 when M-Lab went public and the resulting wide media coverage attracted a large number of users. Figure 4.12 also shows the percentage of Glasnost tests that detected BitTorrent blocking. In contrast to the number of tests, the percentage of blocked tests shows a clear trend: since we first deployed Glasnost in March 2008, Comcast, Cox, and StarHub have stopped blocking BitTorrent flows. This is likely a result of the wide and critical media coverage about these ISPs' traffic management practices after we released results from our Glasnost deployment in May 2008. More, after we published our results, telecommunication regulators started investigating ISPs traffic management practices (some of these regulators actually used Glasnost data in their investigations), which increased the public pressure on blocking ISPs even more. Also, for Comcast our findings coincided with an announcement that it will replace its blocking policy with other traffic management policies by the end of 2008 [Com08a]. In this case, Glasnost can help to verify whether an ISP actually follows through on its announcement that it will stop traffic shaping holds.



(a) Comcast

(b) Cox

(c) StarHub

Figure 4.12: ISPs change their BitTorrent blocking policies over time. Since we published Glasnost test results in May 2008, Comcast, Cox, and StarHub have mostly stopped blocking BitTorrent transfers.

4.7.4 Throttling in Broadband Networks

In the previous section, we discussed Glasnost's findings on the prevalence of BitTorrent blocking in the Internet. Here, we use the data collected during our Glasnost deployment to characterize BitTorrent throttling in the Internet. To our knowledge, such a detailed characterization was not available prior to our study. We limit the analysis in this section to the tests conducted in the two-month period from January and February 2009. We used a short period to avoid effects caused by ISPs that might have changed their differentiation behavior over the time of our measurements, e.g., when an ISP decides to change its traffic management policies and starts or stops differentiation of particular Internet traffic (cf. Section 4.7.3). We selected the two-month period that contains the most datapoints. Further, we considered only ISPs for which we have at least 100 tests in this time period. There are 140 such ISPs.

Figure 4.13 shows the percentage of users for whom we detected throttling in at least one of the four tests that we deployed on Glasnost. The number of Glasnost users per week varied substantially (see Figure 4.9). But aside from a few weeks in the beginning when we did not have enough users, the percentage of throttled users has stayed roughly constant around 10%. Thus, a non-negligible fraction of our testers are subject to differentiation.

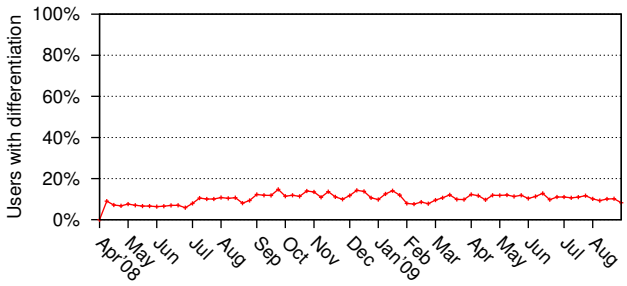


Figure 4.13: Percentage of users that are deemed to suffer differentiation since March 2008.

Note, however, that we cannot use our data to claim that 10% of all Internet users suffer from throttling. Because the users of Glasnost are a self-selecting set, our data may be dominated by those that suspect that their ISP is differentiating against BitTorrent traffic.

Basis for differentiation

Table 4.4 shows the list of the top-30 ISPs ranked based on the fraction of hosts that detected throttling. It also shows how traffic is differentiated. More than half the ISPs throttle only in the upstream direction and 23% of ISPs only in the downstream direction. 20% of ISPs (e.g., Clearwire, TVCABO) throttle traffic in both directions. We also find that most throttling ISPs use both content- and port-based differentiation. For only four ISPs (Free, GVT, Pipex, and Tiscali UK) do we observe an exclusive use of port-

ISP	Loc.	Upstream		Downstream	
		app	port	app	port
Bell Canada (DSL)	CA	×	×		
Brasil Telecom (DSL)	BR			×	×
BT (DSL)	UK	×	×		
Cablecom (Cable)	CH	×	×		
Canaca (DSL)	CA	×	×		
City Telecom (FTTH)	HK	×	×	×	×
Clearwire (WiMax)	US	×	×	×	×
Cogeco (Cable)	CA	×	×		
EastLink (Cable)	CA	×	×		
Free (DSL)	FR				×
GVT (DSL,FTTH)	BR		×		
Kabel Deutschland (Cable)	DE	×	×	×	×
Magix (DSL)	SG			×	×
Oi (DSL)	BR			×	
ONO (Cable)	ES			×	×
PCCW (DSL)	HK	×	×	×	×
Pipex (DSL)	UK		×		
Rogers (Cable)	CA	×	×		
Shaw (Cable)	CA	×	×		
TekSavvy (DSL)	CA	×	×		
Tele2 (DSL)	IT	×	×		
Telenet (DSL)	BE		×	×	
TFN (DSL)	TW			×	×
Tiscali Italia (DSL)	IT			×	×
Tiscali UK (DSL)	UK		×		
TM Net (DSL)	MY	×	×		
TVCABO (Cable)	PT	×	×	×	×
UPC NL (Cable)	NL	×	×		
UPC Poland (Cable)	PL	×	×		
UPC Romania (Cable)	RO	×	×		

Table 4.4: We detected BitTorrent throttling for users of 30 ISPs during January and February 2009. The table shows which of them were differentiated based on application messages (app) or based on TCP port.

based differentiation (which is easier to evade). And only one ISP, Oi, uses content-based differentiation exclusively.

Fraction of users impacted

ISPs that throttle BitTorrent traffic do not do so for every user. For each ISP in Table 4.4, Figure 4.14 shows the fraction of users that tested positive for differentiation. We see that in the median case only 21% of users are affected. Given our tests' low false positive and negative rates, this inconsistent differentiation behavior within an ISP cannot be explained by inference errors alone.

Our data does not allow us to infer why only a fraction of users of an ISP experience traffic differentiation. There are many possible reasons. An ISP might choose to target only customers who generate a lot of P2P traffic, the traffic shapers might be deployed in only a portion of the ISP network, or an ISP might differentiate only during peak hours or periods of high load.

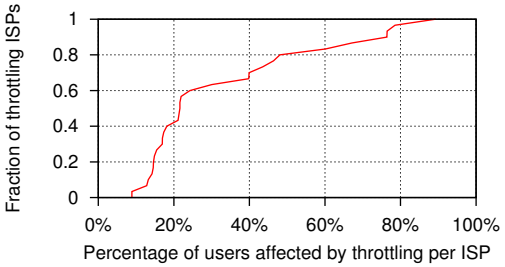


Figure 4.14: For most ISPs we detected traffic differentiation for only a fraction of users.

Dependence on time of day

One potential explanation for why only some users experience differentiation is that ISPs may differentiate only during peak hours, when the network is experiencing the greatest load. To test for the dependence on time of day we divided our dataset into two time periods based on the local time of the user[5]. The *peak period* is the time between 8pm and midnight, and the *off-peak period* is the time between 5am and 9am. These periods are strict subsets of the peak and off-peak durations for access ISPs [LR08, Can08b].

For each period we infer if an ISP differentiated traffic. Our analysis excludes ISPs that have fewer than 100 measurements for either of the two time periods. This leaves us with 30 ISPs. We find that slightly more than half of these ISPs differentiate during both peak and off-peak hours. The other ISPs, e.g., BT, Bell Canada, Kabel Deutschland, ONO, and Tiscali UK, restrict traffic differentiation to the peak period.

Our results in the last two sections show the importance of enabling end users to detect differentiation for themselves and at particular points in time. Other existing tools attempt to discover whether or not an ISP differentiates traffic [TMFA09, ZMZ08]. Since not all users of an ISP are affected by differentiation all the time, ISP-wide information alone is not sufficient for a user to determine if she experiences differentiation.

4.8 Summary

In summary, Glasnost enabled hundreds of thousands of users to test their links for traffic differentiation. The results we published on the prevalence of BitTorrent blocking in the Internet have been widely publicized, e.g., by Reuters and Associated Press, and have been used by regulators in the USA, Canada, Singapore, and Greece to monitor their national ISPs. Interestingly, we found that many ISPs whom we had found to block BitTorrent stopped doing so after we released our results.

Glasnost continues to be very popular. Glasnost currently attracts 1,000 users a day who test their links for traffic differentiation. The ability to create new tests should keep Glasnost attractive for users in the future.

[5]We used an IP-to-geolocation tool to infer the timezone of each tester.

Part II

Evaluating Systems in Broadband Access Networks at Large Scale

5 Background and Related Work

Evaluating new system designs under realistic conditions is important for verifying that these systems work as expected once they are deployed. New system designs are usually based on certain design assumptions about the network environment they are supposed to run in. To verify that these assumptions in fact hold under real-world conditions, evaluation experiments are used. Furthermore, the complex nature of the interaction between a distributed system and the network it runs in is often hard to model. Evaluation experiments help to quantify the performance of distributed systems in the wild, result in optimizations that increase a system's performance, and identify implementation bugs and behavioral anomalies that do not surface during standard, small-scale testing.

Most of today's evaluation techniques focus on academic networks and the Internet backbone and thus only cover a small part of the Internet. We have shown in the previous chapters that residential broadband networks can have very different characteristics than academic or corporate networks. Thus, today's evaluation techniques are not suitable to evaluate systems that will be deployed on broadband hosts.

To enable researchers to evaluate their systems over broadband networks, we present two solutions as part of this thesis. First, we developed Monarch, a tool that enables researchers to study and compare the performance of new and existing transport protocols at large scale in broadband environments. And second, we designed SatelliteLab, a novel testbed design that makes it easy to add arbitrary end nodes, including broadband nodes and even smartphones, to existing testbeds like PlanetLab.

In the remainder of this chapter we present background and related work. We discuss in detail three techniques that are typically used to evaluate networked systems: network measurements, simulation or emulation experiments, and testbed experiments.

5.1 Evaluation Using Measurements of Deployed Systems

Network measurements are often used to study already deployed systems and their performance in the wild. For example, the wide deployment of TCP in the Internet allowed researchers to extensively study the performance of TCP using network measurements. We list some popular examples next. Bolot [Bol93] and Paxson [Pax97] performed some of the initial studies on TCP packet dynamics along a fixed set of Internet paths. Padhye and Floyd [PF01] characterized the TCP behavior of a large set of popular Web servers. Medina et al. [MAF05] investigated the behavior of TCP implementations and extensions. In a different project, Medina et al. [MAF04] characterized the effect of network middleboxes on transport protocols. Based on the traces of TCP flows to a busy Web server, Balakrishnan et al. [BPS+98] presented a detailed analysis of the performance of individual TCP flows carrying Web traffic. Jaiswal et al. [JID+04] used traffic traces of a Tier-1 ISP to investigate the evolution of TCP connection variables over the lifetime of a TCP connection. More recently, Arlitt et al. [AKM05] have used Web traces to investigate the impact of latency on the duration of short transfers.

5.2 Evaluation Using Simulations and Emulations

A key advantage of simulations and emulations is that they offer full control over the experimental parameters, which allows researchers to produce repeatable results. However, this control comes at a cost: the underlying models of the network topologies and the network traffic are synthetic. Consequently, the experiments are of only limited realism. Simulators and emulators simulate arbitrary network topologies using analytical or simplifying models. Also, background traffic is either synthesized using analytical traffic models or replayed from previously captured real-world traces.

The key difference between simulators and emulators is that simulators often model the networking stack down to the physical layer (i.e., also the networking hardware is simulated in software), while emulators build on top of networking hardware and the networking stack of an operating system for experiments. Thereby, emulators typically manage the network traffic to configure an experimental scenario (e.g., by adjusting the bandwidths and delays of flows to resemble a wide-area network connection). Many simulators do not allow the original implementation of a protocol or system to be used. Instead, they provide their own programming environment, which often requires a full reimplementation [ns2]. Emulators like EmuLab [WLS+02], on the other hand, usually extend existing operating systems with a special networking stack that allows emulating arbitrary network topologies, thus creating a virtual, large-scale testbed on a local cluster of nodes. Thus, researchers can often use their existing implementations without changes.

There is a large and impressive number of existing network simulators and emulators; here we list some of the more important ones.

The network simulator ns-2 [ns2] is probably the most-frequently used in academic research. It simulates the network stack down to the MAC layer and includes extensions to simulate wireless networks. Other network simulators include NetPath [ASB05], OMNeT++ [Sim92], and GloMoSim [BTA+99].

For cable links, detailed DOCSIS simulators are available, including the Common Simulation Framework (CSF) from CableLabs and a ns-2 implementation from Shah et al. [SKMM05]. These simulators are useful for exploring the parameter space of DOCSIS. However, it has been unclear how to set the parameters in a way that accurately emulates commercially deployed networks. With the results from our measurement study in Chapter 3, there is now a way to configure these simulators for realistic experiments.

Two of the most popular emulators today are EmuLab [WLS+02] and ModelNet [VYW+02]. They create a virtual experimental environment that allows Internet-scale experiments to be run on a local cluster. Both are based on Dummynet [Riz97], a tool to simulate queue and bandwidth limitations, delays, packet losses, and multipath effects according to a given network model. FlexLab [RDS+07] extends EmuLab to use real-time measurements to dynamically and more realistically configure the emulated Internet links. More recently, Eide et al. [ESL07] presented an enhanced version of EmuLab called Workbench, which adds a facility to replay previous experiments allowing for richer analysis and experimentation.

Other recent emulators include a tool by Sanaga et al. [SDRL09] to emulate the end-to-end behavior of network paths instead of individual network links. Finally, DieCast [GVV08] emulates distributed systems at large scale using virtual machine technology to run many instances of the system on a small number of physical machines.

None of the popular simulators and emulators, such as ns2, EmuLab, and ModelNet, provide built-in models for broadband networks. Thus, at this point they do not allow researchers to evaluate systems for broadband nodes. However, using the results from our broadband studies in this thesis, models for broadband networks can be developed in the future.

5.3 Evaluation Using Testbeds

Internet testbeds have become indispensable for evaluation in networking and distributed systems research. Unlike simulators and emulators, testbeds can accurately capture path properties and thus provide a realistic network environment for evaluating high-level protocols and applications. Testbeds typically consist of tens to hundreds of server-class nodes distributed across the Internet. Thus network traffic is transferred across the Internet under realistic conditions, making experiments running on top of them more realistic than simulations. However, while offering a more realistic environment than simulations, testbeds do not offer the full control of the environment to experimenters. Hence, results are usually not reproducible as the environment is constantly changing.

There are a number of popular testbeds used in academic research. NIMI [PAM02] is an infrastructure for Internet measurements. Researchers can use NIMI to deploy new measurement tools and study the Internet path characteristics between the testbed nodes. RON [ABKM03] is a testbed of about 20 nodes that allows researchers to build and evaluate overlay routing topologies. VINI [BFH+06] is a virtual network infrastructure researchers can build arbitrary network topologies with to evaluate new routing protocols. VINI does not provide its own testbed nodes, but can be run on top of existing testbeds. The state-of-the-art Internet testbed today is PlanetLab [Pla], which is composed of nodes from more than 500 sites worldwide and has been used by more than 1,000 researchers for evaluation experiments. Recently, the success of PlanetLab has inspired numerous efforts to build next-generation testbed environments, such as GENI [GEN] in the USA and FIRE [FIR] in Europe.

Testbeds are supposed to offer a highly realistic evaluation environment that allows systems to be evaluated over real Internet paths. However, today's testbeds are very homogeneous. Most nodes in existing testbeds are located in well-connected academic and corporate networks, and the network paths between them are often restricted to well-provisioned research backbones [BGP04]. As a consequence, these testbeds lack the heterogeneity that characterizes the commercial Internet [SPBP05]. For example, even PlanetLab contains only a handful of broadband nodes and it is already widely accepted that PlanetLab does not capture the characteristics specific to network paths in wireless and mobile environments [BPSK97]. Thus, results from evaluation experiments performed over these testbeds cannot be used to predict how an application will perform in, e.g., broadband networks.

The lack of heterogeneity in today's Internet testbeds is widely known and has been identified as an important concern by the designers of PlanetLab [SPBP05]. Also, many research projects have shown the consequences of the homogeneity of testbeds either directly or indirectly [BGP04, PHM06]. These projects pointed out surprising differences between the results of evaluations performed in testbeds and those performed in highly heterogeneous networks. For example, the paths between PlanetLab nodes and the paths

in access networks have different reliability and packet loss characteristics [GMG+04]. Evaluation of the Vivaldi network coordinate system showed that the accuracy of the system differs vastly when run in a testbed or in the Internet at large [LGS07]. Our own study of residential broadband networks, as presented in Chapter 3, found that transport protocols, such as TCP, can behave differently in broadband access networks than in academic networks. Today's homogeneity of testbeds creates several challenges for networking researchers: networked system designs cannot be rigorously evaluated in an environment that lacks the diversity present in the Internet, protocols are shielded from complications caused by middle boxes (such as proxies and NATs), and network measurement results are not always representative and are difficult to generalize. The obvious solution to this — adding more broadband nodes to existing testbeds — is hard, as testbeds typically have high requirements for new testbed nodes. For example, PlanetLab requires nodes to have server-class hardware and a high-speed Internet connection with a static, public IP address [Pla02]. Such requirements are often too high for ordinary broadband nodes, hence only a small number of broadband nodes are part of today's testbeds.

In the following, we present two approaches that allow evaluations at large scale in heterogeneous environments over the Internet.

In Chapter 6, we present Monarch, a tool that emulates transport protocol flows over live Internet paths. Monarch enables transport protocols to be evaluated in realistic environments and thus complements the controlled environments provided by the state-of-the-art network simulators and emulators, and testbeds. Monarch uses generic TCP, UDP, or ICMP probes to emulate transport protocol flows to any remote host that responds to such probes. By relying on minimal support from the remote host, Monarch enables protocols to be evaluated on an unprecedented scale over millions of Internet paths.

In Chapter 7, we present a new testbed design called SatelliteLab, a system that augments heterogeneity of existing planetary-scale network testbeds by recruiting nodes from Internet edge networks. While state-of-the-art testbeds have high requirements for testbed nodes, SatelliteLab makes it possible to add a diverse set of nodes to the testbed, including broadband nodes and even smartphones, which are rarely present in existing testbeds.

6 Monarch: Emulating Transport Protocol Flows over the Internet at Large

There is a large body of work on designing new transport protocols, such as TCP Vegas [BP95], TCP Nice [VKD02], TFRC [FHPW00], or PCP [ACKZ06]. However, evaluating these protocols on the Internet at large has proved difficult. Current approaches require the protocols to be deployed at both endpoints of an Internet path. In practice, this restricts the evaluation of transport protocols to studies conducted over research testbeds, such as PlanetLab [Pla], RON [ABKM03], or NIMI [PAM02]. Unfortunately, these testbeds are limited in their scale and, as we pointed out in the previous chapter, they are often not representative of the many heterogeneous network environments that constitute the Internet [BGP04, PHM06].

In this chapter, we propose Monarch, a tool that emulates transport protocol flows from a user-controlled end host to any other Internet host. Similar to our methodology in Chapter 3, Monarch leverages the fact that many of these Internet hosts and routers respond to TCP, UDP, or ICMP packet probes. By requiring control of just one of the two end hosts of a path, researchers can use Monarch to evaluate transport protocols in large-scale experiments over a diverse set of Internet paths.

Monarch is based on a key observation about how transport protocols typically work: a sender transfers data to a receiver at a rate determined by the latency and loss characteristics of the Internet path between the two endpoints. Instead of using a real TCP transfer, Monarch uses generic TCP, UDP, or ICMP probes and responses to emulate this packet exchange between a local sender and a remote receiver.

Monarch is accurate because it relies on direct online measurements. For every packet transmission in its emulated flow, Monarch sends an actual probe packet of the same size to the receiver and interprets the response packet as an incoming acknowledgment. Thus, the emulated flows are subjected to a wide range of conditions affecting real network paths, including congestion, delays, failures, or router bugs. However, as Monarch controls only one end host, it can estimate the conditions of the round-trip path but not the one-way paths. Despite this limitation, our evaluation shows that packet-level traces of flows emulated with Monarch closely match those of actual network transfers.

While several previous studies have examined the relative performance of different congestion control algorithms [ADLY95, MLAW99], most of them had access to just a small number of "real" links and therefore had to complement their measurements with simulation results. Moreover, links such as cable and DSL connections have been uncommon in the lab and therefore do not usually appear in measurement studies.

Monarch enhances the state of the art in transport protocol evaluation. Today, researchers can use controlled environments like network emulators [WLS+02, VYW+02] or testbeds [Pla, ABKM03, PAM02] for a systematic analysis of protocol behavior. Monarch complements these tools by providing live access to a real network path. This allows experiments to be conducted with emerging network infrastructures, such as broadband

networks, for which emulators and testbeds are not yet widely available. Using Monarch, researchers can easily obtain results for thousands of "real" links with a wide range of characteristics. Further, Monarch naturally captures the complex protocol interactions with the different configurations of networks and traffic workloads that exist in deployed systems. These tests allow software developers to test or debug the performance and reliability of protocol implementations, which helps to uncover bugs, performance bottlenecks, or poor design decisions in the transport protocol.

6.1 The Design of Monarch

This section focuses on the design of Monarch. We start with an overview of how Monarch emulates transport protocols. Later, we discuss a variety of probing mechanisms Monarch can use, the number of Internet paths it can measure, the types of transport protocols it can emulate, and the factors that affect its accuracy.

6.1.1 How Monarch Works

In a typical transport protocol, such as TCP, a sender on one host sends large data packets to a receiver on another host, and the receiver responds with small acknowledgment packets (Figure 6.1a). Monarch emulates this packet exchange by sending the remote host large *probe packets* that elicit small responses (Figure 6.1b). To emulate a TCP flow, Monarch creates both a TCP sender and a TCP receiver on the *same* local host, but interposes between them (see Figure 6.1d). Whenever the sender transmits a packet, Monarch captures it and, instead of forwarding this original packet, sends a probe packet of the same size to the remote host. As soon as it receives a response from the remote host, Monarch forwards the captured packet to the receiver. Packets in the reverse direction — from the TCP receiver to the TCP sender — are forwarded directly.

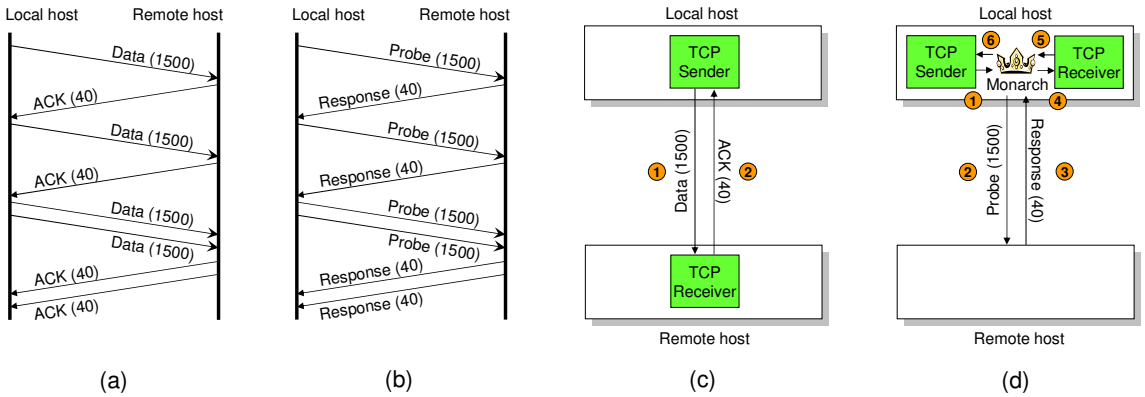


Figure 6.1: The Monarch packet exchange. In a normal TCP flow, large data packets flow in one direction and small acknowledgment packets in the other (a). Monarch emulates this by using large probe packets that elicit small responses (b). While in a normal flow sender and receiver are on different hosts (c), Monarch co-locates them on the same host and interposes between them (d). The numbers in parentheses are packet lengths in bytes.

The sizes of Monarch's probe and response packets match those of TCP's data and acknowledgment packets, and they are transmitted over the same Internet paths. As a result, the sender observes similar RTTs, queuing delays, and loss rates for its packet transmissions as for a TCP real flow. Because Monarch uses online measurements as opposed to analytical models of the network, the characteristics of flows emulated by Monarch closely match those of real TCP flows.

In our simplified description above, we made several assumptions. For example, we assumed that probe packets can be matched uniquely to their response packets, that arbitrary Internet hosts would respond to probe packets, and that an accurate emulation of round-trip (rather than one-way) packet latencies and losses is sufficient for an accurate emulation of transport protocols. Later in this section, we discuss the extent to which these assumptions hold in the Internet at large.

Monarch's output is a packet trace similar to the output of `tcpdump`. Based on this trace, we can infer network path properties, such as packet RTTs, and transport protocol characteristics, such as throughput. We show a particularly interesting use of the collected packet trace in Section 6.2.4: Monarch can analyze its output to detect errors in its own emulated flows.

6.1.2 What Types of Probes Can Monarch Use?

Monarch can use several types of probe packets to emulate transport flows. It is useful to have multiple probe types to choose from because not all hosts respond to all probes. To be accurate, Monarch needs (1) the remote host to respond to *every* probe packet it receives, (2) a way to *match* responses with their corresponding probes, and (3) the sizes of the probe and response packets to be similar to those of the data and acknowledgment packets of a regular flow. Monarch currently supports the following four types of probes:

- **TCP:** Monarch's probe packet is a variable-sized TCP acknowledgment (ACK) sent to a closed port on the remote host. The remote host responds with a small, fixed size TCP reset (RST) packet. According to the TCP standard [Uni81], the sequence number of the RST packet is set to the acknowledgment number of the probe packet header, which enables Monarch to match probes with responses.

- **UDP:** Monarch sends a variable sized UDP packet to a closed port on the remote host, which responds with a small, fixed-size ICMP "port unreachable" message. The response packet contains the first eight bytes of the probe packet, including the IPID field of the probe packet headers [Pos81]. By setting unique IPIDs in its probe packets, Monarch can match probes with responses.

- **ICMP echo request:** Monarch sends a variable-sized ICMP echo request packet to the remote host, which answers with a similarly sized ICMP echo reply packet [Pos81]. The response packet has the same sequence number field in its header as the probe packet, enabling Monarch to match probes with responses.

- **ICMP timestamp request:** Monarch sends an ICMP timestamp request message to the remote host, which answers[1] with a small, fixed size ICMP timestamp reply

[1] Govindan and Paxson [GP02] observed that some routers use a 'slow path' for generating ICMP timestamp responses, which introduces additional delays. Hence, these probes should be used with caution. We use TCP probes whenever possible.

packet [Pos81]. The response packet has the same sequence number field in its headers as the probe packet, enabling Monarch to match probes with responses.

These probes and responses differ in their suitability for evaluating transport protocols. For example, TCP and UDP probes allow the probe packet sizes to be varied, even as the response packet sizes are held fixed between 40 and 60 bytes. They are well suited for matching the sizes of data and acknowledgment packets for many variants of the popular TCP protocol, such as Reno, Vegas, and NICE. The ICMP echo responses on the other hand are of the same size as their probes. Consequently, they are better suited for evaluating transport flows where data flows in both directions.

6.1.3 How Many Internet Hosts Respond to Monarch Probes?

In theory, Monarch could emulate a transport flow to any remote host running a TCP/IP implementation, since the protocol standards require a response to each of the probes presented above. In practice, however, many hosts are either offline or behind NATs and firewalls that block or rate-limit incoming probe packets.

Note that in Chapter 3 we used a methodology similar to Monarch's to study broadband networks using TCP ACK and ICMP echo request probes. Thus, we extended the experiment from Section 3.1.3 to also include UDP and ICMP timestamp request probes. As before, we sent probes to end hosts in commercial broadband ISPs, end hosts in academic and research networks, and Internet routers. We selected end hosts in broadband and academic networks from a 2001 trace of peers participating in the Gnutella file-sharing system [SGG02]. We selected hosts belonging to major DSL/cable ISPs and university domains in North America and Europe using their DNS names. We discovered Internet routers by running `traceroute` to the end hosts in broadband and academic networks.

Table 6.1 presents our results. We probed 1,000 hosts in each of the three host categories. Overall, more than 18% of the academic hosts, 28% of the broadband hosts, and over 90% of the routers responded to at least one of the four types of probes. While this may seem like a small percentage, there are millions of hosts in the Internet, and it should be easy to find thousands of suitable hosts for an experiment.

Using very conservative estimates, our results suggest that Monarch can evaluate transport protocols to at least 18% of Internet hosts, and to at least 7% of hosts when restricted to TCP probes only. This shows that Monarch can evaluate transport protocols over a

	Broadband	**Academic**	**Router**
TCP ACK	7.2%	13.4%	69.6%
ICMP TsReq	18.0%	4.9%	63.0%
ICMP EchoReq	25.0%	8.9%	89.3%
UDP packet	7.4%	4.1%	7.3%
Any probe	28.4%	18.1%	90.3%

Table 6.1: Fraction of Internet hosts responding to our probes. We selected a sample set of 1000 hosts from each of three different categories of hosts: hosts in commercial broadband ISPs, hosts in academic and research environments, and Internet routers.

diverse set of Internet paths, a set which is several orders of magnitude larger than current research testbeds can provide. Note that we measured more than 1,800 hosts from 11 major ISPs in North America and Europe in our characterization of broadband hosts in Chapter 3 using the same TCP ACK and ICMP echo request probes as Monarch. This shows that, while research testbeds like PlanetLab have a very small number of broadband hosts, it is possible to use Monarch's methodology to emulate TCP flows to thousands of hosts in commercial cable and DSL ISPs worldwide.

6.1.4 What Transport Protocols Can Monarch Emulate?

Monarch emulates transport protocols based on real-time, online measurements of packet latencies and losses. Hence, any transport protocol where the receiver feedback is limited to path latencies and losses can be emulated. As shown in Table 6.2, this includes many variants of the widely used TCP protocol, a number of protocol extensions, and several streaming protocols.

Protocol	Usable?
TCP (New)Reno [FHG04], TCP Nice [VKD02], TCP Vegas [BP95], TCP BIC [XHR04], Highspeed TCP [Flo03], Fast TCP [JWL04] Scalable TCP [Kel03], PCP [ACKZ06], TCP Westwood [MCG+01] SACK [MMFR96], FACK [MM96], Window scaling [JBB92] RAP [RHE99], TFRC [FHPW00]	Yes
ECN [RFB01], XCP [KHR02]	No

Table 6.2: Supported protocols. Monarch can be used to evaluate many, but not all, transport protocols.

However, Monarch *cannot* emulate transport protocols that require the receiver to relay more complex information about the network to the sender. For example, Monarch cannot emulate TCP with explicit congestion notification (ECN) [RFB01] because it would require the remote host to echo back the congestion experienced (CE) bit to the Monarch host. We are not aware of any type of probe that could be used for this purpose. Similarly, Monarch cannot be used to evaluate protocols like XCP [KHR02] that require changes to existing network infrastructure.

Monarch currently emulates transport flows in the downstream direction, i.e., connections in which data flows from the Monarch host to the remote host. This mimics the typical usage pattern in which an end host downloads content from an Internet server. Emulating data flows in the upstream direction from the remote host to the Monarch host requires a small probe packet that elicits a large response packet. We have not yet found a probe packet that has this property.

6.1.5 What Factors Affect Monarch's Accuracy?

Monarch is based on round-trip (rather than one-way) estimates of packet latencies and losses. When packets are lost or reordered, Monarch cannot distinguish whether these

events occurred on the downstream path, i.e., from the sender to the remote host, or on the upstream path, i.e., from the remote host to the sender. While this could cause Monarch flows to behave differently than regular TCP flows, our evaluation in Section 6.3 shows that both these events occur rarely in practice and even when they do occur, they tend to have a limited effect on Monarch's accuracy. For example, upstream packet loss and reordering affect less than 15% of all flows. Further, Monarch has a built-in self-diagnosis mechanism that can detect most of these inaccuracies using an offline analysis. Nevertheless, Monarch is not suitable for environments where upstream loss and reordering events occur frequently.

Another source of differences between Monarch and TCP flows is the delayed ACK feature. With delayed ACKs, TCP data packets received in close succession are acknowledged with a single ACK packet. In contrast, in a Monarch flow, the receiver responds to *every* probe packet, which typically doubles the number of packets flowing on the reverse path. However, because the response packets are small, this difference is likely to affect only flows experiencing severe congestion on the upstream path.

When Monarch is used on a path that contains middleboxes such as NATs or firewalls, the probes may be answered by the middleboxes rather than the end host. However, the middleboxes are often deployed close to the end host, and so the resulting loss of fidelity tends to be small. For example, Monarch probes to many commercial cable/DSL hosts are answered by the modems that are one hop away from the end host; however, the network paths to the modems still include the 'last mile' cable or DSL links.

6.2 Implementation

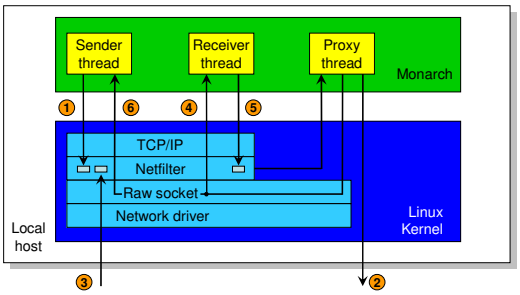
In this section, we first present the details of our implementation of Monarch, which runs as a user-level application on unmodified Linux 2.4 and 2.6 kernels. We then describe how our implementation allows us to test complete, unmodified implementations of transport protocols in the Linux kernel as well as the `ns-2` simulator [ns2]. Next, we show how we avoid unwanted losses that can be caused by network paths with unusual small maximum transfer units (MTU). Finally, we discuss the self-diagnosis feature of our implementation. In particular, we show how Monarch can detect potential inaccuracies in its emulated flows by an offline analysis of its output.

6.2.1 Emulating a TCP Flow

Our Monarch implementation uses three threads: A *sender* and a *receiver*, which perform a simple TCP transfer, as well as a *proxy*, which is responsible for intercepting packets and handling probes and responses. The proxy also records all packets sent or received by Monarch and writes them to a trace file for further analysis.

To emulate a flow to `remoteIP`, Monarch uses the Netfilter [Net03] framework in the Linux kernel. First, the proxy sets up a Netfilter rule that captures all packets to and from that remote IP address. Next, it creates a raw socket for sending packets to the remote host. Finally, the sender thread attempts to establish a TCP connection to the remote host (`remoteIP`), and the packet exchange shown in Figure 6.2 takes place.

As usual, the sender begins by sending a SYN packet to `remoteIP` (step 1). The proxy intercepts this packet, stores it in a local buffer, and sends a similarly-sized probe packet to the remote host (step 2). The remote host responds with a packet that is also intercepted



#	Action	Packet	Source	Destination
1	Sender transmits packet	Data	localIP	remoteIP
2	Proxy intercepts packet, saves it, and transmits a probe	Probe	localIP	remoteIP
3	Remote responds	Response	remoteIP	localIP
4	Proxy forwards saved packet to the receiver	Data	remoteIP	localIP
5	Receiver sends ACK	ACK	localIP	remoteIP
6	Proxy forwards ACK directly to the sender	ACK	remoteIP	localIP

Figure 6.2: Sequence of packet exchanges in Monarch implementation. Monarch consists of a TCP sender, a TCP receiver, and a proxy. The proxy uses Netfilter to interpose between the sender and the receiver. It applies network address translation to create the illusion that the remote host is the other endpoint of the flow.

by the proxy (step 3). The proxy then looks up the corresponding packet in its buffer, modifies its destination IP address, and forwards it to the receiver (step 4). The receiver responds with a SYN/ACK packet that is captured by the proxy (step 5). The proxy then modifies its source IP address and forwards the packet back to the sender (step 6). Figure 6.2 also shows the details of Monarch's packet address modifications among the sender, the receiver, and the proxy.

All further packet exchanges are handled in a similar manner. If a packet transmitted to `remoteIP` is lost, its response is never received by the proxy, and the corresponding buffered packet is never forwarded to the local receiver. Similarly, reordering of packets sent to the remote host results in the buffered packets being forwarded in a different order. During long transfers, Monarch reclaims buffer space by expiring the oldest packets.

The output from Monarch includes a packet trace similar to the output of `tcpdump`. In addition, it also logs how state variables of the protocol vary over time. For example, using a standard interface of the Linux kernel, our current implementation records TCP state variables, such as the congestion window, the slowstart threshold, and the retransmission timeout. The source code of our Monarch implementation is available from the Monarch webpage at `http://monarch.mpi-sws.org`.

6.2.2 Testing Unmodified Transport Protocol Implementations

Our proxy implementation is completely transparent to our TCP sender and TCP receiver. This is critical to Monarch's ability to test unmodified, complex protocol implementations in the Linux kernel. Further, since both the sender and the receiver run locally, we can easily evaluate the effect of different parameter choices on the sender and receiver for a given transport protocol. For example, Monarch can be used to test how sensitive TCP Vegas [BP95] is to different settings of its α and β parameters when transferring data over different Internet paths. We can also run implementations of different TCP protocols simultaneously to understand how the protocols compete with each other. As we show in Section 6.4, this ability to test protocol implementations under a wide range of experimental conditions can be used by protocol developers to discover errors that affect the performance of their implementations.

Since it is common practice for researchers to test new transport protocols using the `ns-2` simulator [ns2], we added a special interface to Monarch that allows it to connect directly to `ns-2`. Thus, researchers can conveniently use a single `ns-2` code base for both their controlled simulation and live emulation experiments. More details about this feature are available at the Monarch webpage.

6.2.3 PMTU Discovery

TCP uses a technique called path MTU (PMTU) discovery [MD90] to detect the maximum segment size it can use on a given path. Initially, packets are sent using the MTU of the first hop and with the don't fragment (DF) bit set. If the path contains a link with a smaller MTU, the corresponding router drops these packets and returns an ICMP destination unreachable message. The sender then reduces its MTU to the value specified in that message and retries. Since Monarch intercepts and translates ICMP messages, this process works for Monarch flows as expected.

However, PMTU discovery can lead to unwanted losses, especially on paths that have an unusually small MTU. For example, some broadband networks use additional headers and therefore have an MTU of 1,492 or 1,488 bytes, instead of the usual 1,500 bytes. Therefore, some hosts use an alternate method — a special option in the SYN/ACK packet — to reduce the sender's MTU. This method does *not* work with Monarch because it never sends a SYN packet to the remote host and therefore cannot observe the special option. The consequence is that Monarch flows to such hosts must use the slower, ICMP-based method and therefore can have a lower throughput.

We solved this problem by adding a small tool that performs PMTU discovery and then passes the result to Monarch using a command-line option. Monarch can then insert the special option into the SYN/ACK packet. The tool works by sending some large ACKs (first-hop MTU) and some small ACKs (200 bytes) to the remote host, both with the DF bit set. The response is interpreted as follows: If the tool observes a TCP RST for the large ACKs, it returns the first-hop MTU, as it has been shown to work for the entire path. If the tool sees an ICMP destination unreachable message, it returns the first-hop MTU specified in that message[2]. If it sees no response at all, the remote host does not respond to our probes. Otherwise, i.e., if only RSTs for the small ACKs are observed, the ICMP message may have been dropped somewhere along the path. In this case, the tool performs a binary search for the correct PMTU.

6.2.4 Self-diagnosis

Monarch is capable of diagnosing inaccuracies in its own emulated flows based on an analysis of its output. As we discussed earlier, the two primary factors that affect Monarch's accuracy are its inability to distinguish loss and reordering of packets on the upstream and the downstream paths, i.e., the paths from the receiver to the sender and vice-versa. These events are difficult to detect online, but their presence can be inferred after the emulation is finished. Monarch runs a self-diagnosis test after each emulation, which either confirms the results or lists any events that may have affected the accuracy.

[2] This step may have to be repeated for paths on which the MTU decreases more than once; however, we have never observed this in practice

Monarch's self-diagnosis uses the IPID field in the IP headers of the response packets to distinguish between upstream and downstream events. Similar to prior techniques [BS02, MSWA03], including our technique in Chapter 3 to detect cross traffic in our measurements, Monarch's self-diagnosis relies on the fact that many Internet hosts increment the IPID field by a fixed number (typically one) for every new packet they create. However, Monarch's analysis is more involved, as it cannot send any active probes of its own and so must extract the information from a given trace. We describe Monarch's IPID analysis technique next.

IPID analysis to detect upstream loss and reordering

When analyzing a trace of n packets, Monarch extracts two kinds of information: The sequence $R = (r_1, r_2, \ldots, r_k)$, $k \leq n$, in which the response packets have arrived, and the sequence $S = (s_1, s_2, \ldots, s_k)$ of IPIDs in these packets. For example, if Monarch sent four probe packets and the trace yields $R = (1, 2, 4)$ and $S = (1, 2, 3)$, Monarch concludes that the third probe must have been lost on the downstream path because probe #4 was the third to arrive at the remote host. If S=(1,2,4), Monarch concludes that the third response was lost on the upstream path because the third probe has evidently reached the remote host and consumed an IPID.

In the following description, we will initially assume that there is no cross-traffic, i.e., that the remote host does not send packets to any other host while the Monarch measurement is in progress. We will relax this assumption later.

Upstream packet loss. To make the problem tractable for long sequences, Monarch first identifies all *in-order* packets, i.e. packets for which neither losses nor reordering have occurred. In-order packets are important because they separate different loss or reordering episodes, which can then be treated independently. Intuitively, response i is in-order if all responses lower than i were received before i, and all responses higher than i were received after i. Formally, i is in-order iff

$$\forall j < i \; ((r_j < r_i) \wedge (s_j < s_i)) \quad \wedge \quad \forall j > i \; ((r_j > r_i) \wedge (s_j > s_i))$$

A special case occurs if the last packet is not in-order. In this case, we append a virtual probe with $r_{k+1} = 1 + \max r_i$ and $s_{k+1} = 1 + \max s_i$, which is guaranteed to be in-order.

Now Monarch considers all pairs p, q of responses, where q is the first in-order response after p. For each pair, it computes the number of probes sent between p and q, $n = q - (p + 1)$, the number of losses, $l = r_q - (r_p + 1)$, and the number of gaps in the IPID sequence, $g = s_q - (s_p + 1)$. We can distinguish five possible cases:

- $n = 0$: Neither losses nor reordering occurred.
- $n > 0$, $l = g = 0$: Reordering, but no loss.
- $n > 0$, $l = n$, $g = 0$: Downstream losses.
- $n > 0$, $l = n$, $g = n$: Upstream losses.
- *Otherwise:* A combination of g upstream losses and $l - g$ downstream losses has occurred.

In the last case, we cannot distinguish between upstream and downstream losses on a per-packet basis. When this happens, Monarch displays an error message to indicate that self-diagnosis cannot be performed. In practice, this affects only 1–2% of all flows.

Upstream packet reordering. The IPID analysis can also be used to distinguish between upstream and downstream reordering. Consider again a pair p, q of successive in-order responses, and assume for now that no losses have occurred. Then we know that the corresponding probes were sent in order $(r_p, r_p + 1, r_p + 2, \ldots, r_q)$, reached the remote host in order $(r_p, r_{p+1}, r_{p+2}, \ldots, r_q)$, and that the responses were seen by Monarch in order $(p, p + 1, p + 2, \ldots, q)$. Therefore, we can reorder the corresponding TCP packets accordingly.

Figure 6.3 shows an example. By sorting the packets in the white area by their IPIDs (s_i), we can conclude that probe packets (3,4,5) arrived at the remote host in the order (5,4,3), so packets 3 and 5 must have been swapped on the downstream path. As shown by the arrival sequence (r_i), the responses arrived at the Monarch host in the order (4,5,3), so responses 4 and 5 must have been swapped on the upstream path.

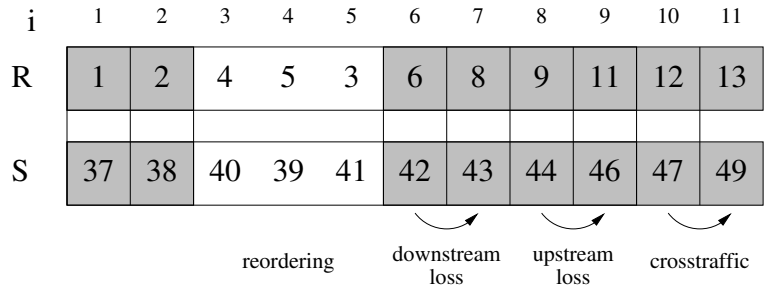


Figure 6.3: IPID analysis example. The probe packets are sent with increasing sequence numbers. R_i is the sequence number of the i^{th} response packet that was received and S_i is the IPID of the packet. Monarch uses S_i to infer the order in which probes have arrived at the remote host.

Not all upstream reordering changes TCP's behavior in a significant way. For example, if the packet exchange in Figure 6.3 had taken place in a real TCP transfer, packets 3–5 would have triggered two duplicate ACKs, since packet 3 was the last to arrive at the remote host. Therefore, TCP would have triggered a retransmission, whether or not the two duplicate ACKs were additionally reordered on the upstream. Monarch takes this into account by labeling upstream reordering episodes as either significant or insignificant. An episode is labeled insignificant if it would have caused the same number of duplicate ACKs in both the Monarch flow and an actual TCP flow.

Cross-traffic. So far, we have assumed that there was no cross-traffic, i.e., that the remote host has not sent any packets to other hosts during the measurement. When cross-traffic does occur, the remote host uses some IPIDs for that traffic, which causes gaps in the IPID sequence and thus increases g in the above analysis. In most cases, this can be detected from the fact that $g > n$. However, if cross-traffic and losses occur simultaneously, some

downstream losses may be classified as upstream losses. Fortunately, the reverse error cannot occur, so Monarch's self-diagnosis is conservative as it can only overestimate the number of inaccuracies in a trace.

Impact of upstream loss and reordering

Upstream packet loss and reordering events affect different transport protocols in different ways. For example, TCP Reno is more strongly influenced by packet loss than packet reordering. Even a single upstream packet loss confused as a downstream packet loss causes TCP Reno to retransmit the packet and halve its future sending rate. On the other hand, only packet reordering on a large scale can trigger retransmissions that affect future packet transmissions in a significant way.

Self-diagnosis tries to estimate the impact of upstream loss and reordering on Monarch flows. This impact analysis depends on the specific transport protocol being emulated. Here, we focus on the analysis we developed for TCP Reno; but similar analysis techniques can be developed for other protocols. Our impact analysis for TCP Reno labels a flow as inaccurate if it sees an upstream packet loss or significant upstream packet reordering that causes packet retransmission; and it confirms the accuracy of all Monarch traces that see no upstream packet loss and no significant upstream packet reordering.

We note that *confirmation* of a Monarch trace by our analysis does not imply that the trace is accurate for *all* usage scenarios. It merely suggests that the trace is likely to be accurate for *many* uses. For example, a Monarch trace that suffers only minor reordering would be confirmed. Such a trace would be accurate with respect to its throughput, latency, or packet loss characteristics, but not with respect to its reordering characteristics.

Output

After detecting upstream events and analyzing their impact, Monarch broadly classifies the result of an emulation as either confirmed, inaccurate, or indeterminate. We illustrate the decision process in Figure 6.4 and discuss it below.

- **Indeterminate:** Results in this category do not contain enough information for Monarch to distinguish upstream events (loss or reordering) from downstream events in all cases. This can happen when downstream losses and upstream losses occur very close together, or when the IPIDs in the response packets are unusable because the remote host randomizes them, or sets the field to zero.

- **Inaccurate:** Monarch warns that its results could be inaccurate when it detects any upstream packet losses, *or* when the observed upstream packet reordering is significant.

- **Confirmed:** In all other cases, Monarch has not detected any upstream losses or significant reordering events. Therefore, it confirms its output.

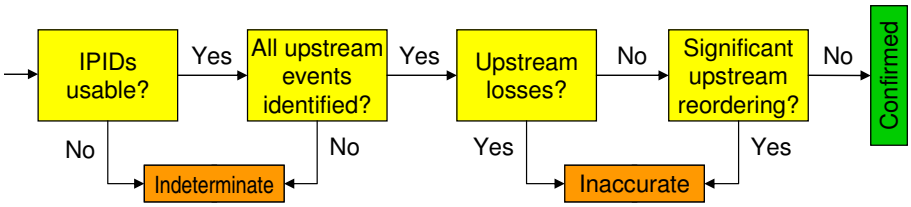


Figure 6.4: Self-diagnosis in Monarch for TCP Reno. The result is confirmed only if no known sources of inaccuracy are present.

Rate-limited responses

In addition to the loss and reordering analysis, Monarch also scans the entire trace for long sequences of packet losses to identify hosts that rate-limit their responses. For example, in our measurements, we observed that some hosts stop sending responses after a certain number of probes, e.g., after 200 TCP ACKs, which could be due to a firewall somewhere on the path. This pattern is easy to distinguish from packet drops due to queue overflows because in the latter case, packet losses alternate with successful transmissions. However, it is hard to distinguish losses due to path failures from end host rate-limiting.

6.2.5 Usage Concerns and Best Practices

Monarch has similar usage concerns as many other active measurement tools, including our measurement methodology from Chapter 3.

Large-scale experiments using Monarch can raise potential security concerns. Internet hosts and ISPs could perceive Monarch's traffic as hostile and intrusive. To address this concern, Monarch includes a custom message in the payload of every probe packet. We use the message to explain the goals of our experiment, and to provide a contact email address. We have conducted Monarch measurements to several thousand end hosts and routers in the Internet in hundreds of commercial ISPs over a period of seven months without receiving any emails about security alarms triggered by our experiments.

Another cause of concern is that Monarch can be used to send large amounts of traffic to a remote host. This can be a great inconvenience to remote hosts on broadband networks that use a per-byte payment model, where any unsolicited traffic costs the host's owner real money. To mitigate this concern, we only measure hosts in broadband ISPs that offer flat rate payment plans. In addition, we never transfer more than a few dozen megabytes of data to any single Internet host.

Finally, we would like to point out that Monarch flows compete fairly with ongoing Internet traffic as long as the emulated transport protocols are TCP-friendly.

6.3 Evaluation

In this section, we present three experiments that evaluate Monarch's ability to emulate transport protocol flows. First, we evaluate the accuracy of its emulated flows, i.e., we verify how closely the characteristics of Monarch flows match those of actual TCP flows. Second, we identify the major factors and network conditions that contribute to inaccuracies in Monarch's emulations, and show that Monarch's self-diagnosis can accurately

quantify these factors. Third, we characterize the prevalence of these factors over the Internet at large.

6.3.1 Methodology

Evaluating Monarch's accuracy over the Internet at scale is difficult. To evaluate Monarch, we need to compare its emulated flows to real transport flows over the same Internet paths. Unfortunately, generating real transport flows requires control over both end hosts of an Internet path. In practice, this would limit our evaluation to Internet testbeds, such as PlanetLab. We deal with this limitation using the following three-step evaluation:

1. In Section 6.3.2, we evaluate Monarch over the PlanetLab testbed. We generate both Monarch flows and real TCP flows, identify potential sources of error, and study how they affect accuracy.

2. In Section 6.3.3, we show that Monarch's offline self-diagnosis can accurately detect these errors from its own traces.

3. In Section 6.3.4, we use this self-diagnosis capability to estimate the likelihood of error in Monarch measurements over a wide variety of Internet paths.

Data collection

We used Monarch to emulate transport flows over three types of Internet paths: (a) paths to PlanetLab nodes, (b) paths to Internet hosts located in commercial broadband ISPs, and (c) paths to Internet routers. Table 6.3 shows statistics about the three datasets we gathered. All measurements involved 500 KBytes data transfers. The TCP senders were located in four geographically distributed locations, three (Seattle, Houston and Cambridge) in the U.S. and one (Saarbrücken) in Germany. The receivers included PlanetLab nodes, broadband hosts, and Internet routers. While gathering the PlanetLab dataset, we controlled both endpoints of the Internet paths measured, so we generated both Monarch and normal TCP flows. In the other two datasets we only controlled one endpoint, so we generated only Monarch flows.

Our *PlanetLab* measurements used 356 PlanetLab nodes worldwide as receivers. To each PlanetLab node, we conducted five data transfers using Monarch interspersed with

	PlanetLab	**Broadband**	**Router**
Sender nodes	4	4	4
Receiver nodes	356	4,805	697
Successful measurements	12,166	15,642	2,776

Table 6.3: Traces used for our Monarch evaluation. For each trace, we used geographically dispersed sender nodes in Seattle (WA), Houston (TX), Cambridge (MA), and Saarbrücken (Germany).

five normal TCP transfers in close succession[3], for a total of ten data transfers from each of the senders in Seattle, Houston, Cambridge, and Saarbrücken. We ran `tcpdump` on both the sending and the receiving node to record packet transmissions in either direction.

Our *Broadband* measurements used 4,805 cable and DSL end hosts in 11 major commercial broadband providers in North America and Europe. We used the same ISPs as for our broadband characterization study in Chapter 3 and selected the hosts by probing hosts measured in a previous study of Napster an Gnutella [SGG02] and used their DNS names to identify the ISPs.

Our *Router* measurements used 1,000 Internet routers we discovered by running `traceroute` to hosts in the broadband data set. Only 697 of these routers responded to Monarch's probe packets.

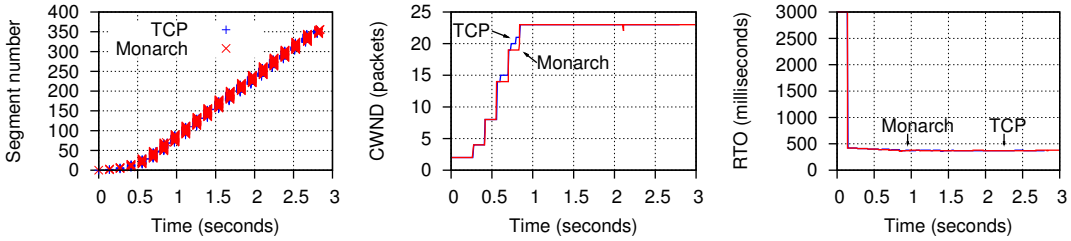
6.3.2 Accuracy over PlanetLab

In this section, we compare the behavior of Monarch flows to TCP flows on Internet paths to PlanetLab nodes. We focus on two different aspects. First, we investigate whether the packet-level characteristics of the emulated flows closely match those of TCP flows. For this, we analyze the sizes and transmission times of individual packets. Second, we compare their higher-level flow characteristics, such as throughput and overall loss rate.

Packet-level characteristics

Monarch emulates transport protocol flows at the granularity of individual packet transmissions. In this section, we compare the packet-level characteristics of Monarch and TCP flows to show that they closely match in terms of number of packets, sizes of packets, packet transmission times, and evolution of important protocol state variables.

We begin by comparing two flows on a single Internet path: one emulated with Monarch and one actual TCP flow. Figure 6.5a shows the times when individual data segments were transmitted. The graph shows that the transmission times of packets in the Monarch flow are almost indistinguishable from those in the TCP flow.



(a) Data segments **(b)** Congestion window **(c)** Retransmission timeout

Figure 6.5: Comparison between typical Monarch and TCP flows. The figures show (a) when each segments was sent and (b) how the congestion window and (c) the retransmission timeout evolved over time. The plots for Monarch and TCP are so close that they are almost indistinguishable.

[3]We also ran the Monarch and TCP flows concurrently, and compared them. The results were similar to those we obtained when we ran Monarch and TCP alternately in close succession. Hence, we show only results from the latter.

Next, we examine how TCP protocol state variables change during the flows. Figure 6.5b and Figure 6.5c show a plot of the congestion window (CWND) and the retransmission timeout (RTO) for both flows. This information is recorded in Monarch's output trace using the `TCP_INFO` socket option. Figure 6.5b shows the typical phases of a TCP flow, such as slowstart and congestion avoidance. Both flows went through these phases in exactly the same way, including the RTO changes shown in Figure 6.5c. The TCP variables of the Monarch flow closely match those of the actual TCP flow, suggesting a highly accurate Monarch emulation.

Next, we compare the properties of aggregate data from all our PlanetLab traces. We begin by comparing the number of packets sent and received in the Monarch flows and their corresponding TCP flows. Figure 6.6 shows the relative difference for each direction, using the number of packets in the TCP flow as a basis. 65% of all flow pairs sent the same number of packets; for 93% of all flow pairs the number of packets differs by less than 5%. This is expected because (a) both Monarch and TCP use packets of the same size, and (b) both flows transfer the same 500 KBytes of data. Moreover, when we compare only flows with no packet losses, the difference disappears entirely (not shown). This suggests that packet losses account for most of the inaccuracies in the number of packets sent.

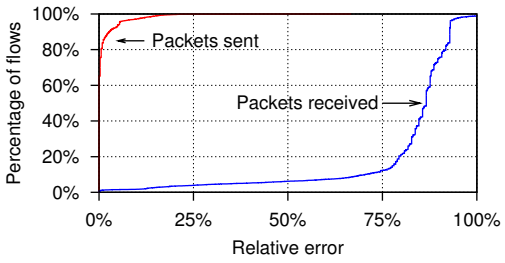


Figure 6.6: Traffic generated by Monarch and TCP. Relative difference between the number of packets sent and received, shown as cumulative distributions. Values greater than 0 indicate that Monarch sent/received more packets than TCP.

Figure 6.6 shows a substantial difference in the number of packets received in the upstream direction. This is due to delayed ACKs: TCP flows acknowledge several downstream packets with a single upstream packet, while Monarch flows contain one response packet for every probe. However, acknowledgment packets are small (typically 40 bytes), and we will show later that this additional traffic has little impact on high-level flow characteristics, such as throughput.

Finally, we repeat our analysis of packet transmission times on a larger scale, across all PlanetLab traces. Our goal is to check whether the rates and times at which packets are transmitted are similar for TCP and Monarch flows.

We compare the times taken to transfer 10%, 30%, 50%, 70%, and 90% of all bytes in the 500 KBytes transfer. Figure 6.7 shows the difference between times taken to complete Monarch and TCP traces relative to the TCP traces. The error stays small for every part of the transfer, suggesting that packets are sent out at similar rates during the flows' lifetimes.

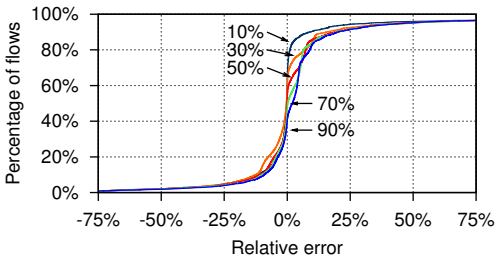


Figure 6.7: Progress of TCP and Monarch flows. Relative error of the time it took to complete a certain fraction of the transfer between pairs of TCP and Monarch flows, shown as a cumulative distribution.

Interestingly, transmission times match more closely at the beginning of the transfer than towards the end. For example, Monarch flows tend to take slightly more time to transfer 90% of their payload than the TCP flows. As we explain in detail later, this is because Monarch flows experience slightly higher loss rates than TCP flows.

To summarize, we find that Monarch and TCP flows match with respect to several packet-level characteristics, including the number and sizes of packets sent, the evolution of important protocol state variables, and the transmission times of individual segments.

Flow-level characteristics

In this section, we investigate whether Monarch and TCP traces are similar with respect to several high-level flow characteristics, such as throughput, RTTs, queueing delay, and packet loss.

Throughput. Figure 6.8 shows the cumulative distributions of the throughput for Monarch and TCP flows using our PlanetLab experiment. While the lines for Monarch and TCP match well, Monarch flows tend to have a slightly lower throughput than TCP flows. The figure also shows a second pair of lines, representing only flows without packet losses and retransmissions. Interestingly, these lines show almost no difference between Monarch and TCP. This suggests that the small errors in Monarch's throughput estimates might be due to packet losses.

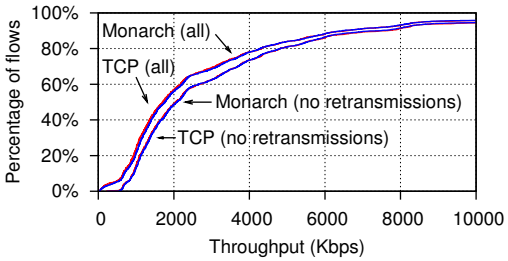


Figure 6.8: Throughput comparison between Monarch and TCP flows. The cumulative distributions are very close; if only flows without packet retransmissions are considered, the distributions are indistinguishable.

To quantify this error, Figure 6.9 shows the relative throughput difference in pairs of consecutive Monarch and TCP flows, using TCP's throughput as a base (recall that we took ten measurements on each path, alternating between Monarch and real TCP flows). In over 50% of the flow pairs, the throughput of the Monarch flow differs from the throughput of the TCP flow by less than 5%, which is a good match. However, not all these differences are due to inaccuracies in Monarch. Figure 6.9 also shows the throughput differences between two consecutive TCP flows along the same paths. The two plots are similar, suggesting that the dominant cause of these differences is unstationarity in the network, e.g., fluctuations in the amount of competing traffic.

Thus, while packet losses can cause Monarch to underestimate the throughput in general, their impact is fairly small, often smaller than the impact of the unstationarity in network path properties during the course of the flow.

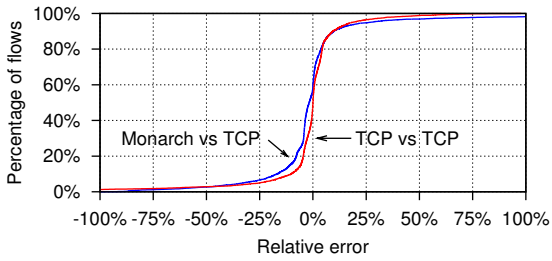


Figure 6.9: Relative throughput error between pairs of TCP and Monarch flows, and between pairs of TCP flows. The error is similar, which suggests that its primary cause is unstationarity in the network, and not a problem with the Monarch emulation.

Latency. Next, we focus on the latencies and delays experienced by packets during Monarch and TCP flows. We compute three types of packet latencies or RTTs: minimum RTT, maximum RTT, and queueing delay. To remove outliers, we take the maximum RTT to be the 95th percentile of all packet RTTs, and compute the queueing delay as the difference between maximum and minimum RTTs. Figure 6.10 shows the difference in the estimates of RTTs between Monarch and TCP traces, as a percentage of TCP estimates. We also show how estimates from successive measurements of TCP flows differ from each other.

There are two take-away points from Figure 6.10. First, Monarch's estimates of minimum and maximum RTT closely match the TCP estimates. In fact, Monarch's errors are indistinguishable from the variation observed in the estimates between successive TCP measurements along the same paths. This points to the efficiency of our Monarch implementation: despite the additional packet processing overhead in our interposing proxy, we add negligible overhead to the packet latencies. Second, queueing delays show much larger variation or unstationarity over time compared to minimum and maximum RTTs. The reason for these large relative differences is that the absolute values are very low.

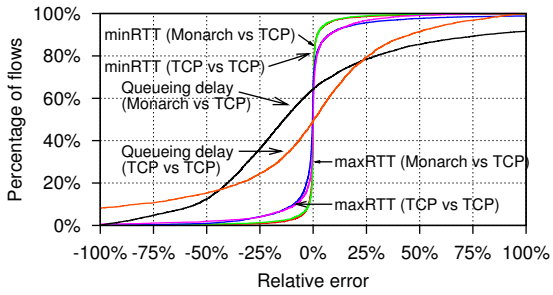


Figure 6.10: Relative RTT difference between successive TCP and Monarch flows. The error in extremely small, except for the queueing delay. Queueing delay was generally low in the PlanetLab trace, which is why even small variations lead to big relative errors.

Over 76% of queueing delay estimates are below 10 milliseconds. Hence, even a small 1-millisecond variation corresponds to a 10% difference.

Packet loss. Finally, we investigate the loss rates in the flows. We note that both Monarch and TCP senders retransmit packets that they *perceive* to be lost, which might be different from the packets that were *actually* lost. For example, TCP might mistake massive packet reordering for a loss and trigger a retransmission. Our interest here is in the perceived loss rates of these flows, so we use the packet retransmission rate for loss rate.

Figure 6.11 shows cumulative distributions of retransmission rates for both Monarch and TCP flows. 75% of all Monarch flows and 88% of all TCP flows do not contain any retransmissions and therefore do not perceive packet loss. Thus, packet retransmissions do not affect a majority of both Monarch and TCP flows. Of the flows that do contain retransmissions, Monarch clearly shows a higher retransmission rate than TCP. This is expected because Monarch flows must retransmit packets for losses in both upstream and downstream directions, while TCP needs to retransmit only packets lost on the downstream, due to cumulative acknowledgments.

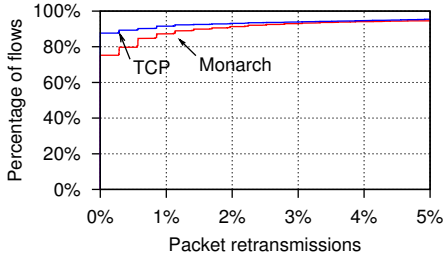


Figure 6.11: Retransmissions per flow for Monarch and TCP. Monarch shows more retransmissions because it must retransmit packets for losses in both upstream and downstream directions, while TCP needs to retransmit only packets lost on the downstream.

Summary. Our analysis shows that Monarch can accurately emulate TCP flows with respect to flow-level properties such as throughput, latency, and queueing delay. However, Monarch's inability to distinguish between upstream and downstream packet loss causes it to over-estimate packet loss. The impact of this inaccuracy is limited to the small fraction of flows that experience upstream packet loss.

6.3.3 Reliability of Self-diagnosis

In the previous section, we showed that the primary source of inaccuracy in a Monarch emulation is upstream packet loss. In this section, our goal is to show that Monarch's self-diagnosis feature (Section 6.2.4) can reliably detect upstream packet loss, and thus, warn the user of potential inaccuracies.

We tested this feature on the Monarch flows in our PlanetLab trace. For each flow, we compared the `tcpdump` traces from the sender and the receiver to determine how many packets had actually been lost on the downstream path and the upstream path. Then we compared the results to the output of Monarch's self-diagnosis for that flow; recall that this uses only the sender-side trace.

Figure 6.12 shows the results: only for a very small number of flows (less than 2%) self-diagnosis could not distinguish between all upstream and downstream losses. In these cases, Monarch printed a warning. For the majority of flows for which self-diagnosis could infer the loss rates, the measured and the inferred loss rates match extremely well in both the upstream and downstream directions. As expected, the measured and inferred total loss rate plots are identical.

We conclude that Monarch's self-diagnosis can reliably detect upstream loss, the major source of inaccuracy in an emulated flow.

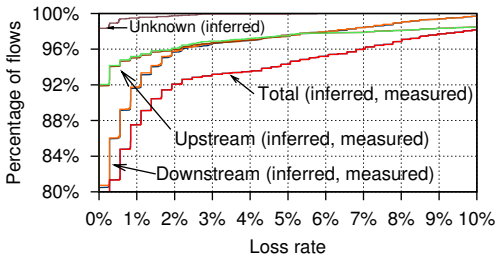


Figure 6.12: Self-diagnosis is accurate. The number of downstream and upstream losses inferred by Monarch's self-diagnosis matches the actual measurements. Only a small number of losses cannot be classified as either downstream or upstream.

6.3.4 Accuracy over the Internet at Large

In the previous two sections, we showed that upstream loss is the most important source of inaccuracies in Monarch emulations, and that Monarch's self-diagnosis can reliably detect the presence of upstream loss. Our goal in this section is to show that upstream losses are rare even when Monarch is used over real Internet paths.

We ran Monarch's self-diagnosis over our two Internet traces. The first trace consists of 15,642 flows to 4,805 broadband hosts and the second trace contains 2,776 flows to 697 Internet routers. Table 6.4 summarizes our results. About 10% of the traces could not be analyzed by Monarch. In the broadband dataset, 7.1% of the traces did not contain usable IPIDs (8.3% in the router dataset), and 1.5% (2.3%) contained a loss that could not be classified as either upstream or downstream. In either of these cases, self-diagnosis was aborted immediately.

Result	**Broadband**		**Router**	
Accurate	13,168	84.2%	2,317	83.5%
Inaccurate	1,130	7.2%	164	5.9%
Indeterminate	1,344	8.6%	295	10.6%
Traces total	15,642	100.0%	2,776	100.0%

Table 6.4: Monarch is accurate over real Internet paths. In the majority of the flows, self-diagnosis did not detect any inaccuracies. Note that even when an inaccuracy is reported, its impact on flow-level metrics such as throughput may be quite small.

Overall, 84.2% of the broadband traces and 83.5% of the router traces were marked as accurate by self-diagnosis because neither upstream losses nor significant reordering were detected. This accurate traces include the 15.8% (24.9% in the router dataset) of the traces that contained only minor reordering that would not have changed the number of duplicate ACKs, and therefore would not have affected any packet transmissions. Only 7.2% of the broadband traces were reported as inaccurate; for the router traces, the fraction was only 5.9%.

We conclude that a majority of our flows to Internet hosts did not suffer from upstream packet loss or significant reordering misinterpreted as packet loss by TCP congestion control. This suggests that Monarch can be used to accurately emulate TCP flows to a large number of Internet hosts. Moreover, our results show that the IPID-based self-diagnosis is applicable in most cases.

6.3.5 Summary

In this section, we showed that Monarch is accurate: its emulated TCP flows behave similarly to real TCP flows with respect to both packet-level and flow-level metrics. We also showed that the most important source of error in Monarch's flows is upstream packet loss, and that this can be reliably detected by Monarch's built-in self-diagnosis. Further, our examination of large sets of Monarch flows to various Internet hosts, including hundreds of routers and thousands of broadband hosts, revealed that less than 10% of these flows suffer from upstream packet loss.

6.4 Applications

Monarch's ability to evaluate transport protocol designs over large portions of the Internet enables new measurement studies and applications. We used Monarch to conduct two

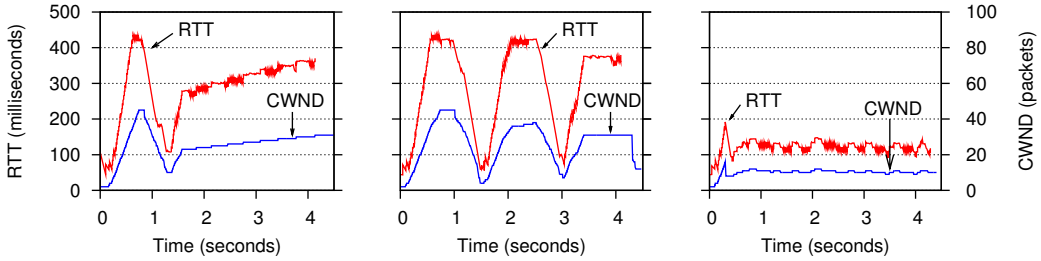
different types of measurement experiments. In this section, we describe these experiments and present some preliminary results from them to illustrate their potential benefits.

6.4.1 Evaluating Different Transport Protocols

New transport protocol designs continue to be proposed as the Internet and its workloads change over time [VKD02, XHR04, ACKZ06, Flo03]. However, even extensive simulation-based evaluations face skepticism about whether their results would translate to the real world. The resulting uncertainty how well these protocols would compete with existing deployed protocols hinders their actual deployment. With Monarch, researchers can evaluate their new protocol designs over actual Internet paths.

We used Monarch to compare three different TCP congestion control algorithms implemented[4] in the Linux 2.6.16.11 kernel: NewReno [FHG04], BIC [XHR04], and Vegas [BP95]. In our experiment, we emulated 500 KByte data transfers from a local machine to several hosts in broadband (cable and DSL) ISPs, using each of the three congestion control algorithms in turn[5]. We examined the traces generated by Monarch for differences in protocol behavior.

Figure 6.13 shows the difference between the algorithms over a single, but typical DSL path. The graphs show how the CWND and the RTT evolve over the duration of the transfer. All flows begin in the slow-start phase, where the CWND increases rapidly until the flow loses a packet and enters the congestion avoidance phase. The TCP NewReno graph shows that the RTT increases from 44 ms at the beginning to well over 300 ms before it loses a packet. This suggests the presence of a long router queue at the congested link on this broadband Internet path. TCP BIC, which has been adopted as the default TCP protocol by Linux since kernel version 2.6.7, shows a similar pattern but ramps up the congestion window much faster after each loss, which results in even higher queueing



(a) TCP NewReno **(b)** TCP BIC **(c)** TCP Vegas

Figure 6.13: Comparing the performance of different TCP protocols over an Internet path between a host in Germany and a host in the BT Broadband DSL network. Variation in packet RTTs and the congestion window (CWND) over the duration of a flow using three different TCP protocol variants. The steep drops in RTT and CWND values are due to packet losses. Compared to Reno, BIC shows higher RTTs and losses, while Vegas shows lower RTTs and losses.

[4]Note that the Linux implementation of TCP protocols may differ significantly from their reference implementation or their standard specification.

[5]In Linux 2.6 kernels, it is possible to switch between different TCP congestion control algorithms at runtime.

delays and packet losses. In contrast to NewReno and BIC, Vegas enters a stable state with a RTT of about 100 ms without suffering a single loss.

Our experiment shows that TCP BIC, the default congestion control algorithm in Linux, exhibits the worst performance both in terms of packet delay and packet loss. This is not particularly surprising because BIC is designed for Internet paths that have a high bandwidth-delay product, but our measurement path includes a broadband link with relatively low bandwidth. However, since many hosts today use broadband Internet connections, it is important to improve BIC's performance over broadband networks or choose a different, better performing TCP variant for broadband networks.

Our Monarch results, while preliminary, show the importance of understanding the behavior of new protocols over a variety of real network paths before deploying them widely.

6.4.2 Testing Complex Protocol Implementations

Modern transport protocols (e.g., TCP NewReno with fast retransmit and recovery) are so complex that it is often difficult to implement them correctly. While program analysis techniques [ME04] could help debug functionally incorrect implementations, it is important to test the performance of these protocols in the real world to find performance problems. Monarch is particularly useful for testing protocols because it can run complete and unmodified protocol implementations.

We used Monarch to emulate TCP flows to several different types of hosts, including broadband hosts and academic hosts. In this process, we discovered bugs in the Linux TCP stack that tend to manifest themselves frequently over certain types of Internet paths. For example, we found that the Linux 2.6.11 implementation of Fast Recovery [FHG04] can cause the congestion window to collapse almost entirely, instead of merely halving it. This problem can severely reduce throughput, and it occurs repeatedly over paths to DSL or cable hosts.

The purpose of Fast Recovery is to allow the TCP sender to continue transmitting while it waits for a retransmitted segment to be acknowledged. Linux uses a variant known as rate halving [SK02], which transmits one new segment for every other ACK received. Thus, one new packet is sent for every two packets that leave the network. Under normal conditions, this has the effect of gradually decreasing the number of packets in flight by half. Linux 2.6.11 implements rate halving by estimating the number of packets in flight, and capping the congestion window at that number. Normally, this has the desired effect because after two ACKs, one new packet is sent and two old packets are known to have left the network; hence, the number of packets in flight, and thus the congestion window, is decreased by one.

However, we found that this approach fails when the congestion window approaches the send buffer size. Figure 6.14 shows an example of a flow that saw its first loss after 0.6 seconds, when the congestion window was 36 packets wide. Initially, Linux was able to send 8 additional segments for every other ACK as expected. But, once it reached the default send buffer size of 64 KBytes (44 packets), it could not transmit more new segments. After this point, with no new segments being transmitted, the number of packets in flight, and consequently the congestion window, decreased rapidly. *Every* incoming ACK reduced the congestion window by one packet, causing it to fall far below the slowstart threshold of 18 packets. Thus, after leaving Fast Recovery, Linux fell back into slowstart for over

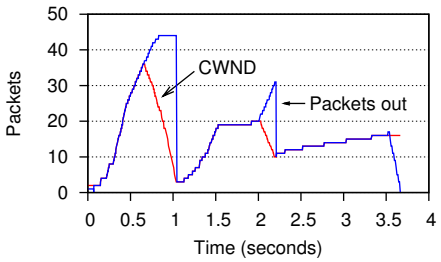


Figure 6.14: Incorrect rate halving in Linux TCP. After the first loss at about 0.7 seconds, the CWND falls below half its original value.

half a second. Note that a second loss at 2.0 seconds was handled correctly because the congestion window was still fairly small.

Monarch helped us discover this problem because it allowed us to test the complete and unmodified protocol implementation (in this case, the NewReno code in the Linux kernel) over a wide range of real links with different characteristics.

6.5 Summary

Monarch emulates transport protocol flows over live Internet paths enabling the evaluation of transport protocols in realistic environments, which complements the controlled environments provided by the state of the art network simulators, emulators or testbeds. Monarch is highly accurate: its emulated flows closely resemble TCP flows in terms of throughput, loss rate, queueing delay, and several other characteristics. By relying on minimal support from the remote host, Monarch enables researchers to evaluate protocols on an unprecedented scale, over millions of Internet paths.

Our preliminary study on the performance of different congestion control algorithms (TCP Reno, TCP Vegas, and TCP BIC) shows that much remains to be understood about the behavior of even widely adopted protocols over the Internet at large. Based on our experience, we believe that Monarch can help the research community conduct large-scale experiments leading to new insights and findings in the design and evolution of Internet transport protocols.

Monarch inspired BlueMonarch [SSW09], a tool that enables realistic evaluation of applications in bluetooth networks at scale. BlueMonarch uses Bluetooth Service Discovery requests as probes to emulate transfers to remote devices without the need to deploy new software on them or to have actual physical access to them.

Monarch allows the evaluation of transport protocols to hosts on the Internet without their cooperation. While this uncooperative approach enables large-scale evaluation experiments, Monarch's methodology is limited to emulating TCP flows and only downloading to the remote hosts. In the next chapter we present SatelliteLab that removes these limitations as it provides a heterogeneous testing environment for protocol and systems evaluation under realistic conditions. However, as SatelliteLab uses a cooperative approach, its scale is limited by the ability to attract nodes to become part of the testbed.

7 SatelliteLab: Adding Heterogeneity to Planetary-scale Testbeds

Internet testbeds, such as PlanetLab [Pla] and RON [ABKM03], have become indispensable for evaluating networking and distributed systems research. Researchers deploy prototypes of new systems over these testbeds and study their performance to estimate how well these systems would work over the real Internet. PlanetLab, the state-of-the-art Internet testbed, has been used by over a thousand researchers for evaluating several hundred Internet-scale distributed systems, including P2P systems [RD01], routing, multicast, and overlays [CDK+03, ABKM03, SSBK03], content distribution networks [WPP02], as well as for network measurements [SWA03].

Here, we focus on a widely recognized problem with existing network testbeds: they lack the heterogeneity that characterizes the commercial Internet [SPBP05]. Most nodes in existing testbeds are located in well-connected academic and corporate networks, and the network paths between them are often restricted to well-provisioned research backbones [BGP04]. This is in sharp contrast to the Internet, where most end nodes connect via a diverse set of edge networks, such as residential broadband, wireless, and cellular networks. Prior studies have shown that the paths in PlanetLab have very different characteristics from the paths in the commercial Internet [BGP04, PHM06]. More alarmingly, researchers have found that the behavior of some systems can vary considerably between Internet and testbed deployments [LGS07].

Our basic idea is to improve the heterogeneity of current testbeds by recruiting nodes from the Internet edges. These nodes can be desktops, laptops, or handhelds, and they can be connected to the Internet via residential broadband, wireless, or cellular networks. However, adding edge nodes with diverse resource constraints to existing testbeds creates new challenges. Current testbeds are designed to be hosted by large institutions in academia and industry, and testbed administrators expect dedicated, well-provisioned nodes that are connected to the Internet via high-speed networks. For example, PlanetLab nodes are required to be server-class machines, they must run the PlanetLab OS, and they must be configured with static, public IP addresses [Pla02]. While these requirements make the testbed easy to manage and easy to use, they also present a high barrier to entry for nodes in Internet edge networks, many of which are hosted and managed by individual end users. For instance, most edge nodes are not server-class machines, and many cannot get a public, static IP address from their ISPs.

The need to support nodes with a wide range of hardware, software, networking, and management resources requires us to fundamentally rethink existing testbed designs. While current architectures treat all nodes equally, we propose a hierarchical testbed architecture that assigns different roles to nodes based on their available resources.

We present the design, implementation, and evaluation of SatelliteLab, a highly heterogeneous testbed that includes nodes from a diverse set of edge networks. SatelliteLab has a two-tier architecture. The nodes in the top tier are called *planets*; they play the role of

classical well-provisioned testbed nodes. The nodes in the bottom tier belong to a new class of light-weight testbed nodes called *satellites*. In existing testbeds, each node performs two tasks: it executes application code, and it forwards traffic over its access links. The key insight behind SatelliteLab's design is to separate these two tasks. Satellites do not run any application code; they delegate this task to well-provisioned planets. Instead, satellites collaborate with the planets to detour traffic along the Internet links to which the satellites are connected. SatelliteLab's routing design thus subjects traffic to the network conditions that it would experience if the application was run on the satellites. In this way, satellites improve the heterogeneity of a testbed by contributing access to edge links without having to contribute any resources for code execution.

SatelliteLab's architecture allows us to leverage existing testbed infrastructures like PlanetLab by augmenting them with satellite nodes from the Internet edges. It also significantly lowers a testbed's barrier to entry while preserving its ease of use and ease of management. We show that an end host can support the limited routing functionality of a satellite by running just a few hundred lines of Java code. Researchers who have been using the existing testbed do not need to modify their application code to use SatelliteLab; they only need to specify which satellites should forward their application traffic. Finally, because satellites do not execute application code, they do not need to be managed or monitored by testbed administrators.

To understand the extent to which our design facilitates recruiting of end hosts in edge networks, we implemented SatelliteLab as an extension to the popular PlanetLab testbed. Within a period of two weeks, we were able to recruit 32 satellite nodes from our friends and colleagues, who were willing to donate their spare network resources to our experiments. These nodes included desktops, laptops, and handhelds that were being actively used by our contributors, and they were connected to the Internet through a variety of residential cable, DSL, ISDN, Wi-Fi, Bluetooth, and cellular links. For comparison, only five out of more than 800 PlanetLab nodes are located in such edge networks. Our experience suggests that SatelliteLab can potentially scale to thousands of nodes if the several hundred researchers using PlanetLab collaborate to grow the testbed.

We used our preliminary deployment to evaluate SatelliteLab's design. Our results show that, despite the limited functionality supported by the satellites, SatelliteLab can forward traffic from testbed experiments over heterogeneous edge networks. Further, we demonstrate that SatelliteLab can be used to conduct realistic experiments in the radically heterogeneous network environments that comprise today's Internet. We illustrate our testbed's benefits by presenting a brief evaluation of network coordinate and overlay multicast systems in residential broadband environments, as well as a preliminary measurement study of UMTS cellular networks.

7.1 Challenges and Requirements

In this section, we discuss the challenges in enabling arbitrary end hosts, including resource-constrained nodes in Internet edge networks, to be used for testbed experiments. From this discussion, we derive two primary requirements that our SatelliteLab design must satisfy.

Our basic design goal is to preserve the numerous benefits of existing testbeds. For example, PlanetLab provides experimenters with a stable software environment, supports

complete management of private virtual slices, and offers an extensive API on top of which useful distributed services can be built. We share all of these goals, and additionally want to support heterogeneous edge nodes. In the remainder of this section, we will outline the important challenges and requirements particular to this last goal.

7.1.1 Challenges

Recruiting volunteer nodes from the edge of the Internet imposes challenges to SatelliteLab's design that are fundamentally different from the challenges faced by existing testbeds such as PlanetLab. We describe three challenges unique to SatelliteLab below.

1. Edge nodes provided by volunteers are not dedicated testbed nodes. Most of the edge nodes we recruited for our testbed (see Table 7.1) were personal computers owned by friends and colleagues, who were willing to forward experiment traffic in the background using their spare resources. Based on our experience, we believe that volunteers will resist giving up administrative control over their systems and will not agree to install a particular OS. This is in contrast to node management in PlanetLab and RON, where sites are required to share (or even relinquish) control over their root account. We also realized that, due to security and accountability concerns, contributors do not want to run arbitrary experiment code on their machines. For example, a BitTorrent experiment could cause others to suspect copyright violations, and a network measurement could generate complaints about unwanted traffic. These concerns must be addressed because they will discourage many volunteers from participating. The challenge is to do so while imposing as few management requirements on edge nodes as possible.

2. Edge nodes often have limited storage and processing resources. We cannot make strong assumptions about the capabilities of participating edge nodes. They may be laptops, handhelds, or cell phones with limited storage and processing resources. Thus, the testbed nodes' software stack must be flexible and light-weight, which contrasts with inflexible policies of existing testbeds. For example, PlanetLab requires nodes to be x86-based server-class machines with fast CPUs and large amounts of memory and storage. The challenge is to allow all sorts of nodes – even restricted ones – to contribute to the testbed.

3. Edge nodes are often located behind middle boxes. It is well known that many Internet users are connected to broadband access networks that almost always use dynamic IP addresses, NATs, and/or firewalls. Existing testbeds like PlanetLab or RON require nodes to have publicly reachable, static IP addresses and DNS names that can be resolved with both forward and reverse lookups. To elicit broad participation and adoption, these requirements must be relaxed. The challenge here is to design a testbed using nodes that may not be able to communicate directly with each other.

7.1.2 Requirements

From our discussion above, we identify two crucial design requirements and describe a testbed design that meets these requirements in the next section.

1. **The testbed design should accommodate nodes that cannot run arbitrary application code.** Even resource-constrained nodes should be able to participate, and node owners should retain administrative control.

2. **The testbed design should accommodate nodes that cannot communicate directly with each other.** For example, nodes behind NATs or firewalls should be able to join the testbed.

7.2 The SatelliteLab Design

In this section, we present the design of SatelliteLab, a testbed that can accommodate nodes from a diverse set of edge networks. We start with a high-level overview of our testbed architecture. Next, we describe two key mechanisms to overcome the design challenges we identified in the previous section. Finally, we show how SatelliteLab leverages these mechanisms to create a highly heterogeneous network testbed.

7.2.1 Overview

At a high level, SatelliteLab has a two-tier architecture. The nodes in the upper tier — the *planets* — are well-provisioned and professionally managed nodes located in high-capacity research networks. The nodes in the lower tier — the *satellites* — belong to a new class of light-weight testbed nodes that is introduced by SatelliteLab. Unlike planets, satellites can be any type of end host with Internet connectivity, such as desktops, laptops, or handhelds, including hosts behind NATs and firewalls. Satellites are owned and managed by individual users.

Separating testbed nodes into powerful planets and light-weight satellites enables us to assign different responsibilities to the two node types. This distinction is crucial to meeting the two design challenges imposed by inclusion of satellites: their inability to execute arbitrary experiment code or to communicate directly with each other. We first describe two mechanisms that address these challenges and then explain how SatelliteLab works.

7.2.2 Delegating Code Execution to the Planets

Since satellites cannot run application code, we associate each satellite with a nearby planet and run the application code on that planet. However, this changes the network path that connects the application instances. In particular, if the traffic were sent directly from one well-connected planet to another, we would miss the access links of the satellites, which are often the bottlenecks on Internet paths and have a significant impact on the path characteristics (cf. Chapter 3).

To subject the application traffic to the network conditions that exist along the direct paths between the satellites, SatelliteLab must re-route packets via the satellites. Figure 7.1 illustrates this through an example in which two satellites, S_A and S_B, delegate the execution of their application instances to two nearby planets P_A and P_B.

There are two problems with this approach: First, the traffic along the path segment between the satellites is often blocked by NATs and firewalls, making the path unusable. Second, compared to the direct path between satellites, the data packets traverse the access

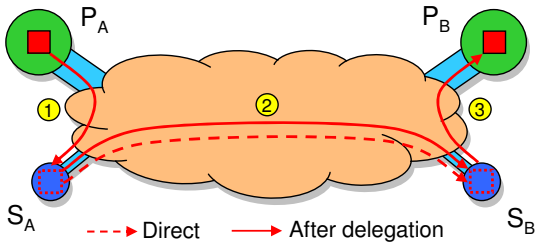


Figure 7.1: Delegating code execution to planets. satellites choose a nearby planet to run application code on their behalf

links of the satellites twice, once each in the upstream and the downstream directions. We propose a detour routing-based mechanism in Section 7.2.3 to solve the first problem. We resolve the second problem in Section 7.2.4, where satellites do not receive and forward actual data packets, but instead they use small 'control' packets in one direction, and 'dummy' packets that are of the same size as the data packets in the other direction.

7.2.3 Detouring Traffic via the Planets

To avoid the path segment between two satellites, we can detour all traffic between satellites via their closest planets. Figure 7.2 shows an example. When satellite S_A needs to send a packet to another satellite S_B, the packet is first sent to the nearest planet P_A, then forwarded to the planet P_B nearest to the target satellite S_B, and finally delivered to satellite S_B.

We expect the detour path latency, loss, and throughput characteristics to be similar to the direct path between the satellites for the following two reasons. First, the detour path and the direct path typically share the same bottleneck links, namely the access links of the satellites. Second, because each satellite is associated with the planet closest to it (in the network topology), the base latency between the planets will approximate the latency between the satellites. Our evaluation results in Section 7.4 show that these expectations typically hold in practice.

Note that when detouring is used, each satellite only needs to communicate with a single node: the nearest planet. In the final protocol, satellites are required to register with this planet a priori, and are not allowed to communicate with any other node. This ensures that satellites will not send traffic to arbitrary nodes, even if a malicious non-testbed node

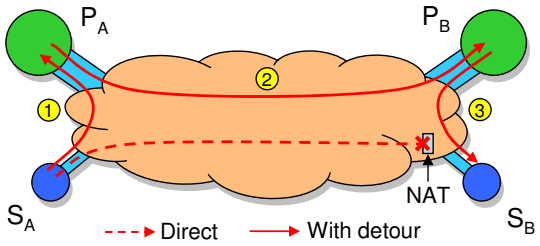


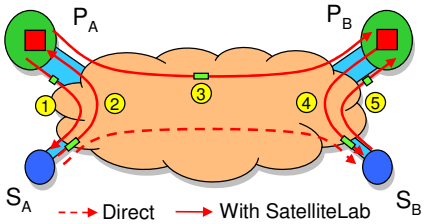
Figure 7.2: Detouring traffic via the planets. traffic is detoured through the planets because satellites cannot communicate directly.

sends them forged packets. This constraint shields contributors from complaints about unwanted traffic, and provides accountability for all traffic generated by the satellites.

7.2.4 How SatelliteLab Works

Our design of SatelliteLab is based on a combination of the two insights discussed in the previous two sections. First, satellites delegate the execution of application code to nearby planets. The planets in turn forward the application traffic via the satellites to subject it to the network conditions that exist on the path between the satellites. Finally, satellites communicate by forwarding their traffic through their nearby planets, overcoming their inability to communicate with each other directly.

Figure 7.3 illustrates in detail how SatelliteLab works. The figure shows two satellites S_A and S_B, as well as their closest planets P_A and P_B. Each planet is running an instance of the application that is being evaluated. When the instance on planet P_A needs to send a data packet to the instance on planet P_B, P_A sends a small control message to its satellite S_A (1) and instructs it to forward a "dummy" packet along the detour path. The dummy packet is the same size as the data packet and is initially empty. When the dummy packet reaches P_A (2), it is filled with the actual data and forwarded to P_B (3). At P_B, the data is removed, and a dummy packet is forwarded to S_B (4). When the packet arrives at S_B, S_B responds with a small control message to P_B acknowledging its receipt (5). Finally, when P_B receives this receipt, it delivers the original packet to its local application instance.



#	Direction	Packet Type	Size
1	$P_A \rightarrow S_A$	Control	40 bytes
2	$S_A \rightarrow P_A$	Dummy	(same as data)
3	$P_A \rightarrow P_B$	Data	(variable)
4	$P_B \rightarrow S_B$	Dummy	(same as data)
5	$S_B \rightarrow P_B$	Control	38 bytes

Figure 7.3: SatelliteLab paths. The application instances run on the planets, but the traffic between them is detoured through the satellites.

The detour path chosen by SatelliteLab includes two path segments, S_A to P_A and P_B to S_B, in addition to the direct path between the planets. While these two segments introduce additional delays and losses, SatelliteLab minimizes their effects by (a) selecting the planets closest to the satellites to minimize the extra latency, and (b) keeping the sizes of the control packets traversing the segments to just a few bytes to minimize the additional network load. Section 7.4 demonstrates that the additional delays and losses introduced by the detour path are minimal in practice.

7.2.5 Incentive Mechanisms

An important aspect of designing cooperative networking testbeds is providing the right set of incentives for people to contribute resources. In the case of SatelliteLab we need to build the incentives for recruiting end users to participate in the SatelliteLab testbed.

SatelliteLab shares the problem of finding the right incentives with other systems that build on user contributions. For example, volunteer grid computing projects — often build on top of BOINC [LLM88, ACK+02, And04] — motivate users to contribute spare processing power of their computers to help solve problems of great impact, e.g., finding extra terrestrial intelligence (SETI@home) or fighting diseases like HIV and Malaria (Rosetta@home). Additionally, contributors are often ranked on the project's webpage according to the amount of processing time they have contributed. Further, systems that allow users to contribute networking resources, like DIMES [SS05], NETI@home [SR04], and SamKnows [Sam03] often give the user access to the collected data, e.g., by displaying statistics of a user's Internet connection. Finally, P2P content distribution networks [Coh01, CGN+04, MPHD06] (e.g., for streaming videos or for downloading software) often require their users to contribute resources — in this case network bandwidth — in order to receive content. Typically, they use a tit-for-tat scheme that bounds the download speed to be proportional to the amount of upload bandwidth contributed. All of these systems are successful in building the right incentives to attract a large number of users.

Based on our experience, we believe that a suitable set of incentives must have three properties to succeed in practice. First, the incentives must outweigh any concerns a contributor may have about participating. This includes technical concerns, such as whether the experiment is going to consume too much bandwidth. Second, joining a testbed must be easy. A cumbersome software configuration process will detract from a volunteer's willingness to participate. Third, once they join, people must have an incentive to continue contributing. Retaining volunteers is an important challenge for SatelliteLab.

We believe that SatelliteLab should support a collection of incentive schemes, each of which is suitable for experiments of a particular scale. For small-scale experiments, such as the ones described at the end of this chapter, researchers may simply convince their friends and colleagues to deploy satellites on their personal machines. In this scenario, SatelliteLab acts as a "private testbed" for the researchers involved. However, if an experiment requires hundreds or thousands of nodes, a different incentive scheme must be used to attract participants beyond the immediate social circle of the researcher.

We describe two examples of incentive schemes that we believe provide adequate volunteer resources for an experiment of this scale.

1. **Tit-for-tat:** SatelliteLab can reward contributors by granting them priority when they utilize the testbed. We envision a scheme in which SatelliteLab records the number of node hours contributed to the testbed by each researcher, and treats the highest contributors preferentially. Because testbeds such as PlanetLab experience excessive load surges close to conference deadlines, such a scheme would motivate many people to contribute more resources to the testbed. We implemented this scheme for SatelliteLab and discuss it in Section 7.3.5.

2. **Providing financial compensation:** Researchers using SatelliteLab can offer to pay contributors per node hour. Amazon.com's MechanicalTurk [Ama05], a system for paying anybody on the Internet for performing various manual tasks, shows that it is possible to recruit people even by offering small financial rewards. For example, SatelliteLab might offer €50 a year to anyone running the SatelliteLab's applet (a €50 reward pays for the cost of a residential broadband connection for one month). Researchers can then pay €100 to rent 100 SatelliteLab nodes for an entire week's

worth of experiments. We believe that such a pricing scheme will appeal to many contributors.

7.3 Implementation

Although our design can leverage any existing testbed infrastructure, we implemented SatelliteLab as an extension to the PlanetLab testbed. In this section, we describe the technical details of our implementation. The code of our SatelliteLab implementation is available from `http://satellitelab.mpi-sws.org`.

7.3.1 Overview

Our implementation of SatelliteLab has two components: a *planet proxy* and a *satellite helper*. Each PlanetLab node runs a planet proxy and each satellite runs a satellite helper.

The role of the planet proxy is threefold: it intercepts network traffic sent by applications, it forwards network traffic along the appropriate detour path, and it communicates with the helpers running on satellites that are assigned to its local node. The satellite helper is assigned to a nearby planet proxy and exclusively communicates with it, responding to its probes.

7.3.2 The Planet Proxy

Our planet proxy runs in user space on the PlanetLab nodes. Restricting our implementation to user space eases deployment, but also prevents us from using fast kernel-level hooks to intercept network traffic [VYW+02]. Thus, our planet proxy is subject to delays associated with the kernel's scheduling policy of user-level tasks.

We implemented the planet proxy as a Linux daemon process in approximately 2,400 lines of C++ code.

Intercepting application traffic

When the proxy is started, it creates a virtual Ethernet device (a TAP device [KY99]) configured with a private subnet, such as `10.0.0.0/8`. This private subnet is shared by all planet proxies and all application instances running on the planets. To run an application on SatelliteLab, the experimenter only needs to configure it to bind to the TAP device's IP address. Thus, all traffic sent by the application is intercepted by its local planet proxy.

When two application instances on the same planet exchange traffic, this traffic would normally be delivered directly by the kernel without being routed through the TAP device. To avoid this, we use a simple technique borrowed from ModelNet [VYW+02]: each application modifies certain bits in all destination addresses to ensure that they appear as remote addresses; once the planet proxy intercepts a packet, these bits are set back to their original value. This technique uses a dynamic library (`libipaddr`) and works with unmodified application binaries.

Communicating with the satellites

The proxy interacts with its satellites using a very simple UDP-based probe/response protocol. The proxy can send a d-byte UDP message `PROBE(i,d,u)`, where i is an identifier

and u is the size of the requested response packet. In response, the satellite sends a u-byte UDP message `RESPONSE(i,u)`. To mask the occasional loss of packets, each probe (response) message includes information about the two probe (response) packets most recently exchanged between the proxy and the satellite. After intercepting a packet of application traffic, the proxy forwards the packet along the detour path using the above protocol on the edges and one extra message between the planets.

Let A and B be application instances running on planets P_A and P_B, respectively; let S_A and S_B be the corresponding satellites, and let σ be the size of the data packet in bytes. The complete packet forwarding process is as follows:

1. P_A chooses an identifier i and stores the packet in an internal buffer indexed by i.
2. P_A sends `PROBE(i,12,`σ`)` to S_A, which responds with `RESPONSE(i,`σ`)`.
3. P_A retrieves the packet and forwards it to P_B.
4. P_B sends `PROBE(i,`σ`,10)` to S_B, which responds with `RESPONSE(i,10)`.
5. P_B retrieves the packet and delivers it to B.

This forwarding process subjects application traffic to network conditions on the access links of the satellites. Since the access links are often the bottlenecks of the direct network path between the satellite nodes, the detour paths have similar access link delays, losses, and bandwidths as the direct paths.

If the probe packets are reordered or delayed, their corresponding data packets are also reordered or delayed. Similarly, if probe packets are lost, their corresponding data packets are not forwarded, and they are eventually dropped from the buffer by the proxy. Finally, the packet sizes of probes (and probe responses) match those of the data packets.

Compensating for packet loss

With the described protocol, SatelliteLab increases the packet loss rate on the access link compared to the direct communication between satellites. The reason is that two control packets – a probe and a response – are needed to emulate the transmission of a single data packet; the data packet is considered lost if *either* of the two control packets is lost.

In our experiments this problem occurred rarely, even on access links such as broadband as these control packets are small. Nevertheless, we developed an additional mechanism to mitigate the effects of this problem. SatelliteLab incorporates information about the control packets already sent in the padded data of subsequent packets. This enables quick detection and recovery of control packet loss. Thus, if a packet is lost in either direction, the recipient can obtain the missing information from one of the subsequent packets[1]. Since this does not help if the last control packet in a sequence is lost, we send this packet twice, but only if the link is otherwise idle.

However, the planet only uses the information about the control packets already sent to compensate for losses that happen on the additional pathways introduced by SatelliteLab. When it emulates an upstream transmission, it compensates for lost probe packets; when it emulates a downstream transmission, it compensates for lost responses. Otherwise

[1]Note that if the access link reorders packets, this can cause additional reordering. If this is an issue, the feature can be disabled.

SatelliteLab would overcompensate and push the loss rate below the access link's actual loss rate.

Adjusting the MTU

When a packet is sent from an application instance on one planet to an instance on another, the proxies encapsulate the packet in a UDP datagram, which adds a 28-byte UDP/IP header and a four-byte SatelliteLab header. If the original packet was already MTU-sized, the resulting UDP datagram must be sent in two packets, which is undesirable because it increases the effective loss rate. Usually, it is possible to avoid this problem by reducing the MTU of the TAP device. However, in some testbeds (e.g., PlanetLab), the MTU cannot be changed. Therefore, we implemented a mechanism in the planet proxies that adds a MSS option to TCP SYN packets, which transparently reduces the packet size for TCP flows. Other transport protocols will still work, but their packets may be split on the planet-to-planet path and thus experience a higher loss rate.

7.3.3 The Satellite Helper

We implemented the satellite helper in Java to ensure its portability across different software and hardware configurations. Our Java implementation has 118 lines of code, as counted by the number of semicolons. The very small size of the satellite code simplifies code reviews. This ensures that the satellite helper does not create a vulnerability on the satellite machines.

Additionally, we developed OS-specific packages – one for Windows, one for Mac OS X, and one for Linux. These packages have installers and are integrated with the respective OSes; for example, the Windows version can be minimized to the system tray and is configured to automatically start at boot time.

To keep the set of potential volunteers as large as possible, the helper must be easy to install and configure, and it must disturb the user as little as possible. During installation, the helper prompts the user once to enter an identifier for the local machine. No other configuration is required. After this, the helper is started automatically on boot and then runs in the background until the machine is shut down; it never actively interacts with the user.

During startup, it first locates the closest testbed node. Each helper is statically configured with a list of DNS names that map to some of the testbed nodes. The helper contacts one of these nodes and obtains from them a list of nearby testbed nodes. It then pings each of these nodes and chooses the one closest to it in terms of network latency. Then it waits for, and responds to, any incoming UDP probes from that planet. To minimize the possibility of complaints or abuses directed at the satellite's owner, the helper does not communicate with any other node.

Handling NATs and DHCP

In our experience, many satellites are behind a NAT. This creates two challenges for the satellite helper: First, it knows its local IP address, but not its publicly visible IP address, and second, control packets from the planet cannot reach it unless it first initiates a connection. We use a heartbeat mechanism to solve both problems. The satellite helper periodically sends a small status message to its planet, which allows the proxy to determine

the satellite's publicly visible IP. These packets also prevent the address translation rule in the NAT from expiring and thus ensure that the satellite remains reachable from the planet. While this mechanism helped us to traverse all the NATs in our testbed, it can be further extended with more elaborate NAT traversal mechanism [FSK05].

Some satellites acquire their IP address using the Dynamic Host Configuration Protocol (DHCP). This can create problems if DHCP reassigns a satellite's IP address. In this case, the planet proxy must no longer send probes to the old address. We also use the heartbeat mechanism to handle this problem: when the proxy no longer receives status messages from a satellite, it stops sending probes to that satellite.

Allocating satellites to experiments

Like the PlanetLab testbed, SatelliteLab can be used by researchers to run different experiments. In our implementation, we allow satellites to participate in only one experiment at a time, which prevents interference between experiments and avoids overloading the satellite's access link. However, researchers can serially allocate satellites to different experiments over time. Our implementation enables a satellite to seamlessly leave one experiment and join another. When switching to another experiment, the planet proxy can either bind the satellite to a different application running on its local planet, or the satellite can re-register with a different planet.

We believe that allocation of satellites to different experiments should be subject to a testbed-wide policy. This policy would depend on SatelliteLab's incentive mechanism for attracting satellites (cf. Section 7.2.5).

7.3.4 Running an Experiment

Running an experiment on SatelliteLab requires four steps. First, the experimenter recruits a sufficient number of satellites. Second, she creates a configuration file that specifies which application instances should run on which planets. Third, she installs and runs the proxy on each planet. Finally, she starts the application instances on the planets. If multiple instances of the application should run on the same planet, the application has to preload `libipaddr` with the appropriate parameters in each case.

SatelliteLab has no particular requirements for applications — if the application runs natively on the planets, it is likely to run in SatelliteLab. If a planet is to run more than one application instance, the only additional requirement is that it needs to work with `libipaddr`. This is also the case for applications that run on ModelNet.

7.3.5 Resource Sharing

Rather than recruiting volunteers for each experiment individually, it seems natural to recruit them for the testbed in general and to share the resources they provide across different experiments. In this section, we describe an extension of our SatelliteLab prototype that enables this type of sharing.

Resource allocation and incentives

Sharing is easy as long as there are no conflicts in resource usage. However, it is only to be expected that experimenters would prefer large-scale experiments and thus would each

request a large number of satellites. If not all requests can be satisfied, it is not clear how the available resources should be allocated.

The straightforward solution is to allocate satellites on a first-come-first-served basis, and to rely on some external mechanism to resolve conflicts. However, this method fails to provide sufficient incentives for experimenters to contribute resources. Such an incentive could be added by assigning satellites proportional to the number of satellites each experimenter has currently recruited. Still, the resulting incentive is for short-term contributions and does not reward volunteers who have provided resources for a long time.

Instead, we chose a time-based system in which volunteers earn credit during the time they contribute and can then use this credit to "buy" time on other satellites. This provides an incentive for recruiting many long-term volunteers with highly available machines, to the benefit of the testbed as a whole. When signing up, each user can specify which group or institution she wants to donate her credit to.

Planet controller and the sun

In order to manage and enforce allocations, we introduce two additional components. The first is the *planet controller*, which runs on each testbed node and arbitrates between planet proxies belonging to different experiments. The second component is a single controller node, which we call the *sun*. Its purpose is to manage and enforce resource allocations, as well as to keep credit accounts.

In a single-experiment deployment, the responses from the satellites are received by the planet proxies. In a shared deployment, these messages go to the planet controllers instead. The controllers maintain a list of all satellites that have recently sent responses, using the user names to distinguish between satellites. Periodically, the sun queries all controllers and updates its credit accounts.

Experimenters can use their credits to reserve time on the satellites. The reservations are kept in a database on the sun, which periodically distributes them to the controllers and also associates each reservation with a specific port number on the planet nodes. When a reservation changes, the controller sends a message to its satellites and asks them to switch to the new planet port number. This ensures that different experiments do not interfere with each other.

The sun's user interface

The sun implements a web-based interface that can be used by volunteers, experimenters, and testbed administrators. Volunteers use the web interface to create a user account which allows them to specify the institution to which they want to donate their credits. Also, the interface suggests a nearby planet to which the new satellite can be connected, and provides additional information about configuring the satellite. Experimenters use the interface to make reservations, and the testbed administrators use it to manage planet nodes and to check the system status.

7.4 Evaluation

At a high level, our evaluation focuses on three aspects about SatelliteLab's design. First, we illustrate one of its key advantages: a lower barrier to entry. Second, we show that

although satellites have lower availability than planets, their session durations are adequate for running most experiments. Lastly, we demonstrate that the characteristics of SatelliteLab's detour paths closely match those of the direct paths between the satellites.

7.4.1 SatelliteLab is Successful in Making Testbeds Heterogeneous

To evaluate whether SatelliteLab is successful in adding heterogeneity to existing testbeds like PlanetLab, we compare the diversity of network paths in the PlanetLab testbed with paths in SatelliteLab. Since all of our 32 SatelliteLab nodes are in Europe and North America, here we only consider PlanetLab nodes in these regions, in order to ensure comparability.

Although PlanetLab continues to attract new participants, additional nodes do not necessarily improve a testbed's path heterogeneity. We illustrate this by using the number of distinct inter-AS links that are covered by paths in a testbed as a proxy for path heterogeneity. Figure 7.4 shows that adding a new node to PlanetLab increases the AS-path heterogeneity by only a small amount. On average, each additional node increases coverage by only 2.7 inter-AS links. We believe this is because most PlanetLab nodes are located in closely coupled academic networks and thus the AS paths between them are similar. However, Figure 7.4 also shows that, if we increase PlanetLab's size by just 10% using nodes from the commercial Internet, we can more than triple its coverage of inter-AS links.

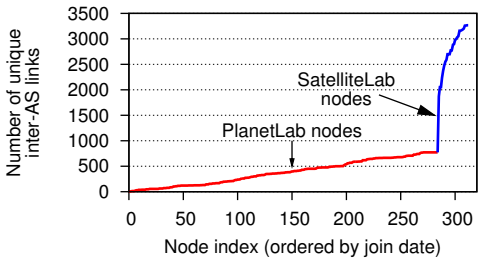


Figure 7.4: Inter-AS links covered by PlanetLab vs. SatelliteLab. By adding a small number of edge nodes to PlanetLab, we can increase the number of inter-AS links covered by the testbed paths by more than a factor of three.

Figure 7.5 further illustrates the diversity of our SatelliteLab testbed. In contrast to PlanetLab, most nodes in SatelliteLab are located behind a diverse set of access links, including cable and DSL. A few are even connected using Bluetooth and cellular links. Two-thirds of the nodes are behind NATs, and half of the nodes are mobile devices, such as laptops and handhelds, that use Wi-Fi.

In summary, SatelliteLab is successful in considerably improving the heterogeneity of Internet testbeds in multiple dimensions by adding nodes from Internet edge networks.

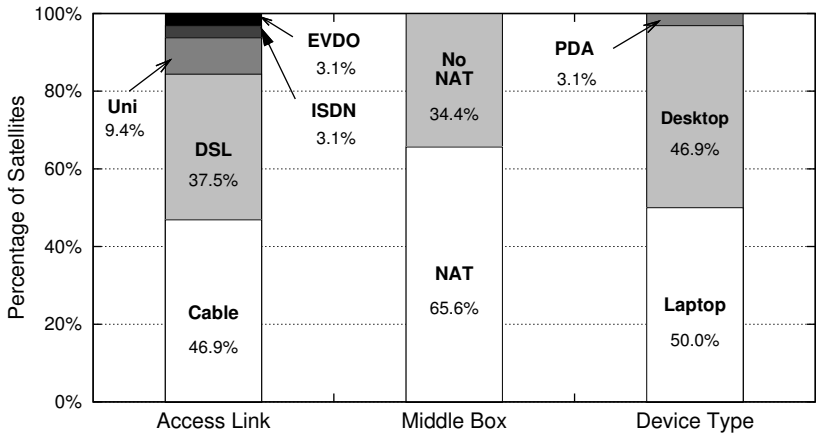


Figure 7.5: Heterogeneity in SatelliteLab. Our 32 testbed nodes were connected to various access networks, such as DSL, cable, and EVDO. Less than 10% of nodes are located in University (Uni) networks. Many of the nodes were located behind NATs, and some of them were mobile.

7.4.2 SatelliteLab Makes it Easy to Recruit Edge Nodes

To support satellite nodes with a variety of software profiles, we implemented the satellite helper in Java. We created easy-to-install packages for Windows, Mac OS X, and Linux, and we set up a webpage with instructions for installing our software. To the SatelliteLab software package we added a test harness that enabled us to send packets directly between the satellites where possible. This test harness allowed us to compare SatelliteLab's detour paths to direct paths. Since this test harness makes the nodes vulnerable to abuse, we disabled it once we had completed our evaluation experiments.

In a period of only two weeks, we recruited 32 satellite nodes by asking our friends, family members, and colleagues to install SatelliteLab on their private machines. Table 7.1 gives an overview of our testbed nodes. Our testbed includes sixteen satellite nodes from the U.S., eight nodes from Germany, five nodes from Canada, and one node each from Hungary, Portugal, and the United Kingdom. These nodes connect via DSL, cable, ISDN, and cellular links; many of them have an extra wireless hop (802.11 or Bluetooth), and several of them are behind NATs.

As demonstrated in Section 7.4.1, SatelliteLab's lower barrier to entry does not just improve access link diversity, it increases the heterogeneity of other testbed characteristics as well. For example, our deployment included mobile nodes (a PDA and several laptops), which were connected to different access networks at different times (e.g., to the university network at work and to a cable network at home). Also, as shown in Figure 7.6, satellite uptimes varied between close to 0% and close to 100% because some of the nodes were switched off overnight and/or were occasionally suspended (e.g., for travel). We believe that this additional heterogeneity could be valuable to experimenters.

Based on our initial experience, we believe that if the several hundred researchers using PlanetLab collaborated and each recruited a handful of satellites, future SatelliteLab deployments could grow to include thousands of nodes in Internet edge networks.

#	Location	Access link	NAT	Type	Mobile
1	Canada	Cable	no	Desktop	no
2	Canada	DSL	no	Desktop	no
3	Canada	Uni+Wi-Fi	no	Laptop	yes
4	Canada	Uni+Wi-Fi	no	Smartphone	yes
5	Canada	Cable+Wi-Fi	yes	Laptop	no
6	Germany	DSL+Wi-Fi	yes	Desktop	no
7	Germany	Cable	no	Desktop	no
8	Germany	DSL	yes	Desktop	no
9	Germany	DSL+Wi-Fi	yes	Laptop	yes
10	Germany	Cable+Wi-Fi	yes	Laptop	yes
11	Germany	DSL	yes	Desktop	no
12	Germany	DSL+Wi-Fi	yes	Desktop	no
13	Germany	ISDN+BT	no	Laptop	no
14	Hungary	DSL	yes	Laptop	yes
15	Portugal	Cable	no	Laptop	no
16	United Kingdom	DSL	no	Laptop	yes
17	CA, USA	DSL+Wi-Fi	yes	Laptop	no
18	CA, USA	EVDO	no	Laptop	no
19	CO, USA	Cable+Wi-Fi	yes	Laptop	no
20	IL, USA	Cable	yes	Desktop	no
21	LA, USA	DSL	yes	Desktop	no
22	MA, USA	Cable+Wi-Fi	no	Laptop	yes
23	MD, USA	Uni	no	Desktop	no
24	MD, USA	Cable+Wi-Fi	yes	Laptop	yes
25	NJ, USA	DSL+Wi-Fi	yes	Laptop	no
26	NJ, USA	Cable+Wi-Fi	yes	Laptop	no
27	TX, USA	Cable+Wi-Fi	yes	Desktop	no
28	WA, USA	Cable	yes	Desktop	no
29	WA, USA	Cable	yes	Desktop	no
30	WA, USA	Cable+Wi-Fi	yes	Desktop	no
31	WA, USA	Cable+Wi-Fi	yes	Laptop	yes
32	WI, USA	DSL	yes	Desktop	no

Table 7.1: Overview of the satellite nodes. Our nodes use a variety of access links such as cable, DSL, wireless (Wi-Fi), Bluetooth (BT), or high-capacity access links located in Universities (Uni). Some of the nodes' access links combine two types of networks; for example, a DSL+Wi-Fi means that the host is connected to a DSL modem via a wireless link. The last column indicates whether or not the host was mobile.

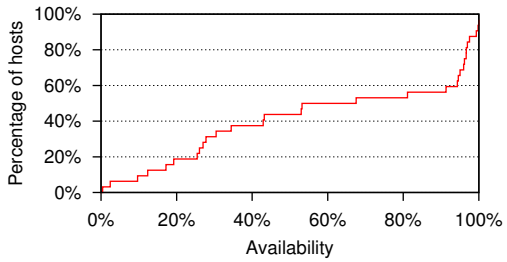


Figure 7.6: Satellite availability. Availability varied widely among the different satellite node types.

7.4.3 The Availability of Satellites is Adequate for Many Testbed Experiments

To test whether the availability of SatelliteLab nodes is sufficient for potential experiments, we compared the median session times of our satellites to run-times of experiments on PlanetLab. We used the CoTop monitor data [PP04] for the month of November 2007 to compute for each slice on PlanetLab the longest continuous session during which a node sent or received data over the network. A session ended when a node sent and received no traffic for 15 minutes. Figure 7.7 compares this to the median node availabilities in our testbed; it shows that 40% of satellites have enough availability to support the workloads of about 60% of the PlanetLab slices. The figure also shows that a significant proportion of slices are active for days or weeks. Most of these slices are services, such as CoMon and OpenDHT, or long-lived measurement experiments. SatelliteLab was not designed to support such workloads (see Section 7.5 for further discussion).

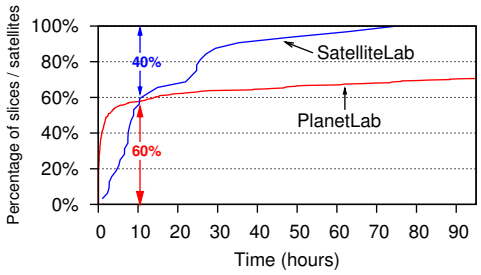


Figure 7.7: Run-time of experiments on PlanetLab. 40% of our satellites had median session times longer than 10 hours, while 60% of PlanetLab slices did not run any experiment that lasted more than 10 hours.

7.4.4 Satellites Can Find Planets in their Close Proximity

Compared to sending data via the direct path, detouring application traffic through satellites increases latency. However, we expect this increase to be small with respect to the overall latency as long as satellites can find a planet in their close proximity. Figure 7.8 shows that 80% of the satellites had a RTT of less than 35 ms to the nearest PlanetLab

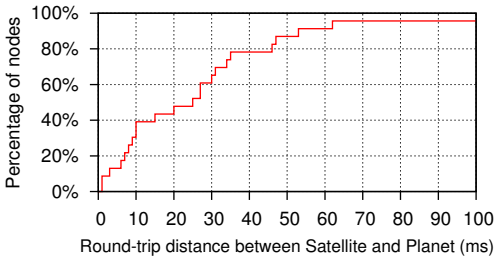


Figure 7.8: Minimum distance between a satellite and its closest PlanetLab node. 80% of the satellites in our testbed had an RTT of 35 ms or less.

host, and all but one had an RTT of less than 62 ms. The satellite using EVDO had an RTT of 109 ms because of the high transmission delays in cellular networks.

7.4.5 Detour and Direct Paths are Bottlenecked at the Same Access Links

The characteristics of Internet paths are driven by their bottlenecks. In edge networks, path bottlenecks often occur on the "last mile" — that is, at or close to the access link (cf. Chapter 3). Because both the direct path and the detour path between two satellites share this "last mile", they are likely to share the same bottleneck. This observation is the key to understanding why the characteristics of SatelliteLab's detour paths closely match those of the direct paths.

We conducted a series of experiments to determine whether the capacity, jitter, and loss rates of detour paths are similar to those of direct paths. We measured five different paths for each pair of satellites (Figure 7.9): the *direct path* between the satellites, the *satellite detour* path used between the two satellites by SatelliteLab, and the three components of the satellite detour, namely the *planetary highway* and two *access pathways*. The planetary highway is the segment of the satellite detour between the PlanetLab nodes, while an access pathway is the segment between a PlanetLab node and a satellite. We used two probe types to measure each of the five paths[2].

- **Small UDP ping/pong probes:** We sent a long sequence of small (100-byte) UDP ping/pong packets along the paths. We paced the ping packets using a Poisson distribution with a mean sending rate of one packet per second; the pongs were returned immediately after the receipt of a ping. Since the average data rate was just 1 Kbps, the probe packets reflect the RTTs experienced by packets under normal operating conditions.

- **Large UDP flood probes:** We sent large (1,000-byte) UDP probes at the rate of 3 Mbps along the paths. Each flood lasted for three seconds and typically saturated the bottleneck links, which were below 3 Mbps in most cases. These probes reflect the conditions of the paths under load.

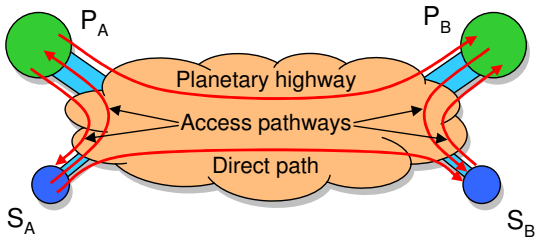


Figure 7.9: Paths used for evaluation. We separated the detour path into two segments, the planetary highway and the access pathways.

[2]The number of paths measured with these probes can be different because not all testbed nodes were available at the same times. For the small ping/pong probes we used only paths for which at least three hours of measurements were available.

Path capacity

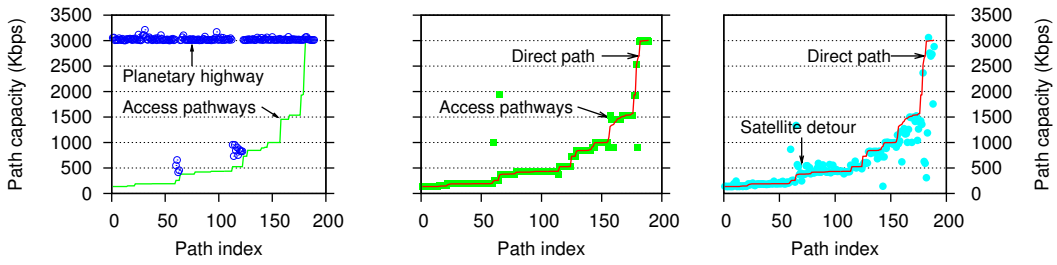
We studied path capacities by measuring the bandwidth of their bottleneck links. Note that measuring bottleneck bandwidth is different from measuring available bandwidth or TCP throughput, both of which are lower than path capacity and can vary over time. When designing SatelliteLab, we expected the detour and the direct path to have the same path capacity because they share the same access links, which often tend to be the bottlenecks. To verify this, we estimated the capacities of paths from the packet delivery rate of the large UDP[3] flood probes. For each path, we took the maximum across all the measurements to remove noise from potential cross-traffic at the bottleneck link.

Our results are as follows:

1. **Access pathways are the bottleneck of the satellite detour paths.** To understand where the capacity bottlenecks in the satellite detours are located, we compared the capacities of their constituent planetary and access pathways. Figure 7.10a shows capacities of 3 Mbps for most planetary highways, which suggests that their bottleneck links were not saturated by our 3 Mbps floods. In contrast, most access pathways show capacities of less than 2 Mbps.

2. **Access links shared by the direct and the detour paths are often the capacity bottlenecks.** Figure 7.10b compares the bottleneck capacities of the access pathways to those of the direct paths between the satellites. The path capacities closely match, suggesting that direct and detour paths share their bottlenecks on the access links.

3. **SatelliteLab's detour routing preserves the direct path capacities.** Finally, we plot the capacities of direct and detour paths in Figure 7.10c. As before, the capacities closely match; the differences are within 10% in almost all cases.



(a) Planetary highway vs. Access pathways

(b) Direct path vs. Access pathways

(c) Direct path vs. Satellite detour

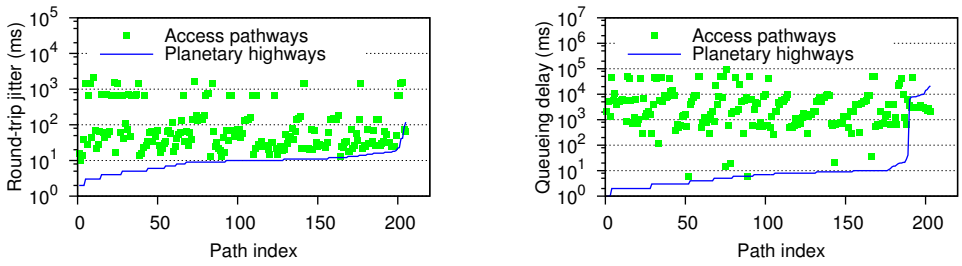
Figure 7.10: Path capacities. The capacity bottlenecks are typically on the access links, so they are shared by both the access pathway and the direct path. As a result, the capacity of the satellite detour matches well with that of the direct path.

[3]We found that most NATs allow UDP-based hole punching, while many have trouble allowing TCP-based hole punching.

Path latency

In this section, we study two path characteristics related to packet latencies: path jitter and queueing delay. We define *path jitter* as the variation in packet RTTs due to queueing at bottleneck routers along the path. We estimate it as the difference between the 95th percentile path RTT and the minimum path RTT, measured with the small ping/pong probes. We define *queueing delay* as the maximum increase in one-way delay for a path under load, and we measure it using the large UDP floods that saturate the path bottleneck router by filling its queue. We estimate queueing delay as the difference between the minimum and the 95th percentile one-way path delay. Our results are as follows:

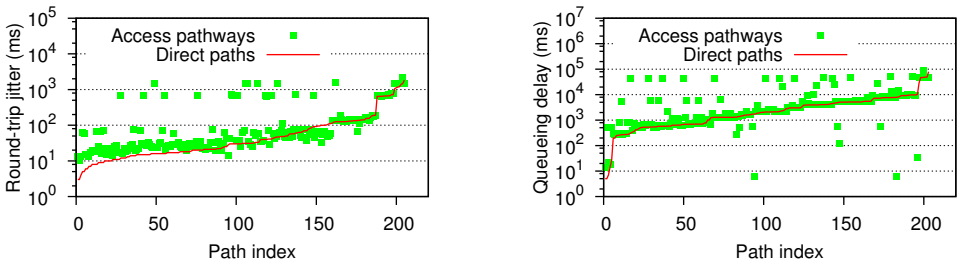
1. **Queueing occurs primarily along the access links.** Since path bottlenecks are likely at the access links, we expect the delays over the access pathways to dominate jitter and queueing delay. Figure 7.11a compares the jitter along planetary highways and access pathways for different satellite detours (note that the vertical axis is a log scale). The results show that the jitter along the access pathways is significantly higher than the jitter along the planetary highways. This indicates that queueing primarily occurs along the access links. Figure 7.11b confirms this result: most planetary highways have low queuing delay, whereas access paths experience significant queuing delays, often exceeding one second. We also found that some planetary highway paths originating in two PlanetLab hosts saw more significant queueing delays, which we suspect to come from their heavy load at the time of our measurements.



(a) Jitter (95th- min) **(b)** One-way queueing delay

Figure 7.11: A comparison of jitter and queueing delay between access pathways and planetary highways.

2. **Access pathways and direct paths have matching jitter and queueing delays.** As detour and direct paths share the access link, we expect them to experience similar jitter and queueing delay. Figure 7.12a demonstrates that the degree of jitter along both paths matches closely. As before, Figure 7.12b confirms the similarity of queueing delays between access pathways and direct paths. These results suggest that queueing is due to properties shared by access pathways and direct paths.



(a) Jitter (95th-min) **(b)** One-way queueing delay

Figure 7.12: A comparison of jitter and queueing delay between access pathways and direct paths.

Path loss rates

To compare loss rates for detour and direct paths, we used the small UDP ping/pong probe experiment data. We recorded a path failure event whenever three or more consecutive packets were lost (most of these loss episodes lasted for three seconds at least). A vast majority of the measured paths (over 90%) did not show any path failure events at all. The remaining paths typically experienced one or two path failures during a single six-hour experiment. Ignoring losses during path failure events, we next computed the paths' average loss rates.

Table 7.2 shows the observed loss rates for different paths. Overall, the loss rates were low across all the paths. The table shows that no measured planetary highways experienced loss rate above 0.5%, while 20.2% of the access pathways, 28.7% of the satellite detours, and 15.3% of the direct paths saw loss rates above 0.5%. Note that all of these three types of paths share the access pathways. This suggests that most of the packet loss occurs at the satellite node access links.

Loss rate	$< \mathbf{0.1}\%$	$\mathbf{0.1 - 0.5}\%$	$\mathbf{0.5 - 1.0}\%$	$\geq \mathbf{1.0}\%$
Planetary highway	95.4%	4.6%	0.0%	0.0%
Access pathway	38.1%	41.7%	7.8%	12.4%
Satellite detour	25.4%	46.0%	10.3%	18.4%
Direct path	66.4%	18.4%	5.3%	9.9%

Table 7.2: Packet loss rates along different paths.

7.4.6 Summary

Our evaluation demonstrates that it is possible to use SatelliteLab to enlist a highly heterogeneous set of nodes as satellites. We showed that the availability of edge nodes is adequate for running many testbed experiments, and our measurements of a 32 node SatelliteLab testbed indicate that important path characteristics, such as capacity, jitter, and loss rates of direct paths between satellites are typically preserved in SatelliteLab's detour paths.

7.5 Applications

Testbeds like PlanetLab have been generally used for three broad classes of experiments.

Evaluations of Networked Systems. A large class of testbed experiments evaluate and test network systems and applications. Researchers deploy prototypes on testbeds like PlanetLab to examine their behavior when run over the Internet at large. With SatelliteLab, network researchers can perform evaluations in an heterogeneous environment. Such evaluations often lead to additional insights into their systems' behavior. Later in this section, we describe how we used SatelliteLab to evaluate two distributed systems: a network coordinate system and an overlay multicast system. For both, we found that the results produced by the SatelliteLab evaluation differ substantially from a PlanetLab-based evaluation. We traced these differences to the heterogeneity of the network paths offered by SatelliteLab.

Internet measurement studies. Another class of experiments often deployed on testbeds like PlanetLab measure the characteristics of network paths between testbed nodes. From these studies, researchers derive much needed insight into the properties of the Internet. By extending the network paths to the edge of the Internet, SatelliteLab increases the diversity of the measured Internet paths. With SatelliteLab, researchers can perform measurement studies on types of networks that are not found in today's testbeds, such as broadband and wireless. Later in this section, we show how SatelliteLab can be used to measure the throughput of TCP downloads in mobile broadband networks, i.e., GPRS/EDGE and UMTS.

Running public Internet services. Planetlab also supports running services continuously, such as Coral [FFM04] and CoDeeN [WPP+04]. Because satellites do not run application code, these services do not benefit from SatelliteLab.

7.5.1 Evaluation of Networked Systems

We used SatelliteLab to evaluate two popular networked systems: a network coordinate system and an overlay multicast system. Although previous evaluation results of both systems over PlanetLab suggested that both perform well over the Internet, our evaluation over SatelliteLab led us to substantially different conclusions. We show that these differences are due to the characteristics of the satellites' network links, which differ drastically from the characteristics of links between PlanetLab nodes.

Network coordinate system

Internet systems use network coordinate systems to cheaply and rapidly obtain estimates of network latency. The basic idea is to assign participating nodes a set of coordinates, which can then be used to obtain rough estimates of network latencies. For example, the Vuze BitTorrent client employs Vivaldi to select nearby peers to download from. Regular BitTorrent packets are used to estimate the RTT between two nodes and to calculate a node's network coordinate. Thus, no extra measurement packets have to be sent, resulting in no additional bandwidth overhead. To account for outliers in the RTT measurements,

Vivaldi uses a simple "moving percentile filter" that estimates the current RTT as the average of the past four RTT measurements.

Although PlanetLab experiments have shown network coordinate systems to be accurate, a recent study found that their accuracy is significantly lower when they are used by BitTorrent participants [LGS07]. In this study, the latency variations between the BitTorrent hosts were so high that the network coordinate system failed to converge on a single coordinate set. However, the authors could not explain the cause of these variations.

We used SatelliteLab to investigate this phenomenon. We began by repeating the experiment described in [LGS07] on a smaller scale. We used eight broadband hosts as satellites, and we installed the Vuze BitTorrent client on their corresponding planets. The *seeder* host served a large file, and seven *leechers* downloaded it. The experiment re-started when all hosts finished downloading the file.

We duplicated the findings of [LGS07] with little effort. Figure 7.13 plots the distribution of the jitter in the latencies measured across the paths among the eight SatelliteLab nodes. More than half of the paths experienced jitter of more than two seconds!

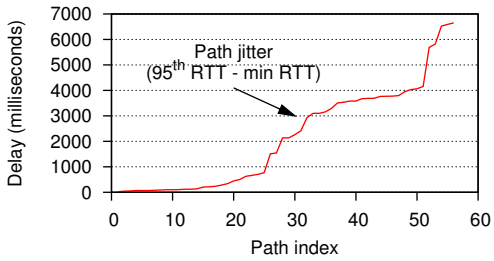


Figure 7.13: Measured jitter in a network coordinate scheme. Many paths between satellites running BitTorrent experience very high jitter.

SatelliteLab allowed us to not only reproduce the results, but also to explain them. We sent probe floods in the upstream and downstream direction that would saturate the link to measure the queue sizes of the access routers for the eight broadband hosts (cf. Chapter 3). The queue size is the difference between the 95th percentile RTT and the minimum RTT measured. We found that all access routers had long downstream and upstream queues. As Figure 7.14 shows, almost all routers had a queue length of at least one second worth of traffic. This explains the findings in [LGS07]: as BitTorrent hosts exchange large amounts of data, bottleneck queues fill up and cause the hosts' latency and jitter to vary by several orders of magnitude.

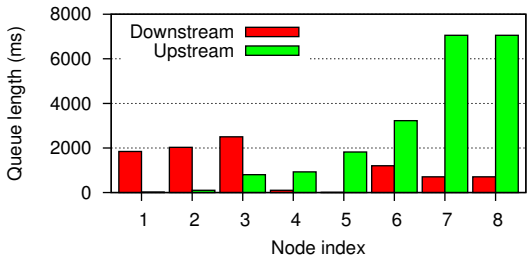


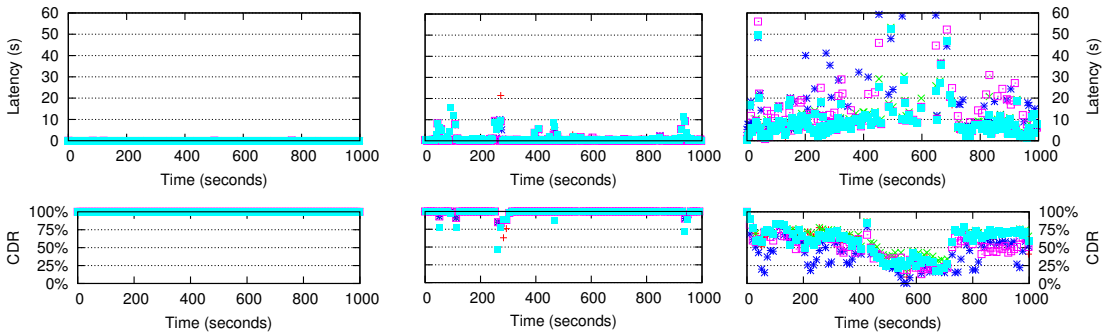
Figure 7.14: Queue sizes of access routers for satellites. Our broadband satellites have very large queue sizes both upstream and downstream.

SplitStream overlay multicast

In another experiment, we used SatelliteLab to evaluate SplitStream, a tree-based overlay multicast system that streams content from a source to a set of client nodes. This improves data redundancy and load balancing by forwarding the content along k different trees that are rooted at the source [CDK+03]. SplitStream was evaluated on PlanetLab, but it has been shown that the performance of such systems depends strongly on the available bandwidth [SGMZ04]. Therefore we expected its performance on SatelliteLab to be very different from its performance on PlanetLab.

In our experiments, we used the FreePastry 2.0_02 [Fre07] implementation of SplitStream configured with $k = 16$ trees. We ran three trials with five nodes each in three different environments: a local cluster, PlanetLab, and SatelliteLab. We used two metrics of SplitStream's performance: the chunk delivery ratio (CDR), which measures the fraction of chunks successfully delivered, and the latency of the chunk delivery.

Figure 7.15 shows our results. We first ran our application on our local cluster, which consists of dedicated machines that are connected via Gigabit Ethernet (Figure 7.15a). As expected on the cluster, performance was almost ideal; each node in the cluster received all content with minimal delay. When we ran the same experiment on PlanetLab



(a) Local cluster **(b)** PlanetLab **(c)** SatelliteLab

Figure 7.15: SplitStream experiment. Latency between transmission and reception of each delivered chunk, and fraction of chunks delivered. The five point types represent the five nodes in the experiment.

(Figure 7.15b), performance was only slightly lower. Even though the nodes experienced occasional losses, the delays and the throughput were consistently high with few CDR drops and delay spikes. Finally, when we ran the experiment over SatelliteLab with five broadband satellites (Figure 7.15c), we observed peak delays three times above those of PlanetLab nodes. Also, these nodes suffered from significant throughput variations, with the CDR dropping below 75%. In fact, the drop in CDR after 400 seconds (Figure 7.15c) occurred because one of the satellite nodes initiated a BitTorrent download, thus competing for the available bandwidth with SplitStream.

7.5.2 Internet Measurement Studies

To illustrate the benefits of SatelliteLab as a platform for Internet measurement, we present a small-scale study of TCP throughput over cellular links. TCP flows are likely to experience highly fluctuating throughput because cellular links suffer from interference, spotty coverage, and poor signal strength. Today, little is known about the characteristics of such links and how they affect the behavior of TCP flows.

Using SatelliteLab, we ran a series of TCP transfers between a mobile laptop and a well-provisioned server. The laptop was equipped with a UMTS modem and communication software that always chose the available network with the highest data rate (GPRS/EDGE or UMTS). Each transfer ran for 30 seconds and was spaced 4 minutes after the previous one. For each transfer we recorded the average throughput.

Figure 7.16 shows the cumulative distribution function of our TCP throughput measurements over UMTS/GPRS for both the upstream and the downstream direction. We find that downstream flows can have a throughput of anywhere between 10 Kbps to 320 Kbps (an order of magnitude difference!). In contrast, upstream flows have a bi-modal distribution. Half of them have very low throughput of up to 10 Kbps, whereas the other half have high throughput of over 250 Kbps. In cellular networks, the available throughput depends on whether UMTS is available or whether GPRS/EDGE is used as a fall-back. Also, the signal quality can greatly affect throughput. This simple experiment illustrates how SatelliteLab can be used to conduct measurement studies of new network environments.

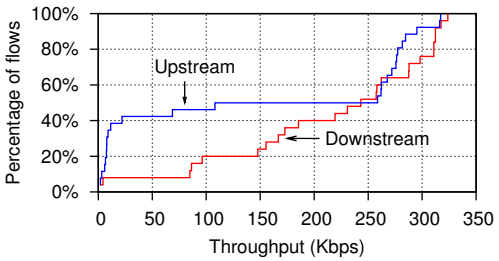


Figure 7.16: Cumulative distribution function of TCP throughput over UMTS. TCP over cellular links can experience very different throughput based on whether UMTS is available or whether it has to fall-back to GPRS/EDGE.

7.5.3 Summary

We have used SatelliteLab to evaluate two networked systems and to perform a measurement study. Our experiments illustrate three key benefits of the SatelliteLab testbed:

1. Evaluations on SatelliteLab and PlanetLab can yield substantially different results. The differences stem from the additional heterogeneity added by satellites. These results shed new insight into the observed performance of the evaluated systems. Even at small scale, SatelliteLab allows networking researchers to evaluate their prototypes over highly heterogeneous networks.

2. SatelliteLab can be used to debug the performance and behavior of deployed Internet systems. When a system behaves in a surprising manner on the Internet, researchers can use SatelliteLab to recreate the network conditions required to reproduce and understand performance issues of deployed systems.

3. SatelliteLab can be used as a measurement testbed for observing characteristics of different network environments, including wireless and broadband networks.

8 Conclusion and Future Work

In this section, we describe the high-level contributions of this thesis and discuss potential future research directions.

8.1 Summary

In the course of this thesis we developed a number of systems and tools designed to provide more transparency in broadband access networks for users, developers, and researchers.

We started by developing a novel measurement methodology that allows the study of broadband networks with minimal end host cooperation. In contrast to previous measurement tools, which have often required control over the measured host thus limiting the number of hosts that can be measured, our technique enabled us to study broadband access networks at scale for the first time. While we used this methodology to study broadband networks, the methodology can be used to study other types of networks as well.

We used this methodology to perform the first large-scale study of the characteristics of residential broadband access networks of major ISPs in Europe and North America. We characterized bandwidth, latencies, and loss rates of broadband networks and were able to show important differences between broadband and academic networks. For instance, we were the first to point out that many broadband hosts have deployed surprisingly long router queues, which can significantly affect the performance of latency-sensitive applications, such as VoIP and VoD.

Next, we developed the Glasnost system that allows users to test their access links for traffic differentiation. One of our design principles for Glasnost was to make it easy-to-use. We believe that this is one of the reasons that Glasnost was able to attract hundreds of thousands of users to date. The success of Glasnost inspired M-Lab, a platform to deploy measurement tools to enhance network transparency, which is supported by Google, PlanetLab, and other researchers. While our original version of Glasnost focused on the detection of blocking and throttling of BitTorrent traffic, we designed Glasnost to be extensible, allowing users to create and run their own Glasnost tests for arbitrary application traffic.

Using the data collected by Glasnost, we conducted the first large-scale study of the prevalence of traffic differentiation in broadband access networks. We were able to identify a number of major ISPs, including Comcast and Cox in the USA and StarHub in Singapore, that blocked BitTorrent traffic by injecting TCP RST packets into the transfer. Our data indicates that most ISPs stopped blocking BitTorrent traffic shortly after our results were widely covered in the media and caught the attention of telecommunication regulators. Since then, we found that ISPs rather than blocking increasingly throttle BitTorrent traffic.

We then turned our attention to realistic evaluation of protocols and systems in broadband networks. To evaluate and study transport protocol behavior at large scale over

diverse Internet paths, we developed Monarch. Monarch allows researchers to emulate transport protocol flows to a large number of hosts on the Internet without requiring direct access to them. This enables researchers to study the behavior of transport protocols in the wild, and in particular in broadband networks. For instance, using Monarch we were able to show that the large router queues that are common in broadband networks can lead to high loss rates and high packet latencies. This can be problematic for widely deployed TCP congestion control algorithms that interpret loss events as an indication of congestion.

Finally, we presented SatelliteLab, a new testbed design that makes it easy to add broadband nodes to existing Internet testbeds such as PlanetLab. SatelliteLab solves the problem that today's testbeds mostly consist of well-connected academic nodes, and thus do not cover the heterogeneity of the Internet. In fact, given the high requirements for testbed nodes, it is often not possible for broadband nodes to join a testbed. SatelliteLab makes it possible to supplement testbeds with arbitrary nodes, including broadband nodes and even smartphones, thus vastly increasing their heterogeneity. Using SatelliteLab, we were able to identify several issues with an overlay multicast system and a network coordinate system in broadband environments that did not occur when evaluating these systems in state-of-the-art testbeds.

8.2 Future Work

While the work presented in this thesis significantly enhances network transparency in broadband networks, there are a number of future directions we are considering.

Although Glasnost is accurate in the detection of traffic differentiation, it cannot infer where on the path network management equipment is deployed. We currently compensate for this problem by presenting aggregate results that point on a specific ISP given that a large fraction of its users are affected by traffic differentiation. To determine which ISP along an Internet path employs traffic differentiation techniques, we want to enhance Glasnost with network tomography techniques.

While today most home users use DSL, cable, or dial-up to access the Internet, other emerging technologies might soon play an important role. ISPs increasingly deploy fiber-to-the-home, which boosts network speeds to 100 Mbps and more. At the same time, the recent popularity of smartphones like the iPhone or Google's Android platform made ISPs offer affordable flat-rate plans for cellular networks. Thus, we expect to see a many people connecting to the Internet using mobile connectivity technologies in the near future.

Similar to DSL and cable before, these networks are not widely studied by the research community and it is not clear how well current popular applications work in these environments. Cellular networks in particular have very different characteristics than other types of networks. They likely suffer from high latencies and loss rates, and it is unclear how these characteristics affect existing transport protocols and how popular applications like VoIP or VoD will work in this environment. As a result, studying these networks at scale with existing or novel measurement tools is an important research direction.

Furthermore, cellular ISPs are known to employ traffic management. But as ISPs are hesitant to reveal details about their practices, cellular networks (similar to other broadband networks) remain opaque to their users, and there is only anecdotal evidence about the traffic management techniques used by cellular ISPs. For instance, many cellular ISPs

use so-called performance-enhancing proxies (PEPs). PEPs improve users' web experience and speed up downloads by compressing webpages and re-encoding images, thus requiring less bandwidth. In addition, some ISPs have announced that they will filter VoIP traffic in their networks [O'B10]. Often, cellular networks are also heavily firewalled or use NATs to prohibit connections from a remote host to a cellular node.

While we already showed how to add cellular nodes to existing testbeds using SatelliteLab, studying these networks at scale is a subject for future work. Also, a mobile version of Glasnost could make these networks more transparent, detecting the presence and effect of PEPs, firewalls, and NATs.

Bibliography

[3GP08a] 3GPP: 3rd Generation Partnership Project. 3gpp specification series: 25 series, 2008. `http://www.3gpp.org/article/umts`.

[3GP08b] 3GPP: 3rd Generation Partnership Project. 3gpp specification series: 36 series, 2008. `http://www.3gpp.org/ftp/Specs/html-info/36-series.htm`.

[ABKM03] David G. Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Experience with an Evolving Overlay Network Testbed. *ACM SIGCOMM Computer Communication Review*, 33(3), 2003.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: An Experiment in Public-resource Computing. *Communications of the ACM*, 45(11):56–61, 2002.

[ACKZ06] Thomas Anderson, Andrew Collins, Arvind Krishnamurthy, and John Zahorjan. PCP: Efficient Endpoint Congestion Control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.

[ADLY95] Jong Suk Ahn, Peter B. Danzig, Zhen Liu, and Limin Yan. Evaluation of TCP Vegas: Emulation and Experiment. In *Proceedings of the ACM SIGCOMM Conference*, Aug 1995.

[AEO03] Mark Allman, Wesley M. Eddy, and Shawn Ostermann. Estimating Loss Rates With TCP. *ACM SIGMETRICS Performance Evaluation Review*, 31(3):12–24, 2003.

[AKM04] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2004.

[AKM05] Martin Arlitt, Balachander Krishnamurthy, and Jeffrey C. Mogul. Predicting Short-Transfer Latency from TCP Arcana: A Trace-based Validation. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2005.

[Alb08] Kamiel Albrecht. Dutch Broadband Q1 2008, 2008. `http://www.telecompaper.com/news/article.aspx?cid=621504`.

[Ama05] Amazon.com, Inc. Amazon Mechanical Turk, 2005. `http://www.mturk.com`.

[And04] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the International Workshop on Grid Computing*, Nov 2004.

[ASB05] Shilpi Agarwal, Joel Sommers, and Paul Barford. Scalable Network Path Emulation. In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep 2005.

[ASS03] Aditya Akella, Srinivasan Seshan, and Anees Shaikh. An Empirical Evaluation of Wide-area Internet Bottlenecks. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2003.

[Azu09] Azureus Wiki. List of ISPs suspected to traffic shape BitTorrent, 2009. `http://www.azureuswiki.com/index.php/Bad_ISPs`.

[BBB07] Robert Beverly, Steven Bauer, and Arthur Berger. The Internet's Not a Big Truck: Toward Quantifying Network Neutrality. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Apr 2007.

[Bee99] Justin Beech. DSLreports.com, 1999. `http://www.dslreports.com`.

[BFH+06] Andy C. Bavier, Nick Feamster, Mark Huang, Larry L. Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of the ACM SIGCOMM Conference*, Sep 2006.

[BGG+08] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. Experimental Study of Router Buffer Sizing. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2008.

[BGP04] S. Banerjee, T. G. Griffin, and M. Pias. The Interdomain Connectivity of PlanetLab Nodes. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Apr 2004.

[BMGS08] Kevin Bauer, Damon McCoy, Dirk Grunwald, and Douglas Sicker. Broadband network management, Apr 2008. `http://systems.cs.colorado.edu/mediawiki/index.php/Broadband_Network_Management`.

[Bol93] Jean-Chrysostome Bolot. Characterizing End-to-End Packet Delay and Loss in the Internet. In *Proceedings of the ACM SIGCOMM Conference*, Sep 1993.

[BP95] Lawrence S. Brakmo and Larry Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communication*, 13(8):1465–1480, 1995.

[BPS+98] Hari Balakrishnan, Venkata N. Padmanbhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings of the IEEE INFOCOM Conference*, Mar 1998.

[BPSK97] Hari Balakrishnan, Venkat Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[BS02] John Bellardo and Stefan Savage. Measuring Packet Reordering. In *Proceedings of the Internet Measurement Workshop (IMW)*, Nov 2002.

[BTA+99] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. GloMoSim: A Scalable Network Simulation Environment. Technical Report 990027, UCLA Computer Science Department, 1999.

[Cab06] CableLabs. DOCSIS 1.1 interface specification, 2006. `http://www.cablemodem.com/specifications/specifications11.html`.

[Can08a] Canadian Radio-television and Telecommunications Commission. Application requesting certain orders directing Bell Canada to cease and desist from throttling its wholesale ADSL Access Services, 2008. `http://www.crtc.gc.ca/PartVII/eng/2008/8622/c51_200805153.htm`.

[Can08b] Canadian Radio-television and Telecommunications Commission. Review of the Internet traffic management practices of Internet service providers, 2008. `http://crtc.gc.ca/PartVII/eng/2008/8646/c12_200815400.htm`.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct 2003.

[CDM] CDMA Development Group. CDMA2000 1xEV-DO Release 0. `http://www.cdg.org/technology/3g_1xEV-DO.asp`.

[CEUB08] Daniele Croce, Taoufik En-Najjary, Guillaume Urvoy-Keller, and Ernst Biersack. Capacity Estimation of ADSL links. In *Proceedings of the CoNEXT Conference*, Dec 2008.

[CEUB09] Daniele Croce, Taoufik En-Najjary, Guillaume Urvoy-Keller, and Ernst Biersack. Fast Available Bandwidth sampling for ADSL links: Rethinking the estimation for larger-scale measurements. In *Proceedings of the Passive Active Measurement Workshop (PAM)*, Apr 2009.

[CFEK06] Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, and Akira Kato. The Impact and Implications of the Growth in Residential User-to-User Traffic. In *Proceedings of the ACM SIGCOMM Conference*, Sep 2006.

[CGN+04] Yang-hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *Proceedings of the USENIX Annual Technical Conference*, Jun 2004.

[CHC+04] Yu-Chung Cheng, Urs Hoelzle, Neal Cardwell, Stefan Savage, and Geoffrey M. Voelker. Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying. In *Proceedings of the USENIX Annual Technical Conference*, Jun 2004.

[Cis03] Cisco Systems Inc. *Internetworking Technology Handbook*. Cisco Press, 4th edition, 2003.

[CKL+04] Mark Claypool, Robert Kinicki, Mingzhe Li, James Nichols, and Huahui Wu. Inferring Queue Sizes in Access Networks by Active Measurement. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Apr 2004.

[CM01] John G. Cleary and H. Stele Martin. Estimating Bandwidth from Passive Measurement Traces. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Apr 2001.

[Coh01] Bram Cohen. Bittorrent, 2001. `http://bittorrent.org`.

[Coh08] Bram Cohen. The BitTorrent Protocol Specification, Version 11031, 2008. `http://bittorrent.org/beps/bep_0003.html`.

[Com07] Comcast Corp. Comcast unleashes its innovative PowerBoost Technology on upstream speed, Aug 2007. `http://comcastcalifornia.mediaroom.com/index.php?s=43&item=170`.

[Com08a] Comcast Corp. Description of planned network management practices to be deployed following the termination of concurrent practices, 2008. `http://downloads.comcast.net/docs/Attachment_B_Future_Practices.pdf`.

[Com08b] Comcast Corp. In the Matter of Broadband Industry Practices: Comments of Comcast Corporation, Feb 2008. `http://fjallfoss.fcc.gov/ecfs/document/view?id=6519840991`.

[CPC+08] Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz. Tupni: Automatic Reverse Engineering of Input Formats. In *Proceedings of the Conference on Computer and Communications Security (CCS)*, Oct 2008.

[Cro07] Jon Crowcroft. Net Neutrality: The Technical Side of the Debate: A Qhite Paper. *ACM SIGCOMM Computer Communication Review*, 37(1), Jan 2007.

[CRSZ01] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2001.

[DCGN03] Michael Dahlin, Bharat Baddepudi V. Chandra, Lei Gao, and Amol Nayate. End-to-end wan service availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of the SIGCOMM Conference*, Aug 2004.

[DHB+08] Marcel Dischinger, Andreas Haeberlen, Ivan Beschastnikh, Krishna P. Gummadi, and Stefan Saroiu. SatelliteLab: Adding Heterogeneity to Planetary-Scale Network Testbeds. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2008.

[DHGS07] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2007.

[DMG+10] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P. Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling End Users to Detect

Traffic Differentiation. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[DMHG08] Marcel Dischinger, Alan Mislove, Andreas Haeberlen, and Krishna P. Gummadi. Detecting BitTorrent Blocking. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2008.

[DRM04] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet dispersion techniques and a capacity estimation methodology. *IEEE/ACM Transactions on Networking*, Dec 2004.

[EBN08] Brian Eriksson, Paul Barford, and Robert Nowak. Network discovery from passive measurements. *ACM SIGCOMM Computer Communications Review*, 38(4):291–302, 2008.

[Ele08] Electronic Frontier Foundation. "Test Your ISP" Project, 2008. `http://www.eff.org/testyourisp`.

[ESL07] Eric Eide, Leigh Stoller, and Jay Lepreau. An Experimentation Workbench for Replayable Networking Research. In *Proceeedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2007.

[Fed05] Federal Communications Commission (FCC). Policy Statement 05-151 on Network Neutrality, 2005. `http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-05-151A1.pdf`.

[Fed06] Federal Department of Environment, Transport, Energy and Communications, Switzerland. Broadband in the universal service, Sep 2006. `http://www.uvek.admin.ch/dokumentation/00474/00492/index.html?lang=en&msg-id=7308`.

[Fed09] Federal Ministry of Economics and Technology, Germany. The Federal Government's Broadband Strategy, 2009. `http://www.bmwi.de/English/Redaktion/Pdf/broadband-strategy,property=pdf,bereich=bmwi,sprache=en,rwb=true.pdf`.

[FFM04] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing Content Publication with Coral. In *Proceeedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Mar 2004.

[FHG04] Sally Floyd, Tom Henderson, and Andrei Gurtov. RFC 3782 – The NewReno Modification to TCP's Fast Recovery Algorithm, Apr 2004. `http://www.rfc-editor.org/rfc/rfc3782.txt`.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2000.

[FIR] FIRE: Future Internet Research & Experimentation. `http://cordis.europa.eu/fp7/ict/fire/`.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[Flo03] Sally Floyd. RFC 3649 – HighSpeed TCP for Large Congestion Windows, Dec 2003. `http://www.rfc-editor.org/rfc/rfc3649.txt`.

[Fre07] The FreePastry Web Site, 2007. `http://www.freepastry.org`.

[FSK05] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer Communication Across Network Address Translators. In *Proceedings of the USENIX Annual Technical Conference*, Apr 2005.

[FXAM04] Jinliang Fan, Jun Xu, Mostafa H. Ammar, and Sue B. Moon. Prefix-Preserving IP Address Anonymization. *Computer Networks*, 46(2):253–272, 2004.

[GEN] GENI: Global Environment for Network Innovations. `http://www.geni.net`.

[GMG+04] Krishna P. Gummadi, Harsha Madhyastha, Steven D. Gribble, Henry M. Levy, and David J. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec 2004.

[Gol08a] Alex Goldman. Top 23 U.S. ISPs by Subscriber: Q3 2008. *ISP-Planet*, 2008. `http://www.isp-planet.com/research/rankings/usa.html`.

[Gol08b] Alex Goldman. Top Seven ISPs in Canada by Subscriber: Q2 2008 . *ISP-Planet*, 2008. `http://www.isp-planet.com/research/rankings/2008/canada+q2+2008.html`.

[GP02] Ramesh Govindan and Vern Paxson. Estimating Router ICMP Generation Delays. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Mar 2002.

[GSG02] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of the Internet Measurement Workshop (IMW)*, Nov 2002.

[GVV08] Diwaker Gupta, Kashi Vishwanath, and Amin Vahdat. DieCast: Testing Distributed Systems with an Accurate Scale Model. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2008.

[GZCF06] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based Geolocation of Internet Hosts. *IEEE/ACM Transactions on Networking*, 14(6):1219–1232, 2006.

[HDGS06] Andreas Haeberlen, Marcel Dischinger, Krishna P. Gummadi, and Stefan Saroiu. Monarch: A Tool to Emulate Transport Protocol Flows over the Internet at Large. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2006.

[ICM+02] Gianluca Iannaccone, Chen-nee Chuah, Richard Mortier, Supratik Bhattacharyya, and Christophe Diot. Analysis of Link Failures in an IP Backbone. In *Proceedings of the Internet Measurement Workshop (IMW)*, 2002.

[Ini09] Initiative D21. (N)ONLINER Atlas, 2009. `http://www.initiatived21.de/category/nonliner-atlas`.

[Int09] International Computer Science Institute. The ICSI Netalyzr, 2009. `http://netalyzr.icsi.berkeley.edu`.

[ipo09] ipoque GmbH. OpenDPI, 2009. `http://www.opendpi.org`.

[ITU] ITU: International Telecommunication Union. IMT-2000. `http://www.itu.int/home/imt.html`.

[JBB92] Van Jacobson, Robert Braden, and David Borman. RFC 1323 - TCP Extensions for High Performance, May 1992. `http://www.rfc-editor.org/rfc/rfc1323.txt`.

[JD03] Manish Jain and Constantinos Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions on Networking*, Aug 2003.

[JID+04] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. Inferring TCP Connection Characteristics Through Passive Measurements. In *Proceedings of the IEEE INFOCOM Conference*, Mar 2004.

[JVC+03] T. Jehaes, D. De Vleeschauwer, T. Coppens, B. Van Doorselaer, E. Deckers, W. Naudts, K. Spruyt, and R. Smets. Access Network Delay in Networked Games. In *Proceedings of the Workshop on Network and System Support for Games (NetGames)*, May 2003.

[JWL04] Cheng Jin, David X. Wei, and Steven H. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. In *Proceedings of the IEEE INFOCOM Conference*, Mar 2004.

[KD10] Partha Kanuparthy and Constantine Dovrolis. DiffProbe: Detecting ISP Service Discrimination. In *Proceedings of the IEEE INFOCOM Conference*, 2010.

[Kel03] Tom Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[Kes91] Srinivasan Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of the ACM SIGCOMM Conference*, Sep 1991.

[KHR02] Dina Katabi, Mark Handley, and Charles Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2002.

[KY99] Maxim Krasnyansky and Maksim Yevmenkin. Universal TUN/TAP Driver for Linux, Solaris, and FreeBSD, 1999. `http://vtun.sourceforge.net/tun/index.html`.

[LGS07] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network Coordinates in the Wild. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2007.

[LLM88] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor – A Hunter of Idle Workstations. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Jun 1988.

[LP03] Karthik Lakshminarayanan and Venkata N. Padmanabhan. Some Findings on the Network Performance of Broadband Hosts. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2003.

[LPP04] Karthik Lakshminarayanan, Venkata N. Padmanabhan, and Jitendra Padhye. Bandwidth Estimation in Broadband Access Networks. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2004.

[LR08] Nikolaos Laoutaris and Pablo Rodriguez. Good Things Come to Those Who (Can) Wait – or How to Handle Delay Tolerant Traffic and Make Peace on the Internet. In *Proceedings of the ACM HotNets Workshop*, Oct 2008.

[LSB+05] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring Bandwidth between PlanetLab Nodes. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Mar 2005.

[MAF04] Alberto Medina, Mark Allman, and Sally Floyd. Measuring Interactions between Transport Protocols and Middleboxes. In *Proceedings of the Internet Measurement Conference (IMC)*, Aug 2004.

[MAF05] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the Evolution of Transport Protocols in the Internet. *ACM SIGCOMM Computer Communication Review*, 35(2), 2005.

[Mar06] John Markoff. 'Neutrality' Is New Challenge for Internet Pioneer. New York Times, Sep 27th, 2006. `http://www.nytimes.com/2006/09/27/technology/circuits/27neut.html`.

[MCG+01] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MobiCom)*, Jul 2001.

[MD90] Jeffrey Mogul and Steve Deering. RFC 1191 – Path MTU Discovery, 1990. `http://www.rfc-editor.org/rfc/rfc1191.txt`.

[ME04] Madanlal Musuvathi and Dawson R. Engler. Model-checking Large Network Protocol Implementations. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Mar 2004.

[Mea] Measurement Lab. `http://www.measurement-lab.net`.

[MFPA09] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the Internet Measurement Conference (IMC)*, Nov 2009.

[MHR08] Matt Mathis, John Heffner, and Raghu Reddy. NPAD: Network Path and Application Diagnosis, 2008. `http://www.psc.edu/networking/projects/pathdiag`.

[MLAW99] Jeonghoon Mo, Richard J. La, Venkat Anatharam, and Jean Walrand. Analysis and Comparison of TCP Reno and Vegas. In *Proceedings of the IEEE INFOCOM Conference*, Mar 1999.

[MM96] Matthew Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *Proceedings of the ACM SIGCOMM Conference*, Aug 1996.

[MMFR96] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. RFC 2018 - TCP Selective Acknowledgment Options, Oct 1996. `http://www.rfc-editor.org/rfc/rfc2018.txt`.

[MPHD06] Alan Mislove, Ansley Post, Andreas Haeberlen, and Peter Druschel. Experiences in Building and Operating ePOST, a Reliable Peer-to-Peer Application. In *Proceedings of the EuroSys Conference*, Apr 2006.

[MSWA03] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson. User-level Internet Path Diagnosis. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, Oct 2003.

[Net03] Netfilter Project. Netfilter: Firewalling, NAT, and packet mangling for Linux, 2003. `http://www.netfilter.org`.

[Nie04] Nielsen/NetRatings. U.S. Broadband Connections Reach Critical Mass, 2004. `http://www.nielsen-netratings.com/pr/pr_040818.pdf`.

[ns2] The network simulator – ns2. `http://www.isi.edu/nsnam/ns/`.

[NZ02] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of the IEEE INFOCOM Conference*, Jun 2002.

[O'B10] Kevin J. O'Brien. Skype in a Struggle to Be Heard on Mobile Phones. The New York Times, Feb 18th, 2010. `http://www.nytimes.com/2010/02/18/technology/18voip.html`.

[OEC09] OECD Broadband Portal, 2009. `http://www.oecd.org/sti/ict/broadband`.

[Ook06] Ookla Net Metrics. The Global Broadband Speed Test, 2006. `http://www.speedtest.net`.

[OSG07] Paul Ohm, Douglas Sicker, and Dirk Grunwald. Legal Issues Surrounding Monitoring During Network Research. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2007. Invited Paper.

[PAM02] Vern Paxson, Andrew K. Adams, and Matt Mathis. Experiences with NIMI. In *Proceedings of the Symposium on Applications and the Internet (SAINT)*, Feb 2002.

[PAPL06] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The Devil and Packet Trace Anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1), 2006.

[Pax97] Vern Paxson. End-to-end Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[Pax99] Vern Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[Pax04] Vern Paxson. Strategies for Sound Internet Measurement. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2004.

[PF01] Jitendra Padhye and Sally Floyd. Identifying the TCP Behavior of Web Servers. In *Proceedings of the ACM SIGCOMM Conference*, Jun 2001.

[PHM06] Himabindu Pucha, Y. Charlie Hu, and Z. Morley Mao. On the Impact of Research Network Based Testbeds on Wide-area Experiments. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2006.

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

[Pla02] PlanetLab. Hosting Requirements, 2002. `http://www.planet-lab.org/hosting/`.

[Pos81] Jon Postel. RFC 792 – Internet Control Message Protocol, 1981. `http://www.rfc-editor.org/rfc/rfc792.txt`.

[PP04] KyoungSoo Park and Vivek Pai. CoTop Monitoring Tool, 2004. `http://codeen.cs.princeton.edu/cotop/`.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the Middleware Conference*, Nov 2001.

[RDS+07] Robert Ricci, Jonathon Duerig, Pramod Sanaga, Daniel Gebhardt, Mike Hibler, Kevin Atkinson, Junxing Zhang, Sneha Kasera, and Jay Lepreau. The Flexlab Approach to Realistic Evaluation of Networked Systems. In *Proceeedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2007.

[RFB01] K. K. Ramakrishnan, Sally Floyd, and David L. Black. RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP, Sep 2001. `http://www.rfc-editor.org/rfc/rfc3168.txt`.

[RHE99] Reza Rejaie, Mark Handley, and Deborah Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the IEEE INFOCOM Conference*, Mar 1999.

[Riz97] Luigi Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM SIGCOMM Computer Communications Review*, 1997.

[Röt08] Janko Röttgers. Internetanbieter bremst Tauschbörsen aus. Focus Online, Mar 6th, 2008. `http://www.focus.de/digital/internet/kabel-deutschland_aid_264070.html`.

[Sam03] SamKnows Ltd. Samknows.com, 2003. `http://www.samknows.com`.

[Sav99] Stefan Savage. Sting: a TCP-based Network Measurement Tool. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Oct 1999.

[SCUB07] Matti Siekkinen, D. Collange, Guillaume Urvoy-Keller, and Ernst Biersack. Performance Limitations of ADSL Users: A Case Study. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, Apr 2007.

[SDRL09] Pramod Sanaga, Jonathon Duerig, Robert Ricci, and Jay Lepreau. Modeling and Emulation of Internet Paths. In *Proceeedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2009.

[SGG02] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, Jan 2002.

[SGMZ04] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs, and Hui Zhang. The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2004.

[Sim92] Simulcraft Inc. OMNeT++, 1992. `http://www.omnetpp.org`.

[SK02] Pasi Sarolahti and Alexey Kuznetsov. Congestion Control in Linux TCP. In *Proceedings of the USENIX Annual Technical Conference*, Jun 2002.

[SKMM05] Neel Shah, Demetres Kouvatsos, Jim Martin, and Scott Moser. A Tutorial on DOCSIS: Protocol and Performance Models. In *Proceedings of the International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs)*, Jul 2005.

[SM09] Hendrik Schulze and Klaus Mochalski. ipoque: Internet Study 2008/2009, 2009. `http://www.ipoque.com/resources/internet-studies`.

[SMWA04] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[Sny08] Alan Snyder. Vuze Plugin: Network Status Monitor, 2008. `http://azureus.sourceforge.net/plugin_details.php?plugin=aznetmon`.

[SPBP05] Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. In *Proceedings of the Workshop on Real, Large Distributed Systems (WORLDS)*, Dec 2005.

[SR04] Charles Robert Simpson, Jr. and George F. Riley. NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements. In *Proceedings of the Passive and Active Measurement Workshop (PAM)*, Apr 2004.

[SS05] Yuval Shavitt and Eran Shir. DIMES: Let the Internet Measure Itself. *ACM SIGCOMM Computer Communication Review*, 35(5):71–74, 2005.

[SS06] Rob Sherwood and Neil Spring. Touring the Internet in a TCP Sidecar. In *Proceedings of the Internet Measurement Conference (IMC)*, Oct 2006.

[SSBK03] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. OverQoS: Offering Internet QoS using Overlays. *ACM SIGCOMM Computer Communications Review*, 33(1):11–16, 2003.

[SSW09] Timothy J. Smith, Stefan Saroiu, and Alec Wolman. BlueMonarch: A System for Evaluating Bluetooth Applications in the Wild. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Jun 2009.

[Sto07] Brad Stone. Comcast: We're Delaying, Not Blocking, BitTorrent Traffic. New York Times Online, Oct 22th, 2007. `http://bits.blogs.nytimes.com/2007/10/22/comcast-were-delaying-not- blocking-bittorrent-traffic`.

[SW02] Subharata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the Internet Measurement Workshop (IMW)*, Nov 2002.

[SWA03] Neil Spring, David Wetherall, and Tom Anderson. Scriptroute: A Public Internet Measurement Facility. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar 2003.

[Tel07] Telco 2.0. Market Dynamics - UK Broadband, Q3 07, 2007. `http://www.telco2.net/blog/2007/11/market_dynamics_uk_broadband_1.html`.

[The07] The Associated Press. F.T.C. Urges Caution on Net Neutrality. New York Times, Jun 28th, 2007. `http://www.nytimes.com/2007/06/28/technology/28net.html`.

[The08] The Associated Press. F.C.C. Chairman Favors Penalty on Comcast. New York Times, Jul 11th, 2008. `http://www.nytimes.com/2008/07/11/technology/11fcc.html`.

[The09a] The 111th Congress of the USA. American Recovery and Reinvestment Act of 2009, 2009. Public Law 111-5.

[The09b] The European Commission. Commission declaration on net neutrality. *Official Journal of the European Union*, 52(C 308):2, Dec 2009. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:C:2009:308:0002:0002:EN:PDF`.

[The09c] The IEEE 802.16 Working Group. IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Broadband Wireless Access Systems. *IEEE Std 802.16-2009 (Revision of IEEE Std 802.16-2004)*, 2009.

[TMFA09] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. Detecting Network Neutrality Violations with Causal Inference. In *Proceedings of the CoNEXT Conference*, Dec 2009.

[Top07] Robb Topolski. Comcast is using Sandvine to manage P2P connections, May 2007. `http://www.dslreports.com/forum/r18323368-Comcast-is-using-Sandvine-to-manage-P2P-Connections`.

[Uni81] University of Southern California, Information Sciences Institute. RFC 793 – Transmission Control Protocol, 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[Uni01] United Kingdom e-Minister and e-Envoy. UK online: The broadband future, 2001. `http://archive.cabinetoffice.gov.uk/e-envoy/reports-broadband/$file/ukonline.pdf`.

[Vel08] Velocix. Velocix Metro: New Generation Content Delivery Network, 2008. `http://www.velocix.com`.

[VKD02] Arun Venkataramani, Ravi Kokku, and Mike Dahlin. TCP Nice: A Mechanism for Background Transfers. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec 2002.

[VS94] Curtis Villamizar and Cheng Song. High Performance TCP in ANSNET. *ACM SIGCOMM Computer Communication Review*, 24(5):45–60, 1994.

[VYW+02] Amin Vahdat, Ken Yocum, Kevin Walsh, Pryia Mahadevan, Dejan Kostic, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[Win09] Windsor Oaks Group, LLC. Market Outlook Report: Global Fixed Broadband Subscriber Forecast, Apr 2009. `http://www.broadbandtrends.com/Report_Summary/BBT_GlobalBBOutlook_091140_Summary.htm`.

[WLS+02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec 2002.

[WPP02] Limin Wang, Vivek Pai, and Larry Peterson. The Effectiveness of Request Redirection on CDN Robustness. *ACM SIGOPS Operating Systems Review*, 36(SI):345–360, 2002.

[WPP+04] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S. Pai, and Larry Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *Proceedings of the USENIX Annual Technical Conference*, Jun 2004.

[WSS07] Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer. Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. In *Proceeedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr 2007.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control for Fast Long-Distance Networks. In *Proceedings of the IEEE INFOCOM Conference*, Mar 2004.

[XYK+08] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4P: Provider Portal for Applications. In *Proceedings of the ACM SIGCOMM Conference*, Aug 2008.

[ZMZ08] Ying Zhang, Zhuoqing Morley Mao, and Ming Zhang. Ascertaining the Reality of Network Neutrality Violation in Backbone ISPs. In *Proceedings of the ACM HotNets Workshop*, Oct 2008.

[ZMZ09] Ying Zhang, Zhuoqing Morley Mao, and Ming Zhang. Detecting Traffic Differentiation in Backbone ISPs with NetPolice. In *Proceedings of the Internet Measurement Conference (IMC)*, Nov 2009.