CALIBRATED NETWORK MEASUREMENT

by

Joel Edward Sommers

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

To Amy.

ACKNOWLEDGMENTS

I left graduate school for the first time in May 1997. I was happy to be finished, having grown weary of the sleep-deprived life of a grad student. My adviser at the time had encouraged me to consider going on for a Ph.D. in the future, and in the back of my mind I wondered if I might ever actually do that. Doubtful, was the answer.

Part way through an odd string of working for companies that fell victim to the corporate buy-out mania of the late 1990s, I began to reevaluate the possibility of becoming a student again. This was prior to the dot-bomb period, so it wasn't because of economic downturn or job uncertainty that I was having second thoughts. Rather, in each job I found that after conquering the steep learning curve of the first several months, I became bored. I came to realize that more than anything, I loved the challenge of the learning process of those first few months. For me, this epiphany meant that I had to seriously consider going to grad school, doing research, and making a career out of teaching and learning.

Looking back, I have benefited greatly from the friendship, encouragement, and wisdom of many people during this initial navigation of a new path. To all, my heartfelt thanks. Now to name some names...

At the University of Wisconsin, I was fortunate to be assigned for my first semester as a teaching assistant for CS640, taught by Paul Barford. Upon finding out that I was interested in networking research, Paul dangled (what I thought at the time) was a small research project in my direction to be worked on in my copious spare time. Paul, thank you for letting me get that proverbial foot in the door, and for pushing me and inspiring me since then at critical points over the past six-odd years. Thanks to your guidance and encouragement I've never had second thoughts (serious ones, at least) about returning to grad school. The conference travel hasn't hurt, either. You are a mentor and a friend, and I sincerely thank you.

I thank Craig Wills for planting the initial seed by encouraging me to think about continuing on for a Ph.D. I thank Mark Quinlivan for his advice, friendship, and mentorship to me. Finally, I thank Ryan Hart and Dani Dudovitz, coworkers of mine for too short a time. Thank you both for your friendship and for keeping my short-timer status at FOC on the down-low for all those months.

I have had the pleasure of working and interacting with some incredibly insightful researchers both within and outside the University of Wisconsin over the past few years. Thanks in particular to Walter Willinger, Amos Ron, Nick Duffield, Vinod Yegneswaran, and Cristian Estan.

A major part of my research while at Wisconsin has been focused in the lab. My sincere thanks to Ana Bizarro and Mike Blodgett for their help in keeping things running. Many headaches were avoided and/or fixed thanks to them.

I thank Eduardo Urbina, Jared Bruckner, and Cliff Pope for helping me to build foundational knowledge and for their encouragement in that process.

I thank Karla Schmidt for placing an advert for a discussion group, which which brought me in contact with Doug Pierce, who encouraged me to take a trip to Costa Rica, where I met Amy. Your friendship and support to me has meant more than my pea-sized vocabulary can express. Thank you.

My love of learning started early in life, thanks to dedicated and loving parents. Thank you, Mom and Dad, for your love and support all the way through. You are the hardest working people I know; thank you for instilling that work ethic in me. I love you and thank you from the bottom of my heart.

Jeff and Michele, thank you for your love, encouragement, and willingness to let me crash when I needed to get out of Madison. Ethan and Leo, your laughter and smiles helped me to regain a small amount of sanity. Thank you.

My dearest Amy, amid all the adventure and insanity of the past couple years, thank you for making me take time to play, to get my hands dirty, and to breathe. Your love and support mean more to me than you (or I) may ever know. With all that I am, thank you. A million times, thank you.

DISCARD THIS PAGE

TABLE OF CONTENTS

DISCARD THIS PAGE

LIST OF TABLES

Table Page

DISCARD THIS PAGE

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

DISCARD THIS PAGE

ACRONYMS

ABET	Available bandwidth estimation tool
ACK	Acknowledgment
AQM	Active queue management
ARP	Address resolution protocol
ARPA	Advanced research projects agency
AS	Autonomous system
BDP	Bandwidth-delay product
BSCL	Buffer-sizing for congested links
CE	Customer edge
CIDR	Classless-interdomain routing
CPU	Central processing unit
DETER	The Cyber Defense Technology Experimental Research testbed
DNS	Domain name system
ECN	Explicit congestion notification
FE	Fast Ethernet
FTP	File transfer protocol
GE	Gigabit Ethernet
GSR	Gigabit Switch Router

HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPMA	Internet Performance Measurements and Analysis
IPPM	Internet Protocol Performance Metrics
ISP	Internet Service Provider
LAN	Local Area Network
MPLS	Multi-protocol label switching
MRTG	Multi-router Traffic Grapher
MTU	Maximum Transmission Unit
NIST	National Institute of Standards and Technology
OC	Optical Carrier
OS	Operating System
PASTA	Poisson Arrivals See Time Averages
PCI	Peripheral Component Interconnect
PCT	Pair-wise comparison test
PDT	Pair-wise difference test
PE	Provider Edge
PING	Packet Internet Groper
POP	Point of Presence
POSIX	Portable Operating System Interface

RAM	Random Access Memory
RED	Random Early Detection
RFC	Request for Comments
RON	Resilient Overlay Network
RPC	Remote Procedure Call
RTT	Round-trip Time
SACK	Selective Acknowledgment
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SPLAT	Scatter and Phase Plot Animation Tool
SQL	Structured Query Language
TCP	Transmission Control Protocol
TTL	Time-to-live
UDP	User Datagram Protocol
VPN	Virtual Private Network
VLAN	Virtual Local Area Network
WAIL	Wisconsin Advanced Internet Laboratory
WAN	Wide-area Network
WRED	Weighted Random Early Detection
XML	eXtensible Markup Language

CALIBRATED NETWORK MEASUREMENT

Joel Edward Sommers

Under the supervision of Assistant Professor Paul Barford
At the University of Wisconsin-Madison

Empirical measurements of Internet paths and systems are essential for understanding the behavior and performance of the Internet, for managing operational networks, and for designing future Internet systems. In this dissertation we propose a framework and set of calibration techniques and tools that can greatly increase our confidence in the validity and accuracy of network measurement methodologies and the measurements they produce.

Our approach is based on performing experiments in a controlled laboratory environment. This setting offers control and repeatability of experiments. It also offers the ability to measure desired aspects of the system or measurement methodology under test, which is critical for evaluating the accuracy of measurements and for gaining insight into system behavior. Our laboratory-based approach also offers realism in terms of end hosts, routers, operating systems, and network protocol implementations. There have previously existed two significant obstacles associated with laboratory-based experiments in the area of realism: the ability to scalably produce a range of realistic traffic conditions, and the ability to recreate Internet-like link and path characteristics. We propose and evaluate two systems for overcoming these problems.

We describe three case studies that leverage the control, transparency, and realism offered in our laboratory calibration environment. These studies are in the areas of end-to-end available bandwidth measurement, measurement of packet loss characteristics, and measurement of the impact of router architecture, buffer size, and queuing policy on transport protocol performance. Through these investigations we illustrate the power and capabilites of our laboratory calibration approach. We develop significantly more accurate available bandwidth and packet loss measurement methodologies. We also illustrate the problems associated with recently proposed buffer sizing methodologies and how consideration of quality of service guarantees made by network providers illuminates important issues in this context.

Paul Barford

ABSTRACT

Empirical measurements of Internet paths and systems are essential for understanding the behavior and performance of the Internet, for managing operational networks, and for designing future Internet systems. In this dissertation we propose a framework and set of calibration techniques and tools that can greatly increase our confidence in the validity and accuracy of network measurement methodologies and the measurements they produce.

Our approach is based on performing experiments in a controlled laboratory environment. This setting offers control and repeatability of experiments. It also offers the ability to measure desired aspects of the system or measurement methodology under test, which is critical for evaluating the accuracy of measurements and for gaining insight into system behavior. Our laboratory-based approach also offers realism in terms of end hosts, routers, operating systems, and network protocol implementations. There have previously existed two significant obstacles associated with laboratory-based experiments in the area of realism: the ability to scalably produce a range of realistic traffic conditions, and the ability to recreate Internet-like link and path characteristics. We propose and evaluate two systems for overcoming these problems.

We describe three case studies that leverage the control, transparency, and realism offered in our laboratory calibration environment. These studies are in the areas of end-to-end available bandwidth measurement, measurement of packet loss characteristics, and measurement of the impact of router architecture, buffer size, and queuing policy on transport protocol performance. Through these investigations we illustrate the power and capabilites of our laboratory calibration approach. We develop significantly more accurate available bandwidth and packet loss measurement methodologies. We also illustrate the problems associated with recently proposed buffer sizing methodologies and how consideration of quality of service guarantees made by network providers illuminates important issues in this context.

Chapter 1

Introduction

> Conducting an Internet measurement study in a sound fashion can be much more difficult than it might first appear.
> —V. Paxson [172].

1.1 Motivation

Accurate and robust methods for measuring characteristics of Internet paths and systems are vital for understanding the behavior and performance of the Internet, for managing operational networks, and for the design of future Internet systems. The Internet has grown dramatically in scale and complexity since the introduction of its predecessor, the ARPAnet, over 30 years ago [40, 137]. With more nearly 500 million hosts, millions of routers, and over 10 thousand networks in today's Internet, it is virtually impossible to predict or understand its behavior and performance without empirically measuring it.

Measurements of Internet paths and systems have been and continue to be instrumental for developing new and improved protocols and systems, for gaining insight into the structure and behavior of the Internet at large, and for managing and provisioning service provider networks. For example, the NewReno and SACK algorithmic improvements to TCP were motivated and developed based on detailed measurements of TCP connections when subjected to loss [103, 116]. Later, Paxson's landmark studies of routes and packet dynamics in the late 1990's were the first to systematically examine the wide-area behavior of the then newly commercialized Internet [168, 169]. Today, network service providers rely more and more on measurements of their networks in order to improve and assure performance for their customers through traffic engineering, deployment, and provisioning of higher-capacity equipment and links.

The critical role of Internet measurement was highlighted by the National Research Council in their 2001 report "Looking Over the Fence at Networks: A Neighbor's View of Networking Research" [85]. In that report, a "back to basics" approach was recommended, with measurement as one of three elements of a framework for future research. New measurement efforts were advocated, including new algorithmic and system-level approaches to measurement,

and a broader effort toward measuring Internet-wide characteristics. Thus, from both a day-to-day operational perspective, and from a longer-term research perspective, the importance of developing accurate and robust methods for measuring aspects of Internet paths and systems cannot be understated.

Methods for measuring aspects of the Internet can be divided into two categories. *Passive measurement* uses monitors that are either attached to network links or available from network nodes such as routers. A standard passive measurement technique is to use the set of Management Information Base (MIB) counters available on network nodes via the Simple Network Management Protocol (SNMP) [76,196]. A benefit of passive monitoring systems is that many of the important details of local traffic behavior can be captured. However, the cost for this detail can be high (*e.g.,* in terms of data storage requirements) and access to links or routers is frequently not possible across administrative domains. Furthermore, although passive measurements may provide high accuracy for certain metrics on a link-by-link basis, they may be insuffient for gauging the performance of end-to-end application traffic.

Active measurement methods, on the other hand, send specially crafted probe packets into the network, measuring subsequent responses of the probe packets exiting at some other instrumented point in the network or reply packets returning to the sender. The benefits of probe-based measurement methodologies are that they can run virtually anywhere in the network, and they give an end-to-end perspective of network conditions, similar to what application traffic might experience. A problem active probing is that the discrete nature of active probing limits the resolution of the measurements (a standard problem in any measurement methodology based on sampling) [182].

Unfortunately, as the opening epigram suggests, Internet measurement is fraught with difficulties. While some of these problems have been mentioned in the previous two paragraphs, the most glaring obstacle is that measurement is not an inherent feature of the Internet. That is, measurement and monitoring of Internet paths and systems are not built into the fundamental architecture or specification of the Internet or its protocols. One implication of this fact is that protocol features and common implementation behaviors are often exploited and subverted for measurement purposes [90, 119, 142]. A related issue is that although end-to-end active measurements give a valuable and important application-like perspective of network path performance, they are often used along with sophisticated inference techniques to estimate link-level characteristics because link-by-link information is unavailable [91, 92, 141]. Second, as noted above, the Internet is huge, complex, and continually growing and changing. Thus, assessing the relevance of measurement data collected in one location of the Internet to understanding characteristics of the broader network is difficult, as is the issue of gauging the features of what is effectively a rapidly moving target. Finally, Internet service providers (ISPs) very often do not want outside researchers or operators to infer characteristics about or reverse engineer their network, such as its topology or reliability. In order to maintain aspects of competitive advantage over rival ISP's, they often work to ensure a shroud of privacy about their network and its performance. This can mean that an ISP will configure its routers not to respond to certain types of packets typically associated with active network measurement (*e.g.*, ICMP echo request packets). Though not usually intended as a cloaking mechanism, traffic engineering practices using MPLS or layer 2 VLANs are also problematic for Internet measurement since standard IP

(layer 3) packets used for active measurement may reveal only a simple path where in reality the path may be more complex. So indeed, measuring characteristics of the Internet can be significantly more difficult than it might first appear.

An approach typically followed by researchers in the design of a new measurement methodology is first to implement an algorithm in a simulation environment such as ns-2 [151], then to deploy an implementation *in situ*. Simulation has been traditionally used as an environment for analysis and study of a measurement algorithm because of its accessibility (ns-2 is free, open-source software), relative ease of use, ability to repeat experiments, and the ability to control, modify, or measure any aspect about the simulated network. This last capability implies that the accuracy of the measurement methodology can be assessed, since *ground truth* measures can be obtained from the simulation.

Although this approach—a combination of simulation studies followed by live deployment—has been employed frequently in network measurement research, there are three important issues that call its continued use into question. First, simulation environments use abstractions, often quite simple ones, of real Internet systems and traffic. For example, a router in ns-2 is implemented as a single collection of input and output "interfaces", with one output queue for each interface. A real Internet router, even a low-end system, is much more complex: there are input queues with virtual output queues, a switching fabric with specific arbitration protocols, multiple levels of output queuing, and many other system-specific features and implementation aspects that can significantly affect, for example, delay and loss characteristics of packets passing through the router, *e.g.*, see [4, 15]. The need and challenge to incorporate aspects of realism in a simulated network and the associated problem of using simulation in order to understand behavior of the live Internet is well known [39, 105, 106, 107]. Even so, simulation continues to be widely used as a mechanism for understanding the Internet. The second problem with the methodological approach described above is that once a measurement tool has been deployed in the live Internet, it is very difficult (typically impossible) to obtain ground truth measures. Thus, measurements obtained from experiments using the live Internet often cannot be judged for accuracy. They simply "are what they are," and researchers must use intuition and experience in order to have confidence that the methodology reports a true behavior of the Internet, rather than an artifact of the measurement technique. The third problem is that translating the measurement methodology software that has been developed in the simulation environment to a tool that can be deployed in the live Internet is a non-trivial task. For example, limitations of system timers, packet timestamps, and network interfaces of real end hosts and operating systems may invalidate basic assumptions of a measurement algorithm.

1.2 Approach

This dissertation explores a new approach for developing and studying Internet measurement methodologies. Our technique, which holds the necessity of ground truth as central, is based on the notion of *calibration*. We describe and examine a framework and an environment for calibration that can improve our understanding of, and increase our confidence in, the accuracy and soundness of network measurement methodologies, and of the measurements themselves.

Calibration refers on the one hand to *comparison and correlation* of measurements with a ground truth standard, and on the other hand to *adjustment* to match a ground truth standard. An example of the first aspect of calibration might be the comparison of estimates of end-to-end delay produced by a proposed methodology with reliable ground truth delay values. Such a comparison might reveal inaccuracies in the estimates, leading to adjustments in the estimation technique to more accurately reflect the true end-to-end delay. This latter activity of modifying the measurement methodology is an example of the second aspect of calibration. Calibration strategies are designed to detect errors in the underlying data, or problems in their analysis [172]. While calibration itself is not intended to achieve perfection, it can aid in our understanding of the limitations and applicability of measurement methodologies.

The foundation for our calibration approach is the design and execution of experiments in a controlled laboratory environment. There are three key benefits associated with this setting: (i) transparency and control over the testbed, (ii) system-level realism, and (iii) the measurement methodologies and tools developed in a realistic laboratory environment can be directly deployed in the live Internet. Transparency implies the ability to obtain ground truth measures while control implies the capability to perform *repeatable* experiments. Repeatability is the hallmark of scientific investigation and implies the opportunity to carefully subject measurement methodologies and systems to a range of experimental conditions in a consistent way in order to assess the reproducibility of measurement results. It also suggests the ability to carefully evalute the properties of a measurement methodology, such as its accuracy and robustness. It is important to note that transparency and control are traditional advantages of a simulation setting. System-level realism means that there are physical end hosts, with hardware and software, including operating systems, network protocol implementations, and applications, that are representative of hosts attached to the live Internet. Besides end hosts, there are also switches, routers, and physical network paths that are representative of the live Internet. While system-level realism is, by definition, not part of a simulation environment, it is clearly an advantage of tests using parts of the live Internet.

Thesis Statement

> We claim that a laboratory calibration environment offers the traditional advantages of simulation, namely transparency (*i.e.*, ground truth) and control, and with key enhancements can also offer the advantage of live environments, namely realism in terms of host system hardware and software, routers, network paths, and traffic. We further assert that such an environment is a valuable setting for the design and testing of network measurement algorithms, and for measurement and performance analysis of Internet systems. We support our claims through three case studies in which we develop new and accurate active measurement methodologies in the areas of available bandwidth and packet loss, and in which we evaluate the performance impact of router architecture, buffer size, and queuing policy (using drop-tail and AQM) on a variety of realistic traffic sources.

Although the advantages of transparency, control, and system-level realism in a laboratory setting are compelling, there are three primary technical obstacles to overcome with a laboratory calibration approach. First, there must be appropriate mechanisms for obtaining ground truth measures. Second, there must be mechanisms for creating realistic network paths and links. For example, propagation delays that are representative of links and paths in the wide-area Internet must be emulated within the physical confines of a laboratory setting. Finally, it is necessary to create a range of realistic network traffic conditions representative of the live Internet. We further discuss each of these three barriers below and presently note that solutions that we propose for the problems of creating realistic path and traffic conditions form key contributions of this dissertation.

For the ground truth measurement system, its main requirement is that it must deliver the required quantity or quantities with *precision* which is at least that of the measurement methodology under examination, and the *accuracy* delivered by this system must be reliable and near-perfect. It may, in fact, be necessary to periodically calibrate the ground truth system by comparing measurements from it with a system with similar capabilities (or even an identical but separate system, in order to guard against imperceptible hardware errors). In the experiments described in this dissertation, the system used for obtaining ground truth measurements is based on Endace DAG monitors [10]. These systems are highly precise and accurate passive monitoring cards which are available for a variety of physical networking interfaces, such as Gigabit Ethernet, OC-3, and OC-12. Each card is equipped with a high-frequency oscillator for timestamping packets and multiple cards can by synchronized either by an external source (*e.g.*, the Global Positioning System) or by pairing cards together. The result is that accurate timestamps with high precision (*i.e.*, at least on the order of a single microsecond) can be applied to packets at multiple monitoring points in a testbed. These high-quality passive packet traces are then used to construct the ground truth measures by which a probe-based measurement methodology's accuracy or the performance of a system under test can be assessed.

Instantiation of wide-area topologies in a laboratory setting requires the recreation of physical link and path conditions representative of the Internet. Emulating these condition is the second barrier that must be addressed for incorporating realism into a laboratory environment suitable for calibration. Emulation of *link* characteristics focuses on recreation of network conditions on a link connecting two physical (or virtual) nodes. In particular, two properties are of interest: *propagation delay* and *bit errors*. Propagation delay is the physical speed-of-light delay imposed on encoded bits as they travel on a physical medium (*e.g.*, optical fiber or copper wire) between two network nodes, and is modeled as a constant per-bit delay. Bit errors may be caused by imperfections in the physical path between two nodes. These are modeled as a probabilistic change of state per bit.

Emulation of *path* conditions in a laboratory environment considers the recreation of more general end-to-end packet-oriented conditions between two hosts in a network. As such, it subsumes the notions of link emulation and further includes the following: (i) *packet latency*, the aggregation of signaling delay caused by the initial transmission of packets by a network interface, propagation delay, and queuing delays caused by buffering of packets at routers

along a path; (*ii*) *packet loss*, the discarding of packets due to full buffers at routers along a path; (*iii*) *packet duplication*, the (generally erroneous) cloning of a packet at a router or routers along a path; and (*iv*) *packet reordering*, or permutating the order in which packets arrive at a router on the path.

We designed and implemented a new system called NETPATH to address the challenge of realistically emulating link and path conditions in a laboratory setting. Prior solutions to this problem are either expensive and inflexible hardware systems or are software-based systems that do not scale to high bandwidth links or large emulated delays. NETPATH is a software system that runs on commodity hardware and utilizes the Click [133] modular router for basic packet-processing functionality. We discuss how NETPATH offers substantially more flexibility in how it may be integrated into a testbed setup than any existing hardware or software solution. Furthermore, we show that NETPATH scales to Gigabit-speed links, accurately emulating desired link and path characteristics at these high bandwidths.

The third issue to address for creating a realistic laboratory environment is that of creating a range of traffic conditions representative of the live Internet. This challenge is significant: new user applications appear on the Internet continually and often arise rapidly, leading to an evolution in Internet traffic volume, composition, statistical features, and dynamic properties. Furthermore, malicious traffic in the form of worms and viruses proliferates across the Internet. Reproducible, realistic traffic conditions are necessary for experiments requiring background traffic representative of the live Internet, *e.g.*, in calibration experiments for testing proposed network measurement methodologies. Realistic traffic conditions are also necessary for tests in which the goal is to subject networking equipment such as routers, switches, and firewalls to a range of conditions that might be expected in a live environment. For such experiments, it is critical to examine the performance, robustness, and stability of networking infrastructure in the face of both expected and extreme conditions. Tools for creating a range of realistic traffic conditions are useful not only for simulation environments [27, 151], but also for laboratory testbeds using real networking hardware [31, 33], and in laboratory emulation environments [203, 209]. Without the capability to consistently create realistic network test conditions, new systems and network measurement methodologies run the risk of unpredictable behavior and unacceptable performance when deployed in live environments.

Previous methodologies for traffic generation have either used simple packet streams or have focused on recreating characteristics of a single application. Standard packet-oriented techniques include constant bit rate UDP streams, such as used by the `iperf` tool [198], and long-lived TCP sources, which have commonly been used in simulation studies. On the other hand, there have been a number of successful application-specific methodologies developed developed [59, 127]. While these latter techniques can be useful for reproducing statistical properties of the applications they model and for providing insight into some aspects of network system behavior, they are clearly designed to recreate only one type of traffic, not a diversity of traffic characteristics observed in today's Internet [108, 109, 174].

Based on these observations and limitations in existing traffic generation methodologies, we designed a network traffic generator called HARPOON that is capable of producing IP traffic flows representative of those observed at routers in the Internet. (A flow, in this context, is defined as a series of packets between a given IP address and port,

using a specific transport protocol.) Our approach is to represent IP flows as a series of application-independent file transfers that use standard TCP or UDP mechanisms for transport. We include both temporal (*i.e.*, modulation of traffic rate over time) and spatial (*i.e.*, relative frequencies of observed source and destination IP address) characteristics of flows in our model. The resulting model can be *self-parameterized* using components of the traffic generation tool along with measured flow records, such as Cisco NetFlow [6], or measured packet traces. HARPOON's capabilities result in traffic observed at the first-hop router (near the server) that is qualitatively the same in terms of byte, packet, and flow volumes as the traffic used to parameterize HARPOON. We note that HARPOON is used extensively for representative background traffic generation in the three case studies described in this dissertation, further demonstrating its usefulness and capabilities.

A fourth, non-technical barrier to the wide adoption of a laboratory calibration approach for network experiments is that of cost. In particular, a realistic laboratory setting requires routers and other types of equipments that are representative of those found in a variety of Internet settings, such as routers typically found in network "edge" environments (*e.g.*, a Cisco 7200 or a Juniper M10) , and those more commonly associated with "core" settings (*e.g.*, a Cisco GSR or CRS-1, or a Juniper M320 or T640). While addressing this issue is beyond the scope of this dissertation, we note that there are a number of laboratory testbeds are available for use by outside researchers, *e.g.*, WAIL [33], Emulab [9], and DETER [8]. While there are certainly limitations associated with the lack of physical access to devices, and other limitations associated with the particulars of each environment, it is likely that methods for sharing testbed resources will improve over time as researchers become aware of the benefits of using such environments.

We propose a framework for calibrating measurement methodologies and evaluating networked systems in a laboratory setting with the capabilities described above, namely, transparency, control, and realism. The framework identifies a set of key issues that must be considered during the calibration and evaluation of a network measurement methodology, or a networked system such as a router. For example, host system hardware and operating systems often impose certain limitations on methodologies for active measurement, and these measurement methodologies often include certain assumptions about the behavior of routers or switches that can vary significantly in practice. The framework also articulates a strategy for designing calibration tests. First, a range of appropriate configurations and repeatable conditions over which to exercise and evaluate the measurement methodology or networked system must be designed. Key to this step is to identify how ground truth measurements are to be obtained. The second step is to identify relevant test suites over which to expose limitations or assumptions of a measurement methodology or networked system. Third, the evaluation of data captured during the tests should be facilitated with flexible analysis and visualization techniques. Such analysis methodologies should provide insight into relevant behaviors or characteristics under study.

With respect to the third component of the calibration strategy, we designed and developed a novel visualization tool called SPLAT (scatter and phase-plot animation tool). Through a series of three examples, we demonstrate the capabilities of SPLAT and its usefulness for calibration studies as well as for visual data mining of large Internet data

sets. We use SPLAT in one of our calibration case studies described below, further underscoring its utility in this context.

Calibration Case Studies

Leveraging this transparent, controllable, and realistic laboratory environment, we designed significantly more accurate algorithms for measuring end-to-end available bandwidth and characteristics of packet loss. Furthermore, we demonstrate another advantage of the laboratory calibration testbed, which is the ability to accurately assess the performance of Internet systems, *e.g.*, routers, firewalls, and other devices, in a realistic setting. In particular, we perform a large-scale study of the impact of router architecture, buffer size, and queuing policy on a variety of traffic sources, examining several key traffic performance metrics. These three case studies comprise further contributions of this dissertation, and evidence supporting the thesis of this dissertation. We now briefly describe each of these case studies.

Our first case study considers the problem of estimating available bandwidth (AB) along an end-to-end network path. Estimating AB is a topic that has received a great deal of attention since the beginning of this decade [43, 75, 117, 121, 135, 152, 169, 180, 187, 188, 197]. Informally, end-to-end AB is defined as the minimum spare capacity on an end-to-end path between two end hosts. To calibrate and validate available bandwidth estimation tools (ABETs), an understanding of the how probe packets interact in network queues with background traffic packets in a realistic environment (*i.e.*, using real routers) is essential. Moreover, analysis of these effects requires fine-grained, time-synchronized measurements of packets as they arrive at and depart from routers along a network path—the kind of measurements available in our laboratory calibration environment.

We detail a series of experiments in which we examine the Spruce [197] and Pathload [121] ABETs using our calibration framework. We chose these two ABETs since they represent the two most common methodologies for available bandwidth estimation. We identify algorithmic shortcomings in these methodologies that lead to estimation inaccuracy. We design a calibrated ABET called YAZ that is based on Pathload. YAZ estimates AB much more accurately than either Spruce or Pathload, and is much less intrusive than Pathload. For example, in an experiment using realistic self-similar background traffic about 80% of YAZ estimates have relative error of at most 10%, whereas only about 55% of Pathload estimates have relative error of at most 10% and only 40% of Spruce estimates fall within this accuracy window. If Pathload and Spruce represent a tradeoff between measurement overhead and measurement accuracy, our results for YAZ show that this tradeoff is not fundamental. For a Spruce-like budget, YAZ is more accurate than Pathload, sometimes significantly so.

In our second case study, we examine the problem of estimating characteristics of packet loss using probes on an end-to-end path through the lens of our calibration framework and environment. Packet loss is a consequence of statistically multiplexed, packet-switched networks and has been extensively studied over the years (*e.g.*, [67, 169,

218]). It continues to be a quantity of interest to estimate since it can have a significant impact on the performance of transport protocols and applications.

We focus on estimating two characteristics of *congestion episodes*: their frequency and their mean duration. Estimation of these characteristics was also a subject of the study of Zhang *et al.* [218]. While most active measurement methodologies have focused on estimation of packet loss rate, we discuss how end-to-end active measurement of this quantity is problematic. In our laboratory setup, we then show how the standard method for measurement of packet loss, which is based on a IETF standard Poisson-modulated probe process [47, 213], does not accurately estimate the frequency or mean duration of congestion episodes (or loss rate, for that matter). Based on these experiments, we designed a new active measurement methodology to more accurately estimate characteristics of congestion episodes. We implemented this new technique in a tool called BADABING.

We examined the accuracy capabilities of BADABING in our laboratory environment. Using a range of background traffic conditions, we find that the BADABING estimates of congestion episode frequency and mean duration are often within 25% of the true values, and many estimates are much closer. Estimates of these characteristics using Poisson-modulated probes are generally inaccurate: frequency estimates are often off by an order of magnitude, and duration estimates tend to be even worse. Thus, the estimates produced by BADABING represent a significant improvement in accuracy over the existing standard approach.

Finally, we developed a heuristic technique based on the BADABING probe process for estimating end-to-end loss rate. Our technique makes the assumption that the loss rate experienced by the BADABING probe stream is similar to what a TCP stream might experience. We compared the loss rate estimates produced by BADABING with the standard Poisson probe stream. Our results show that the BADABING estimates are significantly more accurate than the Poisson estimates. For example, in a scenario using realistic self-similar traffic, the BADABING estimate is 0.07% compared with a true loss rate is 0.08%, while the Poisson estimate is close to zero compared to a true loss rate of 0.17%.

Our final case study considers the question of how router buffers should be sized. A statistically multiplexed network requires routers to have buffers to queue packets in temporary situations in which there is more demand on ingress links than capacity on an egress link. The essential motivation for providing buffers is to improve performance since packets do not need to be discarded during transient periods of congestion. The question of how large router buffers should be has been growing in importance due to the cost of large memories, and the heat dissipation problems associated with them. While a simulation environment such as ns-2 [151] models a router as a relatively simple device with a single queue per egress link, real routers, as mentioned earlier, are quite complex. Thus, our laboratory calibration environment with commercial routers and the capabilities to obtain ground truth performance measures is an ideal testbed for examining this question.

The problem of how to size router buffers has a relatively long history in the network research community (*e.g.* see [53, 112, 120, 156, 164, 205]). It has recently received significant attention following the controversial work of Appenzeller *et al.* [51, 87, 88, 95, 113, 179, 207]. While a number of efforts in this area have focused on how buffer

size affects throughput, an Internet service provider (ISP) in today's marketplace typically offers quality of service guarantees in the form of service level agreements (SLAs) over metrics such as packet loss and delay. Thus, the challenge of sizing router buffers from an ISP's perspective is to decide how to size buffers in routers in the face of a range of expected traffic conditions and offered loads, while minimizing the risk of non-compliance with quality of service guarantees. It is this problem that our efforts are designed to address.

For our first set up experiments, we designed a series of tests that consider router architecture, router buffer size, and queuing policy (tail drop and Random Early Detection, or RED), as well as a variety of traffic mixes and offered loads. We then examine traffic performance through four key metrics: packet loss, delay, throughput, and goodput. We find that throughput is generally insensitive to differences in router architecture, buffer size, queuing policy, and traffic mix. This observation suggests that other metrics such as loss and delay, which tend to be much more sensitive, are more meaningful to consider. Our results demonstrate the risks of the sizing policy advocated by Appenzeller *et al.* [51] that result in small buffers, as well as the problems associated with other methodologies such as Dhamdhere *et al.* [88] that can require quite large buffers. Our results also suggest the potential benefits of an active queue management strategy like RED, that appears to provide improvement in both the aggregate and per-flow performance profiles for smaller buffer configurations.

Next, we consider these raw performance results in the context of service level agreements. We consider a set of simple SLAs to illustrate how these performance requirements affect and contribute to a more informed decision-making process for router buffer sizing. Our results suggest SLAs should play an important role in configuring router buffers, since networks are managed to deliver a level of service specified by these guarantees. Using SLAs as a frame of reference also leads to new problem areas such as measuring SLA (non-)compliance, engineering for robust SLA compliance, or quantifying the risk of SLA non-compliance. For example, we find that for the same SLA, the service provider's risk of SLA non-compliance is typically greater with fine-grained SLA compliance measurement/reporting than with coarse-grained ones. Moreover, we observe that this risk can be quantified in terms of the degree of sensitivity of SLA compliance to uncertainties in traffic mix and volume. At the same time, an analytic treatment of these problems poses formidable technical challenges and is left for future work.

1.3 Summary of Major Contributions

The primary contributions of this dissertation are as follows:

- We describe a new system for emulating characteristics of wide-area Internet links and paths, called NETPATH.
- We describe a methodology for generating representative Internet traffic at the IP flow level. We implemented this methodology in a tool called HARPOON and show that it can qualitatively reproduce the byte, packet, and flow volumes from a measured trace. We also show that it can reproduce the dynamic volume properties and the source and destination address frequencies from a measured trace.

- We articulate a framework for calibrating network measurement techniques and networked systems which relies on the transparency and realism afforded by laboratory environments.

- We describe a new visualization tool called SPLAT that is designed for analysis and exploration of large Internet data sets.

- We present a new active measurement methodology for estimation of end-to-end available bandwidth. We implement this methodology in a tool called YAZ and show that it is significantly more accurate than prior methods while introducing less measurement overhead.

- We introduce a new active measurement algorithm for estimating fundamental characteristics of Internet congestion, in particular, the frequency and mean duration of congestion episodes. We describe the implementation of this new algorithm in the tool BADABING and show that BADABING estimates the frequency and mean duration of congestion episodes with high accuracy.

- We describe a new methodology for estimating end-to-end packet loss rate that builds on our algorithm for estimating characteristics of Internet congestion episodes, and is significantly more accurate than prior methods for estimation of packet loss rate.

- We describe a novel large-scale study of the impact of router architecture, buffer size, and queuing policy (using drop-tail and RED) on a variety of traffic sources. By examining the transport-level performance characteristics packet loss, throughput, delay, and goodput, we demonstrate the risks of recently proposed buffer sizing methodologies such as [51, 88]. In addition, we illustrate the benefits and challenges posed by studying the problem in the context of performance guarantees made by Internet service providers to their customers.

1.4 Outline of this Dissertation

In Chapter 2 we discuss research related to this dissertation. We identify prior work related to incorporating aspects of realism into the laboratory environment. We also discuss work related to our focus on calibration and to specific aspects of our calibration techniques, including the use of the SPLAT visualization tool. We also describe other efforts in the areas of link and path emulation and traffic generation. Finally, we place the work of our three case studies in available bandwidth measurement, packet loss measurement, and router buffer sizing in the context of other research efforts.

Next, in Part I, we discuss aspects of our laboratory calibration approach. Chapter 3 describes a framework for laboratory-based calibration of network measurement methodologies and networked systems. We also discuss the SPLAT visualization tool and demonstrate its capabilities through a set of case studies. In Chapter 4 we describe the HARPOON traffic generation system. We detail HARPOON's generative model, and investigate its capabilities through

a series of laboratory tests. Chapter 5 follows with a description and evaluation of the NETPATH system for scalable emulation of network links and paths.

In Part II, we describe in detail three case studies in which we used the laboratory calibration environment and framework described in the first part of the dissertation.

Our first case study in Chapter 6 considers the problem of estimating end-to-end available bandwidth, or spare capacity, along a network path. We identify shortcomings in existing methodologies and describe a new measurement technique that is significantly more accurate than existing methods while imposing lower measurement overhead on the network.

In Chapter 7, we examine the challenge of estimating characteristics of packet loss along an end-to-end path. We introduce the novel BADABING measurement methodology and using our laboratory calibration environment show how it significantly improves measurement accuracy over the existing standard approach.

In Chapter 8, we describe the setup and results of a large-scale effort to examine the impact of router architecture, buffer size, and queuing policy on a range of traffic flows.

Chapter 9 summarizes the contributions described in this dissertation and charts a set of research areas for future investigation.

Chapter 2

Background and Related Work

> Imagine that a prototype voltmeter has been constructed. It consists of a needle which swings across a scale but as yet the scale is blank. To calibrate the instrument, known voltages are applied to the terminals and the corresponding positions at which the needle comes to rest are recorded. Thus, marks corresponding to known voltages can be inscribed on the scale. Henceforward the meter may be used to measure unknown voltages; the unknown voltage is applied to the terminals and the mark against which the needle comes to rest gives the answer.
> —H.M. Collins in [84].

In this chapter we set our contributions in the context of prior research. This chapter is organized to correspond to the two major parts of this dissertation. In the first section, we describe research related to our efforts in developing a framework and laboratory environment for calibration. In the second section, we discuss research related to each of the three calibration case studies described in the second part of this dissertation.

2.1 Calibration Framework and Environment

We begin by reviewing previous work in the general context of Internet measurement calibration, particularly as it relates to our proposed framework described in Chapter 3. A common need in data analysis and calibration studies is to visually investigate measured data for the purpose of exploratory analysis, and in order to detect anomalies, errors, and other deviations from expected behavior. We discuss our work in this area on a tool called SPLAT.

We next categorize and provide examples of testbeds used for Internet experiments. One of the key requirements of our calibration framework is to evaluate proposed measurement methodologies or networked systems in a *realistic*, *controlled*, and *transparent* environment. We define these terms and evaluate existing experimental environments and how these do or do not satisfy our requirements.

The class of testbed environments that we use in our calibration studies employs commodity routers, end hosts, and software (operating systems, protocol stacks, and applications). In this setting, there are two significant obstacles in the way of achieving a realistic environment: the need to create a range of traffic conditions that are representative

of the live Internet, and the need for realistic link and path conditions. In turn, we examine prior research in these areas.

2.1.1 Calibrating Internet Measurements

Although networking research has not seen systematic use of calibration techniques, they are standard in other scientific disciplines such as chemistry, biology and physics. As an example outside the networking context, the epigram at the beginning of this chapter illustrates the basic process of calibrating a simple measurement instrument such as a voltmeter. The lack of systematic calibration methods does not mean that calibration is an unknown concept in networking. In fact, many network measurement studies directly or indirectly discuss the issues of *accuracy* and *ground truth*; these notions are integral to calibration. (Note that in the next major section below in which we discuss prior work related to our case studies, we describe how previous studies have handled these issues.) Regardless of discipline, the basic goal is the same: the production of methods and tools that lead to consistent and accurate analyses.

Recently, the general activity of calibration in network measurement was highlighted by Paxson [172]. He identifies a series of pitfalls that can lead to inaccurate and/or invalid results, and suggests a list of calibration techniques to cope with these problems. A unifying theme of the suggested techniques is *comparison*; comparing the data or results of analysis with (i) a known ground truth measure, (ii) expected results based on a set of known inputs, or (iii) the output of an alternative measurement or analysis methodology. The goals in this case are a slight refinement of the general goals of calibration mentioned above, namely to detect inaccuracy of results due to (i) the analysis algorithm, (ii) inherent limitations in the data, (iii) data outliers or erroneous data, or (iv) because of limitations or bugs in the software that implements an aspect of the analysis.

Our goals in this work are the same, and our methods have conceptual similarities to Paxson's. We use known ground truth measures based on high-quality passive traces. These are used not only as a reliable basis for comparison, but also as a standard to meet when modifying an algorithm or its parameters in order to improve measurement accuracy. Another aspect of similarity between Paxson's work and ours is the support of visualization techniques to compare data sets and to examine differences between measured data and an expectation of what the data should look like. Below, we describe our efforts in this area on the SPLAT visualization tool.

Our work departs from Paxson's prior work in that we advocate a much broader framework for calibrating network measurement algorithms and tools and for measuring performance aspects of networked systems. The main focus of our proposed calibration framework is to evaluate measurement methodologies and networked systems in a realistic, transparent, and controlled environment, and to use a range of appropriate domain-specific tools for evaluating measurement results, including graphical techniques. We identify a set of issues that should be considered in the design of calibration tests in order to subject the system or algorithm under test to a range of conditions that might be experienced in a live setting, as well as extreme or anomalous conditions. Also, our framework demands ground

truth measures in order to evaluate the relevant characteristics of the experiment (*e.g.*, accuracy, the nature of errors, resiliency under exceptional situations).

There are many different statistical, graphical, or *ad hoc* methods that could be employed to aid in calibration and measurement data analysis. For example, basic first-order statistics are commonly used to compare the results of a proposed measurement methodology with ground truth results, such as comparing a measured average packet loss rate with the true average. In our work, we use a variety of standard techniques to evaluate measurements and compare with ground truth.

Though not designed explicitly for calibration, Floyd designed a set of scripts as a part of the ns-2 simulator in order to ensure the correct behavior of the TCP implementation(s) [151] and for regression testing. For this programmatic calibration technique, there are specific expectations for the output; any deviation implies that something invalid has been introduced into the simulator or implementation of a TCP variant. Clearly, this approach is one example of what Paxson's recent call is targeting: methods to compare and calibrate measurement results. While we were careful to ensure the correct functioning of laboratory equipment via scripts (programmed with expectations of what the output of a test should be), we did not take a fully programmatic approach, *e.g.*, such as is done in software regression tests. The reason is that in a more realistic laboratory setting (as opposed, *e.g.*, to a simulation setting), aspects of randomness can cause differences that, while valid, either cannot or are not easily formalized. A human, however, can quickly verify correct or incorrect functioning.

2.1.1.1 Visualization Tools for Calibration

Empirical measurements are the cornerstone of scientific discovery and evaluation of Internet structure and behavior. However, the quantity, diversity and complexity of Internet data greatly complicate the process of analysis. Many methods for assessing Internet data have been proposed and developed over the years, but most of them are quantitative in nature, suited for specific data types, and designed with a particular purpose in mind. There exist surprisingly few general-purpose tools for qualitative exploratory analysis of Internet data—an important precursor to hypothesis-driven discovery, analysis, and validation. On the other hand, standard plot-based visualization methods are common in Internet data analysis. These include plots to evaluate statistical properties of time series, scaling properties (*e.g.*, [41, 138]), and protocol behavior (*e.g.*, [29]). There are also several visualization tools which have been developed for Internet data analysis; see [3] for a partial listing. These can be classified as tools for connectivity/structural analysis (*e.g.*, [158]), network monitoring (*e.g.*, [11, 97]), and for elucidating the dynamics of particular networking protocols (*e.g.*, [16]).

Similar to regression and validation scripts, humans often conduct visual analysis due to complex or (as yet) unidentified phenomena under study. For example, scatter plots can be used to qualitatively assess the correlation between two variables, *e.g.*, between measured values of a phenomenon and expected values. Indeed, this use is one

intended purpose for the SPLAT visualization tool that we designed. In our case study examining available bandwidth measurement, we employ this visual comparison technique.

More generally, there are many standard textbook references on exploratory data analysis that highlight visual methods: one of the classics is J. Tukey's 1977 book, Exploratory Data Analysis [200]. Likewise, the literature on data visualization is vast, and like many other visualization tools, SPLAT builds upon an array of this prior work. Many of these tools are readily used in a calibration context for comparing measurements, or for verifying an expected behavior. We are not aware of any visualization tools that have the combination of capabilities provided by SPLAT.

While we present scatter and phase plots in SPLAT as a method to assist in calibration, they have been used for different purposes in prior Internet studies. Outside the networking area, phase plots or, more precisely, phase space portraits, are a well-known technique for exploring dynamical systems and analyzing chaotic data. Within the networking domain, the first use of phase plots that we are aware of was by Bolot in [67] who used them as a means for understanding the evolution of RTT delays by comparing measurements of consecutive probe packets. Similar techniques were used by Pásztor and Veitch to understand delay probes in [166]. Erramilli *et al.* [96] relied on phase plots of packet interarrival times to disentangle the effects of short- vs. long-range correlations on performance, and in [131], they are used to assess potential dependencies among successive packet sizes.

2.1.2 Testbeds for Internet Experiments

We advocate the use of laboratory-based experimental environments in our calibration framework since they provide an important set of capabilities not offered by a standard simulator such as ns-2 [151] or *in situ* environments such as PlanetLab [64]. These capabilities include repeatability and control, transparency, and the use of actual implementations of protocols and systems. In this section we first define more specifically the requirements for an environment for performing calibration experiments. We then present a taxonomy of existing testbeds and experimental environments. We discuss how these existing environments do or do not fit our requirements for calibration, including the various challenges faced by researchers performing tests in these settings.

As mentioned previously, there are three primary capabilities required for a calibration testbed, *control*, *transparency* and *realism*:

Control. Scientific inquiry requires the ability to reliably reproduce a particular phenomena. Repeatability, in turn, demands a certain level of control over the experimental environment. The concept of reproducibility is rather complicated. When attempting to reproduce an experiment, most often the goal is not to *precisely* replicate the exact conditions and results of a test, akin to a stereotypic "canned" experiment in a grade school science lab. Rather, the scientific power of an experiment is improved if these exact conditions are *not* met [84]. That is, the predictive strength of the hypothesis under test is enhanced through a broader set of initial conditions. What exactly constitutes a reproducible phenomena is, in the end, defined through consensus within different scientific

communities. We refer the reader to Collins [84] for a more comprehensive and philosophical discussion of the notion of scientific reproducibility.

For the present discussion, we loosely define reproducibility in terms of *quantitative and qualitative consistency* For example, suppose that a new active measurement tool is undergoing testing in a laboratory environment. If the results of running this tool are consistent both quantitatively and qualitatively over a range of network conditions (*e.g.*, topology, background traffic, link loads), they are considered reproducible.

Transparency. Our calibration framework requires the ability to obtain reliable *ground truth* measurements by which to evaluate the accuracy of a measurement methodology or the performance of a system. Transparency implies the opportunity to gain insight into the system under test, and to measure quantities of interest. Transparency and gaining insight into systems under test also suggests the ability to deconstruct or disassemble systems or software in order to understand how they work. This latter ability is not strictly necessary for transparency; one may continue to take a *black box* approach as long as one has the capability to precisely measure the inputs and outputs to the system under test. In our case studies described in Chapters 6–8 we take a black box measurement approach with respect to the commercial routers in our testbeds since we cannot measure internal subsystem behaviors of those systems. However, when appropriate we appeal to known or published internal features of routers or systems under test in order to better explain experimental results. This general approach has been termed a *gray-box* by Arpaci-Dusseau and Arpaci-Dusseau [52] since the device or network path under test (*i.e.*, one router or a series of routers) is not entirely opaque.

Realism. The conditions and setup of a calibration experiment must have a clear relationship to the live Internet: they must be realistic. This requirement is the most challenging one to satisfy for a number of reasons. Most notably, the Internet is huge, and constantly growing and changing in terms of its applications, physical topology and infrastructure, routes, and traffic. Moreover, characteristics that are prevalent in one part of the Internet, may not be in a different part. Thus, the research community has sought to discover *invariants*—properties that are persistent and ubiquitous across the Internet—as a way to understand the Internet's nature, and in particular for understanding which characteristics are most important for creating realistic conditions in simulation or laboratory settings [105, 107]. One example of an invariant is that traffic volumes typically follow a diurnal pattern, in which bit rates are higher during the day and are reduced at night. As we discuss below, we use current understanding of these properties in the design of our HARPOON system for realistic traffic generation.

There are three basic categories of Internet testbeds: simulators, laboratory settings, and *in situ* environments, which typically consist of a collection of hosts situated in the live Internet. Table 2.1 compares these three categories according to the strengths and the challenges associated with them in light of the three requirements identified above, and gives examples of each type.

2.1: A taxonomy of testbeds for Internet experiments.

Testbed	Simulation	Laboratory	*in situ*
Strengths	Control, Transparency	Control, Transparency, System-level Realism (*e.g.*, routers, hosts, protocol implementations)	Realism ("it is what it is")
Challenges	Realism (everything)	Realism (Paths and Links, Traffic)	Control, Transparency
Examples	GloMoSim [13], ns-2 [151], OPNET [183], SSFNet [27]	WAIL [33], DETER [8], Emulab [9], ModelNet [203]	AMP [21], NIMI [173], PingER [30, 149], PlanetLab [23], RON [50], SAMI [24], Surveyor [28, 129], VINI [65]

Simulation environments have been widely used for evaluating new or existing protocols (*e.g.*, [71, 116]), for validating active measurement methodologies (*e.g.*, [121, 141]), and for understanding aspects of Internet behavior (*e.g.*, [99, 104, 128, 215]). The main advantages of a simulation environment are the abilities to measure any desired quantity and to easily and exactly reproduce experiments. Simulators make heavy use of (pseudo-)random number generators, so for experimental replication a typical approach is to change the seed from experiment to experiment. Results are considered reproducible if they are consistent across a variety of seeds and network conditions.

The primary challenge in simulation is how to incorporate aspects of realism. Not only do the issues outlined above need to be considered (*i.e.*, realism of applications, topology, routes, and traffic), but realism of hardware and software implementations must also be taken into account. For example, the implementation of TCP in the ns-2 environment is an abstraction of a production-worthy implementation of TCP in a commodity operating system. While considerable care has been taken to ensure the correct behavior of ns-2's TCP implementations, the instantiations of TCP that exist in the live Internet exhibit a wide range of characteristics different from ns-2. Moreover, there is a rich ecosystem of end host and router operating systems and hardware in the Internet. In simulation settings, a host or router often has a very simple—indeed, simplistic—realization. Thus, results obtained from simulation experiments may be largely irrelevant when the protocol or system under development is deployed in the live Internet.

One approach to address this problem has been to create specialized emulation environments that use production open-source network protocol implementations. For example, Zhang [217] and Jansen and McGregor [125] have created simulation environments that directly used networking stack code from Linux and BSD variants. While these efforts certainly improve upon existing simulators, protocol stack implementations are just one piece of the overall system that also includes the operating system, hardware, and applications. Furthermore, many important systems are not open source, such as routers produced by the two dominant manufacturers, Cisco and Juniper. In the end, while different simulation environments have particular strengths and weaknesses associated with realism they are each inherently limited in the level of realism they can provide.

Deferring a discussion of the laboratory-based settings for now, the next type of testbed setting, *in situ* infrastructures, consists of managed host systems distributed across the Internet. These environments provide highly realistic network conditions since hosts are deployed in the Internet. As such, they enable implementations of network applications, protocols, and measurement methodologies to be evaluated over live end-to-end paths. These testbeds can also be useful for gauging and characterizing Internet structure and end-to-end performance from multiple vantage points. Some testbeds in the past have been developed for use by a single organization for a specific set of objectives, *e.g.*, Surveyor [28, 129], while others have been designed from the outset to serve a general purpose and be shared with a wider set of researchers *e.g.*, RON [50] and PlanetLab [64, 175]. There have been many measurement studies conducted on shared testbeds including RON and PlanetLab, *e.g.*, [56, 178, 197], and specialized systems have been developed for these testbeds to perform or assist with various types of measurements such as ScriptRoute [195]. While they are ideal for addressing questions on Internet structure and behavior, the inherent lack of ability to reproduce conditions across experiments limits their utility. Furthermore, the general inability to obtain ground truth measurements means that they are not appropriate environments for calibration experiments.

Returning now to the laboratory-based testbeds, these include environments such as WAIL, DETER, Emulab, and smaller private University testbeds [8, 31, 33, 136, 203, 209]. A canonical example is the Emulab testbed at the University of Utah which features a large set of commodity workstations that can be flexibly configured to conduct a wide variety of experiments [209]. These environments offer the capabilities of end-to-end node configuration, instrumentation, and repeatability. Environments like WAIL [33] also offer a set of commercial routers and switches, enhancing and broadening what is meant by "real."

Interestingly, there has been very little use of these environments in calibration of measurement tools—an application to which they are ideally suited. The main concern has been the question of how to create conditions representative of wide area networks, including traffic conditions and link/path conditions. This dissertation offers contributions to address these problems. We discuss prior work in these areas below.

Laboratory-based testbeds can be further subdivided into those that employ commercial routing and switching equipment, and those that *emulate* router functionality using commodity hosts. An example of the former is WAIL [33], and an example of the latter is the Emulab system [209]. For certain experiments in which the details of the routing and switching equipment is not critical, an emulation environment may be appropriate. However, for tests that evaluate behavior of detailed packet-level phenomena, *e.g.*, packet loss, delay, available bandwidth, and traffic performance measurements under different buffer sizes and queuing policies, environments using real routers are necessary. Commercial Internet routers are complex machines, *e.g.*, there can be several levels of buffering and queuing of packets and thus different causes of delay and loss [4, 15], and these details can have a substantial impact on certain types of network experiments.

It is also important to note that in testbeds such as Emulab, network topologies exist only in a virtual sense and are typically constructed using Ethernet VLANs [38]. With VLANs, multiple experiments may actually share the same

physical links, leading to possibly unreproducible results or outcomes that may be difficult to interpret. While WAIL can use this same methodology to construct topologies, the experiments described this dissertation do not use this functionality. Instead, we manually construct topologies that are physically isolated from other experiments, and use standard layer 3 routing methodologies (either simple static routes or intra-domain routing protocols such as OSPF).

A common problem in *in situ* environments and laboratory testbeds is to maintain an accurate and synchronized global view of time. This is a well-studied area in networking in which calibration has played a critical role, and it has quite a long history (see, *e.g.*, Mills [153]). A related problem that has also received a great deal of attention has been the question of how to apply accurate, synchronized timestamps to packets across the testbed [155, 166, 167, 171, 216]. Some efforts deal with the problem by using a global time reference such as GPS (*e.g.*, [61, 81, 166]), while others have taken an *ex post facto* approach to apply corrections to timestamps after packet traces have already been captured [155, 171]. In this dissertation, we extensively rely upon Endace DAG packet capture systems [10] which can provide synchronized timestamps with precision of at least 1 microsecond.

2.1.3 Traffic Generation

As mentioned previously, there are two main challenges related to creating realistic conditions in a laboratory testbed environment. We first discuss prior work in the area of generating realistic network traffic.

Creating repeatable, realistic network traffic in a laboratory or simulation environment is a difficult problem due to the diversity of traffic streams in the Internet and the complex dynamic characteristics of aggregate traffic. The emergence of new applications and continually increasing traffic volumes further compound the difficulty. Moreover, malicious traffic from worms, denial-of-service attacks, and botnets also pervades the Internet. Thus, for certain types of network experiments it is important to include these types of traffic in a comprehensive treatment of "realistic" traffic. In this dissertation we focus on benign forms of network traffic, and refer the reader to [18, 159, 193, 194] for work on malicious traffic generation.

There is a large literature on the general topics of Internet traffic characterization and modeling. Some of the most successful models of traffic focus on the correlation structure (self-similarity) that appears over large time scales in both local area [138] and wide area traffic [174]. Based on the model of Willinger *et al.* [210], Crovella and Bestavros presented empirical evidence that self-similarity in Web traffic arises from multiplexing ON/OFF sources with heavy-tailed file sizes transferred [86]. This observation forms the basis of our constructive model and tool (HARPOON) for benign traffic generation at the IP flow level.

Several flow-level network traffic models have been proposed including [58, 73, 110]. These models have been used to study fairness, response times, queue lengths and loss probabilities under different assumptions and using a variety of mathematical techniques. Our work differs from these in that we focus on building a flow-level model based on combining empirical distributions of characteristics that can be measured at a router in a live network. Our model

can be realized in a tool for generating representative packet traffic in a live (or emulated) network environment or for creating traffic in a simulation environment.

An approach similar to ours has been used in a number of application-specific workload generators. In [59], the authors describe the SURGE Web workload generator which was built by combining distributional properties of Web use. A similar approach was used in developing Web Polygraph [35] and GISMO [127] which are used to generate scalable, representative Web cache and streaming workloads respectively. HARPOON differs from these tools in that it is application independent, uses empirical distributions, generates both UDP and TCP traffic and includes both spatial and temporal characteristics. It should also be noted that there are several commercial products that target application-specific workload generation including [25].

A model more closely related to ours was introduced by Feldmann *et al.* in [99]. That model generalizes the SURGE model for Web traffic for the purpose of general background traffic generation in the ns-2 simulator. A similar model based on connection-rate superposition was developed by Cleveland *et al.* in [82]. Our model for HARPOON is more general in that it has no Web-specific parameters, includes the capability to transfer files via UDP and incorporates temporal and spatial request characteristics. Uhlig presents a flow level traffic model in [202] that is also similar to ours. That model, however, uses three parameters (compared with the eight that we use) and does not rely on an underlying transport protocol to transfer individual files. Our work is also distinct from each of these models in that we create and evaluate a tool that can be used for traffic generation on real systems and that can be automatically parameterized from flow records, thus requiring no additional modeling steps.

In contrast to workload generators based on models designed to synthesize distributional characteristics are tools that replay traffic based on raw packet traces. `tcpreplay` and `flowreplay` [201], for example, take a very low-level approach by attempting to generate packets (while optionally rewriting IP addresses) that mirror the specific timings recorded in a packet trace. Monkey-See Monkey-Do [77] takes a slightly higher-level approach by extracting certain features, such as estimated bottleneck bandwidth, delayed ACK behavior, and receiver window from raw packet traces in order to replay HTTP transactions in a test environment while emulating original client protocol and network conditions. Similarly, the `tmix` approach of Hernandez-Campos *et al.* [115, 208], reproduces rather low-level timing aspects of packet arrivals. Tmix uses 3-tuples ("A-B-T") consisting of the number of bytes sent in the forward direction on a path (A), the number of bytes sent in the reverse direction (B), and an amount of time to wait until the next A-B-T tuple. In addition to replaying these 3-tuples, other characteristics are estimated from a trace such as packet loss rate and round-trip delay, and are incorporated into the testbed. Similarly, `Swing` [206] extracts exchanges similar to the A-B-T tuples of `tmix`, but then organizes them in a similar way as HARPOON as higher-level sessions. `Swing` also requires that characteristics from the original trace such as loss rate, RTTs, network interface maximum transmission unit (MTU) sizes, bandwidths are extracted and recreated in the testbed.

The approach taken in HARPOON differs significantly from these traffic generators. First, HARPOON is designed to generate representative background traffic with realism, scalability, and flexibility as fundamental objectives. It is

difficult for tools that mimic network behavior at such a low level to scale to high-speed links. Second, HARPOON uses *source-level* traffic descriptions that do not make inherent assumptions about the transport layer, rather than *packet-level* descriptions based on low-level timings [107]. While it is true that the approach of HARPOON tends to ignore issues of packet arrivals over short (sub-RTT) time scales, this is intentional. Additional testbed parameters could have been required with a HARPOON setup (*e.g.*, a distribution of round-trip times, link capacities, MTUs) to match parameters derived from a live trace, but we currently do not specify such configurations. Creating the necessary volumes over longer time scales to produce the ubiquitous and invariant self-similarity and diurnal patterns in a way consistent with real application traffic is the intent and domain of HARPOON. We believe it is preferable to allow burstiness over short time scales to arise endogenously from TCP closed-loop control and network state (in the spirit of the simulation experiments described in [99, 128]), rather than to exogenously attempt to create such state (e.g., by introducing a random packet dropping element along a path to effect a certain loss rate). Generating packet traffic based on network-induced timings is problematic, especially with closed-loop protocols such as TCP, where transient, context-dependent network conditions can substantially shape individual packet timings.

2.1.4 Path Emulation

The second challenge related to creating a realistic laboratory testbed environment is to create link and path conditions such as propagation delays and packet losses that are representative of the live Internet. The basic task of path emulation can be abstracted to the ability to flexibly buffer packets transmitted between two nodes. Buffering enables delays to be assigned to individual packets. Also, while packets are buffered, they can be duplicated, reordered or dropped.

There are many possible approaches to both link and path emulation. One extreme method for implementing propagation delays is the use of large spools of cable to effect specific delays between systems. While this approach has obvious drawbacks, it provides accurate delays. Spools of cable combined with wavelength-division multiplexing hardware are actually used in CalTech's WAN-in-lab [32]. Another approach is to use specialized hardware developed for the purpose of delay emulation such as that offered by Spirent Communications [25]. Such systems provide precise link delays, probabilistic bit errors and probabilistic packet loss.

The most common approach to link and path emulation is to use general purpose computing hardware to run link and path emulation software. Over the years, many software-based systems have been developed for commodity hardware with the general objective of providing these functions [20, 44, 57, 68, 71, 98, 160, 181, 203]. Each of these systems differs from the others either in terms of their emulation objective, performance, or in how they are meant to be deployed for use in experimental environments.

One of the most widely used software-based network emulation tools is Dummynet [181]. This system can provide network emulation on end nodes that are also running other applications, or it can be run on a dedicated system interposed between other hosts. Dummynet is a kernel-based system built into the FreeBSD operating system that

operates on packets as they pass through the network protocol stack (the system described in [160] is similar). It offers the same basic set of network path effects as NETPATH. One of the biggest contrasts with NETPATH is that Dummynet also offers the ability to realize both bounded FIFO and weighted fair queues. Realizing specific behaviors such as priorities or dropping policies in queues was not a design objective of NETPATH although it would be a simple extension to our implementation[1].

Another popular network emulation system is NIST Net [20]. Like NETPATH and Dummynet, NIST Net can be used on end hosts as well as on dedicated interposition systems. Like other network emulators including [203], NIST Net was developed with the high level design objective of being a "network-in-a-box" [20]. We consider this to be an overly ambitious objective for many reasons (*e.g.,* see [107]). NIST Net offers same basic set of network path effects as NETPATH and its implementation offers several additional features such as derivative random drop (DRD) and explicit congestion notification (ECN) support. NIST Net also offers a graphical user interface for configuration and monitoring.

Path emulation differs from *network emulation* in the ModelNet system which attempts to "subject each packet to a delay, bandwidth, and loss characteristics according to a target topology" [203]. Note that the network emulation approach is integral to the `Swing` traffic generator described above. Path emulation has no notion of a core network topology and is therefore less ambitious in its objectives. Path emulation also does not attempt to emulate router characteristics such as statistical multiplexing, queuing and switching capabilities. Such characteristics vary widely in the Internet depending on specific hardware implementations. Path emulation targets the basic causes of packet variability on multiple-hop, wide-area connections between hosts. Finally, while mimicking link capacity is often listed as a distinct feature in network emulators, we do not consider it a distinct component of path emulation. The essence of capacity limitation is to add latency to each bit equal to the inverse of a specified rate. Thus, experiments requiring links of certain capacities can be accommodated through the propagation delay mechanism.

2.2 Prior Work Related to Calibration Case Studies

The second part of this dissertation presents three case studies that leverage the transparency, control, and realism afforded by our laboratory calibration environment: (i) measurement of available bandwidth, (ii) measurement of packet loss characteristics, and (iii) investigation of the impact of buffer size on transport-level performance. While each of these areas has been well-studied over the years, we find that there are substantial improvements possible in both measurement accuracy and insights gained by employing our laboratory calibration approach.

In this section we discuss work related to each of our case studies in turn. We specifically note how other researchers have handled issues related to calibration, *e.g.*, the problem of evaluating the accuracy of a proposed measurement methodology, and the issues of transparency, control, and realism in experimental evaluation.

[1] In fact, the Click distribution offers several specific queue elements.

2.2.1 Available Bandwidth Measurement

In our first case study we consider the problem of measuring end-to-end available bandwidth. Algorithms for inferring end-to-end available bandwidth have received considerable attention in recent years [43, 75, 117, 121, 135, 152, 169, 180, 187, 188, 197]. End-to-end available bandwidth (AB) can be loosely defined as the minimum spare capacity across each network link between a sender and receiver (more precise definitions are given below). Potential applications of dynamic available bandwidth estimation (ABE) include network overlay maintenance and optimization, network provisioning and capacity planning.

Tools that measure AB are designed to send precisely crafted packet pairs or packet trains and make inferences based on changes to packet spacings measured at a receiver. They are typically designed to measure the rate of cross traffic on the tight link (link with the least AB), by injecting streams of various rates or pairs of packets and applying some function to the input and output streams to determine whether a given stream rate is above or below the AB. The interaction of measurement probe packets with the cross traffic occurs on time scales related to the spacing of packet pairs or packets within a stream, and forms the basis of inference. Such time scales are typically in the range of tens to hundreds of microseconds.

Using pairs or streams of closely-spaced packets to estimate network quantities also has a long and rich history [124]. Jacobson's work on TCP congestion control was an early exposition of the possibility of how a pair of closely-spaced packets can reveal bottleneck link capacity [118]. Other researchers have explored specific algorithms using this technique to perform capacity estimation, *e.g.*, [130, 134, 139], and for estimating, as we consider here, end-to-end available bandwidth. A related study, but with a focus on evaluating available bandwidth estimation tools (ABETs) in the context of high-speed links is found in [188]. Additional contrasts between the work of [188] and our case study is that our focus is on calibration (and thus the *mechanisms* that lead to performance differences between tools), not simply a head-to-head comparison. Furthermore, we use a much more varied range of cross traffic, and our experimental environment is openly available. Lastly, we used significantly more precise measurements than router SNMP counters, and we created a new highly accurate and low-impact ABET as a result of our work.

In Table 2.2, we summarize how different experimental studies have handled the problem of comparing the output of a proposed ABET with known measures of available bandwidth. In general, previous studies have either examined a proposed algorithm in a simulation setting or *in situ* without ground truth. A few authors specifically noted the problem of evaluating the accuracy of their measurement methodology in the live Internet. In addition, while a number of studies have used controlled testbeds, the background traffic used was often simplistic; only Akella *et al.* used closed-loop web-like traffic [43]. Also, only Carter and Crovella note how they obtained ground truth measures in their testbed (using a packet trace) [75]; other authors using testbeds do not explicitly note how these measures were obtained. Finally, the networking equipment used in the testbeds was generally either a simple Ethernet switch, or a software-based router running in a commodity operating system.

2.2: A summary of how proposed methods for estimating available bandwidth have been evaluated. References are listed as tool name, authors, and citation.

Reference	Evaluation Approach
cprobe, Carter and Crovella, 1996 [75]	The authors evaluated cprobe using a combination of a small Ethernet testbed and paths of the Internet. The testbed was a single-hop topology, and synthetic load was generated using long-lived TCP flows (FTP traffic). Ground truth was obtained using a packet trace tool. The authors noted that "beyond the local network validation becomes very difficult."
Paxson, 1997 [169]	Paxson used wide-area measurements using his Network Probe Daemon testbed. There was no capability to obtain ground truth measurements or control network conditions since the network paths were over the live Internet.
TOPP, Melander *et al.*, 2000 [152]	The authors used a combination of ns-2 simulations and simple LAN experiments, with most evaluation performed using simulation. The LAN testbed experiments used a single-hop 10 Mb/s Ethernet topology with UDP cross traffic. Cross traffic packets were sent with uniformly distributed random inter-packet spacings in such a way as to generate certain traffic rates. No details are given on the network setup or how measurements were taken to obtain ground truth.
Pathload, Jain and Dovrolis, 2002 [121]	The authors used a combination of ns-2 simulations and wide-area experiments to evaluate Pathload. ns-2 background traffic was generated using UDP-like packets sent with Pareto-distributed inter-packet spacings. For wide-area experiments, coarse-grained SNMP data in the form of Multi-Router Traffic Grapher (MRTG) [34] plots was used. This data was only available on 5 minute intervals. (Note that the authors did not have the underlying data used in the graphs—they estimated the true available bandwidth by reading the plots.) The authors noted that "even though this verification methodology is not too accurate, it was the only way in which we could evaluate pathload in real and wide-area Internet paths."
BFind, Akella *et al.*, 2003 [43]	The authors tested BFind in ns-2 simulations as well as in the live Internet using PlanetLab hosts and other hosts. For the live Internet tests, the estimates of BFind were compared against the output of Pathload. In simulation, the authors used a combination of web-like and constant bit rate UDP sources for background traffic.
IGI/PTR, Hu and Steenkiste, 2003 [117]	The authors used ns-2 simulations, wide-area Internet experiments, and experiments using a simple testbed consisting of FreeBSD routers. In the testbed, background traffic consisted of `Iperf` long-lived TCP flows. In the wide-area experiments, the authors used a single TCP flow to estimate the actual available bandwidth.
pathChirp, Ribeiro *et al.*, 2003 [180]	The authors used a combination of simulation and Internet experiments. In the simulation experiments background cross traffic was UDP packets sent with Poisson-modulated inter-packet spacings. For the Internet experiments, they used one path from the Stanford Linear Accelerator Center (SLAC) to Rice University. They introduced some additional UDP traffic similar to the simulation background traffic in order to test the effect on pathChirp, but acknowledged that they could not evaluate its accuracy.
Spruce, Strauss *et al.*, 2003 [197]	The authors used wide-area Internet experiments, using PlanetLab and RON testbeds. For one path, they used available SNMP data aggregated at 5 minute intervals in the form of MRTG plots (similar to the Pathload study). For other paths, they designed a "D-test" for evaluating the Spruce's accuracy: once an initial estimate A was obtained, they introduced UDP constant bit rate traffic at $\frac{1}{2}A$ and reran Spruce with the expectation that the new estimate would be half the original.
probegap, Lakshminarayanan *et al.*, 2004 [135]	They used an experimental network (not a private testbed over which they had full control) consisting of commercial Cisco cable head-end system. They relied upon the assumption that the network was lightly loaded, and obtained usage statistics from a network operator. They created a simple tool `udpload` to introduce UDP traffic sent with Poisson inter-packet spacings at different loads for background traffic.
Shriram *et al.*, 2005 [188]	These authors evaluated a number of publicly available tools such as Spruce, Pathload, IGI, and pathChirp in a simple laboratory testbed consisting of commodity routers and switches. Their goal was to examine how well these tools perform head-to-head in a high-speed (Gigabit) environment, *e.g.*, to measure the effect of using jumbo (9000 byte) Gigabit Ethernet MTUs. They used open-loop background traffic produced by tcpreplay and a Spirent SmartBits system, and no mention is made of any propagation delay emulation in their testbed. Furthermore, the measurements by which they compared the ABETs were derived from SNMP counters compared with the more precise measurements we used.
Shriram *et al.*, 2007 [187]	The authors evaluated several ABETs in an ns-2 simulation environment. Their goal was to "eliminate implementation biases" and examine algorithmic performance by focussing entirely on a simulation setting. Within this setting they used dumbbell-like topologies with one or more tight links along a path, and used a range of background traffic conditions. The premise behind their work is that effects caused by real hosts and operating systems are essentially nuisances, masking the true underlying performance of an ABE algorithm. Thus, in their view, simulation is an ideal evaluation environment. While we agree that the use of simulation can yield some insights into measurement methodology behavior, a simulator introduces its own biases and problematic effects (*e.g.*, the simplistic implementation of router queues). Furthermore, we believe that for an active measurement methodology to be useful in a live Internet setting, it must be designed and evaluated in a setting with not only control and transparency, but also in which realistic effects from the network and from host operating systems and hardware are broadly considered.

While the development of fast and accurate ABETs is an active area of research, two recent tools, Pathload [121] and Spruce [197], represent the two most common strategies for probing and two appealing methods for AB inference. Thus, these tools are the focus of our ABET calibration study. We provide an overview of their design below.

2.2.1.1 Definitions and Overview of ABE Techniques

The *available bandwidth*, A, of a single link is defined as the amount of spare capacity over a given time interval, τ. If C is the link capacity and $U(x)$ gives the instantaneous link utilization (0 or 1) at time x, then the available bandwidth over the time interval $[t, t+\tau]$ is

$$A = C\left(1 - \frac{1}{\tau}\int_{t}^{t+\tau} U(x)dx\right). \quad [2.1]$$

The end-to-end available bandwidth is then defined as the minimum AB over all hops in the path:

$$A \equiv \min_{i} A_i, i \in 1..H. \quad [2.2]$$

The link with smallest AB is referred to as the *tight link*, while the link with smallest capacity is called the *narrow link*. These definitions avoid the ambiguous term *bottleneck link* [121]. They also help to avoid any implicit assumption that the tight link is necessarily the narrow link. We use the term capacity to refer to the maximum transmission rate at layer 3, assuming a maximum frame size of 1500 bytes. For example, with standard Ethernet there is a nominal (physical layer) transmission rate of 10 Mb/s. However, for each packet delivered from layer 3 for transmission, there are an additional 18 bytes prepended for the Ethernet header and 8 bytes for the frame preamble. Frames may not be transmitted closer than (in effect) 12 bytes, resulting in an overhead of 38 bytes per layer 3 packet. Thus, the transmission rate available to layer 3 is reduced to approximately 9.75 Mb/s. Note that this definition gives us good reason to probe with maximum possible layer 3 packet size to minimize effects of layer 2 overhead.

Existing tools for measuring AB are generally focused on either estimating the amount of cross-traffic on the tight link, or on direct measurement of AB by sending probe streams at various rates. Both methods are accomplished by specifying an initial set of parameters, sending a series of probes and measuring responses, and finally inferring an estimate of AB from the measurements. All methods for measuring available bandwidth make certain simplifying assumptions. First, they assume first-in first-out queuing at routers. Second, they assume that cross traffic is fluid (cross traffic packets are infinitely small). Finally, cross traffic intensity is assumed to be stationary over the measurement period.

2.2.1.2 Spruce

Spruce estimates AB by sending packet pairs spaced back-to-back according to the capacity of the tight link[2]. Assuming fluid cross traffic, the amount by which the packet pairs are expanded by the tight link is proportional to

[2]With Spruce, the tight link and narrow link are assumed to be the same. Strauss *et al.* claim that the estimates may still be meaningful even when this condition is not satisfied.

the volume of cross traffic. If g_{in} is the spacing of back-to-back probe packets on the tight link and g_{out} the spacing measured at the receiver, the AB is calculated as:

$$A = C\left(1 - \frac{g_{out} - g_{in}}{g_{in}}\right). \quad [2.3]$$

This formulation assumes that the tight-link capacity (C) is known *a priori*. Clearly, negative AB estimates are possible, although Spruce reports zero AB when an estimate is negative. Spruce sends, by default, 100 packet pairs at Poisson-modulated intervals, and reports the average A over those samples. Through the course of a measurement, is unknown whether the estimate has yet converged to the true AB (assuming some stationary average exists).

Spruce Calibration Issues. Liu *et al.* [140] use a single-hop setting to analyze Spruce-like techniques and the bias introduced by the fluid assumption under a range of cross traffic conditions. They claim that bias is minimized when the input gap g_{in} is set to less than or equal to the back-to-back spacing on the tight link. However, a more complex topology with cross traffic on the non-tight links may still introduce bias.

There are practical limitations when considering high-bandwidth links. For the experiments discussed in the following section, the fastest narrow link we consider is an OC-3 (155 Mb/s nominal transmission rate), which requires Spruce to send its packet pairs spaced by approximately 80 microseconds. While we will show that modestly provisioned hosts in our testbed can accommodate this requirement, emitting packet pairs to measure available bandwidth even on OC-12 links (622 Mb/s nominal), where packets must be sent with spacing of 20 microseconds, borders on being infeasible with commodity workstations.

2.2.1.3 Pathload

Pathload attempts to create short-lived congestion conditions in order to directly measure AB. It detects congestion through trends in one-way probe packet delays. Specifically, an increasing one-way delay (OWD) trend is assumed to be indicative of queuing and that the probe stream rate is greater than the end-to-end AB. Through iterative adaptation of its probe stream rate, Pathload converges on an available bandwidth range, the center of which is the average spare capacity over the measurement period.

A simple way to describe the relationship between one-way delay trends and available bandwidth may be made in terms of probe stream rate. If L is the probe packet size (in bytes) and g_{in} the spacing between consecutive packets in a stream, the probe rate (input rate) produced by Pathload is $r_{in} = \frac{L*8}{g_{in}}$. An increasing one-way delay trend is equivalent to saying that there is an increasing inter-packet spacing trend, and an average increase in spacings causes the overall probe rate measured at the receiver (r_{out}) to be less than that introduced at the sender (r_{in}). Such a decrease is evidence that the end-to-end AB is less than the probe stream rate. This relationship can be expressed as follows:

$$\frac{r_{in}}{r_{out}} = \begin{cases} \leq 1 & r_{in} \leq A \\ > 1 & r_{in} > A \end{cases} \quad [2.4]$$

Pathload takes N measurements with probe streams of length K packets, iteratively adapting its send rate to determine whether or not there is an OWD trend. These N streams are referred to as a fleet. Each stream within a fleet is separated by an amount of time designed to allow the path to quiesce. By default, N is set to 12 and K to 100. After each of the N streams, Pathload uses two separate metrics to estimate trends in OWD: the pair-wise comparison test (PCT) and the pair-wise difference test (PDT). The *PCT* and *PDT* metrics operate on blocks of $\Gamma = \sqrt{K}$ packets from each stream. For each block, the median value, $\hat{D}$ is chosen as a robust estimator of OWD. These metrics are defined as follows:

$$A_{PCT} = \frac{\sum_{k=2}^{\Gamma} I(\hat{D}^k > \hat{D}^{k-1})}{\Gamma - 1}, \quad [2.5]$$

$$A_{PDT} = \frac{\hat{D}^{\Gamma} - \hat{D}^1}{\sum_{k=2}^{\Gamma} |\hat{D}^k - \hat{D}^{k-1}|}, \quad [2.6]$$

where $I(x)$ is a function returning 1 if the condition x is true, and 0 otherwise.

PCT returns a value between 0 and 1, where 0 indicates that the stream has not experienced an increasing OWD trend, and 1 indicates that, for $k \in [2..\Gamma], \hat{D}^k > \hat{D}^{k-1}$; that is, there is a consistent increasing trend in OWD. *PDT* detects whether there is a net increasing trend, considering only the first and last OWD samples. It returns a value between -1 and 1, where -1 indicates a strongly decreasing OWD trend (*i.e.*, the probe rate was measured to be much higher at the receiver than was sent), and 1 indicates a strongly increasing OWD trend. For each metric, a threshold value is chosen, above which the OWD trend is said to be increasing. These thresholds are 0.55 for *PCT* and 0.4 for *PDT*. The two metrics must agree (either increasing or non-increasing OWD) on a certain fraction of the streams in a fleet for an overall judgment to hold (by default, this threshold is 0.6). Otherwise, the result is inconclusive and additional fleets are required. The specific algorithm for how the input stream rate is modulated based on the outcome of the *PCT* and *PDT* metrics is described in [121] and in the source code to Pathload.

Pathload Calibration Issues. The *PCT* and *PDT* metrics, along with the specific thresholds, assume congestion only takes the form of *increasing* one-way delays, *i.e.*, expansion in intra-stream packet spacings. In [169], Paxson notes that while expansion is the predominant result of queuing, compression events commonly occur. The *PCT* metric does not consider compression, and the default threshold used for the *PDT* metric eliminates compression as an indication of congestion.

Like Spruce, ABETs that use self-loading techniques may not be able to produce streams on commodity systems that are sufficient for detecting AB at OC-12 link speeds and above. Another potential problem is that congestion-inducing measurement tools may cause significant and/or persistent packet loss during use. To minimize impact, packet streams should be short and should be spaced far enough apart for cross traffic sources to recover from any losses. However, shorter streams result in higher measurement variance, as noted in [121, 122]. An additional problem with longer streams is that there is an increased possibility for operating system context switches, causing a disruption in intended packet spacings and possibly invalidating use of that stream.

Pathload requires threshold parameter selection in its OWD trend detection algorithm. To date, there has been no study of sensitivity of these metrics to different cross traffic, and default thresholds were selected based on experience with network simulations [121].

2.2.2 Measurement of Packet Loss

In our second case study we examine the problem of measuring characteristics of packet loss using active probes. It is well known that packet loss can have a substantial impact on the performance of a wide range of Internet protocols and applications. Understanding the characteristics and impact of packet loss led, for example, to the development of the NewReno [116] and SACK [147] versions of TCP. Loss characteristics are also a fundamental component of TCP throughput modeling [74, 148, 162].

There have been many studies of packet loss behavior in the Internet. Bolot [67] and Paxson [169] evaluated end-to-end probe measurements and reported characteristics of packet loss over a selection of paths in the wide area. Yajnik *et al.* evaluated packet loss correlations on longer time scales and developed Markov models for temporal dependence structures [214]. Zhang *et al.* characterized several aspects of packet loss behavior [218]. In particular, that work reported measures of *constancy* of loss episode rate, loss episode duration, loss free period duration and overall loss rates. Papagiannaki *et al.* [163] used a sophisticated passive monitoring infrastructure inside Sprint's IP backbone to gather packet traces and analyze characteristics of delay and congestion. Finally, Barford and Sommers pointed out some of the limitations in standard end-to-end Poisson probing tools by comparing the loss rates measured by such tools to loss rates measured by passive means in a fully instrumented wide area infrastructure [63].

The foundation for the notion that Poisson Arrivals See Time Averages (PASTA) was developed by Brumelle [72], and later formalized by Wolff [213]. Adaptation of those queuing theory ideas into a network probe context to measure loss and delay characteristic began with Bolot's study [67] and was extended by Paxson [169]. In recent work, Baccelli *et al.* analyze the usefulness of PASTA in the networking context [55]. Of particular relevance to our work is Paxson's recommendation and use of Poisson-modulated active probe streams to reduce bias in delay and loss measurements. Several studies include the use of loss measurements to estimate network properties such as bottleneck buffer size and cross traffic intensity [48, 184] . The Internet Performance Measurement and Analysis efforts [37] resulted in a series of RFCs that specify how packet loss measurements should be conducted. However, those RFCs are devoid of details on how to tune probe processes and how to interpret the resulting measurements. We are also guided by Paxson's recent work [172] in which he advocates rigorous calibration of network measurement tools.

Zing is a tool for measuring end-to-end packet loss in one direction between two participating end hosts [42, 143]. zing sends UDP packets at Poisson-modulated intervals with fixed mean rate. Savage developed the Sting [185] tool to measure loss rates in both forward and reverse directions from a single host. Sting uses a clever scheme for manipulating a TCP stream to measure loss. Allman *et al.* demonstrated how to estimate TCP loss rates from passive packet traces of TCP transfers taken close to the sender [45]. A related study examined passive packet traces taken in

the middle of the network [66]. Network tomography based on using both multicast and unicast probes has also been demonstrated to be effective for inferring loss rates on internal links on end-to-end paths [83, 92].

Table 2.3 summarizes how different researchers have dealt with calibration issues and evaluating the accuracy of their loss measurements. Somewhat surprisingly, only one study refers to experiments in a controlled setting, but does not reveal any details of the setup [185]. Also interesting is the fact that simulation does not play any significant role in evaluating methodologies for measuring packet loss. The operating assumption appears to be that measuring loss is a straightforward, simple process. All studies resort to using uncontrolled wide-area Internet paths. As a result, each either makes the (often tacit) assumption that the measurements are accurate, or the results are compared with an alternative method for measuring packet loss which has also not been evaluated for accuracy.

2.2.3 Router Buffer Performance Measurements

The third and final case study we examine is in the area of sizing buffers in routers. This case study differs from the first two in that we do not evaluate the accuracy of a measurement methodology *per se*. Rather, our goals are (i) to empirically investigate the impact of router architecture, buffer size, and queuing policy on transport performance in a realistic, controllable, and transparent laboratory setting, and (ii) to relate observed transport level performance to a network service provider's need to support competitive service-level agreements (SLAs). In this section we first discuss prior work on the problem of router buffer sizing. Similar our two previous case studies, we discuss how others have evaluated proposed buffer sizing methodologies. Finally, we provide background information on the context and content of service-level agreements.

2.2.3.1 Router buffer sizing

Villamizar and Song are commonly credited for establishing the popular bandwidth-delay product (BDP) formula for router buffer sizing [205]. (Earlier references suggesting the use of BDP for sizing Internet router buffers are by Jacobson [120].) Their work was based on measurement of an RS/6000-based implementation of a router, up to 8 simultaneous long lived TCP flows, varying buffer sizes, and both drop tail and RED queues. These authors found that a buffer size of $Capacity \times RoundTripTime$ or CT guarantees full utilization of the outgoing link. Morris expanded this work in a simulation study by examining the behavior of a large number of long lived TCP flows competing for a buffer sized at CT [156, 157]. He reported performance degradation for the end host, and in terms of fairness and utilization. Several additional studies have considered the effects of router buffer size through queuing analysis and in simulation (see, *e.g.*, [53, 54, 112, 113, 164]). The consideration of Web-like traffic workloads instead of only long-lived TCP flows has been of particular relevance in many of the more recent simulation-based studies.

The recent Stanford study by Appenzeller *et al.* [51] has generated renewed interest in the problem of sizing router buffers. The potential implications of their $B = CT/\sqrt{N}$ result, where N is the number of active long-lived flows, have motivated other researchers to scrutinize the work in [51]. In particular, Dhamdhere and Dovrolis [87, 88] highlight the

2.3: A summary of how the accuracy of active packet loss measurement has been evaluated by different researchers. Note that some studies propose new measurement methodologies, while others use existing approaches.

Reference	Evaluation Approach
Mills, 1983 [154]	One of the first reported evaluations of delay and loss using active probe-based measurements in the Internet. While the focus is on delay, loss measurements are also discussed since the focus is on improving TCP performance. Experiments used `ping` (referred to as the Packet InterNet Groper) ICMP echo packets. Network conditions were not controlled, and the author does not note whether in-network measures are obtained to check the PING measurements.
Bolot, 1993 [67]	Probably the most well-known early study of delay and loss in the Internet. Probes were sent at periodic intervals, similar to `ping`, using different rates. Only wide-area Internet paths were used. The tacit assumption is made that the measurements are accurate.
Paxson, 1997 [169]	Part of Paxson's comprehensive study of end-to-end packet performance in the Internet using his NPD infrastructure. While much care was taken to handle measurement anomalies and errors, there was no ground truth available by which to evaluate the accuracy of the measurements.
Yajnik *et al.*, 1999 [214]	The authors measured loss using probes sent periodically using a range of rates in the wide-area Internet. The implicit assumption is made that the measurements are accurate.
Savage, 2000 [185]	Savage developed the Sting tool to estimate loss rates in both the forward and reverse directions of a network path. Although he notes that Sting was evaluated in a testbed using FreeBSD and the Dummynet link emulator, no details are provided as to the setup or background traffic used. Sting was also evaluated using wide-area Internet paths. Its results were compared with loss measurements obtained using `ping`.
Zhang *et al.*, 2001 [218]	The authors examined loss rates, the duration of loss episodes, and the duration of loss-free episodes along a set of wide-area Internet paths using UDP probes sent according to a Poisson process (consistent with RFC 2680 [47]). The unstated assumption is that the measurements are accurate; no verification of accuracy was possible because of the use of live Internet paths.
Mahajan *et al.*, 2003 [142]	These authors developed the Tulip tool to measure, among other things, loss rates along a path in both the forward and reverse directions, and on a hop-by-hop basis, similar in some respects to Sting. They evaluated Tulip in multiple ways. First, they examined wide-area Internet paths on which they controlled the end-points but did not have control of the full path. At the remote endpoint, they manipulated the stream in order to verify the correct Tulip behavior. Second, they compared the results of Tulip with Sting on live Internet paths. Third, they compared end-to-end results with aggregated hop-by-hop results. They used live Internet paths (often employing PlanetLab hosts) and did not have access to ground-truth measurements.
Pucha *et al.*, 2006 [178]	These authors used Tulip to measure loss characteristics between hosts in the PlanetLab infrastructure. While it was tacitly assumed that the loss measurements were accurate, a comment was made that the results of Tulip were consistent with results from BADABING [190]. It is not clear what is meant by this remark, since BADABING does not report loss rate which was the characteristic under study. As with other studies, no ground truth measurements were available.

fact that the traffic model and performance objective used in making buffer sizing decisions are critical. Of the two, the former is perhaps the most difficult to address due to the well known variability and complexity of packet traffic in the Internet [138]. Determining the value N specified as the number of long-lived TCP flows (*i.e.,* flows that exit slow start) for a given link is challenging (*e.g.*, see [207], and so is estimating the value T for any non-trivial topology. Dhamdhere and Dovrolis also demonstrate effectively the impact that open-loop versus closed-loop TCP transfers for a given buffer size can have on performance. For related earlier work that demonstrated the importance of considering workload models that account for both the feedback regulation inherent in TCP and the heterogeneity of actual traffic see Joo *et al.* [128]. A series of recent papers have been concerned with additional details of sizing buffers, controlling synchronization in routers with small buffers, and fairness [95, 179, 211, 212]. The primary focus of these papers is on core routers with a high degree of statistical multiplexing, and arguments are made for the feasibility of tiny buffers (*i.e.*, tens of packets) under certain conditions. The problems of understanding the behavior of different traffic mixes and active queue management (AQM) schemes are identified as open issues.

The question of the "right" performance metric for buffer sizing decisions has also attracted renewed attention. Appenzeller *et al.* focus in [51] on *link utilization*, which has been shown in [88, 207] to be oblivious to significant loss rates that would likely be unacceptable to network operators. Dukkipati and McKeown [94] argue that *flow completion time* (FCT) is the most important metric since it best reflects what users care about most. FCT is problematic, however, because as an end-to-end performance metric it is beyond the purview of any single network within which specific buffer sizing decisions are made. Gorinsky *et al.* suggest a formulation of the buffer sizing problem that considers end-to-end goodput at the right performance target (directly related to FCT), and show through a series of simulations using different versions of TCP that small buffers can still result in high goodput [113]. The related problem of understanding performance in the context of AQM, specifically Random Early Detection (RED) [104], has been widely examined [101]. Many of these studies have focused on RED tuning and performance evaluation, *e.g.,* [78, 136] which consider the context of Web performance in particular. The RED study by May *et al.* bears some similarity to ours in its use of a commercial router and different traffic mixes [150]. It considers performance with four different buffer sizes and shows that throughput is relatively insensitive to the choice of RED parameters. Interestingly, the results also show that throughput is relatively insensitive to buffer size, but that observation is not highlighted.

A distinguishing feature of our case study is that we advocate a broad experimental approach. We argue that a "one size fits all" model for traffic or performance metric is unlikely to expose the impact of realistic traffic behavior or relevant performance measures on buffer sizing decisions. Therefore, the issues of traffic models and performance metrics must be broadly considered, and the empirical study described in this dissertation addresses them by taking a comprehensive and flexible experimental approach.

Table 2.4 summarizes how different researchers have dealt with evaluating proposed or existing methodologies for sizing router buffers. Nearly all these prior studies were performed in simulation, often with rather simplistic background traffic models. The study of Appenzeller *et al.* [51] is one of the few that used a controlled laboratory

testbed (in fact, we conducted those experiments). It is clear from these prior studies that there exists little empirical data that has been collected in a realistic and controlled environment.

2.2.3.2 Service Level Agreements

SLAs reflect the immense importance of IP networking to today's business enterprises and governmental institutions. A ten minute disruption of network service can cause millions of dollars of loss, or can jeopardize the functionality of essential infrastructure. SLAs spell out the technical and business relationship between network providers and customers, with positive financial consequences if the SLA is met (fees collected for services delivered) and negative ones if the SLA is not met (*e.g.,* penalties and damage to the business relationship). SLAs for IP networks today may span the globe and multiple autonomous systems, under the control of a single network provider, or a set of cooperating providers.

At a technical level, SLAs provide assurances on a plethora of conditions regarding connectivity, time to handle outages or close trouble tickets, and increasingly on network performance, *e.g.,* packet delay, loss and jitter. In our case study of Chapter 8, we concentrate on TCP-based applications where loss and delay play a major role, but jitter is typically of lesser concern.

SLAs are of particular importance for Virtual Private Networks (VPNs). VPNs can be implemented using a variety of networking technologies, but all essentially provide a clear separation of the Provider Edge (PE) and Customer Edge (CE) routers or interfaces. The customer's CEs attach to the provider's PEs, and the provider's core routers offer transport between PEs. VPN services (which may be point to point or any to any among the customers CEs) thus allow customers to out-source their private network to a shared provider infrastructure. The provider manages the PE routers, core routers connecting the PEs, and (depending on the business relationship) the CE routers. The provider can support SLAs within the perimeter it controls, *e.g.*, from CE to CE. SLAs covering performance (*e.g.,* loss and delay) in these networks are of increasing importance to customers who want assurances of little variation from agreed-upon performance targets. For example, large enterprise networks often have a hub and spoke topology (with a small number of hubs and a large number of spokes), where SLAs assure good performance from hubs to associated spokes, as well as between the hubs. However, performance cannot be assured during intervals when resources are oversubscribed. Thus, SLAs may allow for discarding all measurements collected when utilization exceeds a given threshold. The question is, what role does buffer sizing play in these environments?

Performance across today's large IP network cores is largely determined by two factors: (i) transmission characteristics, *i.e.*, fiber layouts and transmission rates, and (ii) PE and CE router configuration and resource management. To design and manage for SLAs, the details of core router behavior play a relatively small role, given the capacity and redundancy built into modern network cores. Transmission characteristics are readily accounted for through understanding lower layer routing and restoration capabilities. The key to engineering to meet SLA targets then quickly

2.4: A summary of experimental approaches in studies of router buffer sizing.

Reference	Evaluation Approach
Jacobson, 1990 [120]	This is one of the first times the bandwidth-delay product rule is mentioned, though from the text it is clear that this rule was at least somewhat commonly known. In particular, Jacobson notes the following: "And, of course, we should always remember that good engineering practise suggests a b-d p worth of buffer at each bottleneck – less buffer and your simulation will exhibit the interesting pathologies of a poorly engineered network but will probably tell you little about the workings of the algorithm (unless the algorithm misbehaves badly under these conditions but my simulations and measurements say that it doesn't). In these days of $100/megabyte memory, I dearly hope that this particular example of bad engineering is of historical interest only."
Villamizar and Song, 1994 [205]	The authors evaluated end-to-end TCP throughput performance in the ANS test network using IBM RS6000-based routers. Background traffic was produced using SGI Indy workstations with TCP implementations based on BSD Reno. Traffic consisted of 1, 4, or 8 long-lived TCP connections created using TTCP. Two paths through the ANS test network were used: a 7 hop path with a round-trip time of 20 milliseconds, and an 8 hop path with a round-trip time of 68 milliseconds. No internal network measurements were used, only end-to-end throughput obtained from TTCP. Four builds of the router software were used to evaluate performance using different buffer sizes and using both tail-drop and RED policies. Interestingly, the authors justification for only using a small number of TCP sources as background traffic was that the resulting traffic was likely to result in a worst-case scenario: "The pragmatic approach taken here is to concentrate on supporting single high speed TCP flows and small numbers of simultaneous TCP flows under the assumption that these cases are likely to be far more bursty than aggregations of very large numbers of slower TCP flows."
Morris, 1997, 2000 [156, 157]	Through these studies, Morris developed and evaluated a new queuing methodology called flow-proportional queuing. These studies were performed using the ns simulator (version 1.4) with the Tahoe variant of TCP, and up to 100 long-lived TCP flows. The metrics used included packet loss rate and throughput.
Avrachenkov *et al.*, 2002 [53]	The goal of these authors was not to come up with a buffer sizing formula, but rather to examine the accuracy of a fixed-point approximation formula for TCP throughput. However, they examined performance using a range of buffer sizes. The evaluation setting was the ns-2 simulator using two different traffic sources: up to 100 long-lived TCP and short file transfers of 10 kBytes. (The distribution of file sizes for the short file transfers is unclear from the paper.) Their results suggest that buffers smaller than the tradition BDP could be used with long-lived sources, but that larger buffers were needed for better performance with the short file transfers.
Appenzeller *et al.*, 2004 [51]	These authors used simulation, laboratory-based experiments, and limited wide-area tests to evaluate the traditional BDP formula. The metric used in all experiments was bottleneck link utilization. Their analysis suggested that substantially smaller buffers could be used without significantly reducing link throughput. In simulation, they used ns-2 to produce up to 500 long-lived TCP flows, or a variety of short flows that did not leave TCP slow-start. We ran the laboratory experiments referred to in this paper. The background traffic used was similar to the simulation setup, and commercial IP routers were used in the testbed. However, simple measurements were taken using the routers themselves rather than an external system that had been calibrated for accuracy. The wide-area tests performed at the Stanford University campus involved modifying the size of a production router attached to the student residence network. Again, measurements were derived from the router itself rather than from an external, calibrated system.
Dhamdhere *et al.*, 2005 [88]	In response to the study by Appenzeller *et al.* [51], these authors developed the "buffer sizing for congested links" (BSCL) strategy for sizing buffers. Their evaluation was based on ns-2 simulations using three different types of background traffic: long-lived TCP flows, bursty web-like flows that arrive in a closed-loop manner, and bursty web-like flows that arrive in an open-loop manner. A range of round-trip times was also used. They considered throughput, loss, and delay metrics in their evaluation.
Gorinsky *et al.*, 2005 [113]	In this simulation-based study, the authors considered two different network topologies with a range of round-trip times, and two different version of TCP (NewReno and Vegas). They used three types of background traffic: long-lived TCP flows, a combination of long-lived TCP flows and constant bit rate UDP flows, and a combination of long-lived TCP flows and short TCP flows. In their evaluation they considered throughput, loss, delay, and goodput, suggesting that buffers could be sized as $2 \times L$, where L is the number of incoming links to a router.
Vu-Brugier *et al.*, 2007 [207]	These authors used a laboratory testbed using FreeBSD and Dummynet for routers to evaluate the loss rate, delay, and utilization for a range of drop-tail buffer sizes. Background traffic was produced using a modified version of `Iperf` [198] to create Pareto-distributed flow sizes and flows that were initiated according to a Poisson process. Round-trip times and buffer sizes were modified in the testbed using the Dummynet system.

reduces to understanding the per-hop performance characteristics of the routers on the edges, the CEs and the PEs, and this is where our experimental study meets SLA engineering.

To meet realistic customer expectations and to engineer their networks effectively and efficiently, providers recognize that SLAs involve trading off risk and expense, and they seek to design for *robust* compliance to an SLA, *i.e.*, configurations that support an SLA in the presence of genuine uncertainties—packet sizes, application mix, or traffic volatility, which even if understood in advance, may change rapidly. Providers also seek simple and *universal* rules for determining router buffer allocations, packet scheduling and shaping algorithms, class of service and drop priority (RED) profiles, and so forth. By universal, we mean that to the largest extent possible, the rules are identical for all routers in the same role (*e.g.,* CE or PE), irrespective of the details of their geographic placement in the network. Of course, geographic details do matter in setting SLA targets (*e.g.,* the delay target between Shanghai and Miami), but providers seek CE configuration rules that would be identical for those two cities.

We are not aware of any detailed treatment of service level agreements in the research literature, though general aspects have been discussed in *e.g.*, [146, 186]. This is largely due to the fact that SLAs are considered proprietary by ISPs. A portion of the tutorial by Shaikh and Greenberg addresses SLAs at a high level [186]. There is also some general information available online from ISPs like Sprint [26], AT&T [2], and NTT [22].

Part I

A Framework for Calibrated Network Measurement

Chapter 3

Calibration Framework

cal · i · bra · tion.
noun.
the action or process of calibrating an instrument or experimental readings: *the measuring devices require calibration | calibrations in the field of electronic measurements.* · each of a set of graduations on an instrument.
—New Oxford American Dictionary, Second Edition.

3.1 Overview

Calibration strategies for Internet measurements are essential for detecting inaccuracy in the underlying data, and misconceptions or errors in their analysis [172]. In this chapter, we describe a set of calibration techniques and tools that can greatly increase our confidence in the validity and accuracy of measurement methodologies and the data resulting from them. Our calibration framework can be applied both to the study of active measurement methodologies designed to estimate path characteristics such as packet delay, packet loss, capacity, available bandwidth and shared congestion, and to the problem of measuring performance aspects of an Internet system, such as a router. Echoing the same sentiment as expressed by Paxson in [172], calibration is not meant to achieve perfection. Rather, it is to aid in our understanding of measurement methodologies and networked systems and their applicability and limitations by evaluating their behavior and performance using measures of "ground truth". Calibration may also illuminate the circumstances under which measurement tools may give inaccurate results or systems may perform poorly.

There are two conventional and complementary aspects to calibration: *comparison* with a known standard, and (if necessary) *adjustment* to match a known standard. The first notion encompasses the tasks of comparing the output of a measurement tool or evaluating the performance of a networked system with "ground truth" measures—a known quantity like the reading of an accurate and precise device. For example, this activity could involve comparing the output of an active measurement tool with accurate ground truth measurements. The second facet of calibration involves changing some feature of a measurement tool so its output matches a standard as closely as possible. This

aspect may involve adjusting parameters of an algorithm realized in a measurement tool, or modifying the algorithm itself in order to produce measurements that are closer to ground truth for a wide variety of scenarios.

Traditional approaches for calibrating and validating active measurement-based performance tools and for understanding networked system behavior almost always employ two basic strategies. One is the use of simple ns-type simulations, and the second consists of small-scale experiments in the "wild," *i.e.,* largely uncontrolled tests that use parts of the live Internet. Ns-type simulations are attractive since they have the advantage of simplified implementations and complete experimental control. However, by definition they are an abstraction of networking reality [105, 106, 107] which may render their results largely irrelevant in situations when the details of live system and protocol implementations or traffic conditions have little in common with their simulation-based counterparts. In contrast, experiments that use parts of the live Internet encounter networking systems, protocols and traffic conditions (depending on the part of the Internet to which they are confined) similar to what would be expected in other parts of the network. However, experiments run in the wide area are largely uncontrolled and typically lack the necessary instrumentation for establishing "ground truth" against which results can be compared. While networking researchers have been generally aware of the pros and cons of these two strategies, the lack of realism in ns-type simulations and the lack of control and instrumentation in the wide area cast serious doubts on this predominant approach to tool calibration and validation, and highlight the need for improved calibration strategies.

In this chapter, we propose an alternative calibration strategy based on conducting experiments in a laboratory setting that is amenable to establishing the "ground truth" for a great variety of Internet-like scenarios. This setting should include, wherever possible, the use of actual hardware found on end-to-end paths in the Internet (*e.g.*, routers, switches, etc.), the use of various versions of the full TCP/IP protocol stack, workload generators capable of exercising systems over a range of realistic conditions, the ability to emulate or recreate wide-area path and link conditions such as propagation delay, and measurement devices that provide a level of accuracy suitable for establishing ground truth. By advocating such an *in vitro*-like experimental environment, we combine the advantages of ns-type simulations (*i.e.,* complete control and full instrumentation) with those offered by experiments in the wide area (*i.e.*, more realistic network systems, protocols and traffic dynamics). *In vitro* calibration techniques are standard in other scientific disciplines such as chemistry and biology, but they have not seen widespread use in the networking area.

We present three case studies in the second part of this dissertation to illustrate the proposed laboratory calibration approach. In the first case study, we consider the problem of estimating the available bandwidth (AB) along a network path, in the second case study, we investigate the problem of active measurement of packet loss characteristics, and our third case study concerns the problem of how to size buffers in routers. For each of these case studies, a detailed understanding of realistic queuing effects and packet-level behavior as they compete and interfere with other packets is essential, *i.e.*, these are fundamental behaviors of the statistically multiplexed, packet-switched Internet. In turn, understanding these effects requires fine-grained time-synchronized measurements of packets as they arrive at *and* subsequently depart from the different routers along a network path.

In our case studies, we address the need for analyzing such detailed measurements from an instrumented laboratory setting with multiple quantitative and qualitative tools, including *phase plot analysis*. Qualitative analysis based on visualizations provides a natural means for exploratory analysis of large complex data sets. The high level objectives in such analyses are to identify both interesting general patterns in the data and relevant domain-specific details in parts of the data. Standard methods for visualizing data are mainly comprised of 2D plots or graphs that may include reference information (*e.g.*, regression curves). However, the key to effective visual analysis is to present the data in ways that offer great flexibility, thereby facilitating the detection and identification of critical characteristics. This suggests several requirements for visualization tools in the context of Internet data, including the ability to handle large, high-dimensional data sets, the flexibility to support a variety of views of the data based on Internet context, and the applicability to a variety of different, network-specific problems. In [172], Paxson also calls for visualization tools to support a range of exploratory analyses.

Phase plots enable visualization and analysis of a variety of complex network traffic behavior at different levels of granularity (*e.g.,* from packets to flows to more application-specific quantities) and over a range of time scales. Although we appeal to additional quantitative information in our case studies, the qualitative insight rendered by phase plots is key to the calibration framework we advocate. We do not claim that phase plots are a panacea, exposing all sources of error or bias in network measurements. We do, however, believe that there is an outstanding need for increased rigor in the evaluation of measurement tools and networked systems; in transforming theories and assumptions into robust programs running on idiosyncratic computers and in idiosyncratic networks.

To close this chapter, we describe a tool that we designed for phase plot and scatter plot analysis, called SPLAT (Scatter and Phase PLot Animation Tool). Through a series of three examples, we illustrate SPLAT's capabilities for exposing interesting network behavior and for assisting in measurement calibration. We further demonstrate SPLAT's usefulness in Chapter 6, our case study on available bandwidth estimation tools.

3.2 Calibration Issues

Active measurement tools send probe packets into the network, measuring subsequent responses of the probe packets exiting at some other instrumented point in the network or reply packets returning to the sender. Comparison with a standard and subsequent adjustment of an active measurement tool's algorithm or parameters are complementary activities of calibration. The basic task of comparing the output of a measurement tool with ground truth measurements typically requires relatively simple measurements. However, to gain insight into *how* a measurement algorithm arrives at a particular estimate we require measurements and analysis suited to the probes produced by the tool and the reported measurements.

Besides active measurement tools, our calibration approach and the associated laboratory test environment is an ideal setting for detailed performance evaluation studies of networked systems such as routers, firewalls, and other

devices. In particular, the ground truth measurements necessary for evaluating the accuracy of an active measurement tool can be used to study the performance and behavior of a router. In either case, we require appropriate test environments to analysis measurement tool accuracy or networked system performance over a range of controlled conditions in order to expose algorithmic, parametric, or system design assumptions that may need adjustment or further investigation.

As part of our framework, we offer a set of issues to consider for network measurement calibration:

1. The hardware (*e.g.*, general-purpose workstations with standard network interface cards, or special-purpose router line cards) and operating system (OS) (*e.g.*, a standard workstation OS such as Linux, Windows, or MacOS X, or a special-purpose OS such as Cisco's IOS or Juniper's JUNOS) used in networked systems and in workstations used to run network measurement tools impose performance and predictability limitations on basic networking tasks such as packet forwarding and on network measurement algorithms. For active measurements, two key considerations are whether probe packet streams (specifically spacing between packets) can be generated with sufficient fidelity, and if timestamp accuracy (and in some cases, synchronization) is sufficient.

2. Assumptions about and/or abstract models to describe the behavior of routers and switches in networks are the foundation for inference methods used to interpret active measurements. The diversity of the implementation details of those systems can limit the effectiveness of the inference methods. Understanding the behavior and generality of how those in-network systems perform has clear implications for the design of measurement algorithms and tools.

3. The heterogeneity and burstiness of traffic can extend beyond the operating bounds of the tool or networked system. For example, routers can only process a limited number of packets per second for a given interface or line card. These limits may not allow a router to effectively process a line rate stream of minimum-sized packets.

4. Probes and response packets generated during active measurement impose a load on the network which can change the conditions on the path of interest and potentially skew results.

5. Many active probe tools require specification of a set of parameters before they are used. A tool's effectiveness can be limited by its sensitivity to configuration parameters. Similarly, modern routers are complex pieces of machinery with many options that can be configured. Suboptimal setting of these parameters can have a tremendous impact on the traffic passing through a router.

These issues imply that certain assumptions, while valid and acceptable in simulation, may lead to unexpected behavior when a measurement tool or networked system is deployed in a live Internet setting. They further imply that fully instrumented environments are critical for understanding the impact and reported measurements of an active

measurement tool or the performance of a networked system. Finally, they suggest that calibration should be performed in a controlled, yet realistic (to the extent possible) environment.

3.3 Calibration Strategy

To address the above issues, we advocate the use of laboratory-based testbeds for calibrating active measurement tools and in-network systems. Such environments provide an important set of capabilities not offered by standard simulation [151] or *in situ* settings such as PlanetLab [23], including control and repeatability, transparency, and the use of actual systems and implementations of actual protocols. The essence of our calibration strategy consists of the following:

1. Design appropriate test environments where a standard can be established over a range of increasingly complex, repeatable test conditions. Essential to this first step is the availability of hardware that provides measurements with a level of accuracy greater than the active measurement tool, or measurements that are sufficiently precise and accurate for understanding the behavior of a networked system. Such ground-truth accuracy is typically not available for *in situ* studies.

2. For the testbed setups defined in the first step, identify relevant test suites for assessing issues such as host system capabilities, loading effects, and networked system behavior over a range of expected conditions. Real systems are generally required to study such issues.

3. The evaluation of data collected in the testbed should be aided by flexible analysis and visualization techniques that provide insight into relevant traffic dynamics and, ultimately, the active measurement methodology or networked system under test.

Below, we provide an example laboratory testbed, detailing the capabilities of various components for supporting this strategy. In the past, there have been two limitations to the laboratory approach that we advocate: the ability to conduct tests with representative traffic conditions, and the ability to scalability and accurately emulate representative link and path characteristics. In Chapters 4 and 5 we describe and evaluate proposed solutions to these problems, respectively.

3.4 Laboratory Calibration Environment

Figure 3.1 shows an example laboratory testbed for conducting calibration experiments. We present this example to illustrate some of the relevant features of our calibration environment. We describe below the common elements of the laboratory testbeds used in our case studies described in the second part of this dissertation. In each case study, we present and describe the specific features of the testbed setups used in those experiments.

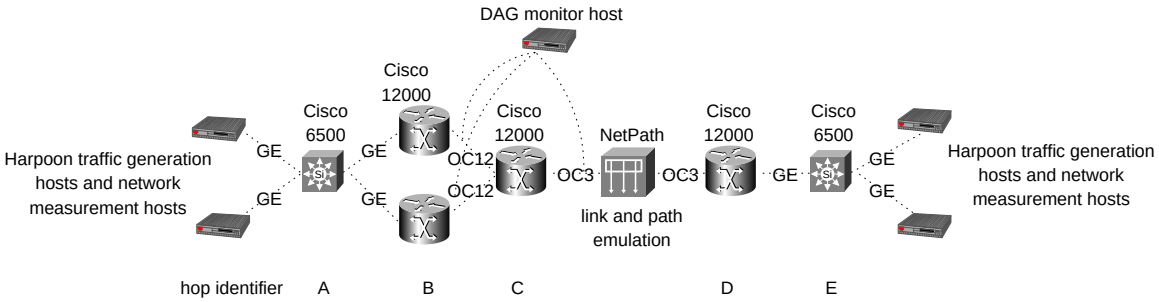
In general, the common features of the laboratory testbeds used in our case studies include: (1) commodity routers from Cisco and Juniper that are representative of systems deployed in the Internet, (2) standard Intel-based (or similar) end hosts running widely-deployed operating systems such as Linux and FreeBSD, (3) a range of network traffic scenarios, an example of which is a bursty, self-similar traffic source that approximates a mix of web-like and peer-to-peer traffic commonly seen in the Internet, (4) emulation of Internet-like link propagation delays, and (5) facilities for collecting ground truth measurements.

To illustrate these elements of our laboratory environment, the setup we show here consists of a simple topology comprised of core-grade Cisco GSR 12000 routers [4] and enterprise-class Cisco 6500 routers [5]. Commodity workstations (standard Intel-based machines with at least 1 GB RAM) for traffic generation and to run active network measurement tools are connected to the Cisco 6500's via Gigabit Ethernet (GE) links. The Cisco 6500's are also connected to the GSRs via GE links. OC-12 (622 Mb/s nominal bandwidth) and OC-3 (155 Mb/s nominal bandwidth) links connect the GSR's. The hosts at the network edges can be used to run the HARPOON traffic generator, which we describe in Chapter 4. HARPOON can be used for generating traffic representative of the Internet across the testbed. Interposed between the GSR's at hops C and D is a system for emulating link and path characteristics, NETPATH, which we describe in Chapter 5. In this testbed, NETPATH might be used to emulate a range of propagation delays for the HARPOON traffic flows traversing the testbed.

To collect detailed measurements for ground-truth comparisons, we use Endace DAG packet capture cards [10]. In the setup shown here, optical splitters are attached to the links incident on the router at hop C. These splitters are then attached to the DAG cards for passive monitoring of the respective links. For this setup, the DAG traces could be used to evaluate the behavior of traffic entering and exiting the router at hop C. For example, the queuing and loss behavior of the router could be examined, and perhaps compared with end-to-end active measurements initiated by the workstations at the end points of the topology. We discuss the specific accuracy characteristics of our ground truth DAG traces in the case studies of Chapters 6–8, below. For now, we note that the DAG card with the most limited capabilities that we use (a DAG 3.5 card with capabilities for OC-3/OC-12 links) offers timestamping accuracy of 1 microsecond[1]. For an OC-3 link, the leading bits of back-to-back 1500 byte packets are separated by about 80 microseconds, while the leading bits of minimum-sized 40 byte TCP packets are separated by about 2 microseconds. More recent cards have higher frequency oscillators, enabling sub-microsecond accuracy at higher line rates, *e.g.*, OC-12, GE, and OC-48 speeds. Thus, these cards provide sufficient accuracy for any packet-level analysis considered in this dissertation.

Although similar environments have been used successfully in recent studies such as [136, 188], they have seen very little use for calibration of measurement tools or detailed study of in-network systems—applications for which they are ideally suited. The open availability of environments that deploy general-purpose workstations and network

[1]The DAG 3.5 has a 16 MHz oscillator, yielding clock ticks every 62.5 nanoseconds. Implementation characteristics of the card reduce the accuracy to 1 microsecond [89].



3.1: An example instantiation of a laboratory calibration testbed.

systems is on the rise including [8, 9, 33]. Our hope is that the benefits demonstrated in this dissertation of conducting experiments in these environments spur their further development, use, and deployment.

3.5 SPLAT: A Scatter and Phase Plot Animation Tool

In this section we describe the tool SPLAT that has been developed for Internet data analysis and to analyze calibration data. As the name suggests, SPLAT offers visualization capabilities based on 2D scatter and phase plots of data. We have enhanced the tool by providing a set of pruning, zooming and feature selection (*e.g.*, filtering) capabilities developed specifically for large high-dimensional Internet data analysis. A distinguishing feature of SPLAT is that it can display animations of phase and scatter plots as they evolve over time. This 3D capability greatly facilitates the discovery and identification of subtle features in the data that may reveal interesting aspects of Internet structure and behavior but would typically be overlooked when using a purely static display of the data. We are not aware of any widely-used visualization tools that have the combination of capabilities provided by SPLAT.

We demonstrate the capabilities of SPLAT in a set of examples that consider (i) characteristics of TCP packet traffic, (ii) characteristics of Internet flows, and (iii) the dynamic structure of source/destination addresses in IP traffic. In these examples, we show how the tool can be used to identify both general patterns and unique features in each of the data sets. These examples illustrate the practical attributes and benefits of this tool in a broad set of problem domains.

3.5.1 Example I: TCP packet traffic characteristics

Many prior studies of packet traffic behavior (*e.g.,* [138]) have been based on measurements taken at a single point in the network such as an ingress link of a router. These measurements usually consist of full IP packet headers and lend themselves directly to many types of analysis, but are generally insufficient for capturing the most basic characteristics of IP networks, *i.e.*, effects of statistical multiplexing and queuing at routers. A simple extension to this basic measurement capability is to gather timestamped IP packet headers at *both* the ingress and egress of a router

(or collaborating end hosts), which provides considerably more information and exposes a much wider range of traffic characteristics.

Given the capability of taking ingress and egress measurements at a router, we can consider two packets from the same source that are emitted close to one another in time. If we measure the time delay between these packets as they arrive at the router ($Ingress\_Spacing = s_i$) and again as they exit the router ($Egress\_Spacing = s_e$), then there are three possibilities for the ratio $s_r = s_e/s_i$. If $s_r = 1$ then spacing remained unchanged by the router. If $s_r > 1$ then other packets enqueued between the two packets causing *expansion*. If $s_r < 1$ then the first packet was delayed because of a queue that has diminished by the time the second packet arrives, causing *compression*. TCP ACK compression is one manifestation of this latter phenomenon[2].

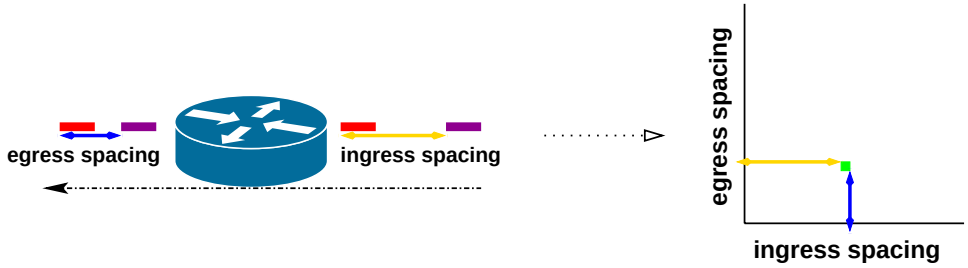
The basic construction of a two-dimensional phase plot from packet traces is depicted in Figure 3.2a. Thus far, we have defined our two measurement points as two links of a path through a router. More generally, these ingress-egress measurements can be considered at arbitrary points along a path. For example, we might collect packet traces at the end hosts of a path, constructing a phase plot that includes effects of all routers and intermediate devices along the path.

As an example of this kind of phase plot analysis, consider the well-known synchronization behavior of long-lived TCP flows with similar round trip times [51, 148, 215]. Reno-based TCP implementations in congestion avoidance cause the bottleneck queue to fill and drain as the sources oscillate between linear increase and multiplicative decrease in sending rate. We used a dumbbell-like topology similar to the one described above and shown in Figure 3.1 and measured the traffic generated by 40 infinite TCP transfers at both ingress and egress of the bottleneck router. The bottleneck link in this setup is an OC-3 (155 Mb/s) and its interfaces are configured with drop-tail queues. A round-trip time of 20 milliseconds is configured using NETPATH, which is described in Chapter 5. Figure 3.3a shows the familiar "sawtooth" characteristic of the time series of packet delays due to queuing. Figures 3.3b and 3.3c show the corresponding phase plots for the traffic at two resolutions. There are two main features in these plots: a cluster of points close to the origin, and a relatively large triangle-shaped element with a lower left corner at an ingress and egress spacing of about 30 milliseconds. From the density indications along the right and upper axes of Figure 3.3b, we see that the majority of packets are sent back-to-back or closely spaced. Although the triangle-like cluster is eye-catching, most packets of each flow arrive closely spaced. Notice also in Figure 3.3a that the maximum queuing delay is about 24 milliseconds, exceeding the configured amount of 20 milliseconds[3].

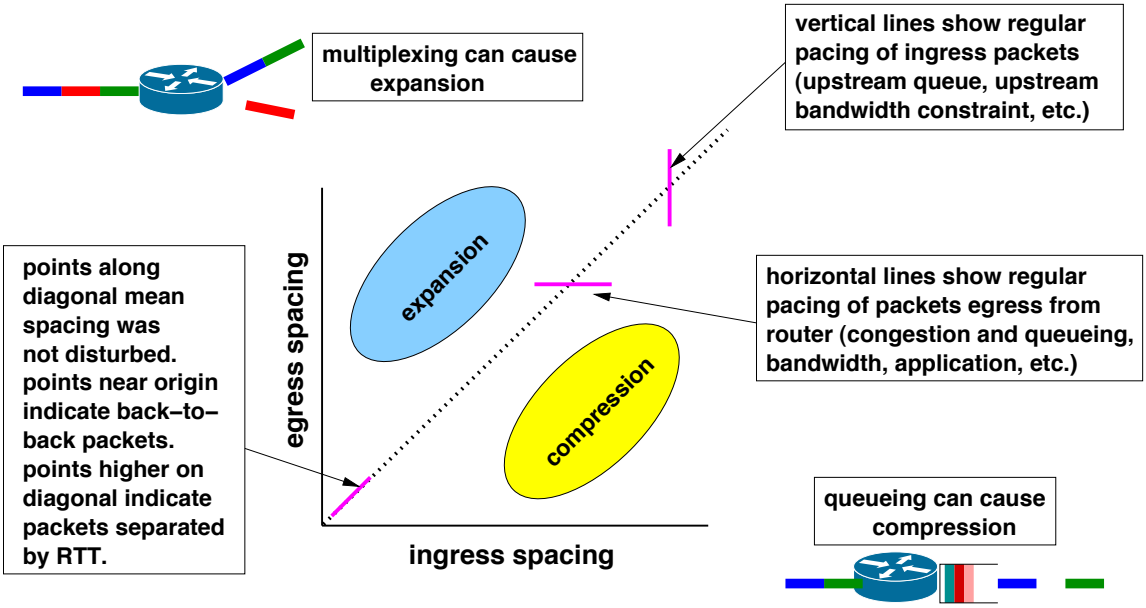
The triangle-like feature of the phase plot has meaningful dimensions and location. As the competing flows increase their congestion windows during the congestion avoidance phase, the queue builds, the round-trip time increases, and so does the spacing between packet trains from each TCP source. This effect results in the line of points

[2]There are, of course, many potential causes of compression and expansion. For example, expansion might be caused by a router-induced delay that is unrelated to congestion, while both compression and expansion may be due to a bandwidth differential between the two measurement points.

[3]The reason is that there is a hidden 128 kByte FIFO on the OC-3 line cards used in these experiments. This buffer accounts for the difference between the configured and observed delay settings.

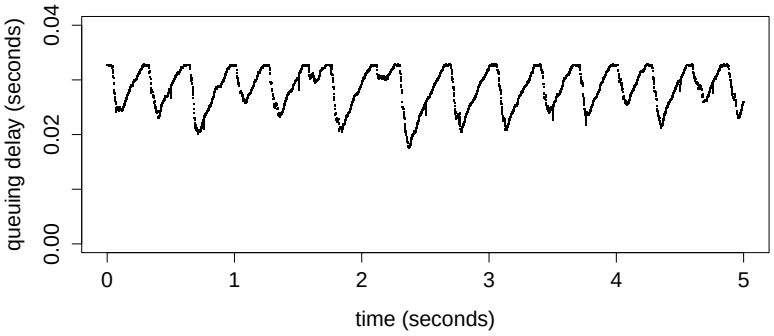


(a) Construction of a phase plot from two synchronized packet traces.



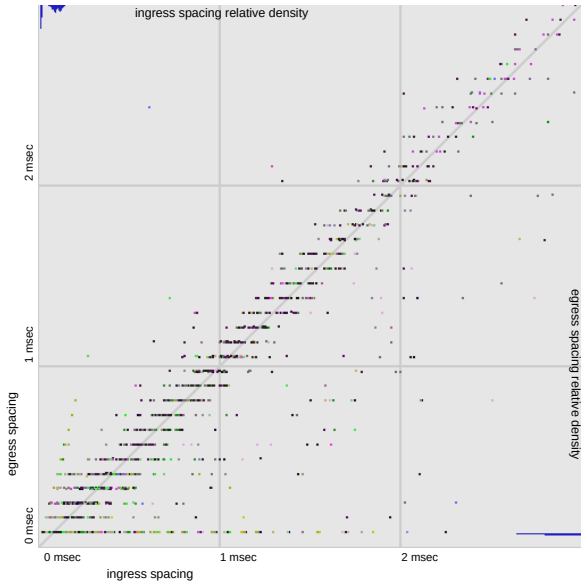
(b) Interpreting phase plot features of flow dynamics.

3.2: Basic construction and interpretation of phase plots for analyzing packet traffic dynamics from measurements taken at the ingress and egress of a router.



(a) Queuing delay over a three second interval of the trace. Well-known oscillating ("saw-tooth") behavior results from synchronization among TCP flows.

(b) Phase plot representation of long-lived TCP connection synchronization. Grid lines are spaced 10 milliseconds apart. Note that most volume is located at lower left (as shown by relative density indications on right and upper axes).



(c) Close-up of lower left corner of (b). Grid lines are spaced 1 millisecond apart.

3.3: Phase plots and an associated queuing delay time series plot created from a laboratory testbed trace of long-lived TCP sources. Phase plot of (c) is a close-up of the lower-left corner of (b). Note that grid lines are not at the same scale for each phase plot. Vertical and horizontal lines along top and left edge of plots indicate density of points along each axis.

along the diagonal of the phase plot[4]. When the queue fills and packet loss occurs, the sources drop their congestion windows, causing the queue to drain, and some packet spacings to be compressed [169, 217]. The triangle feature is approximately 15 milliseconds wide and tall, corresponding to the range of oscillation of the queue. Also, the location of the lower left of the triangle is offset by 10 milliseconds from the round-trip propagation delay of 20 milliseconds. This offset corresponds to the smallest delay measured through the congested queue, as shown in Figure 3.3a.

Finally, we note that when viewing the phase plot as an animated series of consecutive time slices, the points on the triangle of Figure 3.3b appear in a clockwise manner, following the rise and fall of the queue. SPLAT can display a synchronized view of a time series plot similar to Figure 3.3a alongside the phase plot to make this connection visually apparent.

Looking at a close-up of the back-to-back spacing region in Figure 3.3c, we first notice a clear discretization of egress spacings. As closely spaced packets of one flow arrive at the bottleneck link, they are multiplexed with packets from other flows and are respaced by an integral number of 1500 byte packets as they are transmitted at 155 Mb/s. The horizontal striations are separated by approximately 80 microseconds, corresponding to the transmission time of a 1500 byte packet[5].

In this example, SPLAT was helpful in revealing interesting dynamic characteristics of the eye-catching triangular feature and relating those characteristics to the well-understood sawtooth queuing behavior typical for a homogeneous (but unrealistic) network environment (*e.g.*, all TCP flows are long-lived and have similar RTT). The tool also helped to illustrate that while the observed triangular feature is distinctive, it is not the dominant characteristic.

3.5.2 Example II: Flow-level Traffic Characteristics

In this example we illustrate the dynamic characteristics of a scatter plot by comparing flow size and duration. This application of scatter plots could be useful for studies similar to Zhang *et al*. [216], in which they compared characteristics of Internet flow rates with aspects such as flow size and duration. The scatter plots in this section are formed by setting the x-axis to represent flow size in bytes and setting the y-axis to represent flow duration in seconds. A natural choice for the time axis is the starting time of the flow.

Figure 3.4a shows a scatter plot of flow sizes and durations, with a focus on small flows. The data for this example was created using a dumbbell-like topology similar to the one described above and shown in Figure 3.1. The background traffic for this example consists of web-like self-similar traffic created using HARPOON (described in Chapter 4) that varies in utilization over the duration of the experiment, from about 50 Mb/s to 130 Mb/s, averaged over 1 second intervals. Propagation delays in the testbed are configured to be between 35 and 65 milliseconds, with a mean of 50 milliseconds, and the queue is configured to buffer approximately 50 milliseconds of packet data.

[4]The slight expansion in packet spacings evident in this line just above the diagonal is due to the multiplexing of 40 competing TCP flows. The area of expansion extends about three milliseconds, which is about the time needed to transmit 39 packets at OC-3 speed.

[5]There is 149.76 Mb/s available after SONET framing on an OC-3. Transmission time of a 1500 byte packet (plus 9 bytes Cisco HDLC layer 2 overhead) is 80.6 microseconds.

Although the general profile of the figure supports the expected correlation between flow size and duration, we also notice that there are a number of the smallest flows with significantly longer durations than most other flows. If we filter the data to show only time periods when the queue was nearly full (Figure 3.4b), we observe a shift in the data points corresponding to generally longer durations for all flows during congested time periods. If, alternatively, we restrict our view to time periods when the queue was nearly empty (Figure 3.4c), we observe a shift towards generally lower flow durations, as we might expect. Additional filtering capabilities of SPLAT, such as restricting our view to flows with RTTs within a certain range, or confining our view based on the amount of data transferred between a source/destination pair, could be used to draw out subtle characteristics of the plots and determine relevant cofactors.

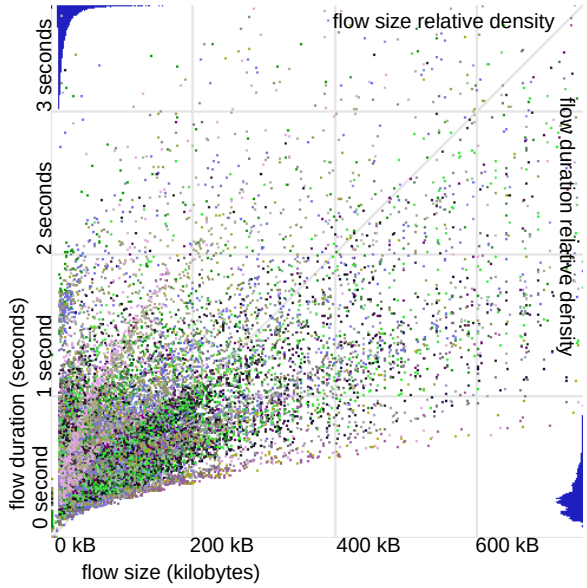
3.5.3 Example III: Structure of Addresses in IP Traffic

In [132], Kohler *et al.* consider the structural characteristics of destination addresses in IP traffic. One way to represent these characteristics in scatter plot form is to consider the source address of a flow as the x dimension, the destination address as the y dimension, and the flow start time as the time dimension. To derive values for the x and y dimensions, IP addresses can be linearly mapped to a given scale. Figure 3.5a depicts this application of scatter plots. In their study, Kohler *et al.* observed that destination address structure was stable over short time scales as seen at a particular location, and that the observed structure can make a practical fingerprint of the aggregate traffic seen at a given vantage point.

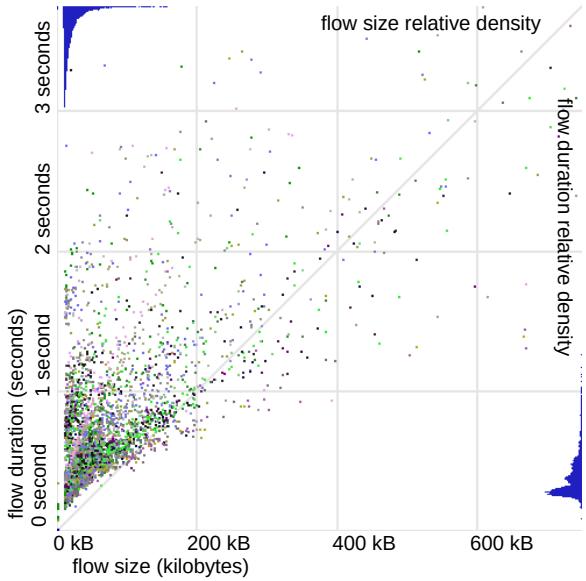
Figure 3.5b shows an IP source/destination address scatter plot created from one hour of flow records collected at a border router of the University of Wisconsin-Madison. The two dominant lines for both source and destination addresses in the plot indicate that most traffic is sourced or sinked by the University's two class B address spaces. The less dominant characteristics are also interesting. For example, a data point in the lower left quadrant indicates a traffic flow between two networks in the traditional class A address range. Also, a set of points in the upper right quadrant indicates one or more multicast sources with a large set of destinations in the class B and C range.

Figure 3.5c shows another IP source/destination address scatter plot created from one hour of flow records collected at the Houston, TX router of the Abilene network [1]. The basic features of this plot compared with Figure 3.5b show the effect of connectivity and perspective of a router on the addresses in traffic observed at that router. Figure 3.5d is produced from the same underlying data as Figure 3.5c, but has been filtered to show only points representing source and destination pairs that transferred more than 1 MByte. From the relatively density indications along the upper and right sides of the plot, we see that there are indeed very few source/destination pairs responsible for most of the traffic. We also see a number of potentially interesting spatial outliers, *e.g.*, in the lower right quadrant.

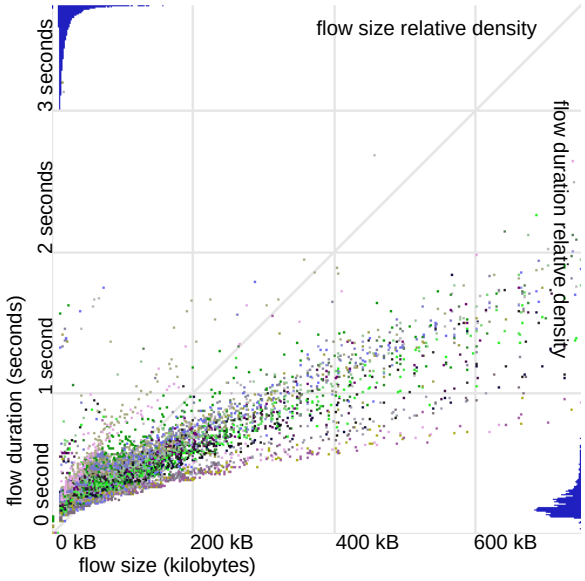
This example suggests that SPLAT can be useful for examining spatial fingerprints or signatures of traffic flows and for helping to understand some of the causes behind these signatures. The animation capabilities of SPLAT make it well-suited as a tool for qualitatively assessing the stability of these signatures over a range of time scales, and also for helping to uncover causes behind possible changes to a signature.



(a) Scatter plot produced from all flows during the experiment. Figure shows a zoom of the smallest flows (up to 750 kBytes).

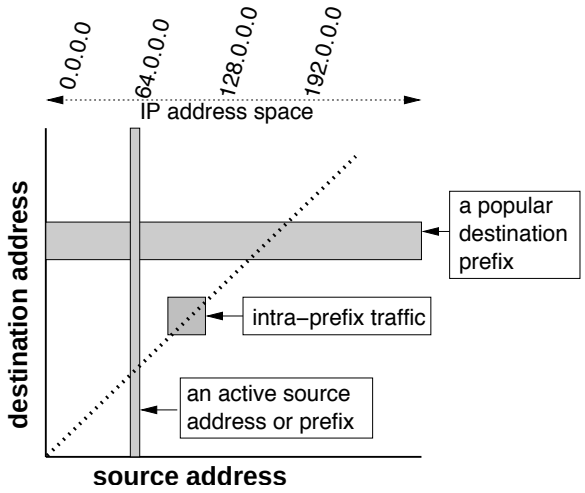


(b) The same data as in 3.4a, but filtered to show periods when the queue was within 90% of its capacity.

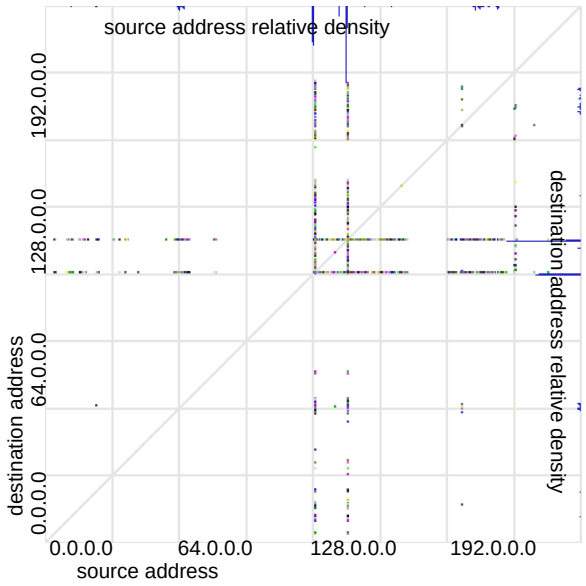


(c) The same data as in 3.4a, but filtered to show periods when the queue was no more than 10% full.

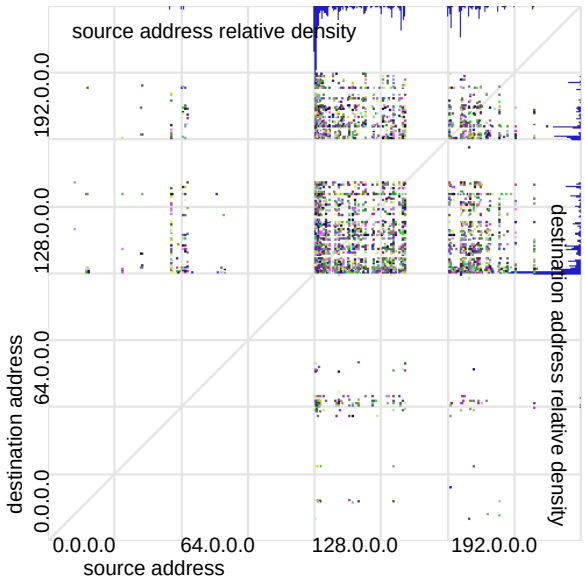
3.4: Scatter plots of flow size (x-axis) versus flow duration (y-axis) generated from traffic produced in a laboratory environment.



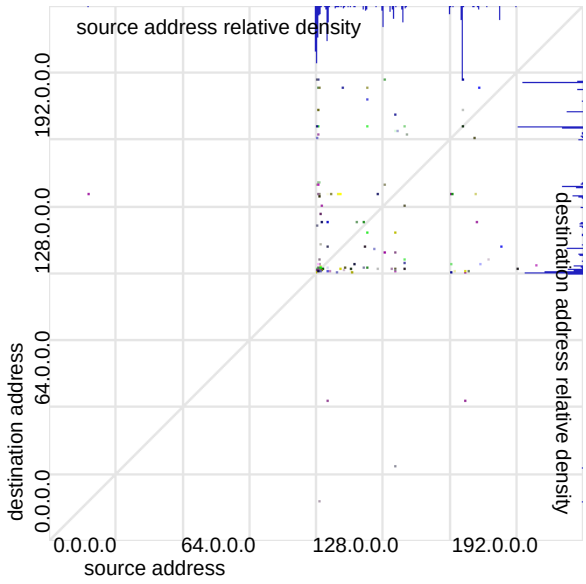
(a) Construction and basic interpretation of an IP source/destination scatter plot



(b) Source/destination scatter plot constructed from flow records collected at a border router of the University of Wisconsin-Madison.



(c) Source/destination scatter plot constructed from flow records produced by the Houston, TX router of the Abilene network.



(d) The same scatter plot as in (c), but filtered to show only source/destination pairs that transferred more than 1 MByte.

3.5: Application of scatter plots to analyzing the structure of source and destination addresses in IP traffic.

Chapter 4

Traffic Generation

> It is our belief that lack of good measurements, lack of tools for evaluating measurement results and applying their results to models, and lack of diverse and well-understood simulation scenarios based on these models are holding back the field. We need a much richer understanding of the range of realistic models, and of the likely relevance of different model parameters to network performance.
> —S. Floyd and E. Kohler [105].

4.1 Overview

The network research community has a persistent need to evaluate new algorithms, systems and protocols using tools that create a range of test conditions similar to those experienced in live deployment and to ensure relevant and reproducible results [39, 105]. Having appropriate tools for generating scalable, tunable and representative network traffic is therefore of fundamental importance. Such tools are critical in laboratory test environments (such as [31, 33]) where they can be used to evaluate behavior and performance of new systems and real networking hardware. They are also critical in emulation environments (such as [203, 209]) and simulation environments (such as [27, 151]) where representative background traffic is needed. Without the capability to create a wide range of realistic network test conditions, new systems run the risk of unpredictable behavior and unacceptable performance when deployed in live environments.

Previous best practices for traffic generation have focused on either simple packet streams or recreation of a single application-specific behavior. Packet streaming methods such as those used in tools like `iperf` [198] consist of sequences of packets separated by a constant interval. These methods form the basis for standard router performance tests such as those recommended in RFC 2544 [70] and RFC 2889 [145]. Another example is an infinite FTP source which is commonly used for traffic generation in simulations. While these approaches can provide some insight into network system capabilities, they lack nearly all of the richness and diversity of packet streams observed in the live Internet [108, 109, 174].

A number of successful application-specific workload generators have been developed including [35, 59, 127]. These tools typically focus on generating application-level request sequences that result in network traffic that has the same statistical properties as live traffic from the modeled application. While they are useful for evaluating the behavior and performance of host systems (such as servers and caches), these tools can be cumbersome for use in router or switch tests and obviously only recreate one kind of application traffic, not the diversity of traffic observed in the Internet. These observations motivate the need for a tool capable of generating a range of network traffic such as might be observed either at the edges or core of the Internet.

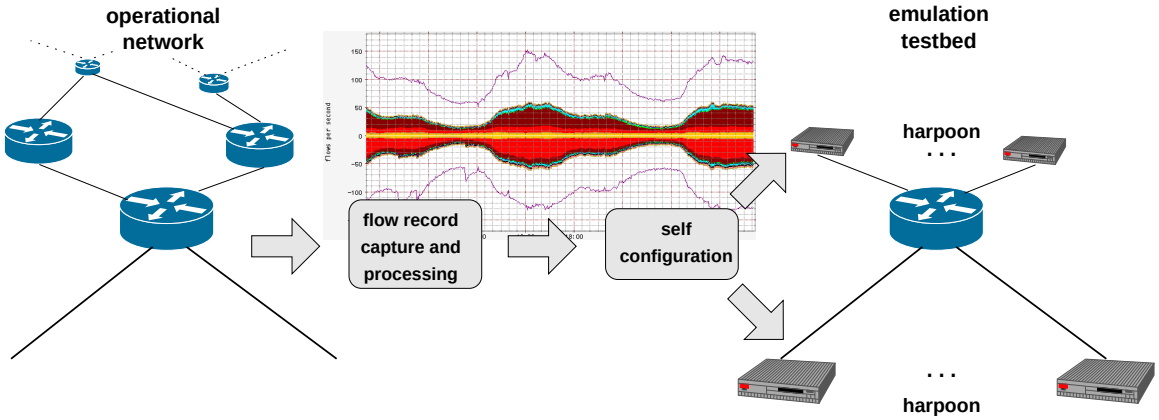
In this chapter, we describe a new network traffic generation method that aims to recreate IP traffic flows (where *flow* is defined as a series of packets between a given IP/port pair using a specific transport protocol—we refine this definition below) representative of those observed at routers in the Internet. We are aware of no other tools that target representative traffic generation at the IP flow level. Our approach is to abstract flow-level traffic generation into a series of application-independent file transfers that use either TCP or UDP for transport. Our model also includes both temporal (diurnal effects associated with traffic volume) and spatial (vis-à-vis IP address space coverage) components. The resulting constructive model can be manually parameterized or can be self-parameterized using NetFlow [6] or packet trace data and components of the traffic generation tool itself. The self-parameterizing capability simplifies use and further distinguishes our approach from other tools that attempt to generate statistically representative traffic.

We realized this model in a tool we call HARPOON that can be used in a router testbed such as our laboratory calibration environment or emulation testbed environment to generate scalable, representative network traffic. HARPOON has two components: *client threads* that make file transfer requests and *server threads* that transfer the requested files using either TCP or UDP. The result is byte/packet/flow traffic at the first hop router (from the perspective of the server) that is qualitatively the same as the traffic used to produce input parameters.

We evaluated HARPOON's capabilities in two ways. First, we conducted experiments in a controlled laboratory environment using two different data sets to show that we can qualitatively recreate the same traffic using HARPOON. We used a one week data set of NetFlow records collected at a border router of the University of Wisconsin-Madison and a series of packet traces (from which we constructed flow records using a standard tool) collected at the border of the University of Auckland. These experiments demonstrate HARPOON's capability to generate representative flow level traffic. Next, we used the same laboratory environment to conduct a throughput evaluation of a Cisco 6509 switch/router following the testing protocol described in RFC 2889. We show that while throughputs achieved using standard packet streaming methods compared with HARPOON are similar, the loads placed on router subsystems are substantially different. The implication of this result is that a tool like HARPOON can augment standard test suites to give both router designers and network operators insight into system behavior under actual operating conditions.

4.2 Architecture

The design objectives of HARPOON are (1) to scalably generate application-independent network traffic at the IP flow level, and (2) to be easily parameterized to create traffic that is statistically identical to traffic measured at a given vantage point in the Internet. Figure 4.1 depicts a high-level process flow of these objectives. We start with the basic definition of an IP flow to create a constructive model for network traffic generation which we describe below.

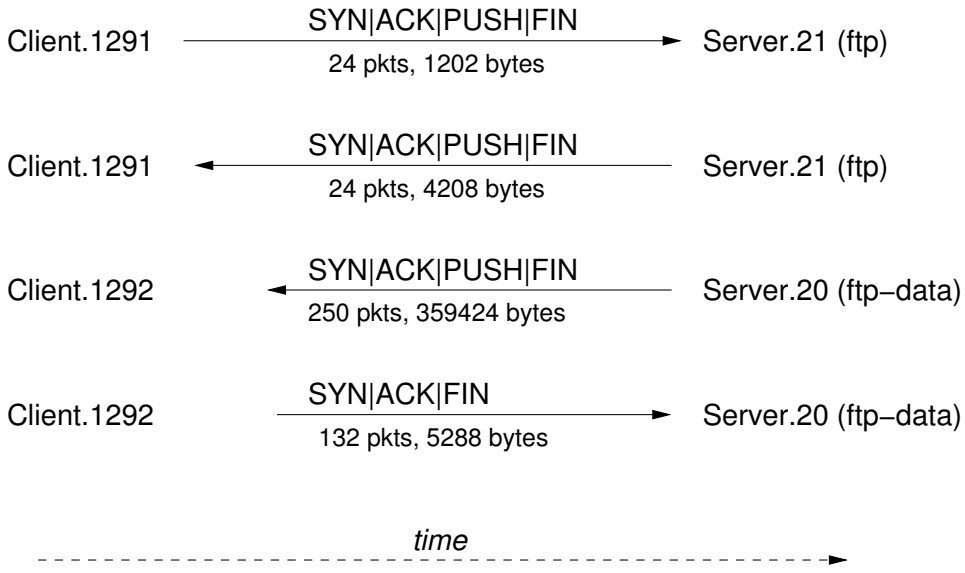


4.1: Flow records are collected at a given vantage point in an operational network using standard software like `flow-tools`. Key aspects of the live flows are extracted during a self-configuration step. These parameters are used to generate traffic in a testbed that statistically matches the temporal (diurnal) volume characteristics as well as the spatial (source and destination IP address frequency) characteristics of the live flows.

An IP flow as defined in [79] is a unidirectional series of IP packets of a given protocol traveling between a source and a destination IP/port pair within a certain period of time. The final condition of this statement is ambiguous, so we tie our definition to the tools we use to gather and analyze network flow data: `flow-tools` [111] and FlowScan [176]. NetFlow data includes source and destination AS/IP/port pairs, packet and byte counts, flow start and end times, protocol information, and a bitwise `OR` of TCP flags for all packets of a flow, in addition to other fields. This data is exported either on timer deadlines or when certain events occur (*e.g.*, a TCP FIN or RST, or a cache becomes full), whichever comes first. While this pragmatically resolves ambiguity in the definition of a flow, specific expiration-related timing behaviors can vary [19]. We discuss how this variation can affect parameterization below. An example of how a single transaction, such as an FTP transfer, can appear in flow records is shown in Figure 4.2. The transaction is represented as multiple data flows between the two hosts. Each direction of the control and data connections is reported, resulting in four flow records. TCP flags, packet and byte counts accumulate over the duration of each connection.

From this operational definition of a flow, HARPOON's architecture begins with the notion of unicast file transfers using either TCP or UDP. HARPOON does not address the packet level dynamics of TCP file transfers. Rather, it relies on the version(s) of TCP running on end hosts to transfer the requested file. Modeling UDP traffic is complicated

by the fact that packet emission behaviors are largely application-specific. At present, HARPOON contains three models of UDP packet transfer: a simple parameterized constant packet rate, a fixed-interval periodic ping-pong, and an exponentially distributed ping-pong. The first source type is similar to some audio and video streams, while the latter two types are intended to mimic the standard Network Time Protocol (NTP) and Domain Name Service (DNS), respectively. UDP traffic in today's Internet is likely to be made up of a wider variety of application level traffic (including voice, SQL worms, *etc*.) whose behavior is not captured in our three source types. Development of a model with a more diverse set of UDP traffic sources is left for future work.

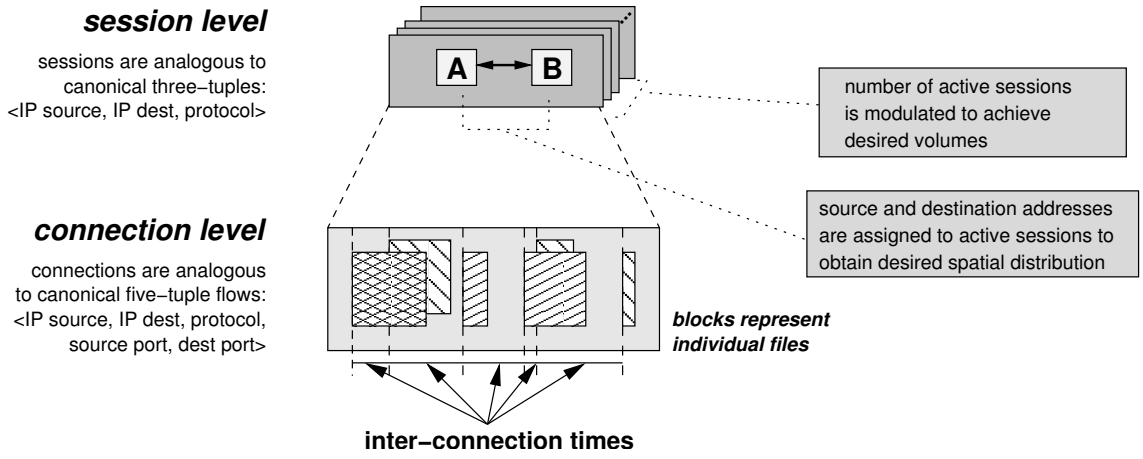


4.2: Flow level decomposition of a simple FTP transaction. Each line is logged as a separate entry by NetFlow.

The HARPOON flow model is a two level architecture and is depicted in Figure 4.3. We refer to the lower level of the HARPOON model as the *connection level*. It is made up of two components that have measurable distributional properties. The first component is the *size* of the file transferred, and the second component is the time interval between consecutive file transfer requests, the *inter-connection time*. HARPOON makes requests for files with sizes drawn from an empirical distribution $P_{FileSize}$. Connection initiations are separated by time intervals drawn from an empirical distribution $P_{InterConnection}$.

The upper level of the HARPOON model is referred to as the *session level*. HARPOON sessions are divided into either TCP or UDP types which then conduct data transfers using that protocol during the time that they are active. The session level has two components: the number of *active sessions* and the *IP spatial distribution*. By modulating the number of sessions that are active at any point in time, HARPOON can match the byte, packet, and flow volumes from the original data and realize the temporal (diurnal) traffic volumes that are a common characteristic of the Internet [170]. The average number of sessions of each type (TCP/UDP) that are active at any point in a day is derived

from a flow data time series for consecutive non-overlapping intervals of length *IntervalDuration* seconds to create an empirical model for $P_{ActiveSessions}$. Scalability is naturally achieved by dividing the number of active sessions across any number of hosts comprising the testbed. For each session, HARPOON picks source and destination addresses from ranges of available addresses to make a series of file transfer requests. The address selection is made preferentially using weights drawn from empirical distributions $P_{IPRange_{src}}$ and $P_{IPRange_{dest}}$. A series of file transfer requests then takes place between the source and destination for *IntervalDuration* seconds. When HARPOON is started, it begins with the average number of sessions in the first interval and proceeds through consecutive intervals for the duration of the test.



4.3: HARPOON's flow-based two-level hierarchical traffic model. Sessions are comprised of a series of connections separated by durations drawn from the inter-connection time distribution. Source and destination IP address selection (A and B in the figure) is weighted to match the frequency distribution of the original flow data. The number of active sessions determines the overall average load offered by HARPOON. A heavy-tailed empirical file size distribution and an ON/OFF transfer model generate self-similar packet-level behavior.

In summary, the HARPOON model is made up of a combination of five distributional models for TCP sessions: file size, inter-connection time, source and destination IP ranges, and number of active sessions. These parameters are summarized in Table 4.1. There are three distributional models for UDP sessions: constant bit rate, periodic ping-pong, and exponential ping-pong. Each of these distributions can be specified manually or extracted from packet traces or NetFlow data collected at a live router. These models enable the workload generated by HARPOON to be application independent or to be tuned to a specific application. The models are combined in a constructive manner to create a series of file transfer requests that results in representative flow-level network traffic.

4.3 Implementation

A key feature of HARPOON is that it is self-configuring. NetFlow logs or packet traces are used for parameterization without any intermediate modeling step, obviating the need for HARPOON users to become experts in distribution

4.1: Summary of HARPOON configuration parameters for TCP sources.

Parameter	Description
$P_{FileSize}$	Empirical distribution of file sizes transferred.
$P_{InterConnection}$	Empirical distribution of time between consecutive TCP connections initiated by an IP source-destination pair.
$P_{IPRange_{src}}$ and $P_{IPRange_{dest}}$	Ranges of IP addresses with preferential weights set to match the empirical frequency distributions from the original data.
$P_{ActiveSessions}$	The distribution of the average number of sessions (IP source-destination pairs) active during consecutive intervals of the measured data. By modulating this distribution, HARPOON can match the temporal byte, packet and flow volumes from the original data.
$IntervalDuration$	Time granularity over which HARPOON matches average byte, packet and flow volumes.

and parametric estimation. In this section, we first discuss the relevant implementation issues and limitations in transforming flow records into a suitable configuration for HARPOON. We follow with a description of the implementation of the traffic generation component of HARPOON.

4.3.1 Self-Configuration

The self-configuration phase of HARPOON takes flow records as input and generates the necessary parameters for traffic generation. The key parameters are distributional estimates of (1) file sizes, (2) inter-connection times, (3) source and destination IP addresses, (4) and the number of active sessions. We divide the input flow records into a series of intervals of equal duration to generate the number of active sessions in order to match average byte, packet, and flow volumes of the original data over each interval. We also discuss below how the interval duration, a configurable parameter, is set. For each parameter, we use the empirical distribution derived from the original flow or packet data and do not attempt fitting to a known distribution (although HARPOON could be trivially enhanced to generate variates from known distributions).

File Sizes Flow records contain packet and byte counts, so a first approximation of file sizes (flow payload sizes) can be extracted by $ByteCount - PacketCount * 40$ (assuming no IP or TCP options). To this calculation we make the following refinements. First, due to flow record timeouts, a single flow may be split into multiple flow records. To counter this effect, we perform "flow surgery" [93] on the raw flow records, coalescing records where addresses, ports, protocol, timestamps and TCP flags indicate that the records refer to the same flow. Second, we only perform the calculation if there are start and end markers present in the TCP flags, *i.e.*, a SYN flag and a RST or FIN flag. This check ensures that we do not underestimate application payloads because of missing the beginning or end of a flow. Third, we discard flows that appear to be "ACK flows" or flows that are very small (*e.g.*, the request direction for an HTTP transfer).

There are two practical complications to calculating file sizes from flow records. First, some routers do not record TCP flags in flow records. In this implementation of HARPOON, we assume that these flags are available. Also, it is common practice on high-bandwidth links to perform packet sampling (*e.g.*, by recording every *N*th packet or recording a packet with probability $\frac{1}{N}$) to reduce the storage and processing requirements of maintaining flow records. Duffield *et al.* [93] describe methods for recovering the distribution of flow sizes from sampled flow records. Application of these methods to the self-configuration step of HARPOON and relaxing assumptions regarding presence of TCP flags are areas for future work.

Inter-connection Times To extract the inter-connection time distribution, we again make use of TCP flags in the flow records. For each source and destination IP address pair encountered, we create an ordered list of start times for flows that contain a SYN flag. The collection of differences between consecutive SYNs for each address pair constitutes the inter-connection time empirical distribution. In practice, we impose a bound on the maximum inter-connection time (*e.g.*, 60 seconds). We discuss implications of this bound below.

Source and Destination IP Addresses To emulate the spatial characteristics present in flow records captured in a live environment, we first extract the empirical frequency distributions of source and destination IP addresses. We map the resulting rank-frequency distributions onto source and destination address pools (from the perspective of a HARPOON client) used in the testbed. For example, if the source address pool for a testbed host is specified as a class C network, we map the frequency distribution of the top 254[1] addresses from the live flows to HARPOON configuration parameters.

Number of Active Sessions One way to calculate the number of sessions that should be active over a series of intervals is to start with the observation that each source and destination IP address pair (the analog of a session in HARPOON) contributes to the overall load during one or more intervals. For each host pair, we find the earliest flow start time and latest end time and distribute values proportional to the lifetime of that session over the corresponding intervals.

While the above technique might seem to be the most direct way of calculating the number of active sessions, it fails because the end timestamp of flow records frequently does not reflect the precise time of the final packet of a flow. The inaccurate timestamps extend flow durations and therefore cause the number of sessions that should be active over a series of time intervals to be overestimated. This inflation could cause the byte, packet and flow volumes generated by HARPOON to exceed the original volumes. The inaccuracy in the end timestamp may be caused by delays from waiting until a flow export packet or cache fills, or by lingering out-of-order packets[2] [19]. In contrast, flow start timestamps appear to accurately reflect the first packet of a flow (allowing us to use them in calculating the inter-connection times).

To quantify the timing inaccuracies introduced by NetFlow, we generated traffic through a Cisco 6509, capturing NetFlow records from the router and simultaneously taking a packet trace using a high precision DAG 3.5 capture

[1] 256 addresses in a full class C minus host address (.0) minus broadcast address (.255) equals 254 usable addresses.

[2] There are also inaccuracies that are related to the coarse-grained nature of flow records. Since we subtract contributions of headers (including connection setup and teardown packets and associated latency) to file sizes, we would really like the start and end timestamps to reflect the first and final payload packets of a flow.

4.2: Summary statistics of differences (in milliseconds) between NetFlow timestamps from a Cisco 6509 and flow records generated from a DAG 3.5 packet trace.

timestamp	**mean**	**median**	**standard deviation**
flow begin	19	0	8
flow end	454	461	254

card[3] [10]. Table 4.2 shows the sample mean, median, and standard deviation for begin and end timestamp differences in milliseconds. While most differences in the begin timestamp are zero (note the median) and otherwise quite small, differences in the end timestamp are, on average, significantly larger. These end timestamp differences, when compounded over several thousand flows per minute (see Figure 4.7c, for example) cause our initial algorithm to fail. On the other hand, using flow records constructed from raw packet traces with accurate timestamps leads to a good match. Results for the Auckland trace, described in Section 4.4, were generated in this way.

The revised approach we take for tuning the number of active sessions is outlined in the pseudocode shown in Listing 4.1. We first make the assumption that, in the overwhelming majority of cases, a file request made during interval I_j is also completed during interval I_j. This assumption is reasonable based on relatively large values for $IntervalDuration$, such as 300 or 600 seconds, and a reasonable bound on round-trip times. We determine how many sessions are required to generate the required byte volume over the duration of an interval by mimicking the action of HARPOON sessions as they alternate between waiting for an amount of time drawn from the $P_{InterConnection}$ distribution and requesting files whose sizes are drawn from the $P_{FileSize}$ distribution[4]. We successively increment the number of sessions, imitating the action of each session and noting when we surpass the requisite byte volume for the given interval.

In practice, we run the algorithm for each interval N times, using the mean number of sessions required to generate the necessary volume. As shown in the validation experiments of HARPOON, tuning the number of active sessions to the required byte volume using the technique we outline here leads to an accurate match with the original byte, packet, and flow volumes over relatively coarse intervals.

Interval Duration The value $IntervalDuration$ is the time granularity over which we match byte, packet, and flow volumes between the originally measured flow data and HARPOON. The duration can be selected somewhat arbitrarily, but there are two practical constraints. First, since we choose a source and destination address for each active session at the start of each interval, there is a tradeoff between the length of an interval and how well the spatial distribution can be approximated over the course of a full test. With longer intervals, there is less opportunity for sampling a spatial distribution. With shorter intervals, there is an increased internal overhead from managing sessions and timers. In our experience, intervals such as five minutes work well. Five minutes also happens to be a common interval over

[3]The DAG capture point was situated one hop after the 6509. Differences in timestamps due to propagation and forwarding delays were assumed to be negligible.

[4]Note that the connection initiation schedule defined by the $P_{InterConnection}$ distribution is made independent of network feedback.

Listing 4.1: Pseudocode for algorithm used to determine the number of sessions that should be active over a given interval to produce a certain byte volume.

```
# Input:
#    1) IntervalDuration (in seconds)
#    2) Intended byte volume for the interval
#    3) FileSize and Interconnection distributions

BytesGenerated = 0                          # Total bytes generated by mimicked
                                            # sessions.
SessionsRequired = 0                        # Sessions required to create the
                                            # intended volume.

while BytesGenerated < IntendedByteVolume:  # We want to generate at least as many
                                            # bytes that were originally sent.

    SessionsRequired += 1                   # One more session is required...

    ElapsedTime = 0                         # ElapsedTime holds amount of time the
                                            # current session has been active
                                            # during this interval.

    while ElapsedTime < IntervalDuration:   # This loop mimics the action
                                            # of a single session.

        ElapsedTime += InterConnectionTimes.next()  # Get the next inter-connection
                                            # time from the empirical distribution.

        BytesGenerated += FileSizes.next()  # Get the next file size from
                                            # the empirical distribution (and
                                            # assume the file is transferred
                                            # in the current interval).
    end while
end while

# Output:
#    The number of sessions need to generate the intended byte volume for an
#    interval of duration IntervalDuration is stored in the variable SessionsRequired.
```

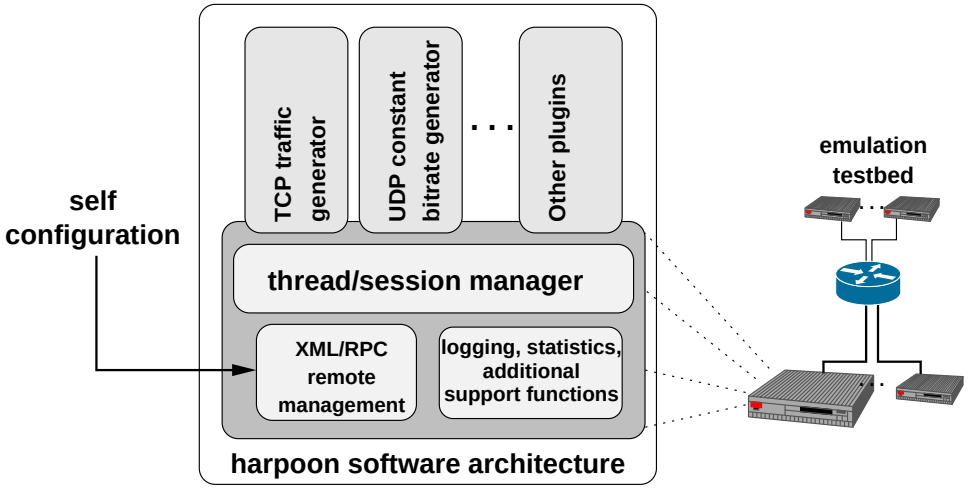
which flow records are aggregated (as implemented by the widely used `flow-tools` [111]). The second practical consideration is that *IntervalDuration* should be at least as long as the maximum inter-connection time. One reason is that a session may randomly get an inter-connection time that causes it to be idle for the entire interval. Another reason is that at the end of an interval, there may be some sessions that are waiting for an inter-connection time to expire before initiating a connection—we do not prematurely halt sessions at the end of an interval to ensure sampling the longest inter-connection times. The sessions that are waiting for the next connection time at the end of an interval are technically active but not engaged in file transfers and count against the target number of active sessions for the new interval. The effect of these temporarily frozen sessions is that HARPOON may not generate the intended volumes for every interval after the first. Setting *IntervalDuration* to a large enough value along with setting an appropriate cap on the maximum inter-connection time when processing the original flow data resolves this potential problem.

The value *IntervalDuration* is used during parameterization for determining the $P_{ActiveSessions}$ distribution and is also used to control traffic generation. The values used for each phase need not be the same. This decoupling enables time compression (or expansion) relative to the original data during traffic generation. HARPOON can either be set to match the aggregate volume of the original intervals or to match the bit rate. For example, if N TCP sessions are measured from flow logs in interval I_j and M TCP sessions are measured in interval I_{j+1}, HARPOON could be configured to initiate $N+M$ sessions over a test interval I_k to realize similar aggregate volume. To match the original bit rate, the traffic generation *IntervalDuration* is simply set to a smaller value than the duration used for parameterization. However, there is a potential pitfall when matching bit rate by using a shorter traffic generation *IntervalDuration*. If the interval is too short, there may be insufficient sampling of the $P_{InterConnection}$ and $P_{FileSize}$ distributions with the configured number of active sessions. The result is that the byte and packet volumes generated over the interval may vary far from the expected value.

While it is true that our use of rather coarse intervals for matching original volumes tends to ignore issues of packet arrivals over short (sub-RTT) time scales, this is intentional. Additional testbed parameters could have been required with a HARPOON setup (such as a distribution of round-trip times, link capacities, MTUs, etc.) to match parameters derived from a live trace, but at this point we do not specify such configurations. Creating the necessary volumes over longer time scales to produce self-similarity and diurnal patterns in a way that real application traffic is generated is the intent and domain of HARPOON. We believe it is preferable to allow burstiness over short time scales to arise endogenously from TCP closed-loop control and network state, rather than to exogenously attempt to force such state (*e.g.*, by introducing a random packet dropping element along a path to effect a certain loss rate). We demonstrate limitations of HARPOON with respect to short time scales below.

4.3.2 Traffic Generation

HARPOON is implemented as a client-server application. Figure 4.4 illustrates the conceptual aspects of the implementation. A HARPOON client process consists of threads that generate file requests. The hierarchical session/connection model is realized by individual threads, and the distribution for active sessions maps to a number of threads actively making file requests. In each thread, connections are initiated according to the inter-connection time distribution. The duration of any file transfer is dictated by the dynamics of the transport protocol and the underlying testbed configuration. Inter-connection times, on the other hand, are followed independently of the transport layer so an active session may be multiplexing multiple concurrent connections. Note that at the connection level, HARPOON operates in an *open loop* manner since subsequent connections are made independent of network feedback. (HARPOON can be easily modified for *closed loop* behavior, *e.g.*, if inter-connection times are considered as OFF periods between connections for a given session.) HARPOON server process consists of threads that service HARPOON client requests. The server controls the sizes of files transferred according to the input distribution.



4.4: HARPOON software architecture. A core session manager controls dynamically loadable traffic generator plugin modules and through an XML/RPC interface indirectly handles remote requests to stop or start plugins, load new configurations, or retrieve plugin statistics.

While TCP file transfers are controlled by protocol dynamics, UDP dictates no control over packet emissions. Currently, HARPOON can send UDP datagrams at roughly constant bit rates (configurable, but by default 10 kb/s), at fixed intervals, and at exponentially distributed intervals. Parameters used for these traffic sources in our tests are summarized in Table 4.3. Unlike the constant bit rate source type, the latter two UDP source types have no notion of multi-packet files or inter-connection times; all "files" consist of a single datagram whose size depends on file size distributions for each source type. As such, fixed interval and exponentially distributed interval sources do not require any control messages to coordinate when packets are sent. The same is not true for the constant bit rate sources, where we must ensure that file requests take place according to the file size and inter-connection request distributions. Each

4.3: Summary of HARPOON configuration parameters for UDP sources used in validation tests.

Source Type	**Key Parameter**
Constant Packet Rate	Rate = 10 kb/s
Fixed-interval Periodic Ping-Pong	Interval = 64 seconds
Exponentially Distributed Ping-Pong	λ = 30 milliseconds

client maintains a TCP control connection with the target server over the duration of an active UDP session. File requests are made over this channel and UDP data is then sent by the server. The client sends the port number of a locally bound UDP socket in order for the server to set the proper destination of UDP traffic. The client additionally sends a datagram size and rate for the server to use when sending data, and a unique request identifier. Once the server finishes sending the requested file, it sends a completion indication to the client for the original request identifier.

In addition to the distributional input parameters, each HARPOON client is given a range of local IPv4 addresses to use as a source address pool, and a range of IPv4 addresses and associated ports of target HARPOON servers. Address ranges are specified as CIDR prefixes to HARPOON. The source addresses may be bound to physical interfaces on the client host or to address aliases assigned to an interface. When starting a new user, a thread picks a source address and destination address from this pool. The address pools are constructed in such a way that random selection of addresses generates the required spatial frequency distribution.

HARPOON is designed in a modular fashion to enable relatively easy addition of new traffic models. Traffic generation modules for TCP, UDP constant bit rate stream, and the fixed and exponential interval UDP datagram sources are implemented as plugin modules that are dynamically loaded and controlled by a core HARPOON thread manager. HARPOON reads XML parameter files produced by the self-configuration components, loading and starting the necessary plugin modules.

Management of resources and tools within large-scale testbed environments can be challenging. To counter this problem we have implemented an HTTP server in HARPOON, along with an XML parser to enable remote management via XML-RPC [36]. Users can remotely reset, stop, and gather statistics from running HARPOON processes. Traffic module object files can be disseminated to remote HARPOON processes and configured by distributing XML parameter files from a central location.

Currently, a single HARPOON process can produce a few hundreds of megabits per second of network traffic using reasonably provisioned commodity workstations. Performance is largely dependent on the nature of the inter-connection time distribution (because of timer management) and the number of active sessions. The memory footprint of HARPOON (client or server) is relatively small, normally less than 10MB with modestly sized input distributions. The code is written in C++ and currently compiles and runs under FreeBSD, Linux, MacOS X, and Solaris. Porting to new platforms is limited only by support for POSIX threads, capability for dynamic loading of objects (*e.g.*, `dlopen()` and friends), the C++ standard template library, and the eXpat XML library[12].

4.4: Summary of Data Sets Used in Validation Experiments

Data Set	Total Flows	TCP Flows
Wisconsin 31 July–6 August 2002	423,836,790	241,484,962
Auckland 11–12 June 2001	12,912,462	12,144,434

4.4 Validation

In this section we validate the ability of HARPOON to generate traffic that qualitatively exhibits the same statistical signatures as the distributional models used as input.

We used two different data sets to self-parameterize HARPOON in our validation tests. Our first data set consisted of one week of NetFlow records collected between July 31, 2002 and August 6, 2002. The second data set was a series of packet traces from the University of Auckland, taken on June 11 and 12, 2001, from which we constructed flow records. We modified the `crl_flow` tool in the CoralReef software suite [7] to produce wire-format NetFlow version 5 records from the packet traces. We refer to these data sets as "Wisconsin" and "Auckland" below, and they are summarized in Table 4.4.

4.4.1 Tests

Our validation test environment consisted of two workstations running the FreeBSD 5.1 operating system. Each machine had 2 GHz Intel Pentium 4 processors, 1 Gigabyte of RAM, and used kernel defaults for TCP/IP parameters, which meant that the NewReno congestion control algorithm, time stamping, and window scaling TCP options were enabled, and the default receive window size was 64kB. The default Ethernet MTU of 1500 bytes was used. Each machine was dual-homed, with a separate Intel Pro/1000 interface used solely for experiments. The machines were each connected to a Cisco 6509 router via 1 Gb/s Ethernet.

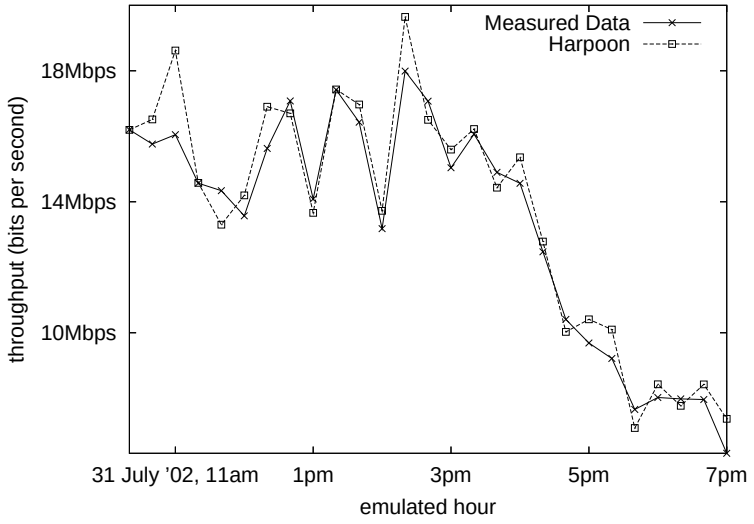
One FreeBSD machine was used as HARPOON data server and two were used as HARPOON clients for generating requests and receiving the resulting data packets. We monitored each host during our tests to ensure the systems did not exhaust all available CPU or memory resources.

The client machines were configured to generate requests using an IPv4 class C address space (2^8 addresses). Likewise, the server machine was configured to handle requests to an IPv4 class C address space. In each case, the address creation was accomplished by creating aliases of the loopback interface, and creating static routes at the HARPOON clients, server and intermediate switch. Aliases of the loopback adapter were created to avoid ARP overhead with the switch.

4.4.2 Results

Figures 4.5 and 4.8 examine HARPOON's capability to generate the desired temporal characteristics for the Wisconsin and Auckland data sets, respectively. Figure 4.5 compares the bit rate over a period of 8 hours from the

original Wisconsin data with HARPOON. Figure 4.8 compares the bit rate over a period of two days of the original Auckland data with the bit rate produced by HARPOON when emulating the same time period. The level of traffic is determined by the input distribution specifying the number of active users ($P_{ActiveSessions}$). In each test, the HARPOON traffic exhibits burstiness due to the distribution of file sizes, inter-connection times, and the closed-loop nature of TCP. Nonetheless, over the duration of each test the hourly pattern emerges due to control over the number of active sessions at the HARPOON client process. Some variability is introduced because HARPOON does not immediately truncate an active session at each emulation interval. Instead, it lets the final connection finish naturally, thus avoiding abrupt shifts between each interval.



4.5: Emulation of temporal volume characteristics for Wisconsin data.

Figure 4.6 compares the inter-connection time, file size, and destination frequency empirical distributions derived from the Wisconsin data set with the distributions generated by HARPOON. For inter-connection times, shown in Figure 4.6a, there is a good overall match except at the shortest inter-connection times. The visible "bumps" for the HARPOON tests at these inter-connection times are an effect of a coarse-grained operating system scheduler interacting with the shortest durations of this distribution. FreeBSD (as well as Linux), by default, uses time slices of 10 milliseconds and the steps in Figure 4.6a are at multiples of this time slice. In tests with operating systems that have different scheduling mechanisms (*e.g.*, MacOS X), these particular artifacts do not appear. Results for the Auckland data set are qualitatively similar.

Figure 4.6b compares file sizes extracted from the Wisconsin data set with the file sizes transferred by HARPOON. There is a close qualitative match, and results for the Auckland data set are similar.

Figure 4.6c plots frequency vs. rank on a log-log scale for destination addresses for the Wisconsin data set. We observe a close match between the original data and the HARPOON data. Results for source addresses are qualitatively similar, as are results for the Auckland data set.

Our final validation test was to compare the traffic volumes generated by HARPOON against the original flow traces used to generate parameters for each data set. In Figure 4.7 we compare the distributions of packets, bytes, and flows per measurement interval to those derived from the Wisconsin data set. We make the same comparisons for the Auckland data set in Figure 4.9. As shown in these plots, HARPOON accurately approximates the number of bytes and flows per interval. For each data set, there are more packets sent in the original trace than from HARPOON. The reason is that our testbed is homogeneous with respect to link-layer maximum transmission unit sizes, resulting in average packet sizes that are larger than the original trace. Figure 4.10a shows a time series of the mean packet size over 60 second intervals of a one hour segment (12pm-1pm, 11 June 2001) of the Auckland data set and a similar time series for HARPOON. HARPOON packet sizes average almost twice as large as the original trace. As shown in Figures 4.7b and 4.9b, when we scale the HARPOON volumes by the ratio of average packet sizes between HARPOON and the measured data, we observe a close match.

4.4.3 Limitations

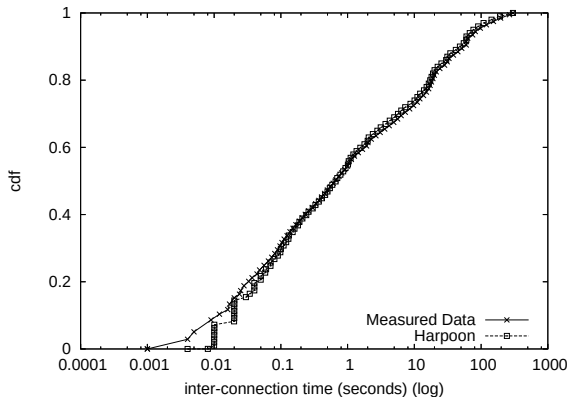
Two limitations to HARPOON's traffic model are:

1. it is designed to match byte, packet, and flow volumes over relatively coarse intervals (*e.g.*, 300 seconds) and may not match over shorter intervals;

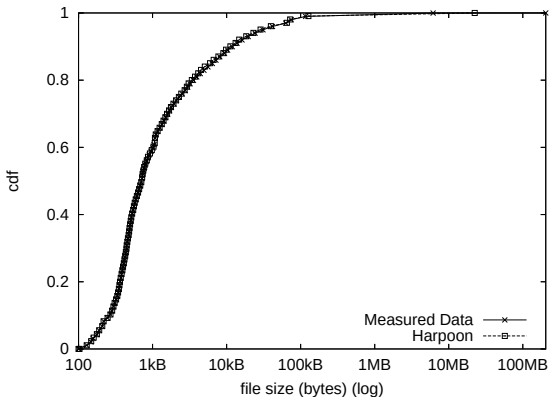
2. since packet-level dynamics are not specified, traffic produced by HARPOON may not match other metrics of interest, such as scaling characteristics, queue length distribution for the first-hop router, packet loss process, and flow durations.

Packet-level dynamics created by HARPOON arise from the file size distribution, the inter-connection time distribution, TCP implementations on end hosts, and testbed parameters such as round-trip time distribution, link capacities, and MTU sizes. Round-trip time is a key parameter because it affects flow durations for TCP sources. As a consequence, the nature of the ON/OFF process and therefore the correlation structure of packet arrivals over both short and long time scales are affected. It was shown in [100] that dynamics over short time scales differ between LAN and WAN environments, and in [99] that short time scale effects arise due to the TCP feedback loop and network environment variability characteristic of WANs. Effects across both short and long time scales are of interest in testbed traffic generation because performance measurements can differ substantially between LAN and WAN settings (*e.g.*, see [60]).

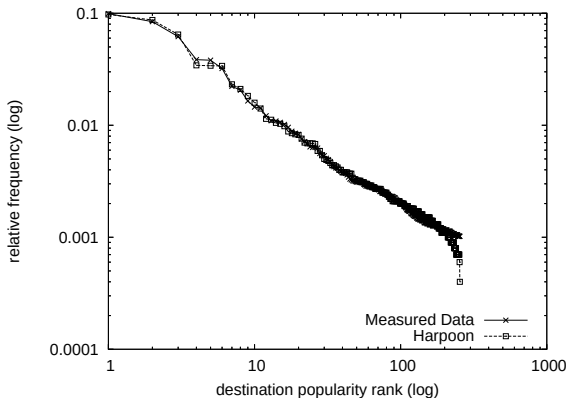
In Figure 4.10b we compare time series of bytes transferred over 1 second intervals between a segment of the Auckland trace and HARPOON using an $IntervalDuration$ of 300 seconds. There is no propagation delay emulated in



(a) Inter-connection time distribution.

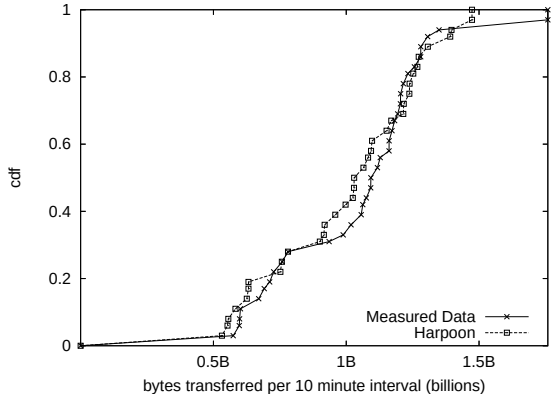


(b) File size distribution.

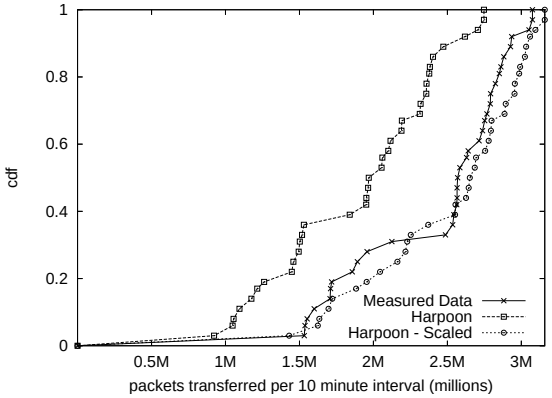


(c) Destination IP address frequency distribution.

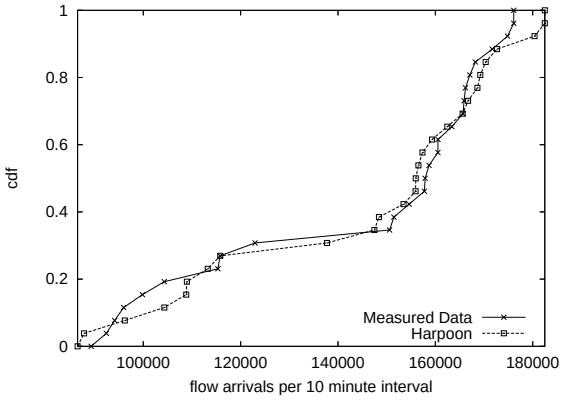
4.6: Comparison of empirical distributions extracted from Wisconsin data with distributions produced during HARPOON emulation. Results shown for one day of Wisconsin data (31 July 2002). Results for other days and for the Auckland data are qualitatively similar.



(a) Byte volumes.

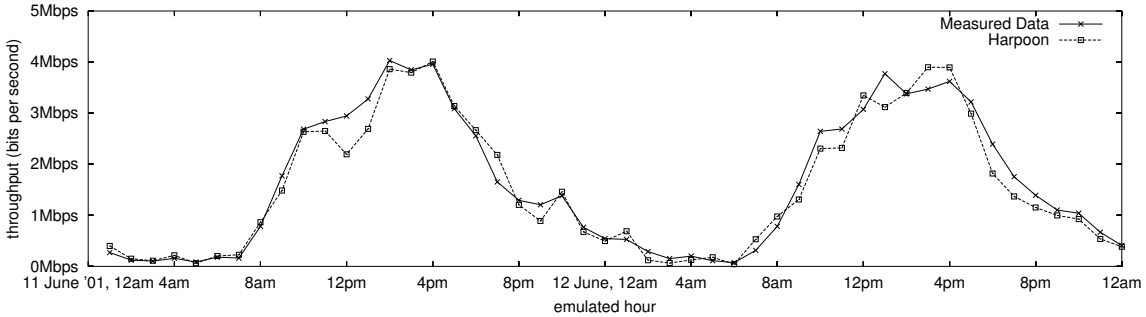


(b) Packet volumes. HARPOON data also shown scaled by ratio of average packet size between HARPOON and measured data.

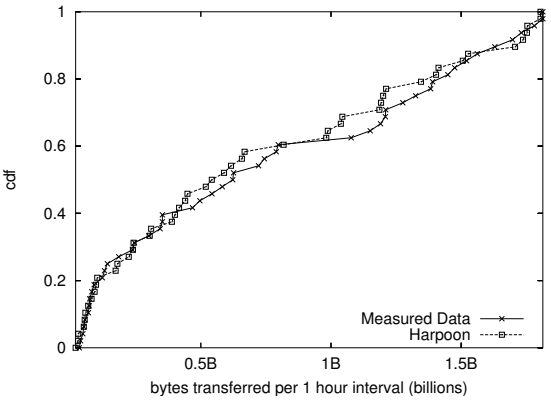


(c) Flow volumes.

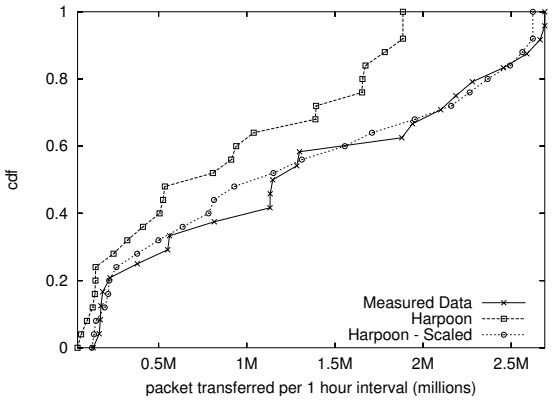
4.7: Comparison of byte, packet, and flow volumes extracted from of Wisconsin data with volumes produced during HARPOON emulation. Results shown for one day of Wisconsin data (31 July 2002). Results for other days are qualitatively similar.



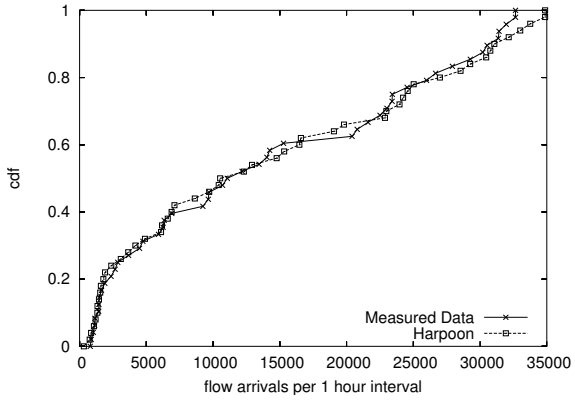
4.8: Emulation of temporal volume characteristics for Auckland data.



(a) Byte volumes.



(b) Packet volumes. HARPOON data also shown scaled by ratio of average packet size between HARPOON and measured data.



(c) Flow volumes.

4.9: Comparison of byte, packet, and flow volumes extracted from original Auckland data with volumes produced during HARPOON emulation.

the testbed, so the round-trip time for HARPOON sources is on the order of 1 millisecond. There is greater variability evident in the testbed configuration, in part, from a tight TCP feedback loop resulting in higher throughput and shorter flow durations. For the Auckland trace, dominant round-trip times of around 200 milliseconds (roughly the RTT to North America) lead to longer flow durations, a greater degree in flow concurrency, and less observable variability in utilization.

A standard method of characterizing behavior of traffic dynamics over a range of time scales is the log-scale diagram described in [41]. The log-scale diagram was developed as a means for identifying self-similar scaling in network traffic and for estimating the Hurst parameter. Evidence for self-similar scaling is associated with a range of scales where there exists a particular linear relationship between scale and the log of a normalized measure of energy (variability). A different scaling regime has been shown to exist in measured WAN traffic and is separated from the linear self-similar region by a pronounced dip in energy at the time scale associated with dominant round-trip times.

In Figure 4.10c we compare log-scale diagrams based on wavelet decompositions of the time series of byte volumes over 500 microsecond intervals for a one hour segment of the Auckland data set (12pm-1pm, 11 June 2001) and for HARPOON with two different values of *IntervalDuration* (60 or 300 seconds) and two emulated round-trip times (0 or 200 milliseconds). (We do not plot confidence intervals for clarity.) We first note the difference between HARPOON configurations using different round-trip times. Comparing HARPOON using an RTT of 200 milliseconds with the original trace, there is a common range of time scales indicative of self-similar scaling (scales 9-14) and a clear dip at the 256 millisecond time scale (scale 9) because of dominant round-trip times in the original trace, and the singular round-trip time for HARPOON. For the HARPOON configurations with no emulated propagation delay, there is more energy across almost all time scales and nothing in common with the original trace. Finally, we note that over sub-RTT time scales, HARPOON does not match the Auckland trace for any configuration. Since our testbed lacks diversity in round-trip times (as well as MTUs and link capacities), we do not expect a match over these time scales. Even though HARPOON accurately approximates the original volumes over 300 second intervals for both round-trip time values (not shown here), there are vast differences between the original correlation structure and that produced by HARPOON.

Comparing the two configurations of *IntervalDuration*, there is slightly less overall energy when using 60 second intervals because the byte volumes produced by HARPOON are less than the original volumes. The reason is that the interval is too small compared with the default maximum inter-connection time of 60 seconds: a session may randomly get an inter-connection time that causes it to be idle for the length of the entire interval. The end effect is that HARPOON produces less traffic than intended (see also Section 4.3).

These differences have important implications not only for HARPOON, but in configuring any laboratory testbed and in the interpretation of results. First, it is clear that queuing dynamics will be different depending (at least) on configured round-trip times, resulting in different delay, jitter, and loss characteristics than might have been measured in a live environment. Second, such differences will very likely affect transport and application performance, requiring careful interpretation of measurements, and characterization of their applicability outside the laboratory. While

HARPOON represents a step forward in testbed traffic generation, the goal of predicting performance in a live environment based on laboratory experiments remains somewhat out of reach, since specifying the ingredients necessary to produce realistic packet dynamics over all time scales in both simulation and laboratory testbeds is, in general, an open problem [165].

4.5 Comparison with Packet-Oriented Traffic Generators

The first and most obvious application for HARPOON is in scalable background traffic generation for testbed environments such as our laboratory calibration setting. In this section we investigate a related application of HARPOON in a router benchmarking environment. We compare router performance using workloads generated by HARPOON with workloads generated by a standard packet level traffic generator[5]. Our motivation for comparing HARPOON with a standard traffic generation system is to both demonstrate the differences between the two approaches, and to consider how the standard tools might be tuned to exercise routers more comprehensively.

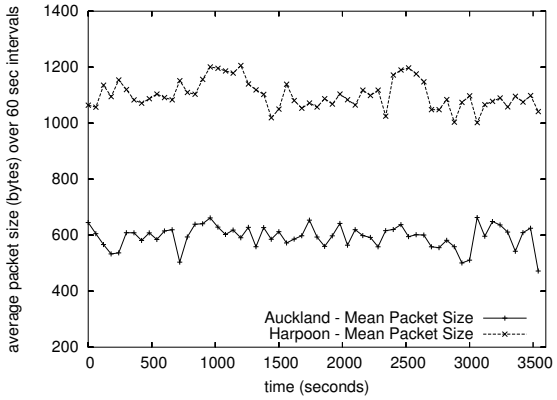
4.5.1 Environment and Methodology

For the hardware environment used in this series of experiments we used the two HARPOON source machines and one HARPOON sink machine configured in the same way as our validation tests described above. Each host was connected to a Cisco 6509 router[6] [5] via 1 Gb/s Ethernet links. Each link was connected to separate physical line cards on the 6509, forcing traffic to cross the internal switching fabric of the router. During tests, the 6509 was connected only to the traffic generation machines and to our management network, over which we polled specific MIB variables via SNMP. No other traffic traversed the router. The system we used to generate the standard packet traffic was a Spirent AX/4000. The AX/4000 generated traffic over two 1 Gb/s Ethernet links and the traffic returned to the AX/4000 over another 1 Gb/s link. While our tests contrast HARPOON with the AX/4000, the model of traffic generation in the AX/4000 is representative of many tools that generate packets with constant spacing. We therefore use the term "constant spacing packet generator" in the discussion and graphs of results.

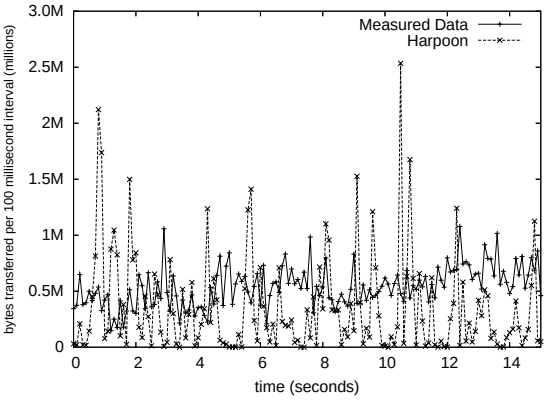
Our experiments were based on a series of tests described at `http://www.lightreading.com` [161] and on benchmarking terminology and methods described in RFCs 2285 [144] and 2889 [145]. These tests were designed to evaluate packet forwarding performance of Internet backbone routers under a variety of conditions, including the use of minimum-sized packets, and forwarding tables comparable in size to those found in the Internet backbone. The LightReading tests also investigated many other performance capabilities of routers, such as BGP table capacity, packet filtering, and forwarding performance under unstable path conditions. Our tests focused on packet forwarding performance and the impact of workloads on router subsystems.

[5] Standard packet generation tool range from free software tools such as `iperf` [198] to large-scale, feature-rich hardware platforms such as the Spirent AX/4000 [25].

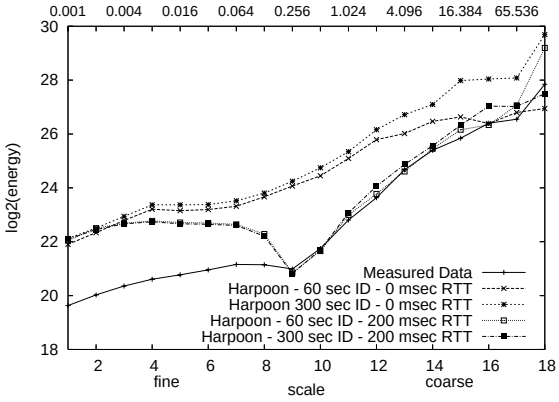
[6] The Cisco 6500 series of devices can be configured either to run Cisco CatOS, Cisco IOS, or a hybrid of the two. The choice depends on anticipated traffic, customer requirements, and modules installed. We configured our 6509 to run native IOS, version 12.1(13)E6.



(a) Average packet sizes for one hour segment of Auckland data and equivalent HARPOON trace. For the Auckland data, the overall sample mean and standard deviation are 610 and 629, respectively. For HARPOON, the overall sample mean and standard deviation are 1106 and 615, respectively.



(b) Time series of bytes transferred for a 15 second segment of Auckland trace and for HARPOON (no RTT configured in testbed).



(c) Log-scale diagrams for one hour segment of Auckland data set (12pm-1pm, 11 June 2001) and HARPOON with two different *IntervalDuration* (ID) values and two different round-trip times.

4.10: Limitations of HARPOON.

Our basic metric for comparison was packet forwarding rate. As RFC 2285 points out, it is problematic to report forwarding rates in the *absence* of packet losses as the earlier RFC 1242 [69] had defined. Obviously, packet loss rates depend on the offered load and router or switch implementation artifacts. Our challenge was to determine how to match offered loads between two fundamentally different packet generation methods. Instead of attempting to compare forwarding rates across a broad range of offered loads, we chose two different regimes, a "low" load and a "high" load. With each offered load, we polled the 6509 every 20 seconds using SNMP to obtain the forwarding rate, packet loss counts and other system utilization variables available via SNMP on each interface. SNMP variables are updated internally every 10 seconds in the 6509. Our polling period of 20 seconds was chosen to minimize additional load placed on the 6509 while ensuring each SNMP request would obtain updated counters. The low and high offered load regimes were roughly tuned to 60% ($\approx$ 600 Mb/s) of egress bandwidth and 90% ($\approx$ 900 Mb/s) of egress bandwidth, respectively[7]. For HARPOON, we used the $P_{FileSize}$ and $P_{InterConnection}$ distributions extracted from the Wisconsin data and used the algorithm of Figure 4.1 to determine how many session threads should be active to generate the intended load levels on the 6509. We then calculated the average offered load from SNMP measurements of the ingress interfaces and used this average to configure the packet-level generator. Because of this tuning approach, including other application-level generators in this evaluation would have required imprecise trial-and-error tests to tune them to the same level. Thus, we chose only to compare HARPOON with the constant load generator. For all tests described in this section, HARPOON was initialized with the same random number generator seed to ensure identical sequences in input distributions.

We compared forwarding rates in a series of tests run for 10 minutes each using the two different offered loads while separately changing traffic source burstiness, traffic source packet size, and forwarding table size. Note that changes in traffic source burstiness and packet size are only applicable to the constant spacing generator; HARPOON (intentionally) does not dictate these parameters[8]. Packet-level burstiness in HARPOON arises from the distributional parameters configured in HARPOON and the closed-loop dynamics of TCP.

For the constant spacing generator, we used burst sizes of 1 (uniformly spaced packets), 232, 465, 697, and 930. The burst size is defined by RFC 2285 [144] as a sequence of frames emitted with the minimum legal inter-frame gap, and a maximum recommended burst size is given as 930. The range of burst sizes we used covers $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ of the maximum burst size.

The packet sizes (number of bytes excluding link-layer framing) used in the constant spacing generator were 40, 1500, and a trimodal distribution configured so that 61% of the generated packets were 40 bytes, 17% were 576 bytes, and 22% were 1500 bytes. The empirical basis for this mix of packet sizes roughly approximates the values used in

[7] With a deterministic packet generation tool like the AX/4000, tuning the offered load is straightforward. With HARPOON, the load is tuned by adjusting the number of client and server threads.

[8] It is possible to change the MTU at each interface (physical or alias) used by HARPOON, indirectly allowing HARPOON to generate different packet sizes. We did not modify the default Ethernet MTU of 1500 bytes on the FreeBSD hosts running HARPOON.

the LightReading tests, and comes from measurements taken at Merit Networks Inc. between 28 August, 2000 and 13 September, 2000 [17].

The sizes of forwarding tables used were 32, 1,024, 32,768, or 65,536 entries. The two traffic sources (two server workstations in HARPOON configuration, two separate GE cards in the AX/4000) were each configured with a single IP address. The traffic sink (one client workstation in HARPOON, a single GE card in the AX/4000) was configured to use an entire IPv4 class B address range (2^{16} addresses). With the AX/4000, this is simply a parameter in configuring the generated packets. With HARPOON, we created 2^{16} interface aliases. This address space was reached by installing static routes on the traffic source workstations and the forwarding tables on the 6509. We aliased the loopback adapter rather than the physical Ethernet adapter to avoid ARP overhead between the client host and the 6509, and to force a destination address lookup in the forwarding tables of the 6509.

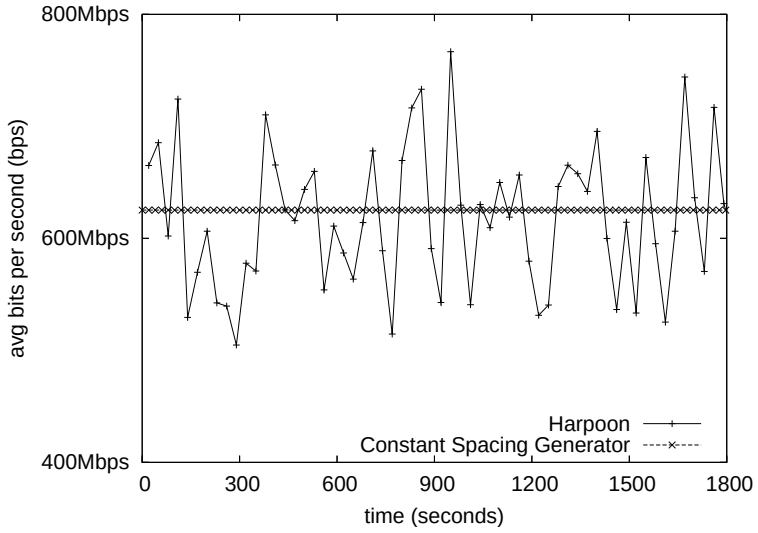
4.5.2 Results

We begin by showing a qualitative comparison of the time series of bit forwarding rates for HARPOON and the constant spacing generator shown in Figure 4.11. In this test, HARPOON was configured to generate roughly 600 Mb/s of traffic over a 30 minute test period. The constant spacing generator was configured to match the average offered load of HARPOON and emitted 1500 byte packets with uniform spacing. HARPOON maintains a high level of offered load in the presence of variable file sizes transferred and inter-connection times while still generating burstiness in packet traffic. As expected, the constant spacing generator presents a virtually unwavering load to the router. In further tests, we explore how these fundamental differences in packet emission processes affect the performance of the router.

We examine the effects of burst sizes of 1, 465, 697, and 930 on forwarding rate in the results shown in Figure 4.12. The vertical bars on each data point indicate the range of one standard deviation above and below the average. The router was configured with a forwarding table size of 2^{15} and the high regime of offered load was used for these tests. HARPOON results for a forwarding table size of 2^{15} and high offered load are shown for comparison. The drop in bit forwarding rate for the constant spacing generator at a burst size of 697 accompanies a corresponding increase in the number of packets dropped at the egress line card of the router. The drop in forwarding rate between a burst size of 465 and a burst size of 930 is roughly 180 Mb/s. This increase in stress placed on the router does not come with an increase in forwarding rate variability, as noted by the low variation in the height of vertical bars drawn for the constant spacing generator tests.

Figure 4.13 shows the effects of different packet sizes on forwarding performance. The constant spacing generator was configured to emit packet sizes of 40, 1500, and a trimodal distribution as described above. As with the burst tests above, the router was configured with a forwarding table size of 2^{15} and the high regime of offered load was used for these tests. HARPOON results for a forwarding table size of 2^{15} and high offered load are plotted for comparison. With minimum sized IPv4 packets, the forwarding rate of the test router under the same offered load drops by roughly 140

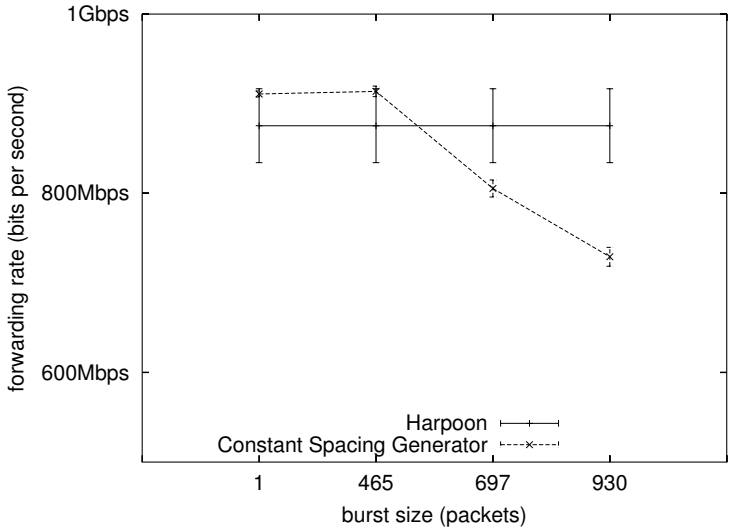
Mb/s while exhibiting very low variability. Packet loss was measured for the 40 byte case, but there was no packet loss measured for the 1500 byte or trimodal case.



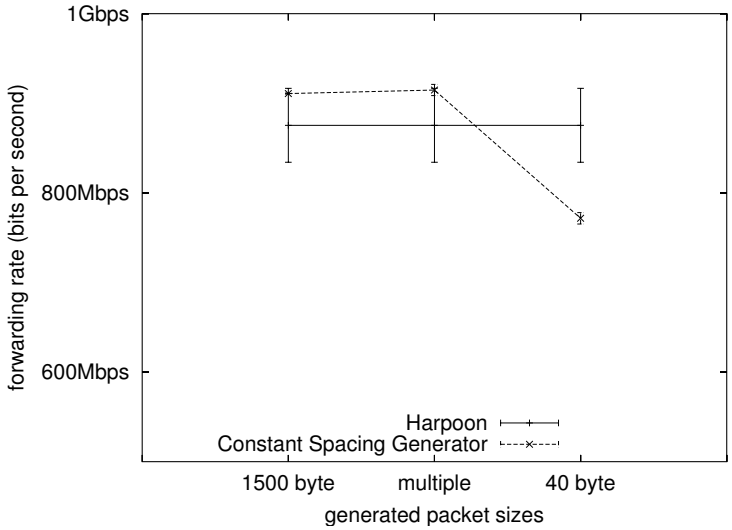
4.11: Packet forwarding rate time series for HARPOON and the constant spacing packet generator.

Figure 4.14 shows the affect of different forwarding table sizes on forwarding rates. The results from the low offered load test are shown in Figure 4.14a and results from high offered load test are shown in Figure 4.14b. The obvious features in the figures are the near-perfect matching of average bit forwarding rates between HARPOON and the constant packet spacing generator at low loads, and the poor match for the test using a high offered load. Recall that the offered load used in the constant packet spacing tool was configured using the average offered load generated by HARPOON. For the high load test shown in Figure 4.14b, the difference in forwarding rate comes as a direct result of packet loss that occurred during the HARPOON tests, but not during the tests using the constant spacing generator. Though the same average load is offered to the router by each traffic generator, the forwarding rate is clearly lower for the HARPOON test. Another important feature of these graphs is the trend in variance for the HARPOON tests. As the forwarding table size increases, so does the variance in bit forwarding rate. For the constant packet spacing generator, variance remains very low for all tests. Since HARPOON was configured so that each test used the same sequence of input parameters, we conclude that the burstiness in the traffic presented by HARPOON results in distinctly different and potentially highly variable forwarding performance at the router.

Figure 4.15a shows time series of packet loss for selected tests described above. All tests shown used a 2^{15}-entry forwarding table at the router and high offered load. It is clear from Figure 4.15a that even when the constant spacing generator emits bursts of packets, the loss rate measured at the router remains basically constant. The same is true for the constant spacing generator test with 40 byte packets. This observation is not surprising considering the packet-level generator operates in an open-loop manner at the transport level, making no adjustment in offered load in response to



4.12: Average forwarding rates using different burst sizes for the constant spacing traffic generator. Configuration is 2^{15} entries in forwarding table and high offered load. HARPOON results for 2^{15} forwarding table entries and high offered load shown for comparison. Vertical bars span one standard deviation above and below the mean.



4.13: Average forwarding rates using different packet sizes for constant spacing packet generator. Configuration is 2^{15} entries in forwarding table and high offered load. HARPOON results for 2^{15} forwarding table entries and high offered load shown for comparison. Vertical bars span one standard deviation above and below the mean.

packet loss. Because of the closed-loop nature of TCP, HARPOON is likely to generate some packet loss with *almost any* configured offered load, and it is likely that these losses will be correlated and bursty [214]. Finally, we note that all packet losses measured during our tests occurred as output buffer drops on the line card connected to the HARPOON client machine or the module on the packet-level generator acting as the data sink.

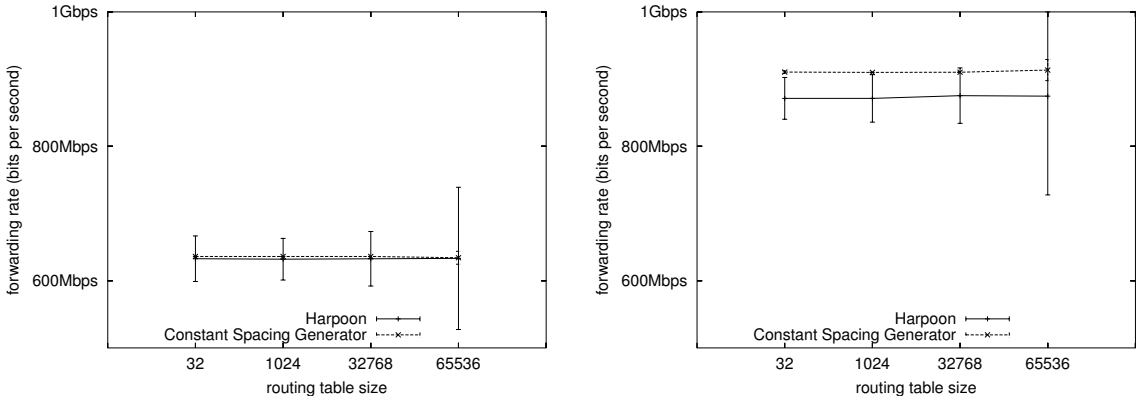
The effect of four test configurations on switch fabric utilization is shown in Figure 4.15b[9]. Surprisingly, the results for the constant spacing generator using uniformly spaced 1500 byte packets exhibits a wider range of fabric utilization levels than for the same generator using uniformly spaced 40 byte packets. The fluctuation in utilization measured during the 1500 byte test is likely due to limited precision of the utilization calculation internal to the 6509 (it is simply an integer). The highest level of switch fabric utilization is measured during the constant packet spacing generator test using 40 byte packets. It is interesting to note that despite sending much larger packets on average and despite a much lower packet loss rate (see Figure 4.15a), utilization during the HARPOON test comes within two percent of the highest measured fabric utilization. Fabric utilization during the HARPOON test also exhibits much greater variability than any of the constant spacing generator tests.

While not shown here, we ran experiments using the constant spacing generator and a limited cross-product of parameters explored above (burst size, packet size, forwarding table size, and offered load). As one might expect, loss rates when sending 40 byte packets in bursts are greater than when sending uniformly spaced 40 byte packets. The key observation from all these tests is that variability in bit forwarding rate, loss rate, and switch fabric utilization remains very low for the constant spacing generator.

4.5.3 Implications

It is clear that precisely controlled traffic streams are useful for Internet RFC conformance testing and for subjecting network systems to extreme conditions along certain dimensions. However, our experiments demonstrate that a workload based on measured characteristics of real Internet traffic generates a fundamentally different and more variable load on routers. Our results suggest ranges of behaviors that can be expected for given average loads. These ranges could be used to tune constant bit rate streams to explore an appropriate operational space. Finally, the subsystem load variability imposed by HARPOON should provide insights to system designers on the stresses that these systems might be subjected to under real operating conditions. This could inform the allocation of resources in future system designs.

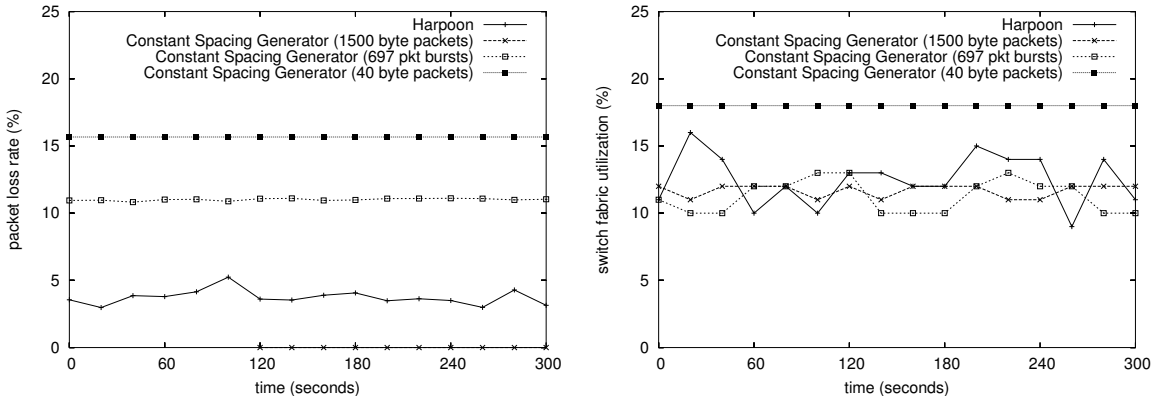
[9] Switch fabric utilization on the 6509 is reported for both input and output channels to each module installed in the chassis. We show results for the output channel on the module where traffic is multiplexed on the path to the HARPOON client machine or packet-level generator data sink module.



(a) Low Offered Load

(b) High Offered Load

4.14: Average forwarding rates for HARPOON and the constant spacing traffic generator for different router forwarding table sizes. Vertical bars span one standard deviation above and below the mean.



(a) Packet Loss Rate

(b) Output Channel Fabric Utilization

4.15: Packet loss rates and switch fabric utilization using 2^{15} forwarding table entries and high offered load. Results shown for HARPOON, constant spacing generator with uniformly spaced packets of 1500 bytes, bursts of 697 packets of 1500 bytes, and uniformly spaced packets of 40 bytes.

Chapter 5

Path Emulation

> There is no more familiar dictum than that experimental results must be repeatable. On my view that works out as something of a tautology. Experiment is the creation of a phenomena; phenomena must have discernible regularities—so an experiment that is not repeatable has failed to create a phenomenon.
> —Ian Hacking [114].

5.1 Overview

There are two main challenges to address for creating realistic wide area network conditions in a laboratory calibration environment. In the previous chapter, we described and evaluated the HARPOON system designed to overcome the first obstacle, that of creating a range of realistic traffic conditions similar to those that might be experienced in different locations of today's Internet. We now turn to the second challenge: to accurately emulate wide-area link and path characteristics. We define *link emulation* as the task of recreating the conditions experienced by packets on a single link physically (or virtually) connecting two nodes in a network. As such, link emulation is primarily focused on the *impact* of layers 1 and 2 in the network protocol stack. In a lab environment this means emulation of two properties:

Propagation delay the speed of light delay for each bit due to the physical distance between two nodes, is modeled as a constant delay per bit.

Bit errors are caused by noise and other factors along the physical path between two nodes and are modeled as a probabilistic change of state per bit.

Link emulation capability is of particular importance in experiments with topological configurations that are meant to recreate a specific wide area network. In these cases, some nodes are configured to act as routers and others as end hosts, so the lab infrastructure must be able to emulate the desired link characteristics between these nodes.

In experimental configurations that do not attempt to instantiate specific topologies, it is often desirable to emulate the network path characteristics between hosts. We define *path emulation* as the task of recreating the physical conditions experienced by packets on an end-to-end path between hosts in a network. Path emulation is a superset of link emulation and includes the following properties:

Packet latency is the aggregate of signaling delay (caused by the network interface cards), the propagation delay (caused by the physical distance between two hosts) and the queuing delays (caused by routers along the path). The sum of these delays is modeled as a probabilistic delay per packet.

Packet loss is the drop of a packet due to a full bottleneck queue on the path between two hosts, and is modeled as a probabilistic event.

Packet duplication is modeled as a probabilistic cloning of a packet on the path between two hosts.

Packet reordering shuffles packets from the order in which they arrive on a link and is modeled as a probabilistic event.

In this chapter we describe NETPATH, a network path emulation system designed to provide performance approaching expensive hardware-based solutions like the Adtech SX-14, and well beyond the capabilities of other software-based systems like Dummynet, NIST net, and Modelnet [20, 181, 203]. NETPATH offers the ability to add fixed and probabilistic packet delay, probabilistic packet loss, probabilistic packet duplication and packet reordering to links in a network. NETPATH's implementation is based on the Click modular router platform [133]. Click is an extensible, high performance packet processing system that runs on commodity hardware. Click offers several of the basic capabilities required for path emulation, making it a natural platform for our work. The primary challenge in NETPATH's development was to create a reliable, high capacity queuing system that significantly extends the basic packet buffering capabilities in Click. We extended Click to seamlessly and efficiently utilize both RAM and disk resources on the NETPATH host system. We also briefly describe VICT (virtual interposition configuration tool), a configuration tool we developed to allow NETPATH to easily be used in a variety of testbed setups.

The objectives of our evaluation of NETPATH are to demonstrate its raw performance capabilities from the perspectives of precision, throughput, and scalability. We also compare these capabilities with Dummynet, NIST Net and ModelNet [203]. Our results show that our new queuing module yields high throughput rates and supports over three orders of magnitude longer emulated delays without packet loss versus the standard Click queue. We also show that under a moderate traffic load NETPATH's delay emulation precision is consistently within 1% of a hardware-based reference emulator. Our comparative analysis shows that NETPATH offers over three times the loss free throughput capacity and orders of magnitude better delay precision than other software-based reference emulators. We demonstrate how NETPATH improves application traffic behavior over other software-based emulators, relative to our hardware-based reference. Finally, we investigate the scalability of NETPATH under multiple link configurations.

5.2 Architecture and Implementation

In this section, we describe the design objectives and general architecture of NETPATH. We then describe implementation details of NETPATH, including specific challenges faced in its development. We also briefly describe the configuration tool VICT.

5.2.1 Architecture

Network laboratory testbeds can contain thousands of individual links between network interfaces on routers, switches, and end hosts. The design of NETPATH is framed in the context of balancing the desire of testbed operators to use a limited number of commodity host systems for path emulation and the need of researchers to accurately create a range of network path conditions.

The design objectives for NETPATH were to create a system that could (1) offer high precision link and path emulation capability, (2) operate effectively on network links beyond 100 Mb/s, and (3) operate effectively on multiple network links at the same time. Practical considerations related to the use of NETPATH in lab environments led to extending the design objectives to include the ability to easily configure VLANs and/or MPLS paths to deflect packets on specific links through a node running NETPATH.

5.2.2 NETPATH Implementation

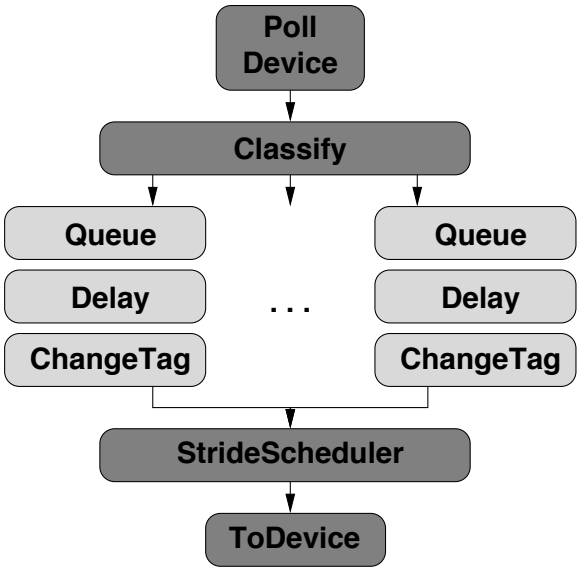
Path emulation as defined above implies the need for a packet processing system as the basis for our implementation. For this functionality we chose the Click modular router[1]—an open source platform for building packet switching system that runs on commodity hardware [133]. Click offers three key features that make it a natural choice for NETPATH. First, Click has high performance packet switching capabilities due to its kernel-based implementation and the ability to operate in polling (instead of interrupt) mode with respect to the network interface card (NIC). Second, Click is designed for extensibility. Click-based systems consist of primitive elements that are arranged to perform some desired function. The Click distribution provides a library of elements that can be extended for new functionality. Third, Click is designed to be easily configured through a declarative language that defines inter-element connectivity. The configuration language also supports higher level abstractions through definitions of compound elements.

5.2.2.1 Basic Link Emulation

Basic link emulation consists of the following: read packets from the NIC, delay them in a queue for a specified fixed interval and then send them out on a network interface ensuring that they are headed for the proper destination. These functions require the selection of appropriate Click elements, which are illustrated in Figure 5.1. For good performance, the first element in our system is `PollDevice` which polls a specified NIC and reads and timestamps

[1] We developed NETPATH using the v1.3pre1 distribution of Click.

packets when they arrive. Deferring discussion of the `Classify` and `StrideScheduler` elements for now, the next step is to delay packets. This function is implemented through the `Queue` and `Delay` elements. `Queue`, a simple FIFO, is required since the interarrival time of packets can be shorter than the specified delay interval. `Delay` is a single-packet buffer which, if empty, pulls packets from the head of `Queue`, computes the elapsed time since the packet's arrival, then holds it according to the desired delay setting. Our definition of link emulation includes the ability to cause random bit errors in packets. If enabled by the user, this function is included in the `Delay` element by toggling the first bit in the packet payload based on sampling a precomputed table of uniformly generated random numbers. The final step in basic link emulation is to place the packet on a network interface with the proper addressing information. The `ChangeTag` element takes a packet from `Delay`, encapsulates it with the appropriate addressing information, then sends it to the final element. `ToDevice` accepts packets from `ChangeTag` and is responsible for sending them to the NIC.



5.1: Click elements used in basic and multiple link emulation. Lighter elements were modified and extended for NETPATH.

5.2.2.2 Emulation of Multiple Links

In addition to the elements related to basic link emulation, Figure 5.1 includes the `Classify` and `StrideScheduler` elements. The design requirements for flexible and efficient use of host systems implies the ability to assign different delays to packets based on their source and destination addresses and port information. `Classify` pulls packets from `PollDevice` and, based on addressing information, passes them to the appropriate queue. The `StrideScheduler` multiplexes packets from the different link emulation paths and sends them on to the `ToDevice` element. The Click

distribution includes all of these basic elements except for `ChangeTag`, and `Delay` only required a minor modification to enable probabilistic bit errors.

The `ChangeTag` element insures that outgoing packets from the emulator are correctly forwarded. Our primary consideration in developing `ChangeTag` was for NETPATH to be easily used in any one of two physical configurations: *direct interposition* between hosts (*i.e.*, through the use of two network interface cards on the NETPATH host and no other hub, switch or router) or *virtual interposition* between hosts in both switched and routed environments. We developed the `ChangeTag` element to operate at the link layer of the protocol stack, and to facilitate packet routing through NETPATH via Virtual Local Area Networks (VLANs), in the case of switched links, or via Multiprotocol Label Switching (MPLS), in the case of routed links. While configuration of the required VLANs and/or MPLS routes in the switches or routers can be done by hand, we developed VICT to facilitate this task.

5.2.2.3 High Performance through Enhanced Buffers

While the Click platform enabled us to bootstrap our development effort, the performance of the basic elements fell short of our design objectives. As we will see in the next section, when larger delays are specified, the basic `Queue` element quickly overflows and packets are lost. Eliminating bottlenecks (and by extension unwanted packet loss), while necessary for maintaining precise control over packets passing through NETPATH, was one of the primary challenges of this project. In comparing NETPATH to other emulators, we found that they are also vulnerable to unwanted loss. Thus, we use loss as one of our metrics for comparison in our evaluation.

Our initial step in addressing unwanted loss was to modify the basic `Queue` element. `Queue` is implemented as a queue of packet pointers and is restricted to a maximum of 32K packets. Our implementation consists of an array of circular queues in which each queue has a maximum size of 32K (due to kernel restrictions on memory allocation). With this implementation, users can specify the desired quantity and size of each queue up to the total amount of available RAM on the host. While this modification has an important positive impact on performance, the overall scalability of the system is still limited by available RAM.

For greater scalability we created a seamless buffering mechanism that couples `Queue` to storage allocated on disk. We created the `DiskQueue` element to allocate and manage auxiliary queues on disk. When a queue in RAM fills, `DiskQueue` enables subsequent packets to be written to an associated queue on disk. As space is recovered in the RAM queue, `DiskQueue` shifts packets from the auxiliary queue to the RAM queue. Our challenges in developing this element were in keeping track of the boundaries of packets written to disk and in efficiently interacting with the virtual memory system of the host. The tasks of writing to and reading from the auxiliary queues are managed by a single thread which uses configurable thresholds to determine the quantity of data to transfer between RAM and disk. When these tasks execute, other tasks may be preempted which, depending on the traffic conditions, can lead to packet loss. Thus, the thresholds must be selected based on the NETPATH configuration, host capabilities and expected traffic load.

5.2.2.4 Queue Management on Two Disks

While the ensemble of `DiskQueue` and `Queue` enhances the scalability of NETPATH, random accesses to disk can cause performance degradation. To address this problem, we enhanced `DiskQueue` to take advantage of a host system with two disk drives. Our approach is to begin writing on disk *A* until a read request arrives. Subsequent write requests are then directed to disk *B* until reads have exhausted all of the data that had been previously written to disk *A*. Reading then switches to disk *B* and writing switches back to disk *A*. We call this *opportunistic disk queue management*. This simple algorithm reduces the amount of seek time for disk heads and we show in § 5.3.2 that it increases performance. The caveat for this algorithm is that it will not necessarily improve performance when multiple disk queues are in use. The reason is that the physical queues on disk may not be contiguous, resulting in non-local seeks and thus reducing performance.

5.2.2.5 Other Path Emulation Features

NETPATH's configuration of standard Click elements plus our modified queuing capabilities implement the basic link emulation functions scalably and with good performance. However, further additions were required to meet our objectives for path emulation. Specifically, in addition to applying fixed constant delays to packets we wanted the ability to probabilistically delay, drop, duplicate and reorder packets. Dropping and duplication is a simple matter of making a binary decision based on sampling a precomputed table of uniformly distributed random numbers (distributions other than uniform could be trivially added to NETPATH). Probabilistic delay values are assigned from a precomputed table of normally distributed random numbers (with user specified mean and standard deviation). A subtle problem in our implementation causes head-of-the-line blocking to occur in the queues for some packet arrival processes. This can skew the shape of the resulting delay distribution. As future work, we are investigating different methods for addressing this problem. Probabilistic reordering of packets also depends on the packet arrival process and requires us to create a customized combination of the `Queue` and `Delay` elements. This combined element only makes a reordering decision when multiple packets are enqueued. If a reordering decision is made (based on sampling the uniform random number table), then the second packet in the queue is serviced before the first. More complex reorderings are certainly possible but are beyond the scope of this work.

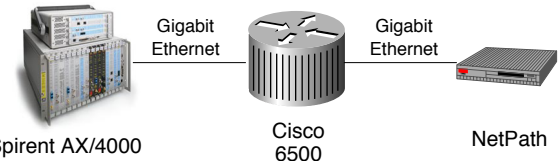
5.2.2.6 VICT implementation

NETPATH can be used for path emulation on multiple links. To configure such setups in both routed and switched environments, we developed VICT. To implement NETPATH's virtual interposition, we employ VLAN tagging in switched environments and MPLS tagging in routed environments. VLAN tags are 12-bit fields identifying virtual LAN membership information. Similarly, MPLS tags are 20-bit virtual circuit identifiers. VLAN and MPLS tags, along with MAC addresses, are the primary mechanism for directing packets between other lab systems and NETPATH.

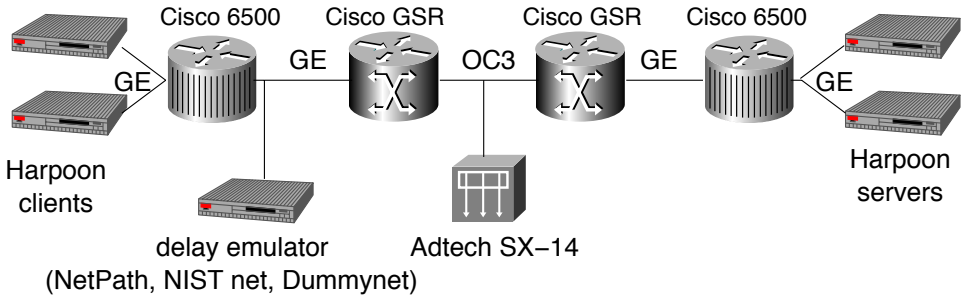
Obviously, VICT and NETPATH will not be able to be used on multiple links in environments that do not support VLANs or MPLS. However, both of these protocols are commonly available in network systems today so our tools should be widely applicable. Our VICT implementation is about 1,000 lines of Perl, and currently supports Cisco's IOS configuration language. As future work we intend to modify and expand VICT to support configuration languages of other popular network hardware vendors.

5.3 Performance Evaluation

Our objectives in evaluating NETPATH were to assess its raw performance characteristics and to compare its performance with other similar tools. We verified the probabilistic bit errors, loss, duplication and reordering capabilities of NETPATH for correctness but do not present results on their performance other than to say that the functions work as described above. The focus of our evaluation efforts was on the throughput, scalability and precision properties of NETPATH. As we previous discussed, one of the primary limitations of path or network emulation systems is their ability to handle high traffic loads without dropping packets. Thus, instead of simply considering raw performance as a function of load, we consider *loss-free* performance as a function of load as one of our primary evaluation metrics.

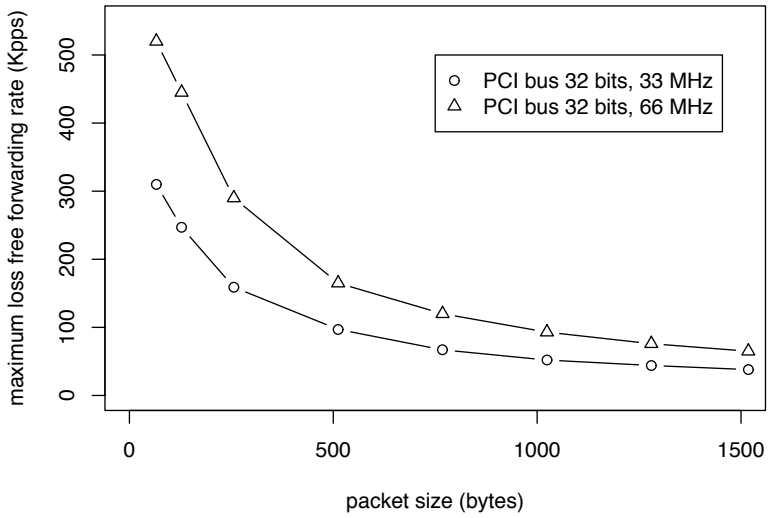


(a) In our basic setup, constant rate UDP streams produced by the AX/4000 were redirected to NETPATH via a Cisco 6500 switch. After NETPATH processing, packets returned to the AX/4000, where propagation delays and loss rates were measured.

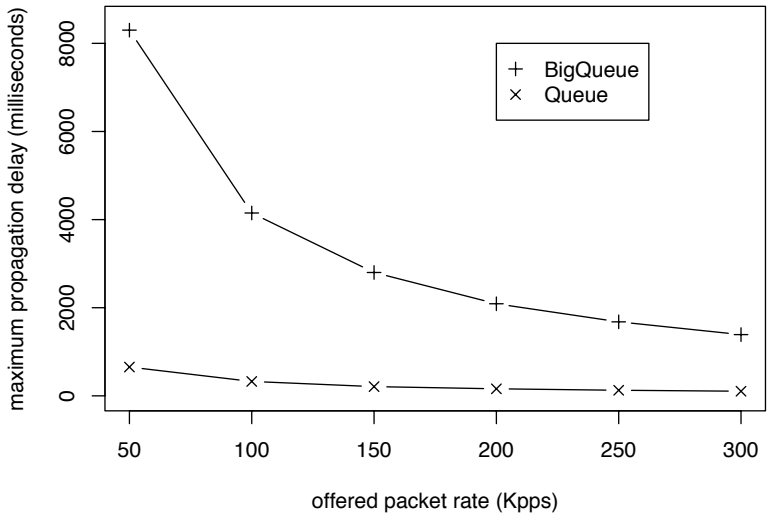


(b) An extended setup was used for evaluating application traffic performance under different path emulation systems. We generated self-similar web-like TCP flows using the HARPOON traffic generator across a dumbbell-like topology.

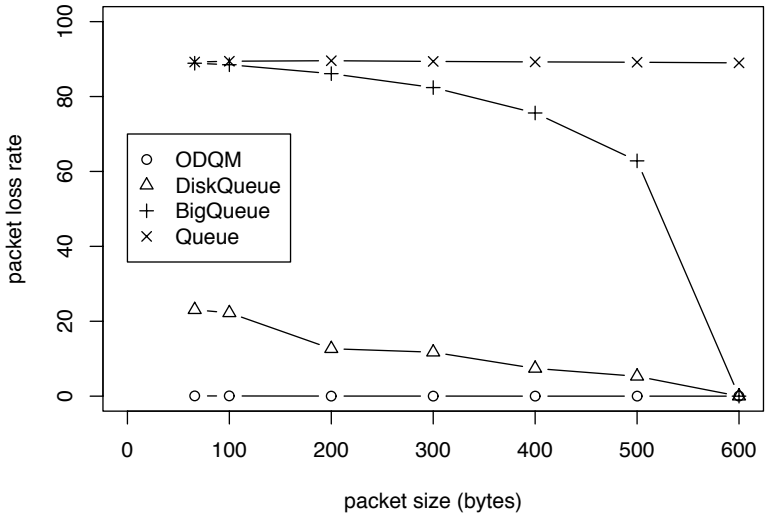
5.2: Experimental testbeds used to evaluate NETPATH.



5.3: NETPATH's throughput as a function of packet size for two different host PCI bus configurations.



5.4: Maximum propagation delays possible with the standard Click `Queue` element compared with the enhanced BigQueue element.



5.5: Packet loss rates for four different queue implementations. Offered loads are fixed at 100 Mb/s with varying packet sizes. NETPATH is configured to limit link capacity to 10 Mb/s.

5.3.1 Experimental Testbeds

Our basic experimental testbed, shown in Figure 5.2a consisted of a commodity workstation, Cisco 6500 switch, and a Spirent AX/4000 traffic generator [25]. The workstation contained a 2 GHz Intel Pentium 4 with 1 GB main memory and a 32 bit, 33 MHz PCI Bus, two Intel/Pro 1000 Gigabit Ethernet adapters, and ran Linux (2.4.18 kernel)[2]. The host was also configured with two disk drives, both 30 GB IDE with DMA/ATA-133 interfaces. The Spirent AX/4000 is a high performance network test appliance with precise traffic stream generation and measurement capability. Our system was configured with OC-3 (149.76 Mb/s) and Gigabit Ethernet interfaces. Unless otherwise noted, the experiments below used the Gigabit Ethernet interfaces. All ports are synchronized to one clock. Thus, using the AX/4000 as both a traffic source and sink we were able to measure one way propagation delays with precision on the order of single microseconds. Unless otherwise noted, the traffic for each test was a constant rate stream of 66 byte UDP packets. Our extended testbed, shown in Figure 5.2b, was used for evaluating application traffic performance. It consisted of a dumbbell-like topology with a bottleneck OC-3 link between two Cisco GSRs, with Cisco 6500s used to aggregate end host traffic and divert packets to the path emulation host. The testing protocol for our experiments consisted of generating traffic streams and taking measurements over a period of 10 minutes.

5.3.2 NETPATH Raw Performance Characteristic

We first consider the performance of NETPATH by evaluating its throughput in terms of maximum loss-free packet forwarding rate (MLFFR). The MLFFR is the maximum rate at which no loss is measured in the received packet stream for the entire duration of the experiment. The focus of this experiment is on measuring the MLFFR of NETPATH for packets of various lengths. As discussed in [133] throughput in Click can be limited by various hardware components such as CPU, PCI bus or the network adapter. To consider this issue, we evaluated throughput performance on two different workstations which differed only in their PCI bus configuration (both 32 bits wide, running at 33 or 66 MHz). As shown in Figure 5.3, NETPATH can sustain a MLFFR of 310K minimum sized packets per second in the 33 MHz configuration and at least a 50% higher rate on the 66 MHz PCI system. This suggests that the PCI bus bandwidth plays an important role in limiting throughput[3].

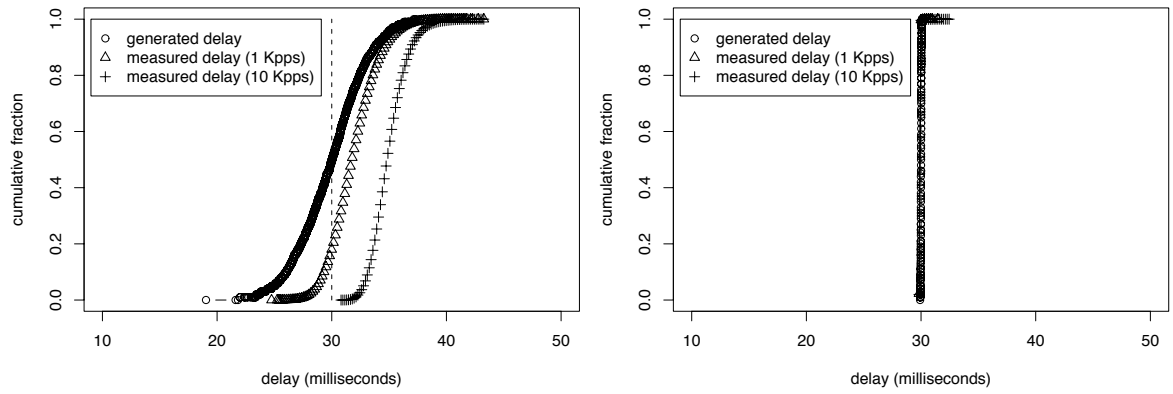
Next, we evaluated our modifications to the `Queue` element. For these experiments, we tested four different instances of `Queue`: the base element provided in the standard Click distribution (which we refer to as `Queue`); the extension enabling all of RAM to be used (which we refer to as BigQueue); the extension enabling auxiliary queues on disk (which we refer to as DiskQueue); and the extension enabling auxiliary queues on two disks (which we refer to as opportunistic disk queue management—ODQM). We begin by showing the maximum possible propagation delay that can be realized on a link without packet loss for various input rates in Figure 5.4. We can see that BigQueue enables an order of magnitude longer delays than `Queue`. Since typical transcontinental propagation delays in the Internet are

[2]For the NICs, we used the Intel e1000 device driver and set `TxDescriptors` to 4096, the maximum possible value.

[3]The fact that the CPU is always near 100% utilization due to polling makes it somewhat more difficult to do bottleneck analysis.

of the order of 10's of milliseconds, BigQueue is capable of providing this capacity on high throughput links. We will show in our evaluation below that these propagation delays are also highly precise.

DiskQueue and ODQM were created to enhance NETPATH's scalability for applications such as large propagation delay emulation and capacity emulation. We evaluated the capabilities of these systems by fixing the incoming traffic rate to 100 Mb/s, and configuring NETPATH to emulate a capacity of 10 Mb/s. In this experiment, tests are run for 3 minutes—what might be considered to be an exceptionally long burst in the Internet. In Figure 5.5, we show the packet loss rates for packet streams with different size packets for all of the queue element implementations. We experimented with different packet sizes since Click allocates memory based on packets received—thus the effective allocation rate and the amount of space allocated goes down as packet size increases. The figure shows that the basic `Queue` is not useful for this application. When BigQueue is used, packets are lost because of overflow. DiskQueue enhances performance by about 300% and ODQM achieves 4 times better throughput than the DiskQueue. Packet loss still occurs in both DiskQueue and ODQM due to buffer overflows at the NIC. This is due to the overhead of the read/write tasks in these implementations which occasionally block the polling task. The implication of this result is that the auxiliary queues created by the `DiskQueue` element enable bursty traffic, common in the Internet [138], to be handled effectively.



(a) Generated delay of mean 30 milliseconds and standard deviation 3 milliseconds.

(b) Generated delay of mean 30 milliseconds and standard deviation 30 microseconds.

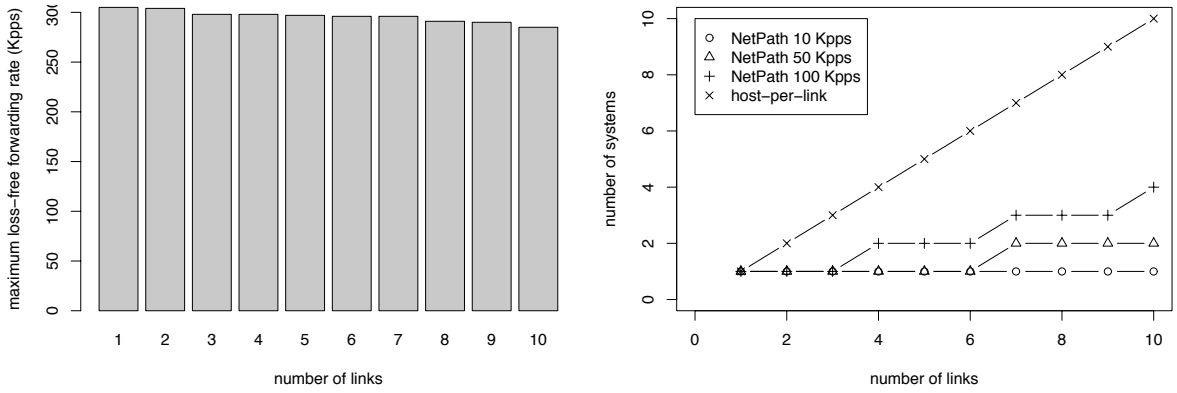
5.6: Probabilistic delays produced by NETPATH under two different offered loads for generated delays of mean 30 milliseconds and two standard deviations of 3 milliseconds (left) and 30 microseconds (right).

As explained above, our implementation of probabilistic delays follows a normal distribution based on user specified mean and standard deviation. We demonstrate NETPATH's ability to create delays that follow the specified distributional characteristics in Figures 5.6a and 5.6b. These figures show the cumulative distribution function of generated delays and compare them to the measured distribution. We considered two different configurations, each with mean delay of 30 milliseconds, one with standard deviation 3 milliseconds (*i.e.,* variable delays) and one with standard deviation 30 microseconds (*i.e.,* relatively stable delays). Each test was run with two different offered loads, 10 Kpps

and 1 Kpps. From the figure, as variability increases, the mean of the measured delay is shifted by about 10 milliseconds for 10 Kpps and about 5 milliseconds for packet rate 1 Kpps. This effect is due to head-of-line blocking. Clearly, the distributional difference between the specified delay values and measured delay values depend on the packet inter-arrival times and the variability in delay. This result indicates that the current functionality for probabilistic delays is adequate for applications in which typical arrival times are longer than typical delay times.

One of the design goals for NETPATH was that it support emulation of multiple point-to-point links at the same time. NETPATH's design enables multiple `Queue-Delay-ChangeTag` sequences to be operational on a single host. Packets exiting each of these element sequences are subject to the `StrideScheduler` element prior to arrival at the `ToDevice` element. Thus, increasing the number of links not only increases classifying overhead but also the scheduling overhead which can affect both the cumulative forwarding rate and the precision of the system.

Figure 5.7a demonstrates these effects. In this experiment, we increased the offered packet rate evenly across all links until packet loss occurred on any link. Increasing the number of links from one to ten results in about a 7% reduction in the overall forwarding rate. This result indicates that NETPATH can be effectively used for simultaneous path emulation on multiple links as long as the overall input rate for all paths is less than about 90% of the maximum throughput rate for one link. An important implication of a highly scalable system is in reduction of the number of hosts required for path emulation in a laboratory testbed. Figure 5.7b illustrates the number of systems required if one per link are used (as is the case in some network research labs today) compared with assigning multiple links per NETPATH host for given offered loads. Clearly, substantial savings are possible.



(a) Maximum loss-free forwarding rate of one NETPATH host with multiple links. Offered load is spread evenly across all links.

(b) Number of host systems required for NETPATH under different maximum link loads. A one-host-per-link reference line is also shown as this is the approach used in some network testbeds today.

5.7: Scalability of NETPATH when multiple links are configured.

5.1: Comparison of delay precision of NETPATH versus the Adtech SX-14 hardware-based emulator. Values indicate mean delay (standard deviation) delivered by each system, in milliseconds.

offered load (Kpps)	NETPATH μ (σ)	Adtech SX-14 μ (σ)
50	10.03 (0.01)	9.99 (0.00)
100	10.03 (0.01)	9.99 (0.00)
150	10.04 (0.01)	9.99 (0.00)
200	10.05 (0.01)	9.99 (0.00)
250	10.06 (0.01)	9.99 (0.00)
300	10.09 (0.03)	9.99 (0.00)

5.3.3 Comparative Performance

Our first comparative performance test of NETPATH considers its capability of generating high precision delays compared with a hardware-based emulator, an Adtech SX-14 [25]. Our SX-14 was configured with two OC-3 interfaces enabling it to be physically interposed between two nodes. We used OC-3 ports on AX-4000 for traffic generation and measurement of the SX-14. We configured the system for a fixed delay of 10 milliseconds, and, like our other experiments, measured the delays of packets passing through the system using different offered packet rates over a period of 10 minutes. We ran the same series of tests using NETPATH (with the configuration shown in Figure 5.2a). Table 5.1 shows that NETPATH's mean delay is within 1% of the SX-14 mean delay. The table also shows that NETPATH's mean delay is relatively higher than the SX-14 due to the inherent overhead of software-based systems. We also see from the table that there is very low variability in delays generated by NETPATH. An implication of these results is that as input rates approach the maximum loss-free forwarding rate, NETPATH provides precise delays—approaching those of dedicated systems like the SX-14.

Next, we compare the performance of NETPATH with several widely-used software-based network emulation tools including NIST net, Dummynet, and ModelNet[4]. Our assessment of performance considers the throughput and precision of each tool through a series of four identical experiments. Each experiment measures the software-based emulator's performance under a variety of input rates using 66 byte UDP packets. We configured each tool to provide a modest 10 millisecond delay on a single link in each experiment.

Figure 5.8 shows the mean and standard deviation of measured delays for NETPATH, NIST net, Dummynet, and ModelNet respectively. Table 5.2 contains selected values from these experiments. The results indicate that NIST net's delay precision is fairly accurate up to 80 Kpps but beyond this the deviation from the target delay begins to increase. Degradation at this threshold can be attributed to the interrupt driven implementation of NIST net. Similar to NIST net, Dummynet shows good performance up to 80 Kpps but beyond this, performance degrades dramatically. ModelNet's performance is the best of the software-based emulators up to about 140 Kpps, but beyond 150 Kpps, we were unable

[4]On the same host system as NETPATH, we configured NIST net (version 2.0.12), and on a separate partition installed FreeBSD version 4.8 for Dummynet and ModelNet (version 0.96). On the FreeBSD system, we set the system clock rate to 1 kHz and set NMBCLUSTER to 65K, according to Dummynet and ModelNet documentation.

5.2: Delay precision for emulation systems over a range of offered loads (incoming packets per second). Values indicate mean delay (standard deviation) delivered by each system, in milliseconds.

offered load (Kpps)	NETPATH μ (σ)	**NIST net** μ (σ)	**Dummynet** μ (σ)	**ModelNet** μ (σ)
40	10.04 (0.01)	10.14 (0.03)	10.00 (0.35)	10.06 (0.13)
80	10.04 (0.01)	10.52 (0.40)	10.23 (0.46)	10.10 (0.20)
120	10.04 (0.01)	12.75 (0.84)	29.08 (1.68)	10.56 (0.45)
160	10.04 (0.01)	16.47 (1.96)	399.00 (3.50)	-failure-

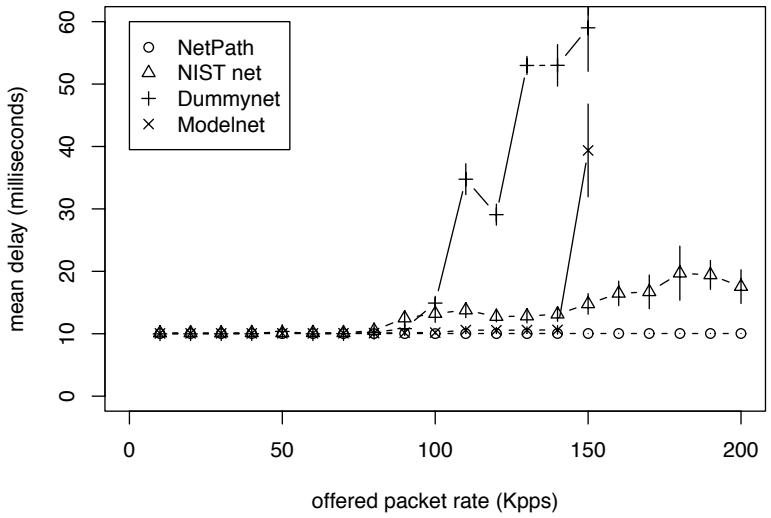
to get any response. A further point of comparison of delay precision is the variability of delays. Table 5.2 shows that standard software-based emulators will not provide precise delays on links under heavy load, and that NETPATH is stable over the range of these conditions.

Delay distribution is a useful metric for comparing software-based emulators but is incomplete since there is no notion of loss-free forwarding. Figure 5.9 shows the packet loss characteristics of each tool as a function of offered load. NETPATH does not drop packets over the entire range. Dummynet and ModelNet begin to drop after 40 Kpps and NIST net after 80 Kpps[5]. Implications of these results are significant since inadvertent packet loss can have a great negative impact on many types of experiments.

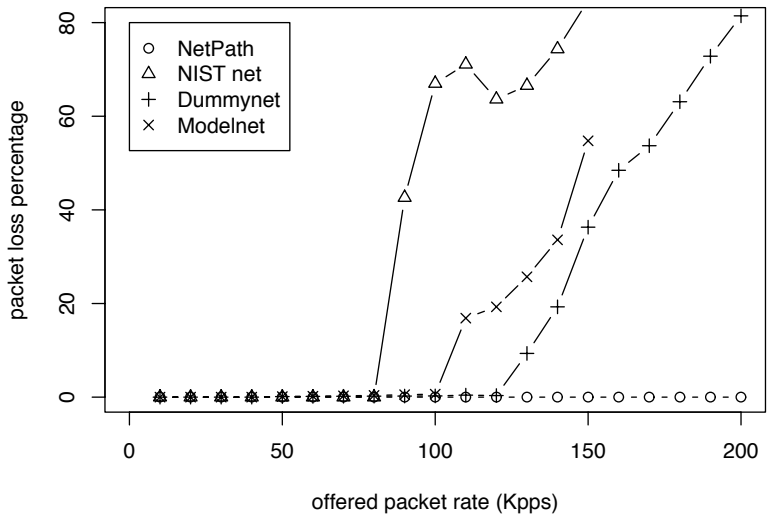
As a final point of comparison, we looked at effects on application traffic by NETPATH, NIST net, and Dummynet using the testbed shown in Figure 5.2b. We configured each path emulation system to introduce an 80 millisecond round-trip time propagation delay and used the HARPOON traffic generator [189] to create self-similar web-like TCP traffic over a range of bit and packet rates, from approximately 60 Mb/s (20 Kpps) through 150 Mb/s (45 Kpps). At HARPOON clients, we measured response times (time between the initial SYN to the first data response packet from the server). We also captured all packet headers on the bottleneck link using an Endace DAG 3.5 card and measured packet drops at the bottleneck queue. In addition to the three software emulators, we ran experiments using the Adtech SX-14 to obtain a hardware-based reference point. All experiments were run for 10 minutes.

Figure 5.10 shows the distribution of response times measured by the HARPOON clients for the lowest offered load (60 Mb/s, 20 Kpps). Response times are shown relative to the SX-14 reference. We see that response times through NIST net are, on average, slower than the SX-14, and are quite variable. NIST net drops packets during this experiment, causing retransmissions and an overall shift in response times. For Dummynet, while the mean response times are closer to the reference point than NIST net, there is very high variance indicated by the tails in the distribution. Dummynet drops fewer packets than NIST net for this experiment, but with much more variable delays. NETPATH delivers response times that are quite close to the hardware reference point, with relatively low variability. For higher data rates, NETPATH delivers performance similar to that shown in Figure 5.10, while the other two systems degrade further.

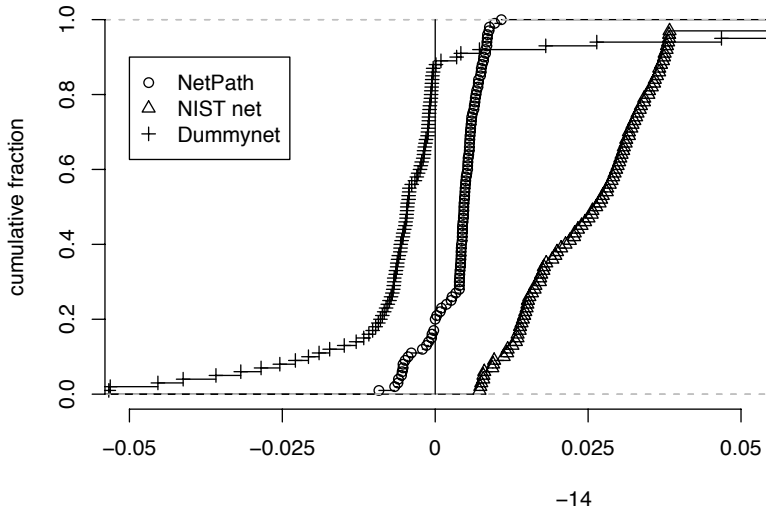
[5]The bump in the NIST net curve is caused by interrupt coalescence.



5.8: Delay precision of NETPATH, NIST net, Dummynet and ModelNet respectively for a configured delay of 10 milliseconds. Lines above and below each curve represent one standard deviation away from mean.



5.9: Comparison of packet loss characteristics of NETPATH versus other popular software-based emulators under different offered packet rates.



5.10: Distribution of response times (time between initial SYN and first data response packet) for web-like TCP traffic using NETPATH, NIST net, and Dummynet. Response times are all relative to the Adtech SX-14.

It appears that the more realistic packet arrival process presented by HARPOON elicits somewhat different performance from each system than with constant bit rate streams. While NETPATH exhibits behavior nearly identical to the SX-14 under CBR traffic, it performs somewhat differently under this setup. Still, NETPATH's performance is far superior to both NIST net and Dummynet. As future work, we intend to examine NETPATH's behavior more closely to further improve performance.

Part II

Case Studies in Calibrated Network Measurement

Chapter 6

Calibration Case Study 1: Available Bandwidth Estimation

> "I often say that when you can measure what you are speaking about, you know something about it; when you cannot measure it . . . your knowledge is of a meagre and unsatisfactory kind."
> —Lord Kelvin, as quoted in [114].

6.1 Overview

Dynamic estimation of end-to-end available bandwidth has important potential for network capacity planning and network overlay monitoring and management. Tools designed for active measurement of available bandwidth (AB) send precisely crafted packet pairs or streams and infer the bandwidth available along a given end-to-end path by measuring changes in the spacings of the packet pairs or streams at a receiver. Over the past few years, the development of fast and accurate available bandwidth estimation tools (ABETs) has become an active area of research (see, for example, [43, 75, 117, 121, 135, 152, 169, 180, 187, 188, 197]).

In this chapter, we apply the calibration strategy described in Chapter 3 through a series of experiments to two ABETs, Spruce [197] and Pathload [121], which we consider to be canonical representatives of two basic methods for ABE. Recall that calibration is comprised of two complementary activities: *comparison* with a known standard, and *adjustment* to match a known standard. For ABETs, the first facet involves comparison with measurements of available bandwidth that have been obtained through, *e.g.*, packet traces with timestamps of sufficient quality. The second aspect of calibration may involve adjusting parameters of a given algorithm, or the algorithm itself.

As with the case studies we consider in subsequent chapters, a detailed understanding of realistic queuing effects experienced by individual packets as they compete and interact with other packets is essential for gaining insight into behavior ABET, and requires fine-grained, time-synchronized measurements of packets as they arrive at *and* subsequently depart from the different routers along the network path. Our analysis of these measurements includes the use of *phase plots*, as described in Chapter 3. Within the context of ABETs, we advocate phase plot analysis as a natural means for assessing two characteristics critical to measurement tool calibration. The first is the ability to identify and isolate sources of bias in active measurement tools themselves. The second is the ability to assess the

basic design assumptions of ABETs concerning the nature of complex queuing effects that individual packets or flows experience at routers as they traverse the network and interact with competing traffic.

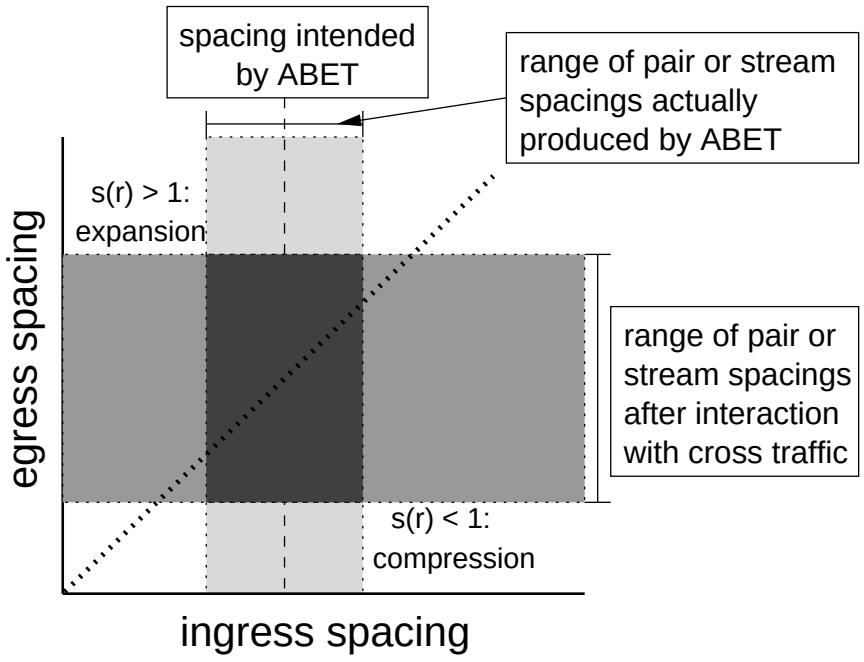
We analyze the detailed arrival and departure measurements of Spruce and Pathload available in our testbed, showing why and how both tools are prone to measurement bias and errors over a range of increasingly complex cross traffic and network path conditions. With the insights gained from analyzing the detailed arrival and departure measurements, we designed and implemented a calibrated ABET, called YAZ, that builds on the basic insights of Pathload. Through an additional set of laboratory-based calibration tests, we show that (1) YAZ compares well with respect to ground truth measures of AB, (2) it is significantly more accurate than both Spruce and Pathload, while remaining much less intrusive than Pathload, and (3) it produces available bandwidth estimates faster than the other tools. If Pathload and Spruce represent a tradeoff between measurement overhead and measurement accuracy, our results for YAZ show that this tradeoff is not fundamental. For a Spruce-like budget, YAZ is more accurate than Pathload, sometimes significantly so.

6.2 Calibration Measurements and Analysis

Available bandwidth estimation tools such as Spruce [197] and Pathload [121] are designed to send packet pairs or streams at well-defined spacings (on time scales of tens to hundreds of microseconds), measure the spacings of the same packets at a receiver, and infer the amount of available bandwidth along the path according to a tool-specific model. To gain insight into ABET behavior, we capture time-synchronized packet headers *before and after* interaction between probes and cross traffic at a congested queue. From these measurements, we can compare *measured* packet level characteristics of a probe stream with characteristics that the tool *should have produced*. From the arrival measurements, we can construct a true measure of AB over a given time interval as a standard by which to judge an ABET.

To analyze the probe arrival (ingress) and departure (egress) measurements, we use a phase plot representation. To construct the phase plot, we consider the time delay between consecutive probe packets as they arrive at the router ($Ingress\_Spacing = s_i$) as the x dimension on the plot and consider the spacing of the same packets as they exit the router ($Egress\_Spacing = s_e$) as the y dimension. From these measurements, there are three possibilities for the ratio $s_r = s_e/s_i$. If $s_r = 1$ then spacing remained unchanged by the router. If $s_r > 1$ then other packets enqueued between the two packets causing *expansion*. If $s_r < 1$ then the first packet was delayed because of a queue that has diminished by the time the second packet arrives, causing *compression*.

Figure 6.1 depicts how we use phase plots for ABET calibration. The ingress dimension of the plot should reveal any differences between spacings that are intended by the ABET, and the spacings actually produced. This provides the ability to assess bias introduced into the measurement process by imprecise commodity hardware and operating systems. The egress dimension of the plot shows the spacings on which inferences are made by the receiver after interaction with cross traffic, though they may differ from the spacings actually measured by the receiver (again, due



6.1: Interpreting phase plot features for available bandwidth estimation tool analysis and calibration.

to inaccuracies introduced by commodity hardware and operating systems). Note that while some ABETs do not make inferences directly from these spacings (*e.g.*, Pathload), they play a key role in AB estimates. Therefore, these spacings enable calibration of both the inference method as well as providing a baseline for calibrating the receiving host.

6.3 Calibration Experiments

The objective our calibration study of Pathload and Spruce was to examine the tools under a number of increasingly complex traffic and path conditions and to understand where they work well, where they work poorly, and why. For each experiment we evaluate the tool's ability to report AB within a range of 10% of the tight link capacity. This threshold is chosen as an arbitrary reference point in the sense that a threshold would typically be chosen based on specific requirements of a target application. We required that estimates be consistently within this window of accuracy for a series of estimates reported by an ABET over the duration of an experiment. Without the property of *consistent accuracy*, ABETs are unlikely to be used in applications such as in re-optimization of an overlay network.

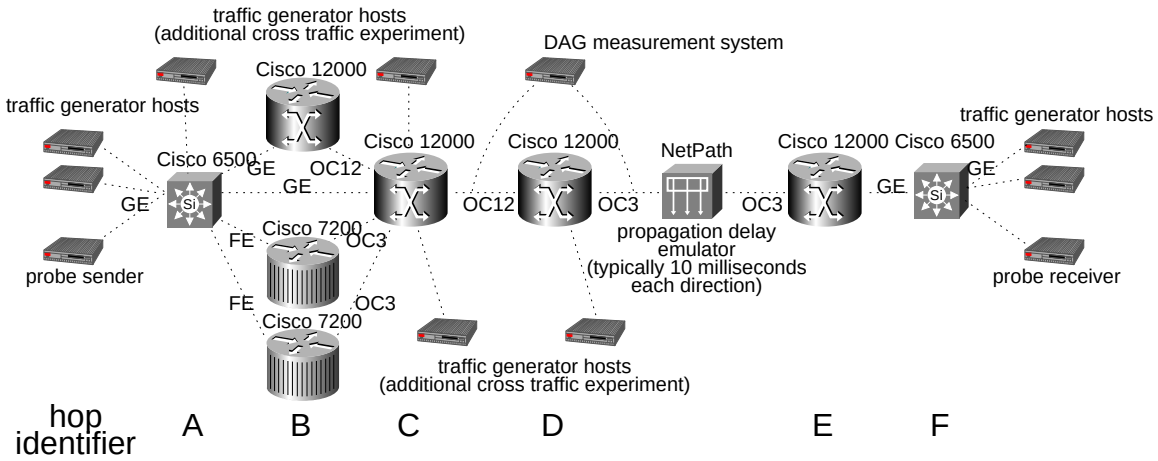
6.3.1 Testbed Setup

Our testbed configuration consisted of variations on a dumbbell-like topology with an OC-3 narrow link as depicted in Figure 6.2. We used a total of six setups, including three traffic scenarios: constant bit rate (CBR) traffic of 50 Mb/s (UDP traffic with uniformly spaced 1500 byte packets), 19 long-lived TCP flows in a single direction, 19 long-lived TCP flows in each direction, and three variants of a setup using bursty traffic generated by HARPOON with file sizes drawn from a heavy-tailed distribution to produce self-similar traffic to approximate a mix of web-like and peer-to-peer traffic. In all cases, the direction of interest was left to right in the figure (*i.e.*, CBR, single direction long-lived TCP connections, and web-like traffic traveled left to right). Cross traffic was generated by hosts running HARPOON to create bursty traffic or Iperf [198] to create infinite source and constant bit rate traffic[1]. We used NETPATH to add a delay of 10 milliseconds in each direction for all experiments unless otherwise specified.

To create increasingly more complex path conditions, we considered the following three topological setups.

Topology 1 (narrow and tight link are the same, homogeneous round-trip time): Probe traffic was configured to cross the Gigabit Ethernet link directly connecting routers at hops A and C. No cross traffic was routed across this link. CBR and long-lived TCP connection traffic crossed the Cisco 12000 at hop B, while web traffic was configured to use the two Cisco 7200's and the Cisco 12000 at hop B, but not the direct link to hop C. Our decision to route probe traffic direction from hop A to hop C caused the tight link and narrow link to be identical in the CBR, long-lived TCP

[1]Our traffic generator hosts were identically configured workstations running either Linux 2.4 and FreeBSD 5.3. The workstations had 2 GHz Intel Pentium 4 processors, 2 GB of RAM and Intel Pro/1000 cards (with interrupt coalescence disabled). Each system was dual-homed, so that all management traffic was on a network separate from the one depicted in Figure 6.2. Probe traffic systems were identical to the traffic generators and ran FreeBSD 5.3.



6.2: Experimental testbed. Cross traffic scenarios consisted of constant bit rate traffic, long-lived TCP flows, and bursty web-like traffic. The tight link (link with least available bandwidth) is the OC-3 between hops D and E. Cross traffic flowed across one of three routers at hop B, while probe traffic normally flowed directly between hop A and hop C over a Gigabit Ethernet (GE) link. In a setup forcing the narrow (link with smallest capacity) and tight links to be distinct physical links, probe traffic crosses a Fast Ethernet (FE) link between hops A and B. In a setup considering additional web-like cross traffic, hosts shown attached at hops A, C, and D generate traffic that persists on shared links for one hop. Optical splitters connected Endace DAG 3.5 or 3.8 measurement cards to the testbed between hops C and D, and hops D and E. The bulk of cross traffic and probe traffic flowed left to right.

source, and basic web-like traffic scenarios. When using web-like cross traffic in this setup, we configured HARPOON to produce an average of 50 Mb/s.

Topology 2 (narrow and tight link are not the same, homogeneous round-trip time): Using web-like cross traffic, we routed probe traffic across a Fast Ethernet link between hops A and B, but configured cross traffic not to use this link. In this experiment, we also configured the cross traffic sources to produce approximately 100 Mb/s of traffic on the OC-3 link between hops D and E, causing the Fast Ethernet link to be the narrow link, but the OC-3 to be the tight link[2].

Topology 3 (narrow and tight link are not the same, heterogeneous round-trip time): Using again web-like cross traffic, we configured our Linux traffic generation hosts with NETPATH to emulate round-trip times of 20, 50, 80 and 110 milliseconds. We also attached additional hosts at hops A, C, and D to generate cross traffic that traveled across all links between hops A and C (sharing the link with probe traffic) or the OC-12 link between hops C and D.

Critical to our calibration methodology was the ability to take high accuracy measurements in our test environment. To do this we attached optical splitters and Endace DAG 3.5 packet capture cards (affording timestamping accuracy on the order of a single microsecond [89][3]) to monitor the links between hops C and D, and hops D and E. We used these monitoring points to create phase plots and measure utilization on the tight OC-3 link. This configuration gave us ground truth measurements well beyond coarse-grained SNMP and similar types of measurements used in prior *in situ* studies of ABETs.

6.3.2 ABET Calibration: Comparison

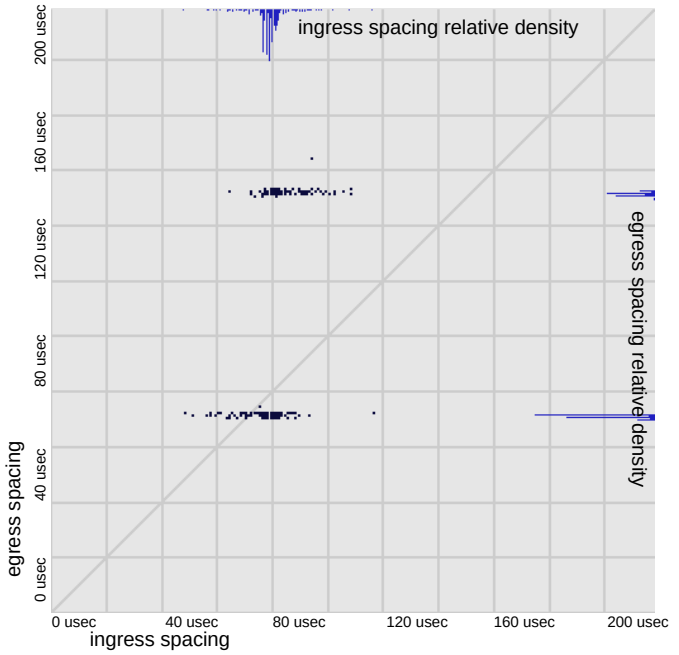
The calibration framework described in Chapter 3 directs our evaluation process. We begin by assessing the capabilities of the end hosts running the ABETs. Sources of potential bias introduced by end hosts include OS context switches and other system capability/OS effects such as network adapter interrupt coalescence [121, 126, 177]. Our interest is not in untangling the details of each source of host system bias, rather it is in understanding the overall impact.

In our experiments below, we considered topology 1 and collected traces from a single Pathload fleet (1200 probe packets of 1309 bytes), and a series of 12 Spruce runs (1200 packet pairs, each packet of length 1500 bytes) with constant bit rate cross traffic of 50 Mb/s flowing across the narrow link during each test. If the host systems emitted packets without bias, we would expect ingress spacings for both tools to be tightly clustered around the intended value of 80 microseconds.

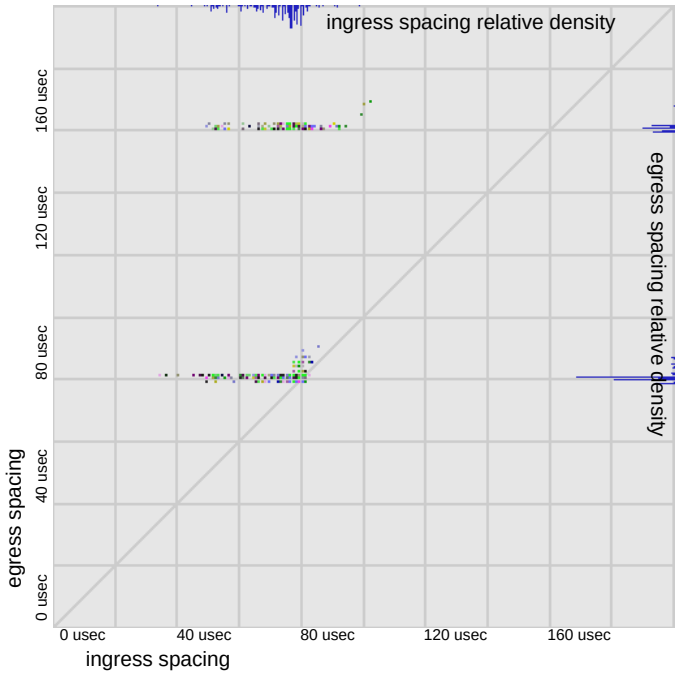
The phase plots created using SPLAT for these experiments and shown in Figure 6.3 immediately expose two potential sources of measurement bias. First, it is easy to see that for each ABET there is a wide range of interpacket

[2] We verified in each experiment that, over each tool measurement interval, the tight link was always the OC-3 link.

[3] As a consistency and calibration check, we also captured traces using Endace DAG 3.8 cards, which employ a higher frequency clock, and have somewhat different architectural features than the DAG 3.5. The resulting phase plots were consistent with those produced using the DAG 3.5. Experiments described below employ the DAG 3.5 cards unless otherwise specified.



(a) Phase plot produced from one Pathload fleet (1200 probe packets of length 1309 bytes).



(b) Phase plot produced from 12 Spruce runs (1200 packets pairs, packets of length 1500 bytes).

6.3: Phase plots of Pathload and Spruce streams created using SPLAT. Grid lines are separated by 20 microseconds for each plot. CBR cross traffic of 50 Mb/s, with uniform UDP packets of 1500 bytes (not shown in plots) causes bimodal output spacing distribution of probe traffic. Target input spacing for each tool is 80 microseconds. Note the slightly different scale for each plot.

spacings on ingress which can be attributed to the sending host. Second, it is also evident that an effect of the CBR cross traffic is to cause a respacing of probe packets on egress to either back-to-back (70 microseconds for Pathload packets, 80 microseconds for Spruce packets) or with one cross traffic packet interposed (150 microseconds for Pathload, 160 microseconds for Spruce). Closer examination reveals that packets spaced farther apart by the ABET are more likely to experience expansion by a cross traffic packet than to be transmitted back-to-back on the tight link. This can be seen in Figure 6.3a by the perceptible shift to the right in the upper cluster of points. A similar shift exists in Figure 6.3b. Also, observe that points below the diagonal line in Figure 6.3a represent evidence for compression in Pathload streams. We quantify the prevalence of this effect below. Finally, we note that additional benefits of using SPLAT to analyze these types of measurements include the possibility for dynamic analysis of ABET bias through SPLAT animations and the potential to filter the data, *e.g.*, to focus on time segments of high intensity background traffic or rapidly changing background traffic conditions.

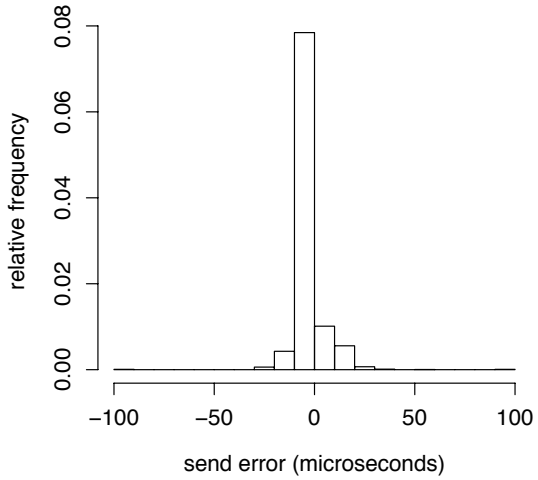
To further explore the problem of bias imposed by probe senders, we collected several thousand packet spacing measurements from Spruce and Pathload and compared each spacing with the spacing measured at the DAG monitor between hops C and D. Figure 6.4a shows a representative histogram of differences between the spacing measured using `gettimeofday()` at Pathload[4]. From these measurements, we conclude that while the magnitude of individual errors can be quite significant, the mean deviation is close to zero.

Next, we examined the measurements at the receiving application. Pathload timestamps outgoing/incoming packets using the `gettimeofday()` operating system call. Spruce timestamps outgoing packets using the `gettimeofday()` system call and incoming packets receive timestamps in the OS interrupt handler (it uses the `SO_TIMESTAMP` option when initializing its receive socket). Timestamps used for both these tools are of microsecond precision (though not necessarily microsecond accuracy). Comparing timestamps measured upon packet receive with timestamps measured at the DAG monitor between hops D and E (*i.e.*, the egress spacings of Fig. 6.3 compared with application-measured receive spacings), we obtain a result similar to the sender. Figure 6.4b shows a representative histogram of differences in packet spacings measured at the probe receiver versus the same spacings measured at the DAG monitor. The magnitude of error is smaller than that on the sender and the mean deviation is close to zero.

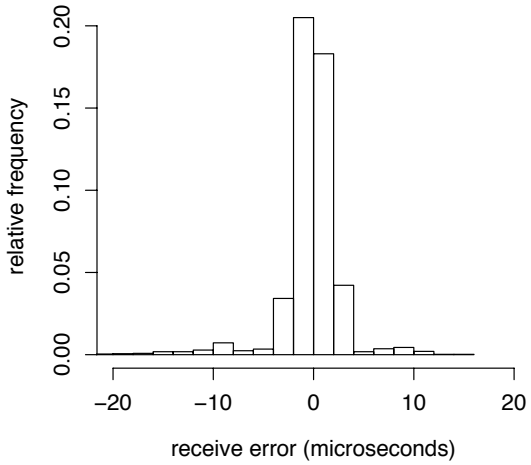
As a final calibration check, and to test whether these results were unique to the hardware and OS configuration used, we attached a DAG 4 (Gigabit Ethernet) monitor directly to the Intel Pro/1000 on a Linux 2.4 workstation and collected additional measurements using Spruce. A histogram of differences between spacings measured at Spruce and spacings measured at the DAG 4 is shown in Figure 6.4c. Again, the mean deviation is close to zero. Packet receive errors on the Linux 2.4 system (not shown) are also close to zero mean. Table 6.1 summarizes these results.

Even though the averaged behavior of probe streams tends toward the intended value, bias on individual probes can still have a significant detrimental effect on the operation of Pathload and Spruce. Since the source code for each tool treats each timestamp as accurate, while utilizing the microsecond precision supplied by the `gettimeofday()` system

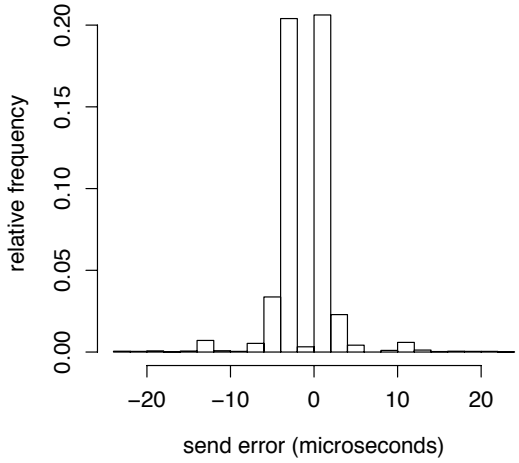
[4]We modified Spruce and Pathload to log these timestamps.



(a) Distribution of errors at FreeBSD 5.3 sender with Intel Pro/1000 Gigabit Ethernet adapter and DAG 3.5 between hops C and D.



(b) Distribution of errors at FreeBSD 5.3 receiver with Intel Pro/1000 Gigabit Ethernet adapter and DAG 3.5 between hops D and E.



(c) Distribution of errors at Linux 2.4 sender with Intel Pro/1000 Gigabit Ethernet adapter and DAG 4 attached directly to network interface.

6.4: Relative frequencies of errors between send or receive packet spacings and spacings measured at DAG monitor.

6.1: Summary of errors between packet spacings measured at application send and receive, and DAG monitors. All values are in microseconds. Negative values indicate that a larger spacing was measured at the DAG monitor than in the application.

	DAG 3.5/3.8 (OC-3/12) FreeBSD 5.3		DAG 4 (Gigabit Ethernet) Linux 2.4
	Send Error	Receive Error	Send Error
Min	-93.00	-20.00	-24.00
Median	-2.00	0.00	-2.00
Mean	-1.54	0.15	-0.61
Max	100.00	18.00	23.00

call, even very tiny differences may lead to poor AB estimation. Also, since errors are widely distributed around zero and may not be uniformly distributed, simple filtering schemes may be ineffective.

6.3.3 ABET Calibration: Algorithmic Adjustment

Although phase plots are not new to the networking community their use in ABET calibration as illustrated above has been very beneficial. They not only helped us to expose end-host limitations of generating precise streams and to identify the resulting bias, but by studying the egress spacings, we were also able to gain an expectation of what the receiving host *should have* measured and realized that considering compression events is important. In summary, phase plot analysis resulted in the following observations:

- The error introduced by end hosts has approximately zero mean when multiple measurements are taken.
- The relationship between input and output probe rates and available bandwidth at the heart of Pathload's algorithm (Equation 2.4, Chapter 2) invites refinements.
- Both compression and expansion are indicative of congestion along a measured path.

These observations lead us to propose a calibrated algorithm for measuring available bandwidth. We first test how quickly the mean deviation of measurements converges, on average, to zero. That is, how many packets should comprise a stream, at minimum, in order for the error to be less than some threshold? To answer this question, we created a tool to send packet streams of length 100 packets at four target spacings of 60, 80, 100, and 120 microseconds, in separate experiments. We ran the tool under topology 1 with no cross traffic to collect approximately 1000 packet stream measurements (*i.e.*, about 100,000 packets per experiment)[5]. For each packet stream, we counted the number of packets required for the mean deviation between spacings measured at the DAG monitor and timestamps generated by the application to be less than 1 microsecond. Figure 6.5 plots the cumulative distribution of stream lengths required for each target spacing. The distributions show that the mean error converges to zero quite quickly and that

[5]Phase plots from these experiments are similar to those shown in Figure 6.3.

packet streams of at least length 20–40 packets appear to be sufficient[6]. However, there remain tradeoffs for ABET methodologies using packet streams. While shorter streams may reduce the intrusiveness of the tool, and may reduce measurement latency, the averaging time scale is also reduced, theoretically resulting in greater measurement variance (Equation 2.1).

At the base of our proposed algorithm is a version of Equation 2.4, modified to consider absolute difference in average input and output spacings. The absolute difference is used because both compression and expansion (or the combination of both in a given measurement period) must be considered as an indication of congestion. Average spacings are used since we have shown that individual spacings are subject to significant error. In the formulation below, we consider the absolute difference between the average input and output spacings as an indication of whether the input rate was above the available bandwidth along the path (analogous to Equation 2.4):

$$|\overline{g_{in}} - \overline{g_{out}}| = \begin{cases} \leq \zeta * \overline{g_{in}} & r_{in} \leq A \\ > \zeta * \overline{g_{in}} & r_{in} > A. \end{cases} \quad [6.1]$$

The value ζ is a threshold parameter used to determine how far apart the average send and receive spacings can be while still considering the input stream to be below the level of spare capacity along the measurement path.

Like Pathload, our algorithm for finding the available bandwidth is iterative. First, we set the target send spacing, g_{target}, to be some minimum value (effectively setting the maximum AB measurable), then proceed as follows.

1. Send probe stream, measuring $\overline{g_{in}}$ and $\overline{g_{out}}$ at send and receive hosts, respectively.

2. If the absolute difference in average input and output spacings is above the ζ threshold of the input spacing Equation 6.1, increase g_{target} by $\frac{|\overline{g_{in}} - \overline{g_{out}}|}{2}$, wait a configurable amount of time, and go to previous step.

3. Otherwise, update an exponentially-weighted moving average (EWMA) (with parameter α) with the estimate r_{in}. Report the updated EWMA as the estimate of AB.

The reason for using an exponentially-weighted moving average on the individual estimates is that there will still be outliers in measurements. Even though Figure 6.5 implies that a stream length of 20 packets is generally sufficient, due to the unpredictability of commodity systems, care is still required.

We consider the above algorithm to be a "calibrated Pathload" and have implemented it in a tool called YAZ[7]. The source code for YAZ is available to the research community for evaluation at `http://wail.cs.wisc.edu`.

[6] Although we do not show detailed results here, we also ran experiments with interrupt coalescence (IC) enabled on packet transmit, with an absolute timer of 64 microseconds (the default value). The absolute timer sets the maximum amount of time that a packet will be held awaiting another packet to send before triggering an interrupt in the device driver. The results for this configuration show that target spacings that are approximately 20-40 microseconds above the IC absolute timer cannot be met, and no packet stream of reasonable length can reduce the error to an acceptable range. We therefore ran all experiments with IC disabled.

[7] Although the name YAZ is reminiscent of tool names starting with "yet another…", our tool is actually named after the baseball great, Carl Yastrzemski.

6.3.4 Experimental Evaluation

We compared the accuracy of Pathload, Spruce, and YAZ using the different traffic and topological scenarios described above. For the CBR and long-lived TCP source experiments, we continuously collected AB estimates from each tool for 10 minutes, discarding the first 30 seconds and last 30 seconds. For the web-like traffic setups, we continuously collected AB estimates from each tool for 30 minutes, also discarding the first 30 seconds and last 30 seconds. For the comparisons below, we compute the actual available bandwidth using the DAG monitor between hops D and E for the exact interval over which a tool produces an estimate[8]. For each experiment, we consider the fraction of estimates that fall within a range of 10% of the tight link capacity (*i.e.*, the absolute difference between the reported estimate and the actual available AB). Since our tight link is OC-3 (149.76 Mb/s before Cisco HDLC overhead), this window is $\approx$ 15 Mb/s.

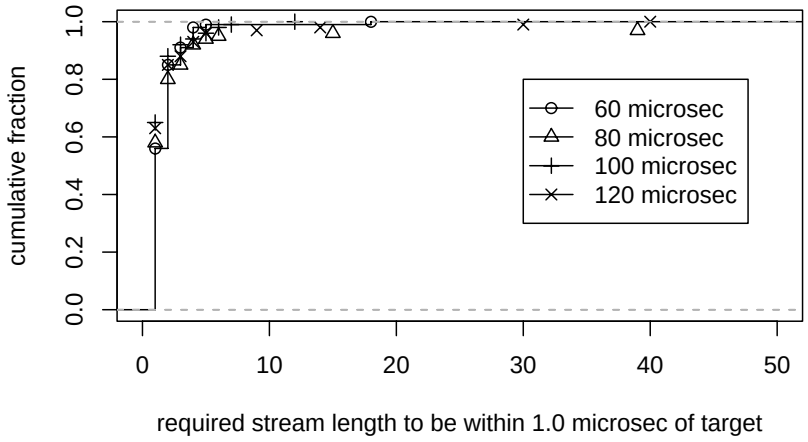
For all experiments, YAZ was configured with $\alpha = 0.3$ in its exponentially-weighted moving average and the threshold parameter ζ was set to be equivalent to a rate of 1 Mb/s, which we found to be a robust setting over our topologies and traffic scenarios. α of 0.3 produced minimum MSE over the collection of experiments. We set YAZ's stream length to 50 packets. For Spruce, we use 149.76 Mb/s as the tight link capacity in Equation 2.3 for all experiments except for the second web-like traffic scenario, in which we set it to 97.5 Mb/s (the narrow link is Fast Ethernet) and use the default value of 100 samples to compute an estimate of AB. For Pathload, we used default parameters, and in the initial comparison with YAZ, we set the stream length to 50 packets, while leaving the number of streams per fleet at the default value of 12. We report the midpoint of Pathload's estimation range as the AB estimate which, as we discuss below, is generally favorable to Pathload.

Results for all the experiments are shown in Figure 6.6. The results for constant bit rate traffic in topology 1 (Figure 6.6a) show that both YAZ and Pathload perform with similar accuracy, coming quite close to the true AB. However, fewer than 60% of Spruce estimates are within the 10% acceptance range.

The two long-lived TCP traffic scenarios in topology 1, in some ways, create the most pathological cross traffic conditions due to the frequent traffic oscillations on the tight link. Figure 6.6b plots results for the setup with TCP flows in a single direction. The YAZ estimates are fully within the 10% threshold, while more than 90% of Pathload's estimates are within this bound. Only about 20% of Spruce estimates fall within the acceptable range. For the bi-directional long-lived TCP flows, YAZ and Pathload perform similarly, with approximately 90% of estimates falling within the 10% acceptance range. Again, very few estimates produced by Spruce fall within the 10% range.

For the web-like cross traffic in topology 1 experiment (Figure 6.6c), approximately 75% of estimates produced by YAZ are within the acceptance range compared to about 50% of Pathload estimates and about 40% of Spruce estimates. We also ran Pathload in this setup again, setting the stream length to be 100 packets (the default in the Pathload source

[8] We include Cisco HDLC overheads (9 bytes per packet) in this computation. Since we control packet payloads, we limit any hidden effects due to SONET character stuffing.



6.5: Cumulative distribution of stream lengths required to cause mean sending error to be within 1 microsecond of target. Target spacings are 60, 80, 100, and 120 microseconds.

6.2: Mean and standard deviation of *PCT* and *PDT* values for streams of length 50 or 100 packets upon departure (prior to interaction with cross traffic) for web-like traffic scenarios.

stream length	*PCT*		*PDT*	
	μ	σ	μ	σ
50	0.4	0.14	0.04	0.28
100	0.26	0.14	-0.07	0.27

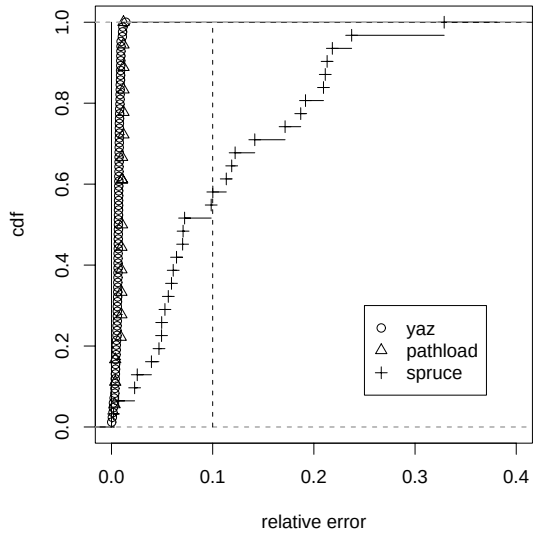
code). Figure 6.6e shows the result of this experiment, comparing the YAZ and Spruce results from Figure 6.6d. We see that the accuracy of Pathload improves by about 15%.

The results for the case of web-like cross traffic in topology 2 are shown in Figure 6.6f. In this setup, Pathload underperforms both YAZ and Spruce, with about 65% of YAZ estimates and about 55% of Spruce estimates falling within the 10% threshold, but only about 40% of Pathload estimates falling within this range. A closer look at the Pathload results revealed that it took longer on average to converge on an estimation range, and convergence times were more variable than in any other setup. Since AB is a moving target, these increased convergence times led to poor estimates. Finally, Figure 6.6g shows results for the web-like cross traffic in topology 3. In this setup, about 80% of YAZ estimates are within the acceptance range, compared with about 50% for Pathload and 40% for Spruce.

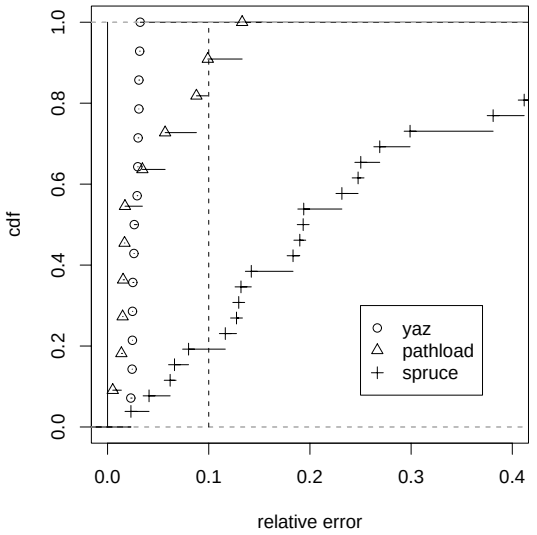
Detailed Analysis of Experimental Results

Reported Range of Pathload. We also examined how often the actual AB fell in the range reported by Pathload. For the constant bit rate and long-lived TCP experiments, the actual value is rarely within Pathload's range. The reason is that its range was often a single point, but not equal to the actual AB. For the three self-similar web-like traffic scenarios, the actual AB is, at best, within Pathload's range 58% of the time (53/92 estimates, for topology 3). For these experiments, the width of the range varies greatly, preventing general explanation. In the end, our focus on comparing the midpoint of Pathload's estimation range with the actual AB is favorable to Pathload. We plan to more carefully evaluate AB variation in future work.

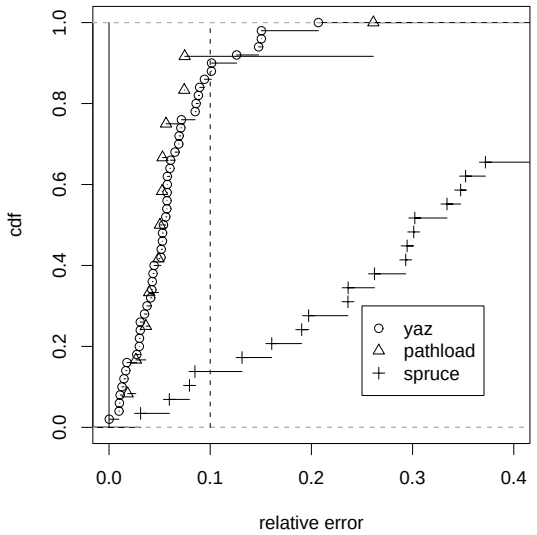
The Effect of Pathload's Stream Length. To better understand the underlying reason for Pathload's improved AB estimates with a larger number of probes, we examined a series of streams emitted by Pathload in the web-like traffic scenarios. We calculated the *PCT* and *PDT* metrics for each stream using the DAG monitor between hops C and D—*before* any interaction with cross traffic. Table 6.2 summarizes these results. The mean *PCT* for stream lengths of 50 is close to the threshold of 0.55 that considers the stream to reflect an increasing trend in OWD. With the longer stream length of 100, the mean *PCT* is further away from the threshold. This shift suggests that the longer stream has an important side-effect on the *PCT* metric, and that there is substantial bias introduced at the sending host. From the results in Table 6.2, stream length appears to have less of an impact on the initial bias of the *PDT* metric. For each stream length, at least 15% of the streams departed the sending host with at least one of the metrics exceeding its threshold.



(a) Constant bit rate cross traffic of 50 Mb/s (topology 1).

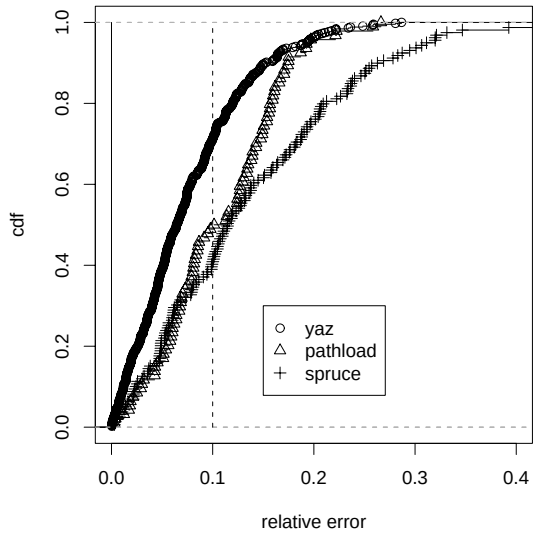


(b) Long-lived TCP sources in one direction (left to right in Figure 6.2) (Topology 1).

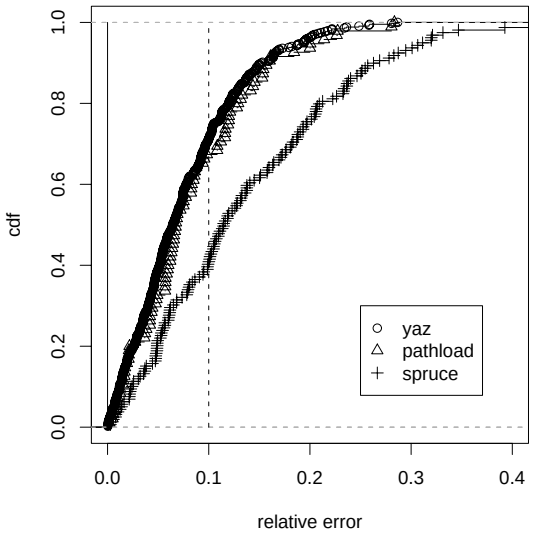


(c) Long-lived TCP sources in two directions (Topology 1).

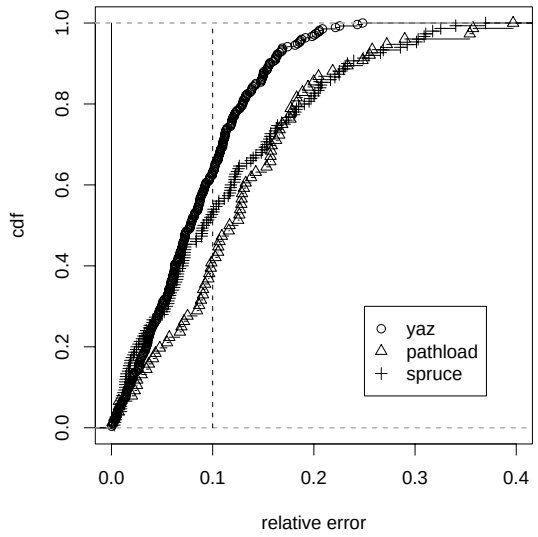
6.6: Comparison of available bandwidth estimation accuracy between YAZ, Pathload, and Spruce for the constant bit rate and long-lived TCP traffic scenarios. Plots show empirical cumulative distribution functions of measurement error, relative to the tight link (OC-3, 155 Mb/s) of the testbed, for each ABET considered. Error is computed by taking the absolute difference between estimated and true available bandwidth. True AB is computed using DAG traces over the same interval on which a tool estimation is performed. Dashed vertical line at $x = 0.1$ indicates 10% desired accuracy threshold.



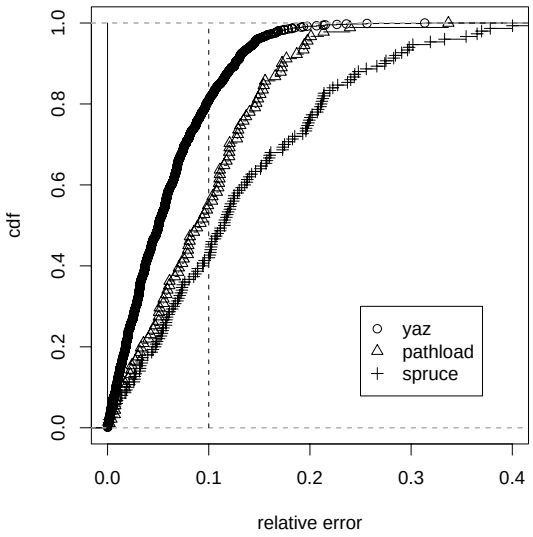
(d) Web-like cross traffic produced by HARPOON with average rate of 50 Mb/s (Topology 1).



(e) Comparison of YAZ, Pathload, and Spruce, for web-like traffic when Pathload is configured for streams of length 100 packets (Topology 1). (YAZ and Spruce curves are same as in Figure 6.6d.)



(f) Web-like traffic with narrow link (Fast Ethernet) and tight link (OC-3) as distinct physical links (Topology 2).



(g) Web-like traffic with additional points of cross traffic and a diversity of round-trip times (Topology 3).

6.6: Comparison of available bandwidth estimation accuracy between YAZ, Pathload, and Spruce for the web-like traffic scenarios. Plots show empirical cumulative distribution functions of measurement error, relative to the tight link (OC-3, 155 Mb/s) of the testbed, for each ABET considered. Error is computed by taking the absolute difference between estimated and true available bandwidth. True AB is computed using DAG traces over the same interval on which a tool estimation is performed. Dashed vertical line at $x = 0.1$ indicates 10% desired accuracy threshold.

6.3: Prevalence of compression in Pathload streams for all six cross traffic scenarios.

Traffic Scenario	Fraction of Compressed Streams
CBR traffic of 50Mb/s (Topology 1)	0.212
Long-lived TCP, one direction (Topology 1)	0.077
Long-lived TCP, two directions (Topology 1)	0.260
Web-like traffic (Topology 1)	0.233
Web-like traffic (Topology 2)	0.220
Web-like traffic (Topology 3)	0.219

Pathload Stream Compression. Table 6.3 quantifies the prevalence of compression in Pathload streams. We compared the spacing intended by Pathload for each stream with the spacings measured at the DAG monitor between hops C and D. The values for each traffic scenario in the table show the fraction of streams for which there was an overall effect of compression. We see that, in general, about 20% of all streams are compressed. In the case of long-lived TCP flows in one direction, the queue at the tight link is usually increasing as the flows increase their windows until loss occurs. The fact that a non-negligible fraction of streams over each scenario experience compression supports Equation 6.1 as a key distinguishing feature between YAZ and Pathload. It also, in part, explains why YAZ outperforms Pathload.

Tool Accuracy *Without* Cross Traffic. For Pathload, using the median OWD over a window of samples appears to be a critical component of at least the *PCT* formulation. A technique like Spruce, on the other hand, is not insulated from individual spacing and measurement errors; even a difference of a single microsecond (assuming a target spacing of 80 microseconds) can lead to an estimation error of nearly 2 Mb/s. For example, we ran Spruce in the testbed while not introducing any cross traffic. In this setup, the tool should invariably report close to 149.76 Mb/s. Over 10 consecutive runs, Spruce produced estimates ranging from 134.4 Mb/s to 153.8 Mb/s, with a mean of 149.4 Mb/s and a standard deviation of 5.42. These inaccuracies are entirely due to measurement error. While most estimates are close to the true AB, the worst estimate is just beyond the desired 10% accuracy range. (For similar zero cross traffic experiments using Pathload and YAZ, the estimates were consistently within 1-2% of the true AB.)

The Asymptotic Nature of Spruce. Lastly, as we noted above, the Spruce formula is an asymptotic result. During both long-lived TCP source experiments, Spruce reported negative estimates. Even with perfect measurement devices, it is not clear, *a priori*, how long a tool like Spruce should collect samples. While the default value of 100 may be sufficient for certain traffic scenarios, it is clearly insufficient for others.

Estimation Latency and Overhead. Lastly, we compare estimation latency, the average number of probes emitted per estimate, and the number of estimates produced during the first web-like traffic scenario. Table 6.4 summarizes these results, which are qualitatively similar for other traffic scenarios. We see that YAZ produces estimates more quickly, thus producing many more estimates over the duration of the experiment. Pathload and YAZ operate in an iterative fashion, and we see from the table that YAZ, on average, requires fewer cycles to arrive at an estimate.

6.4: Comparison of number of estimates produced, latency, number of packets emitted per iteration (Pathload and YAZ), and average number of packets emitted per estimate for each ABET for web-like traffic in topology 1.

	Estimates Produced	Latency μ (σ) (seconds)	Iterations per Estimate μ (σ)	Mean Pkts per Estimate
Pathload ($K = 100$)	96	17.7 (3.8)	8.4 (4.8)	10080
Pathload ($K = 50$)	97	17.6 (3.8)	8.8 (4.2)	5280
Spruce	156	10.9 (0.9)	NA	200
YAZ	446	3.8 (1.5)	6.1 (8.8)	366

Considering tool parameters and the mean number of iterations, we arrive at the mean number of packets required for each estimate. For much higher accuracy, YAZ uses packets roughly of the same order of magnitude as Spruce, but at least an order of magnitude fewer packets than Pathload. If Pathload and Spruce represent a tradeoff between measurement accuracy and overhead, our results for YAZ suggest that this tradeoff is not fundamental. For a Spruce-like budget, YAZ is more accurate than Pathload, sometimes significantly so.

6.3.5 Limitations of YAZ

There are some limitations to YAZ that we have yet to fully examine. First, we do not yet have a complete understanding of how to set the ζ threshold parameter and its sensitivity to particular operating system and hardware characteristics and cross traffic conditions. Second, since we use the mean spacing measured at sender and receiver, we cannot detect intra-stream indications of congestion that may be "washed out" over the duration of the stream. We can only detect either persistent expansion or compression of a probe stream. Given our current understanding of the nature of the errors introduced by commodity end hosts, we may not be able to do significantly better. Third, the initial minimum value of g_{in} is specified by the user. Determining how best to automatically set this parameter for a range of environments is an area for future work. Finally, our calibration study has focused on average AB over a time interval. Pathload reports a variation range for AB, and Jain *et al.* [122, 123] have shown that the variation of AB is an important measurement target. Extending our calibration study to consider AB variation is a subject for future work.

Chapter 7

Calibration Case Study 2: Measurement of Packet Loss

> Network parameters (*e.g.* delay, loss, and throughput) are relatively easy to measure over an end-to-end path.
> —K.G. Anagnostakis and M.B. Greenwald [49].

7.1 Overview

Measuring and analyzing network traffic dynamics between end hosts has provided the foundation for the development of many different network protocols and systems. Of particular importance is understanding packet loss behavior since loss can have a significant impact on the performance of both TCP- and UDP-based applications. Consequently, packet loss rate is one of the metrics typically specified as part of service level guarantees (*cf.*, Chapter 2, Section 2.2.3.2). Despite efforts of network engineers and operators to limit loss, it will probably never be eliminated due to the intrinsic dynamics and scaling properties of traffic in packet switched network [138]. Network operators have the ability to passively monitor nodes within their network for packet loss on routers using SNMP. End-to-end active measurements using probes provide an equally valuable perspective since they indicate the conditions that application traffic is experiencing on those paths.

The most commonly used tools for probing end-to-end paths to measure packet loss resemble the ubiquitous `ping` utility. `Ping`-like tools send probe packets (*e.g.,* ICMP echo packets) to a target host at fixed intervals. Loss is inferred by the sender if the response packets expected from the target host are not received within a specified time period. Generally speaking, an active measurement approach is problematic because of the discrete *sampling* nature of the probe process. Thus, the accuracy of the resulting measurements depends both on the characteristics and interpretation of the sampling process as well as the characteristics of the underlying loss process (see, *e.g.*, Roughan's recent work [182]).

Despite their widespread use, there is almost no mention in the literature of how to tune and calibrate active measurements of packet loss to improve accuracy or how to best interpret the resulting measurements. One approach is suggested by the well-known PASTA principle [213] which, in a networking context, tells us that Poisson-modulated probes will provide unbiased time average measurements of a router queue's state. This idea has been recommended as

a foundation in IETF standard methods for active measurement of end-to-end delay and loss in RFCs 2679 and 2680, respectively [46, 47]. However, the asymptotic nature of PASTA means that when it is applied in practice, the higher moments of measurements must be considered to determine the validity of the reported results. A closely related issue is the fact that loss is typically a rare event in the Internet [218]. This reality implies either that measurements must be taken over a long time period, or that average rates of Poisson-modulated probes may have to be quite high in order to report accurate estimates in a timely fashion. However, increasing the mean probe rate may lead to the situation that the probes themselves skew the results. Thus, there are trade-offs in packet loss measurements between probe rate, measurement accuracy, impact on the path and timeliness of results.

The goal of this case study is to understand how to accurately measure loss characteristics on end-to-end paths with probes. In general, we are interested in two specific characteristics of packet loss: *loss episode frequency*, and *loss episode duration* [218]. Using a heuristic that we develop, the commonly referred-to notion of *loss rate* [67, 169, 218] (number of packets lost divided by total number of packet over a give time interval) can be accurately estimated. This chapter consists of four parts: (*i*) empirical evaluation of the currently prevailing approach of RFC 2680, (*ii*) development of estimation techniques for loss episode frequency and mean loss episode duration that are based on novel experimental design, novel probing techniques, and simple validation tests, (*iii*) empirical evaluation of this new methodology, and (*iv*), extending our methodology with a heuristic for estimating end-to-end loss rate.

We begin by testing RFC 2680-compliant Poisson-modulated probing in our laboratory calibration environment consisting of commodity workstations separated by a series of commercial IP routers. Background traffic is sent between end hosts at different levels of intensity to generate loss episodes thereby enabling repeatable tests over a range of conditions. Our tests reveal two important deficiencies with standard RFC 2680 probing. First, individual probes often incorrectly report the absence of a loss episode (*i.e.,* they are successfully transferred when a loss episode is underway). Second, they are not well suited to measure loss episode durations over limited measurement periods.

Our observations about the weaknesses in the standard Poisson probing approach motivate the second part of our study: the development of a new approach for end-to-end loss measurement that includes three key elements. First, we design a probe process that is geometrically distributed and that assesses the likelihood of loss experienced by other flows that use the same path, rather than merely reporting its own packet losses. The probe process assumes FIFO queues along the path with a drop-tail policy. Second, we design a new experimental framework with estimation techniques that directly estimate the mean duration of the loss episodes without estimating the duration of any individual loss episode. Our estimators are proved to be consistent under mild assumptions of the probing process. Third, we provide simple validation tests (that require no additional experimentation or data collection) for some of the statistical assumptions that underly our analysis.

The third part of our study involves the empirical evaluation of our new loss measurement methodology. To this end, we developed a one-way active measurement tool called BADABING. BADABING sends fixed-size probes at specified intervals from one measurement host to a collaborating target host. The target system collects the probe

packets and reports the loss characteristics after a specified period of time. We also compare BADABING with a standard tool for loss measurement that conforms to RFC 2680 and emits probe packets at Poisson intervals. The results show that our tool reports loss episode estimates much more accurately for the same number of probes. We also show that BADABING estimates converge to the underlying loss episode frequency and duration characteristics.

Based on BADABING's methodology for estimating loss episode frequency and duration, we developed a heuristic approach for estimating end-to-end loss rate, which forms the fourth part of our study. Loss rate is an important quantity to measure accurately since it is a metric typically specified in SLAs. We show that BADABING estimates the end-to-end loss rate with high accuracy and with good confidence bounds. For example, in a scenario using self-similar background traffic, the true loss rate over a 15 minute period is 0.08% and the BADABING estimate is 0.07%. In contrast, the standard RFC 2680 method for estimating loss rate can have errors of more than two orders of magnitude.

Through the experiments and results of this case study we demonstrate the benefits and potential of our laboratory calibration approach. In particular, the use of commercial routers, a scalable traffic generation system, and a highly accurate ground-truth measurement system enabled critical insights into packet-level behavior that led to a significantly more accurate measurement methodology. An important outcome of this case study is that there is now a methodology and tool available for wide-area studies of packet loss characteristics, enabling researchers to understand and specify the trade-offs between accuracy and impact. Furthermore, the tool is self-calibrating in the sense that it can report when estimates are poor. Practical applications could include its use for path selection in peer-to-peer overlay networks and as a tool for network operators to monitor specific segments of their infrastructures.

7.2 Definitions of Loss Characteristics

There are many factors that can contribute to packet loss in the Internet. We describe some of these issues in detail as a foundation for understanding our active measurement objectives. The environment that we consider is modeled as a set of N flows that pass through a router R and compete for a single output link with bandwidth B_{out} as depicted in Figure 7.1a. The aggregate input bandwidth (B_{in}) must be greater than the shared output link (B_{out}) in order for loss to take place. The mean round trip time for the N flows is M seconds. Router R is configured with Q bytes of packet buffers to accommodate traffic bursts, with Q typically sized on the order of $M \times B$ [51, 205]. We assume that the queue operates in a FIFO manner, that the traffic includes a mixture of short- and long-lived TCP flows as is common in today's Internet, and that the value of N will fluctuate over time.

Figure 7.1b is an illustration of how the occupancy of the buffer in router R might evolve. When the aggregate sending rate of the N flows exceeds the capacity of the shared output link, the output buffer begins to fill. This effect is seen as a positive slope in the queue length graph. The rate of increase of the queue length depends both on the number N and on sending rate of each source. A *loss episode* begins when the aggregate sending rate has exceeded B_{out} for a period of time sufficient to load Q bytes into the output buffer of router R (*e.g.,* at times a and c in Figure 7.1b). A loss episode ends when the aggregate sending rate drops below B_{out} and the buffer begins a consistent drain down to

zero (*e.g.,* at times b and d in Figure 7.1b). This typically happens when TCP sources sense a packet loss and halve their sending rate, or simply when the number of competing flows N drops to a sufficient level. In the former case, the duration of a loss episode is related to M, depending whether loss is sensed by a timeout or fast retransmit signal. We define *loss episode duration* as the difference between start and end times (*i.e.,* $b-a$ and $d-c$). While this definition and model for loss episodes is somewhat simplistic and dependent on well behaved TCP flows, it is important for any measurement method to be robust to flows that do not react to congestion in a TCP-friendly fashion.

This definition of loss episodes can be considered a "router-centric" view since it says nothing about when any one end-to-end flow (including a probe stream) actually loses a packet or senses a lost packet. This contrasts with most of the prior work discussed in Chapter 2 which consider only losses of individual or groups of probe packets. In other words, in our methodology, a loss episode begins when the probability of some packet loss becomes positive. During the episode, there might be transient periods during which packet loss ceases to occur, followed by resumption of some packet loss. The episode ends when the probability of packet loss stays at 0 for a sufficient period of time (longer than a typical RTT). Thus, we offer two definitions for *packet loss rate*:

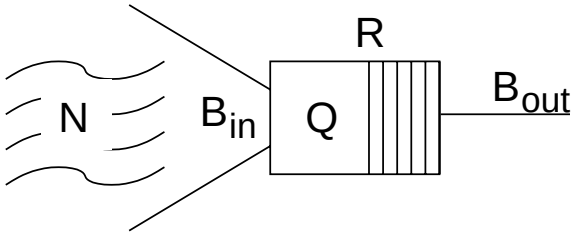
- **Router-centric loss rate.** With L the number of dropped packets on a given output link on router R during a given period of time, and S the number of all successfully transmitted packets through the same link over the same period of time, we define the router-centric loss rate as $L/(S+L)$. Note that with active measurements, it is not clear how to estimate router-centric loss rate. The reason is that demand $(S+L)$ during congestion periods can be, in essence, arbitrarily large.

- **End-to-end loss rate.** We define end-to-end loss rate in exactly the same manner as router-centric loss-rate, with the caveat that we only count packets that belong to a specific flow of interest.

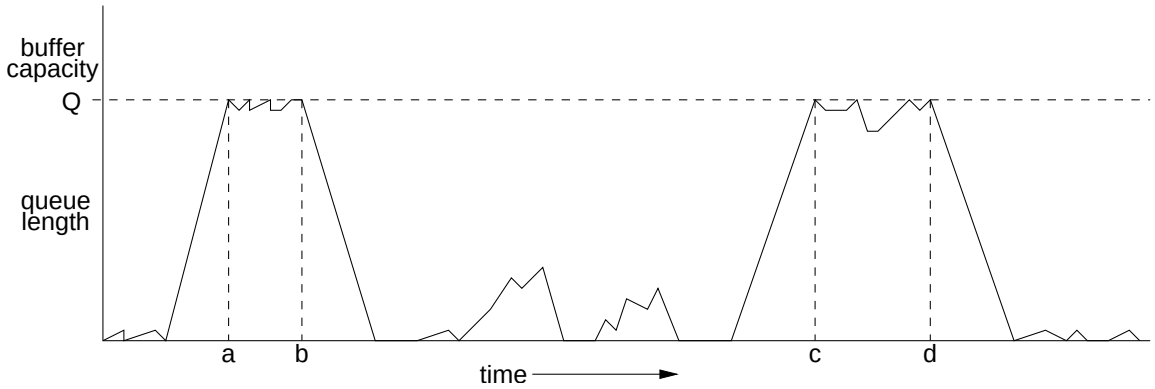
It is important to distinguish between these two notions of loss rate since packets are transmitted at the maximum rate B_{out} during loss episodes. The result is that during a period where the router-centric loss rate is non-zero, there may be flows that do not lose any packets and therefore have end-to-end loss rates of zero. This observation is central to our study and bears directly on the design and implementation of active measurement methods for packet loss.

As a consequence, an important consideration of our probe process described below is that it must deal with instances where individual probes do not accurately report loss. We therefore distinguish between the *true loss episode state* and the *probe-measured* or *observed state*. The former refers to the router-centric or end-to-end congestion state, given intimate knowledge of buffer occupancy, queuing delays, and packet drops, *e.g.*, information implicit in the queue length graph in Figure 7.1b. Ideally, the probe-measured state reflects the true state of the network. That is, a given probe P_i should accurately report the following:

$$P_i = \begin{cases} 0: & \text{if a loss episode is not encountered} \\ 1: & \text{if a loss episode is encountered} \end{cases} \qquad [7.1]$$



(a) Simple system model. N flows on input links with aggregate bandwidth B_{in} compete for a single output link on router R with bandwidth B_{out} where $B_{in} > B_{out}$. The output link has Q seconds of buffer capacity.



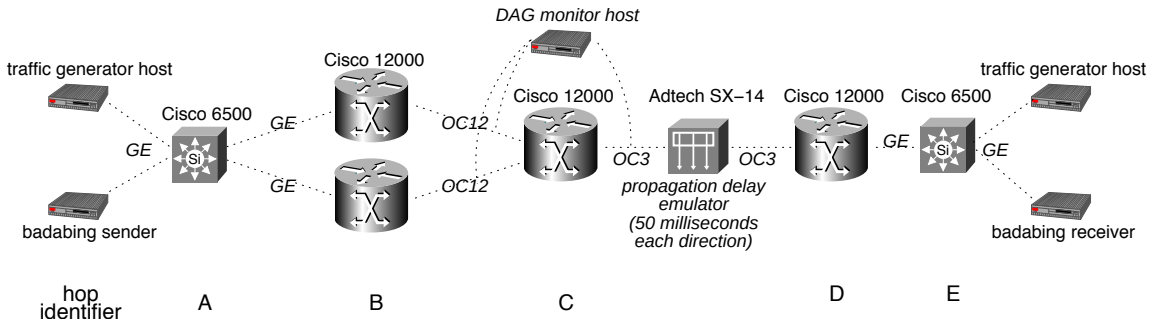
(b) Example of the evolution of the length of a queue over time. The queue length grows when aggregate demand exceeds the capacity of the output link. Loss episodes begin (points a and c) when the maximum buffer size Q is exceeded. Loss episodes end (points b and d) when aggregate demand falls below the capacity of the output link and the queue drains to zero.

7.1: Simple system model and example of loss characteristics under consideration.

Satisfying this requirement is problematic because, as noted above, many packets are successfully transmitted during loss episodes. We address this issue in our probe process model and heuristically, as described below.

Finally, we define a *probe* to consist of one or more very closely spaced (*i.e.*, back-to-back) packets. Note that all other loss measurement tools of which we are aware, *e.g.*, `ping`, implicitly define a probe as a single packet. As we will see in our evaluation and results below, the reason for using multi-packet probes is that not all packets passing through a congested link are subject to loss; constructing probes of multiple packets enables a more accurate determination to be made.

7.3 Laboratory Calibration Testbed



7.2: Laboratory testbed. Cross traffic scenarios consisted of constant bit rate traffic, long-lived TCP flows, and web-like bursty traffic. Cross traffic flowed across one of two routers at hop B, while probe traffic flowed through the other. Optical splitters connected Endace DAG 3.5 and 3.8 passive packet capture cards to the testbed between hops B and C, and hops C and D. Probe traffic flowed from left to right and the loss episodes occurred at hop C.

The laboratory testbed used in our experiments is shown in Figure 7.2. It consisted of commodity end hosts connected to a dumbbell-like topology comprised of Cisco GSR 12000 routers. Both probe and background traffic were generated and received by the end hosts. Traffic flowed from the sending hosts on separate paths via Gigabit Ethernet to separate Cisco GSRs (hop B in the figure) where it transitioned to OC-12 (622 Mb/s) links. This configuration was created in order to accommodate our measurement system, described below. Probe and background traffic was then multiplexed onto a single OC-3 (155 Mb/s) link (hop C in the figure) which formed the bottleneck where loss episodes took place. We used a hardware-based propagation delay emulator on the OC-3 link to add 50 milliseconds delay in each direction for all experiments[1], and configured the bottleneck queue to hold approximately 100 milliseconds of packets. Packets exited the OC-3 link via another Cisco GSR 12000 (hop D in the figure) and passed to receiving hosts via Gigabit Ethernet.

[1]Note that at the time these experiments were run, NETPATH was under development. For the final experiments described in this chapter examining the packet loss heuristic, NETPATH was used in place of the Adtech SX-14.

The probe and traffic generator hosts consisted of identically configured workstations running Linux 2.4. The workstations had 2 GHz Intel Pentium 4 processors with 2 GB of RAM and Intel Pro/1000 network cards. They were also dual-homed, so that all management traffic was on a separate network than depicted in Figure 7.2.

At the heart of our laboratory calibration environment is the passive measurement system used to establish true loss episode state. Optical splitters were attached to both the ingress and egress links at hop C and Endace DAG 3.5 and 3.8 passive monitoring cards were used to capture traces of packets entering and leaving the bottleneck node. By comparing packet header information, we were able to identify exactly which packets were lost at the congested output queue during experiments. Furthermore, the fact that the measurements of packets entering and leaving hop C were time-synchronized on the order of a single microsecond enabled us to easily infer the queue length and how the queue was affected by probe traffic during all tests.

7.4 Evaluation of Simple Poisson Probing for Packet Loss

We begin by using our laboratory testbed to evaluate the capabilities of Poisson-modulated loss probe measurements using the RFC 2680-compliant `zing` tool [42, 143]. `Zing` measures packet delay and loss in one direction on an end-to-end path. The `zing` sender emits UDP probe packets at Poisson-modulated intervals with timestamps and unique sequence numbers and the receiver logs the probe packet arrivals. Users specify the mean probe rate λ, the probe packet size, the number of packets in a "flight", and the distribution of time intervals between flights.

To evaluate standard Poisson probing, we configured `zing` using the same parameters as in Zhang *et al.* [218]. Namely, we ran two tests, one with $\lambda = 100ms$ (10 Hz) and 256 byte payloads and another with $\lambda = 50ms$ (20Hz) and 64 byte payloads. To determine the duration of our experiments below, we selected a period of time that should limit the variance of the loss rate estimator $\bar{X}$ where $Var(\bar{X}_n) \approx \frac{p}{n}$ for loss rate p and number of probes n.

We conducted three separate experiments in our evaluation of `zing`. In each test we measured both the frequency and duration of packet loss episodes. Again, we used the definition in [218] for loss episode: "a series of consecutive packets (possibly only of length one) that were lost."

The first experiment used 40 infinite TCP sources with receive windows set to 256 full size (1500 bytes) packets. Figure 7.3a shows the time series of the queue occupancy for a portion of the experiment; the expected synchronization behavior of TCP sources in congestion avoidance is clear. The experiment was run for a period of 15 minutes which should have enabled `zing` to measure loss rate with standard deviation within 10% of the mean [62, 63].

Results from the experiment with infinite TCP sources are shown in Table 7.1. The table shows that `zing` performs poorly in measuring both loss frequency and duration in this scenario. For both probe rates, there were no instances of consecutive lost packets, which explains the inability to estimate loss episode duration.

In the second set of experiments, we used `Iperf` to create a series of (approximately) constant duration (about 68 milliseconds) loss episodes that were spaced randomly at exponential intervals with mean of 10 seconds over a 15 minute period. The time series of the queue length for a portion of the test period is shown in Figure 7.3b.

7.1: Results from `zing` (RFC 2680) experiments with infinite TCP sources.

	frequency	duration mean (std. dev.) (seconds)
true values	0.0265	0.136 (0.009)
`zing` (RFC 2680) (10Hz)	0.0005	0 (0)
`zing` (RFC 2680) (20Hz)	0.0002	0 (0)

7.2: Results from `zing` (RFC 2680) experiments with randomly spaced, constant duration loss episodes.

	frequency	duration mean (std. dev.) (seconds)
true values	0.0069	0.068 (0.000)
`zing` (RFC 2680) (10Hz)	0.0036	0.043 (0.001)
`zing` (RFC 2680) (20Hz)	0.0031	0.050 (0.002)

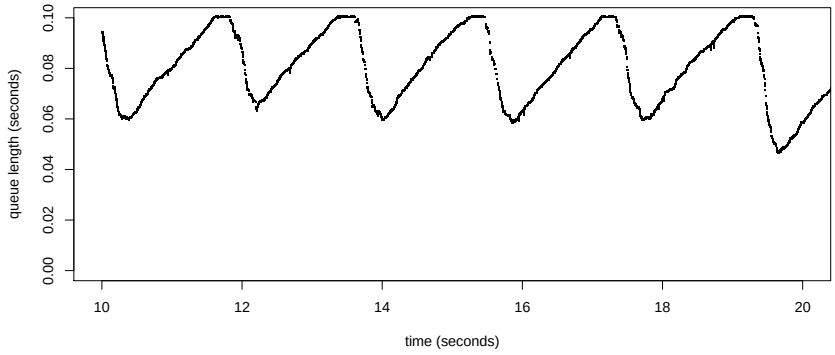
Results from the experiment with randomly spaced, constant duration loss episodes are shown in Table 7.2. The table shows that `zing` measures loss frequencies and durations that are closer to the true values.

In the final set of experiments, we used HARPOON to create a series of loss episodes that approximate loss resulting from a mix of web-like and peer-to-peer traffic. HARPOON was configured to briefly increase its load in order to induce packet loss, on average, every 20 seconds. The variability of traffic produced by HARPOON complicates delineation of loss episodes. To establish baseline loss episodes to compare against, we found trace segments where the first and last events were packet losses, and queuing delays of all packets between those losses were above 90 milliseconds (within 10 milliseconds of the maximum). We ran this test for 15 minutes and a portion of the time series for the queue length is shown in Figure 7.3c.

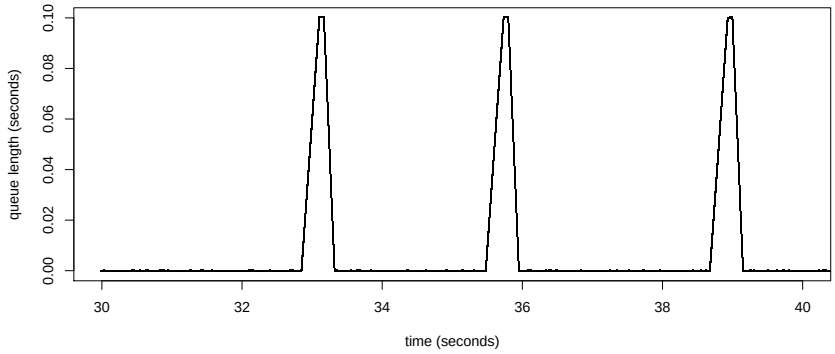
Results from the experiment with the bursty HARPOON traffic are shown in Table 7.3. For measuring loss frequency, neither probe rate results in a close match to the true frequency. For loss episode duration, the results are also poor. For the 10 Hz probe rate, there were no consecutive losses measured, and for the 20 Hz probe rate, there were only two instances of consecutive losses, each of exactly two lost packets.

7.5 Probe Process Model

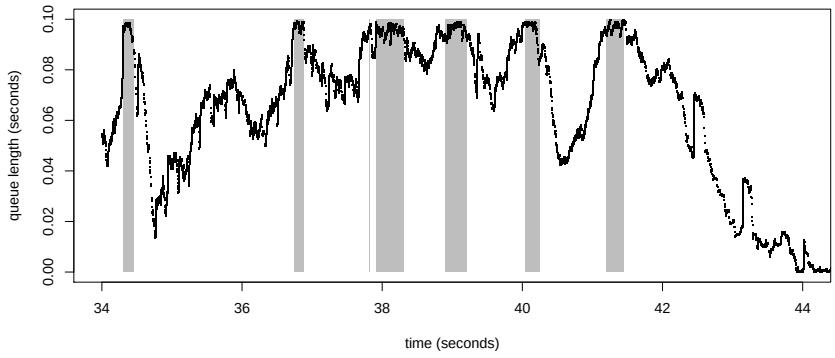
The results from our experiments described in the previous section show that the standard RFC 2680 Poisson probing methodology is generally poor for measuring loss episode frequency and loss episode duration. These results, along with deeper investigation of the reasons for particular deficiencies in loss episode duration measurement, form the foundation for a new measurement process.



(a) Queue length time series for a portion of the experiment with 40 infinite TCP sources.



(b) Queue length time series for a portion of the experiment with randomly spaced, constant duration loss episodes.



(c) Queue length time series for a portion of the experiment with HARPOON web-like traffic. Time segments in grey indicate loss episodes.

7.3: Queue length time series plots for three different background traffic scenarios.

7.3: Results from `zing` (RFC 2680) experiments with HARPOON web-like traffic.

	frequency	duration mean (std. dev.) (seconds)
true values	0.0093	0.136 (0.009)
`zing` (RFC 2680) (10Hz)	0.0014	0 (0)
`zing` (RFC 2680) (20Hz)	0.0012	0.022 (0.001)

7.5.1 General Setup

Our methodology involves dispatching a sequence of probes, each consisting of one or more very closely spaced packets. The aim of a probe is to obtain a snapshot of the congestion state of the network at the instant of probing. As such, the record for each probe indicates whether or not it encountered a loss episode, as evidenced by either the loss or sufficient delay of any of the packets within a probe (we elaborate these details below). The reason for using multi-packet probes is that not all packets passing through a congested link are subject to loss; using multiple packets enables a more accurate determination to be made.

The probes themselves are organized into what we term *basic experiments*, each of which comprises a number of probes sent in rapid succession. The aim of the basic experiment is to determine the dynamics of transitions between the congested and uncongested state of the network, *i.e.*, beginnings and endings of loss episodes. Below we show how this enables us to estimate the duration of loss episodes.

A *full experiment* comprises a sequence of basic experiments generated according to some rule. The sequence may be terminated after some specified number of basic experiments, or after a given duration, or in an open-ended adaptive fashion, *e.g.,* until estimates of desired accuracy for a loss characteristic have been obtained, or until such accuracy is determined impossible.

We formulate the probe process as a discrete-time process. This decision is not a fundamental limitation: since we are concerned with measuring loss episode dynamics, we need only ensure that the interval between the discrete time slots is smaller than the time scales of the loss episodes. A *congested slot* is simply a time slot during which congestion occurs. A *congestion episode* (alternatively *loss episode*) is a maximal set of consecutive slots that are congested.

There are three steps in the explanation of our loss measurement method (*i.e.*, the experimental design and the subsequent estimation). First, we present the *basic algorithm* version. This model is designed to provide estimators of the frequency of time slots in which loss episodes is present, and the duration of loss episodes. The frequency estimator is unbiased, and under relatively weak statistical assumptions, both estimators are consistent in the sense they converge to their respective true values as the number of measurements grows.

Second, we describe the *improved algorithm* version of our design which provides loss episode estimators under weaker assumptions, and requires that we employ a more sophisticated experimental design. In this version of the

model, we insert a mechanism to estimate, and thereby correct, the possible bias of the estimators from the basic design.

Third, we describe simple validation techniques that can be used to assign a level of confidence to loss episode estimates. This enables open-ended experimentation with a stopping criterion based on estimators reaching a requisite level of confidence.

7.5.2 Basic Algorithm

For each time slot i we decide whether or not to commence a basic experiment; this decision is made independently for each slot with some fixed probability p over all slots. In this way, the sequence of basic experiments follows a geometric distribution with parameter p. (In practice, we make the restriction that we do not start a new basic experiment while one is already in progress. This implies that, in reality, the random variables controlling whether or not a probe is sent at time slot i are not entirely independent of each other.) We indicate this series of decisions through random variables $\{x_i\}$ that take the value 1 if "a basic experiment is started in slot i" and 0 otherwise.

If $x_i = 1$, we dispatch *two* probes to measure congestion in slots i and $i+1$. The random variable y_i records the reports obtained from the probes as a 2-digit binary number, *i.e.*, $y_i = 00$ means "both probes did not observe a loss episode", while $y_i = 10$ means "the first probe observed a loss episode while the second one did not", and so on. Our methodology is based on the following fundamental assumptions, which, in view of the probe and its reporting design (as described below) are very likely to be valid ones. *These assumptions are required in both algorithmic versions.* The basic algorithm requires a stronger version of these assumptions, as we detail later.

7.5.2.1 Assumptions

We do **not** assume that the probes accurately report loss episodes: we allow that a true loss episode present during a given time slot may not be observed by any of the probe packets in that slot. However, we do assume a specific structure of the inaccuracy, as follows.

Let Y_i be the *true loss episode state* in slots i and $i+1$, *i.e.,* $Y_i = 01$ means that there is no loss episode present at $t = i$ and that a loss episode is present at $t = i+1$. As described in our definitions above, *true* means the congestion that would be observed were we to have knowledge of router buffer occupancy, queuing delays and packet drops. Of course, in practice the value of Y_i is unknown. Our specific assumption is that y_i is correct, *i.e.*, equals Y_i, with probability p_k that is independent of i and depends only on the number k of 1-digits in Y_i. Moreover, if y_i is incorrect, it must take the value 00. Explicitly,

(1) If $Y_i = 00$ (= no loss episode occurring) then $y_i = 00$, too (= no congestion reported), with probability 1.

(2) If $Y_i = 01$ (= loss episode begins), or $Y_i = 10$ (= loss episode ends), then $\mathsf{P}(y_i = Y_i | (Y_i = 01) \cup (Y_i = 10)) = p_1$, for some p_1 which is independent of i. If y_i fails to match Y_i, then necessarily, $y_i = 00$.

(3) If $Y_i = 11$ (= loss episode is on-going), then $\mathsf{P}(y_i = Y_i | Y_i = 11) = p_2$, for some p_2 which is independent of i. If y_i fails to match Y_i, then necessarily, $y_i = 00$.

As justification for the above assumptions we first note that it is unlikely that a probe will spuriously measure loss. That is, assuming well-provisioned measurement hosts, if no loss episode is present a probe should not register loss. In particular, for assumptions (1) and (2), if $y_i \neq Y_i$, it follows that y_i must be 00. For assumption (3), we appeal to the one-way delay heuristics developed below: if $y_i \neq 00$, then we hold in hand at least one probe that reported loss; by comparing the delay characteristics of that probe to the corresponding characteristics in the other probe (assuming that the other one did not report loss), we are able to deduce whether to assign a value 1 or 0 to the other probe. Thus, the actual networking assumption is that the delay characteristics over the measured path are stationary relative to the time discretization we use. We provide implementation details on this part below. Specifically, assuming FIFO queuing and a corresponding correlation between loss and delay, it is unlikely that a probe does not *at least* experience high delay during a true loss episode.

7.5.2.2 Estimation

The basic algorithm assumes that $p_1 = p_2$ for consistent duration estimation, and $p_1 = p_2 = 1$ for consistent and unbiased frequency estimation. The estimators are as follows:

Loss Episode Frequency Estimation. Denote the true frequency of slots during which a loss episode is present by F. We define a random variable z_i whose value is the first digit of y_i. Our estimate is then

$$\widehat{F} = \sum_i z_i / M, \qquad [7.2]$$

with the index i running over all the basic experiments we conducted, and M is the total number of such experiments.

This estimator is unbiased, $\mathsf{E}[\widehat{F}] = F$, since the expected value of z_i is just the congestion frequency F. Under mild conditions (*i.e.*, $p_1 = p_2 = 1$), the estimator is also consistent. For example, if the durations of the loss episodes and loss-free episodes are independent with finite mean, then the proportion of lossy slots during an experiment over N slots converges almost surely, as N grows, to the loss episode frequency F, from which the stated property follows.

Loss Episode Duration Estimation is more sophisticated. Recall that a loss episode is one consecutive occurrence of k lossy time slots preceded and followed by no loss, *i.e.,* its binary representation is written as:

$$01\ldots10.$$

Suppose that we have access to the true loss episode state at all possible time slots in our discretization. We then count all loss episodes and their durations and find out that for $k = 1, 2, \ldots$, there were exactly j_k loss episodes of length k. Then, loss occurred over a total of

$$A = \sum_k k j_k$$

slots, while the total number of loss episodes is

$$B = \sum_k j_k.$$

The average duration D of a loss episode is then defined as

$$D := A/B.$$

In order to estimate D, we observe that, with the above structure of loss episodes in hand, there are exactly B time slots i for which $Y_i = 01$, and there are also B time slots i for which $Y_i = 10$. Also, there are exactly $A+B$ time slots i for which $Y_i \neq 00$. We therefore define

$$R := \#\{i : y_i \in \{01, 10, 11\}\},$$

and

$$S := \#\{i : y_i \in \{01, 10\}\}.$$

Now, let N be the total number of time slots. Then $\mathsf{P}(Y_i \in \{01,10\}) = 2B/N$, hence $\mathsf{P}(y_i \in \{01,10\}) = 2p_1B/N$. Similarly, $\mathsf{P}(Y_i \in \{01,10,11\}) = (A+B)/N$, and $\mathsf{P}(y_i \in \{01,10,11\}) = (p_2(A-B)+2p_1B)/N$. Thus,

$$\mathsf{E}(R)/\mathsf{E}(S) = \frac{p_2(A-B)+2p_1B}{2p_1B}.$$

Denoting $r := p_2/p_1$, we get then

$$\mathsf{E}(R)/\mathsf{E}(S) = \frac{r(A-B)+2B}{2B} = \frac{rA}{2B} - r/2 + 1.$$

Thus,

$$D = \frac{2}{r} \times \left(\frac{\mathsf{E}(R)}{\mathsf{E}(S)} - 1 \right) + 1. \qquad [7.3]$$

In the basic algorithm we assume $r = 1$, the estimator $\widehat{D}$ of D is then obtained by substituting the measured values of S and R for their means:

$$\widehat{D} := 2 \times \frac{R}{S} - 1. \qquad [7.4]$$

Note that this estimator is *not* unbiased for finite N, due to the appearance of S in the quotient. However, it is consistent under the same conditions as those stated above for $\widehat{F}$, namely, that congestion is described by an alternating renewal process with finite mean lifetimes. Then the ergodic theorem tells us that as N grows, R/N and S/N converge to their expected values (note, *e.g.*, $\mathsf{E}[R/N] = p\ \mathsf{P}[Y_i \in \{01,10,11\}]$ independent of N) and hence $\widehat{D}$ converges almost surely to D.

7.5.3 Improved Algorithm

The improved algorithm is based on weaker assumptions than the basic algorithm: we no longer assume that $p_1 = p_2$. In view of the details provided so far, we will need, for the estimation of duration, to know the ratio $r := p_1/p_2$. For that, we modify our basic experiments as follows.

As before, we decide independently at each time slot whether to conduct an experiment. With probability 1/2, this is a basic experiment as before; otherwise we conduct an *extended experiment* comprising *three* probes, dispatched in slots $i, i+1, i+2$, and redefine y_i to be the corresponding 3-digit number returned by the probes, *e.g.,* $y_i = 001$ means "loss was observed only at $t = i+2$", etc. As before Y_i records the true states that our ith experiment attempts to identify. We now make the following additional assumptions.

Additional Assumptions

We assume that the probability that y_i misses the true state Y_i (and hence records a string of 0's), does not depend on the length of Y_i but only on the number of 1's in the string. Thus, $\mathsf{P}(y_i = Y_i) = p_1$ whenever Y_i is any of $\{01, 10, 001, 100\}$, while $\mathsf{P}(y_i = Y_i) = p_2$ whenever Y_i is any of $\{11, 011, 110\}$ (we address states 010 and 101 below). We claim that these additional assumptions are realistic, but defer the discussion until after we describe the reporting mechanism for loss episodes.

With these additional assumptions in hand, we denote

$$U := \#\{i : y_i \in \{011, 110\}\},$$

and

$$V := \#\{i : y_i \in \{001, 100\}\}.$$

The combined number of states $011, 110$ in the full time series is $2B$, while the combined number of states of the form $001, 100$ is also $2B$. Thus, we have

$$\frac{\mathsf{E}(U)}{\mathsf{E}(V)} = r,$$

hence, with U/V estimating r, we employ (Equation 7.3) to obtain

$$\widehat{D} := \frac{2V}{U} \times \left(\frac{R}{S} - 1\right) + 1.$$

7.5.4 Validation

When running an experiment, our assumptions require that several quantities have the same mean. We can validate the assumptions by checking those means.

In the basic algorithm, the probability of $y_i = 01$ is assumed to be the same as that of $y_i = 10$. Thus, we can design a stopping criterion for on-going experiments based on the ratio between the number of 01 measurements and the number of 10 measurements. A large discrepancy between these numbers (that is not bridged by increasing M) is an indication that our assumptions are invalid. Note that this validation does not check whether $r = 1$ or whether $p_1 = 1$, which are two important assumptions in the basic design.

In the improved design, we expect to get similar occurrence rate for each of $y_i = 01, 10, 001, 100$. We also expect to get similar occurrence rate for $y_i = 011, 110$. We can check those rates, stop whenever they are close, and invalidate

the experiment whenever the mean of the various events do not coincide eventually. Also, each occurrence of $y_i = 010$ or $y_i = 101$ is considered a violation of our assumptions. A large number of such events is another reason to reject the resulted estimations. We leave experimental investigation of stopping criteria for future work.

7.5.5 Modifications

There are various straightforward modifications to the above design that we do not address in detail at this time. For example, in the improved algorithm, we have used the triple-probe experiments only for the estimation of the parameter r. We could obviously include them also in the actual estimation of duration, thereby decreasing the total number of probes that are required in order to achieve the same level of confidence.

Another obvious modification is to use unequal weighing between basic and extended experiments. In view of the expression we obtain for $\widehat{D}$ there is no clear motivation for doing that: a miss in estimating V/U is as bad as a corresponding miss in R/S (unless the average duration is very small). Basic experiments incur less cost in terms of network probing load. On the other hand, if we use the reports from triple probes for estimating $\mathsf{E}(S)/\mathsf{E}(R)$ then we may wish to increase their proportion. Note that in our formulation, we cannot use the reported events $y_i = 111$ for estimating anything, since the failure rate of the reporting on the state $Y_i = 111$ is assumed to be unknown. (We could estimate it using similar techniques to those used in estimating the ratio p_2/p_1. This, however, will require utilization of experiments with more than three probes). A topic for further research is to quantify the trade-offs between probe load and estimation accuracy involved in using extended experiments of 3 or more probes.

7.6 Probe Tool Implementation and Evaluation

To evaluate the capabilities of our loss probe measurement process, we built a tool called BADABING[2] that implements the basic algorithm of our probe model described above. We then conducted a series of experiments with BADABING in our laboratory testbed with the same background traffic scenarios described above.

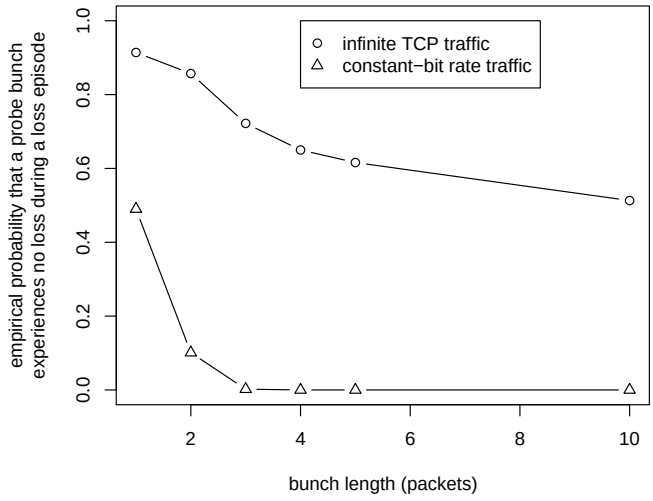
The objective of our lab-based experiments was to validate our modeling method and to evaluate the capability of BADABING over a range of loss conditions. We report results of experiments focused in three areas. While our probe process does not assume that we always receive true indications of loss from our probes, the accuracy of reported measurements will improve if probes more reliably indicate loss. With this in mind, the first set of experiments was designed to understand the ability of an individual probe (consisting of 1 to N tightly-spaced packets) to accurately report an encounter with a loss episode. Next, we examine the accuracy of BADABING in reporting loss episode frequency and duration for a range of probe rates and traffic scenarios. In our final set of experiments, we compare the capabilities of BADABING with simple Poisson-modulated probing.

[2]Named in the spirit of past tools used to measure loss including `ping`, `zing`, and Sting. This tool is approximately 800 lines of C++ and is available to the community for testing and evaluation at `http://wail.cs.wisc.edu/`.

7.6.1 Accurate Reporting of Loss Episodes by Probes

We noted in our definitions that, ideally, a probe should provide an accurate indication of the true loss episode state (Equation 7.1). However, this may not be the case. The primary issue is that during a loss episode, many packets continue to be successfully transmitted. Thus, we hypothesized that we might be able to increase the probability of probes correctly reporting a loss episode by increasing the number of packets in an individual probe. We also hypothesized that, assuming FIFO queuing, using one-way delay information could further improve the accuracy of individual probe measurements.

We investigated the first hypothesis in a series of experiments using the infinite TCP source background traffic and constant bit rate traffic described above. For the infinite TCP traffic, loss event durations were approximately 150 milliseconds. For the constant bit rate traffic, loss episodes were approximately 68 milliseconds in duration. We used a modified version of BADABING to generate probes at fixed intervals of 10 milliseconds so that some number of probes would encounter all loss episodes. We experimented with probes consisting of between 1 and 10 packets. Packets in an individual probe were sent back to back per the capabilities of the measurement hosts (*i.e.,* with approximately 30 microseconds between packets). Probe packet sizes were set at 600 bytes[3].



7.4: Results from tests of ability of probes consisting of N packets to report loss when an episode is encountered.

Figure 7.4 shows the results of these tests. We see that for the non-reactive UDP constant bit rate traffic, longer probes have a clear impact on the ability to detect loss. While about half of single-packet probes do not experience loss during a loss episode, probes with just a couple more packets are much more reliable indicators of the true loss

[3]This packet size was chosen to exploit an architectural feature of the Cisco GSR so that probe packets had as much impact on internal buffer occupancy as maximum-sized frames. Investigating the impact of packet size on estimation accuracy is a subject for future work.

episode state. For the infinite TCP traffic, there is also an improvement as the probes get longer, but the improvement is relatively small. Examination of the details of the queue behavior during these tests demonstrates why the 10 packet probes do not greatly improve loss reporting ability for the infinite source traffic. Since the TCP flows react to packet loss by reducing their rates, probes consisting of several packets tend to alter the loss process under measurement. As shown in Figure 7.5, longer probes begin to have a serious impact on the queuing dynamics during loss episodes.

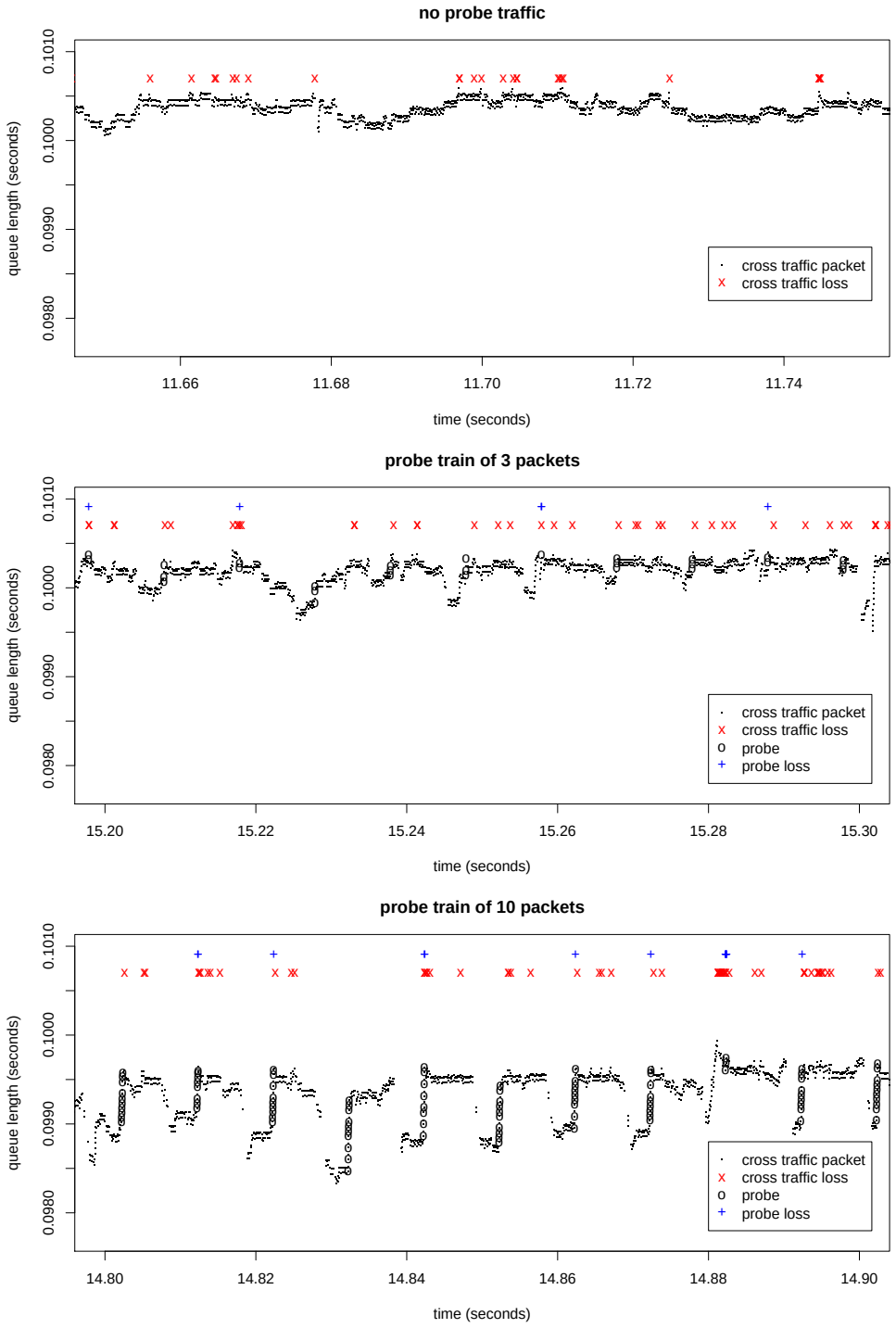
This observation, along with our hypothesis regarding one-way packet delays, led to our development of an alternative approach for identifying loss events. Our new method considers both individual packet loss with probes *and* the one-way packet delay as follows. For probes in which any packet is lost, we consider the one-way delay of the most recent successfully transmitted packet as an estimate of the maximum queue depth (OWD_{max}). We then consider a loss episode to be delimited by probes within τ seconds of an indication of a lost packet (*i.e.*, a missing probe sequence number) and having a one-way delay greater than $(1-\alpha) \times OWD_{max}$. Using the parameters τ and α, we mark probes as 0 or 1 according to Equation 7.1 and form estimates of loss episode frequency and duration using Equations 7.2 and 7.4, respectively. Note that even if packets of a given probe are not actually lost, the probe may be considered to have experienced a loss episode due to the α and/or τ thresholds.

This formulation of probe-measured loss assumes that queuing at intermediate routers is FIFO. Also, we can keep a number of estimates of OWD_{max}, taking the mean when determining whether a probe is above the $(1-\alpha) \times OWD$ threshold or not. Doing so effectively filters loss at end host operating system buffers or in network interface card buffers, since such losses are unlikely to be correlated with end-to-end network congestion and delays.

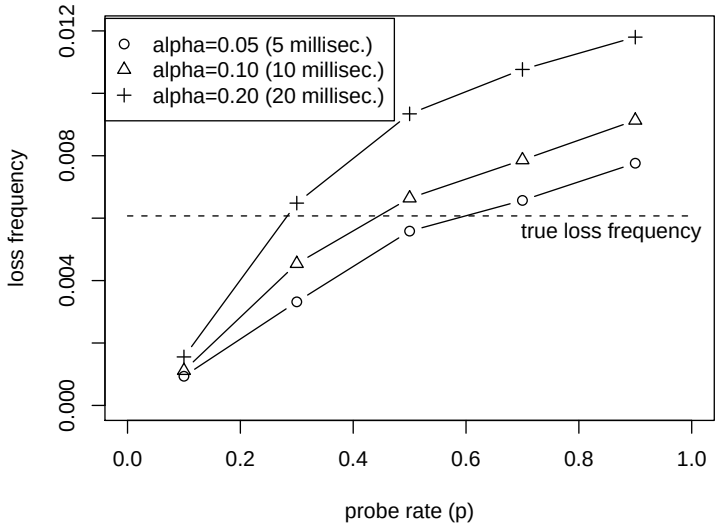
We conducted a series of experiments with constant bit rate traffic to assess the sensitivity of the loss threshold parameters. Using a range of values for probe send probability (p), we explored a cross product of values for α and τ. For α, we selected 0.025, 0.05, 0.10, and 0.20, effectively setting a high-water level of the queue of 2.5, 5, 10, and 20 milliseconds. For τ, we selected values of 5, 10, 20, 40, and 80 milliseconds. Figure 7.6a shows results for loss frequency for a range of p, with τ fixed at 80 milliseconds, and α varying between 0.05, 0.10, and 0.20 (equivalent to 5, 10, and 20 milliseconds). Figure 7.6b fixes α at 0.10 (10 milliseconds) while letting τ vary over 20, 40, and 80 milliseconds. We see, as expected, that with larger values of either threshold, estimated frequency increases. There are similar trends for loss duration (not shown). We also see that there is a trade-off between selecting a higher probe rate and more "permissive" (larger) thresholds. It appears that the best setting for τ comes around the expected time between probes plus one or two standard deviations. The best α appears to depend both on the probe rate and on the traffic process and level of multiplexing, which determines how quickly a queue can fill or drain. Considering these issues, we discuss parameterizing BADABING in general Internet settings below.

7.6.2 Measuring Frequency and Duration

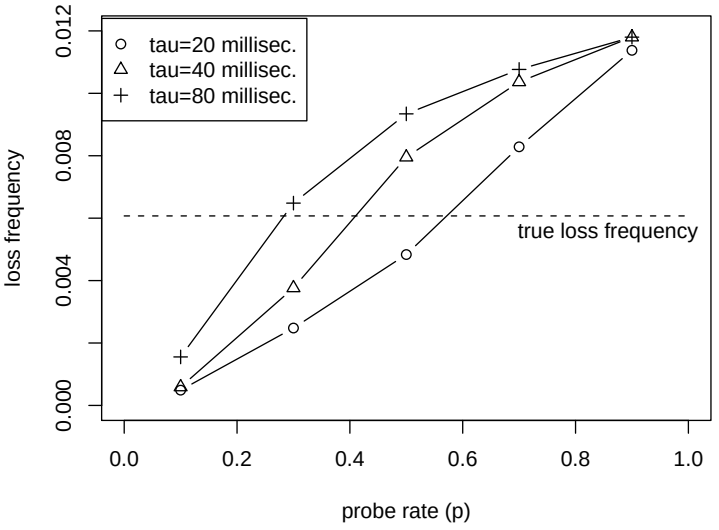
The formulation of our new loss probe process calls for the user to specify two parameters, N and p, where p is the probability of initiating a basic experiment at a given interval. In the next set of experiments, we explore BADABING's



7.5: Queue length during a portion of a loss episode for different size loss probes. The top plot shows infinite source TCP traffic with no loss probes. The middle plot shows infinite source TCP traffic with loss probes of three packets, and the bottom plots shows loss probes of 10 packets. Each plot is annotated with TCP packet loss events and probe packet loss events.



(a) Estimated loss frequency over a range of values for α while holding τ fixed at 80 milliseconds.



(b) Estimated loss frequency over a range of values for τ while holding α fixed at 0.1 (equivalent to 10 milliseconds).

7.6: Comparison of the sensitivity of loss frequency estimation to a range of values of α and τ.

ability to report loss episode frequency and duration accurately for a fixed N, and p using values of 0.1, 0.3, 0.5, 0.7, and 0.9 (implying that probe traffic consumed between 0.2% and 1.7% of the bottleneck link). With the time discretization set at 5 milliseconds, we fixed N for these experiments at 180,000, yielding an experiment duration of 900 seconds. We also examine the loss frequency and duration estimates for a fixed p of 0.1 and N of 720,000 from an hour-long experiment.

In these experiments, we used three different background traffic scenarios. In the first scenario, we used `Iperf` to generate random loss episodes at constant duration as described above. For the second, we modified `Iperf` to create loss episodes of three different durations (50, 100, and 150 milliseconds), with an average of 10 seconds between loss episodes. In the final traffic scenario, we used HARPOON to generate self-similar, web-like workloads as described above. For all traffic scenarios, BADABING was configured with probe sizes of 3 packets and with packet sizes fixed at 600 bytes. The three packets of each probe were sent back-to-back, according to the capabilities of our end hosts (approximately 30 microseconds between packets). For each probe rate, we set τ to the expected time between probes plus one standard deviation (namely, $\tau = \lceil \frac{1-p}{p} + \sqrt{\frac{1-p}{p^2}} \rceil$ time slots). For α, we used 0.2 for probe probability 0.1, 0.1 for probe probabilities of 0.3 and 0.5, and 0.05 for probe probabilities of 0.7 and 0.9.

For loss episode duration, results from our experiments described below confirm the validity of the assumption made in Section 7.5.4 that the probability $y_i = 01$ is very close to the probability $y_i = 10$. That is, we appear to be equally likely to measure in practice the beginning of a loss episode as we are to measure the end. We therefore use the mean of the estimates derived from these two values of y_i.

Table 7.4 shows results for the constant bit rate traffic with loss episodes of uniform duration. For values of p other than 0.1, the loss frequency estimates are close to the true value. For all values of p, the estimated loss episode duration was within 25% of the actual value.

Table 7.5 shows results for the constant bit rate traffic with loss episodes randomly chosen between (approximately) 50, 100, and 150 milliseconds. The overall result is very similar to the constant bit rate setup with loss episodes of uniform duration. Again, for values of p other than 0.1, the loss frequency estimates are close to the true values, and all estimated loss episode durations were within 25% of the true value.

Table 7.6 displays results for the setup using HARPOON web-like traffic to create loss episodes. Since HARPOON is designed to generate average traffic volumes over relatively long time scales, the actual loss episode characteristics over these experiments vary. For loss frequency, just as with the constant bit rate traffic scenarios, the estimates are quite close except for the case of $p = 0.1$. For loss episode durations, all estimates except for $p = 0.3$ fall within a range of 25% of the actual value. The estimate for $p = 0.3$ falls just outside this range.

In Tables 7.4 and 7.5 we see, over the range of p values, an increasing trend in loss frequency estimated by BADABING. This effect arises primarily from the problem of selecting appropriate parameters α and τ, and is similar in nature to the trends seen in Figures 7.6a and 7.6b. It is also important to note that these trends are peculiar to the open-loop (non-reactive) CBR traffic sources: such an increasing trend in loss frequency estimation does not exist for

7.4: BADABING loss estimates for constant bit rate traffic with loss episodes of uniform duration.

p	loss frequency		loss duration (seconds)	
	true	BADABING	true	BADABING
0.1	0.0069	0.0016	0.068	0.054
0.3	0.0069	0.0065	0.068	0.073
0.5	0.0069	0.0060	0.068	0.051
0.7	0.0069	0.0070	0.068	0.051
0.9	0.0069	0.0078	0.068	0.053

7.5: BADABING loss estimates for constant bit rate traffic with loss episodes of 50, 100, or 150 milliseconds.

p	loss frequency		loss duration (seconds)	
	true	BADABING	true	BADABING
0.1	0.0083	0.0023	0.097	0.034
0.3	0.0083	0.0076	0.097	0.076
0.5	0.0083	0.0098	0.097	0.090
0.7	0.0083	0.0102	0.097	0.074
0.9	0.0083	0.0105	0.097	0.059

the significantly more bursty HARPOON web-like traffic, as seen in Table 7.6. We also note that no such trend exists for loss episode duration estimates. Empirically, there are somewhat complex relationships among the choice of p, the selection of α and τ, and estimation accuracy. While we have considered a range of traffic conditions in a limited, but realistic setting, we have yet to explore these relationships in more complex multi-hop scenarios, and over a wider range of cross traffic conditions. We intend to establish more rigorous criteria for BADABING parameter selection in our ongoing work.

Finally, Table 7.7 shows results from an experiment designed to understand the trade-off between an increased value of p, and an increased value of N. We chose $p = 0.1$, and show results using two different values of τ, 40 and 80 milliseconds. The background traffic used in these experiments was the simple constant bit rate traffic with uniform loss episode durations. We see that there is only a slight improvement in both frequency and duration estimates, with most improvement coming from a larger value of τ. Empirically understanding the convergence of estimates of loss characteristics for very low probe rates as N grows larger is a subject for future experiments.

7.6.3 Dynamic Characteristics of the Estimators

As we have shown, estimates for a low probe rate do not significantly improve even with rather large N. A modest increase in the probe rate p, however, substantially improves the accuracy and convergence time of both frequency and duration estimates. Figure 7.7 shows results from an experiment using HARPOON to generate self-similar, web-like TCP traffic for the loss episodes. For this experiment, p is set to 0.5. The top plot shows both the dynamic

7.6: BADABING loss estimates for HARPOON web-like traffic (Variability in true frequency and duration is due to inherent variability in background traffic source.)

p	loss frequency		loss duration (seconds)	
	true	BADABING	true	BADABING
0.1	0.0044	0.0017	0.060	0.071
0.3	0.0011	0.0011	0.113	0.143
0.5	0.0114	0.0117	0.079	0.074
0.7	0.0043	0.0039	0.071	0.076
0.9	0.0031	0.0038	0.073	0.062

7.7: Comparison of loss estimates for $p = 0.1$ and two different values of N and two different values for the τ threshold parameter.

N	τ	loss frequency		loss duration (seconds)	
		true	BADABING	true	BADABING
180,000	40	0.0059	0.0006	0.068	0.021
180,000	80	0.0059	0.0015	0.068	0.053
720,000	40	0.0059	0.0009	0.068	0.020
720,000	80	0.0059	0.0018	0.068	0.041

7.8: Comparison of results for BADABING and `zing` with constant bit rate (CBR) and HARPOON web-like traffic. Probe rates matched to $p = 0.3$ for BADABING (876 kb/s) with probe packet sizes of 600 bytes. (BADABING results copied from row 2 of Tables 7.4 and 7.6. Variability in true frequency and duration for HARPOON traffic scenarios is due to inherent variability in background traffic source.)

traffic scenario	tool	loss frequency		loss duration	
		true	measured	true (sec)	measured (sec)
CBR	BADABING	0.0069	0.0065	0.068	0.073
	`zing`	0.0069	0.0041	0.068	0.010
HARPOON web-like	BADABING	0.0011	0.0011	0.113	0.143
	`zing`	0.0159	0.0019	0.119	0.007

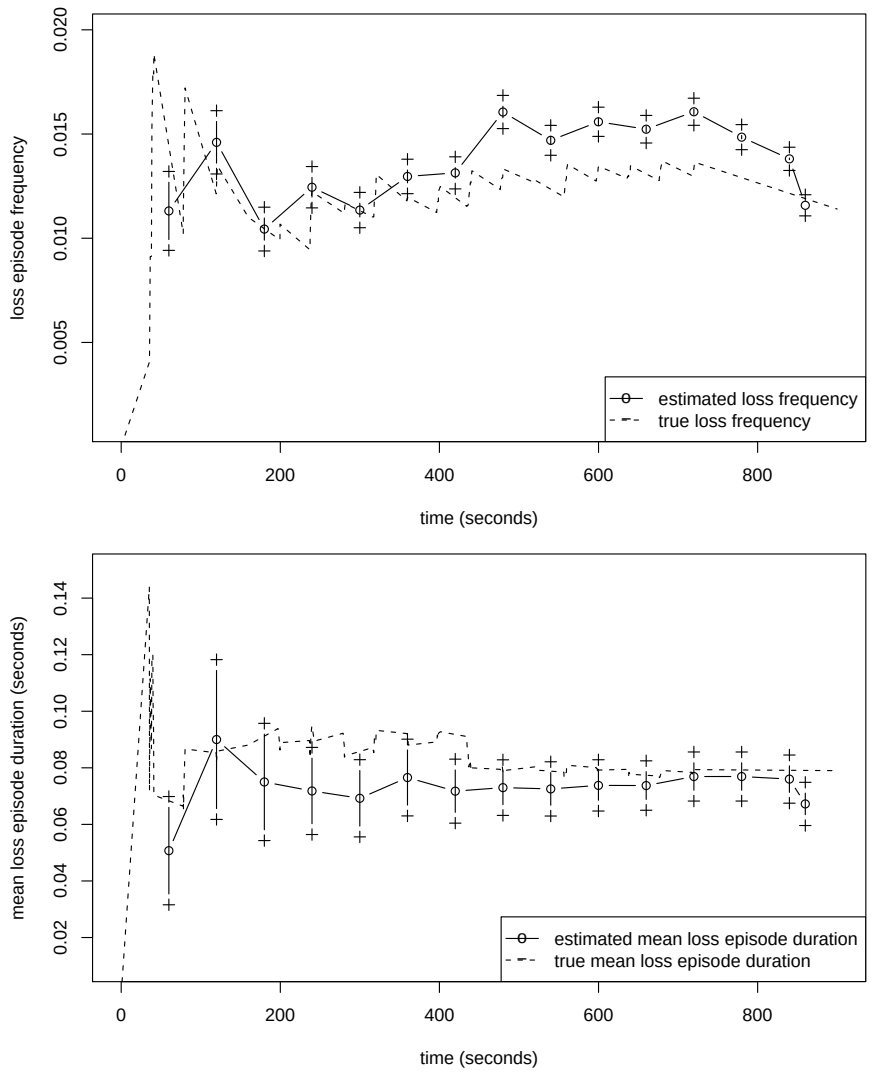
characteristics of both true and estimated loss episode frequency for the entire 15 minute-long experiment. BADABING estimates are produced every 60 seconds for this experiment. The error bars at each BADABING estimate indicate a 95% confidence interval for the estimates, computed using the methodology detailed in Sommers *et al.* [191]. We see that even after one or two minutes, BADABING estimates have converged close to the true values. We also see that BADABING tracks the true frequency reasonably well. The bottom plot in Figure 7.7 compares the true and estimated characteristics of loss episode duration for the same experiment. Again, we see that after a short period, BADABING estimates and confidence intervals have converged close to the true mean loss episode duration. We also see that the dynamic behavior is generally well followed. Except for the low probe rate of 0.1, results for other experiments exhibit similar qualities. Note that the capabilities of BADABING to quickly and accurately estimate loss episode frequency and mean duration suggest that it could be used to reevaluate the loss constancy findings of Zhang *et al.* [218].

7.6.4 Comparing Loss Measurement Tools

Our final set of experiments compares BADABING with `zing` using the constant bit rate and HARPOON web-like traffic scenarios. We set the probe rate of `zing` to match the link utilization of BADABING when $p = 0.3$ and the packet size is 600 bytes, which is about 876 kb/s, or about 0.5% of the capacity of the OC-3 bottleneck. Each experiment was run for 15 minutes. Table 7.8 summarizes results of these experiments, which are similar to the results of shown above. (Included in this table are BADABING results from row 2 of Tables 7.4 and 7.6.) For the CBR traffic, the loss frequency measured by `zing` is somewhat close to the true value, but loss episode durations are not. For the web-like traffic, neither the loss frequency nor the loss episode durations measured by `zing` are good matches to the true values. Comparing the `zing` results with BADABING, we see that for the same traffic conditions and probe rate, BADABING reports loss frequency and duration estimates that are significantly closer to the true values.

7.7 Estimating End-to-End Loss Rate with BADABING

We now describe the basic assumptions and method for estimating packet loss rate along an end-to-end path. Our objective is to develop an accurate, robust estimator based on BADABING's calculations of congestion event frequency



7.7: Comparison of loss frequency and duration estimates with true values over 15 minutes for HARPOON web-like cross traffic and a probe rate $p = 0.5$. BADABING estimates are produced every minute, and error bars at each estimate indicate the 95% confidence interval computed using the methodology of Sommers *et al.* [191]. Top plot shows results for loss episode frequency and bottom plot shows results for loss episode duration.

($\widehat{F}$) and duration ($\widehat{D}$). It was noted above that the primary difficulty in estimating end-to-end *packet loss rate*—the loss performance metric specified in SLAs—is that it is unclear how to measure overall *demand* along the path, particularly during congestion periods. Therefore, we propose the following heuristic approach.

Starting from BADABING's geometric probe stream, we measure the loss rate $\widehat{l}$ of the probes *during loss episodes*. Since the estimation techniques described above do not directly identify individual congestion episodes we take an empirical approach, treating consecutive probes in which at least one packet is lost as indication of a congestion episode. Also, as above, we assume that the end-to-end loss rate L is stationary and ergodic. Given an estimate of the frequency of congestion $\widehat{F}$, we estimate the end-to-end loss rate as

$$\widehat{L} = \widehat{F}\widehat{l}.$$

The key assumption of this heuristic is that we treat the probe stream as a *marker flow*, namely, that the loss rate observed by this flow has a meaningful relationship to other flows along the path. As a basis for this assumption, we note that BADABING probes consist of multiple packets (3 by default), which has some similarity to a TCP stream where delayed ACKs cause a sender to release two very closely-spaced packets. While we do not claim that the probe stream is, in general, the same as a TCP stream, our results below demonstrate that such an assumption may be reasonable in this context.

To evaluate this technique, we used four background traffic scenarios in our laboratory calibration testbed. For the first scenario, we used `Iperf` [198] to produce constant bit rate (CBR) UDP traffic for creating a series of approximately constant duration (about 68 milliseconds) loss episodes that were spaced randomly at exponential intervals with mean of 10 seconds over a 10 minute period. The second scenario consisted of 100 long-lived TCP sources run over a 10 minute period. For the final two scenarios, we used HARPOON with a heavy-tailed file size distribution to create self-similar traffic approximating a mix of web-like and peer-to-peer traffic. For the HARPOON experiments, we used two different offered loads of 60% and 75% of the bottleneck OC-3, and ran experiments for 15 minutes. For all scenarios, we discarded the first 30 and last 30 seconds of the traces.

We then examined the accuracy of the loss rate estimates for BADABING, comparing BADABING's accuracy with a standard RFC 2680 Poisson-modulated [47] stream of the same rate, and a periodic probe stream of the same rate. Periodic probes such as those produced by the ubiquitous `ping` tool are also commonly used for measuring packet loss rate.

Table 7.9 compares the true loss rate with the loss rate estimates of BADABING and the Poisson and periodic probe streams. Values are shown for each of the four traffic scenarios and are average loss rates over the duration of each experiment. Note that differences in true values are due to inherent variability in traffic sources. We see that for all four scenarios, the RFC 2680 and periodic streams yield very poor estimates of the true loss rate. In all but one case, the estimates are off by more than two orders of magnitude—a significant relative error. In fact, these estimates are generally close to zero—a phenomenon consistent with results shown above and primarily due to the fact that single

7.9: Comparison of loss rate estimation accuracy for BADABING, Poisson (RFC 2680), and periodic probe streams. Values are average loss rates over the full experiment duration. Differences in true values for the self-similar experiments are due to inherent variability in the underlying traffic source.

Probe stream → **Traffic scenario ↓**	BADABING		Poisson (RFC 2680)		periodic	
	true	estimate	true	estimate	true	estimate
CBR	0.0051	0.0073	0.0051	0.0017	0.0051	0.0017
Long-lived TCP	0.0163	0.0189	0.0163	0.0062	0.0163	0.0050
HARPOON self-similar (60% load)	0.0008	0.0007	0.0017	0.0000	0.0018	0.0000
HARPOON self-similar (75% load)	0.0049	0.0050	0.0055	0.0000	0.0060	0.0011

packet probes generally yield poor indications of congestion along a path. The estimates produced by BADABING are significantly better, with a maximum relative error in the case of the CBR background traffic. Both BADABING loss rate estimates for the self-similar background traffic have relative errors of about 10% or less.

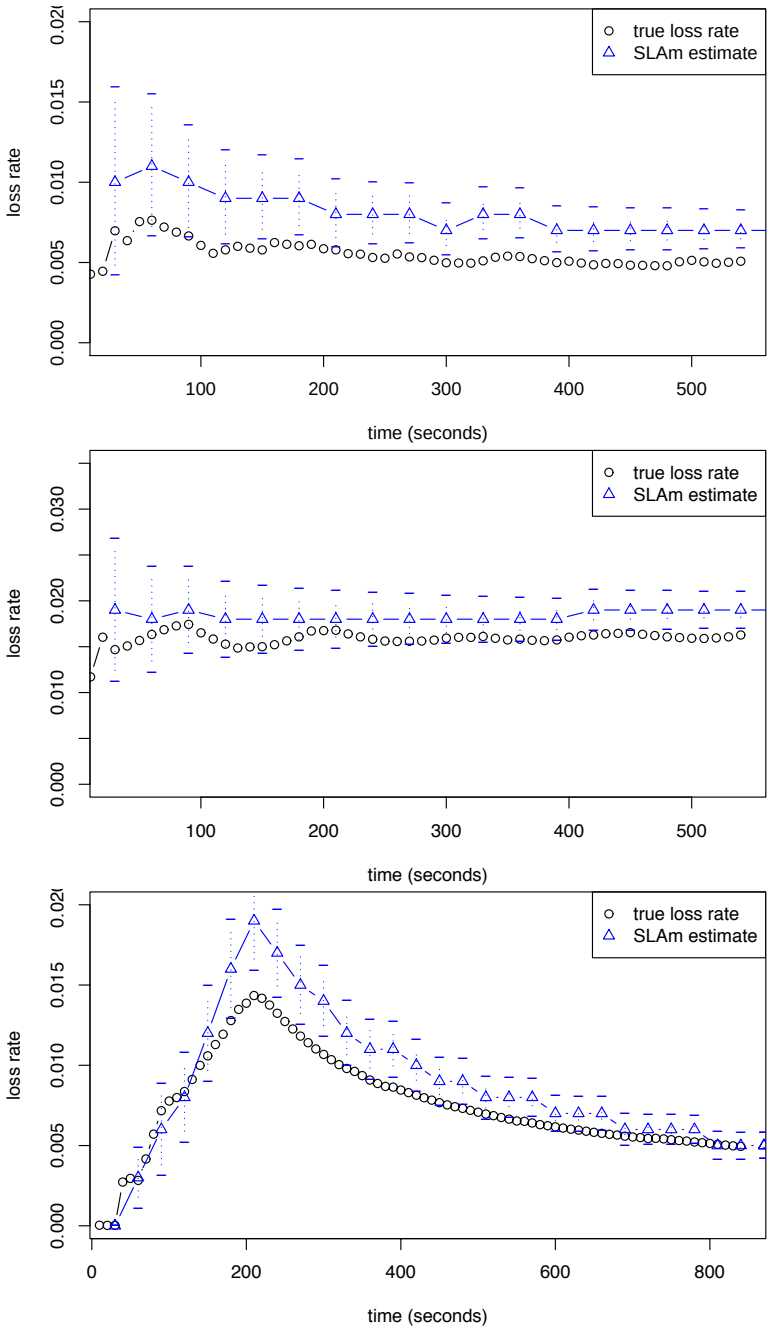
Estimation accuracy over relatively long time periods (*e.g.*, 10 minutes) is clearly desirable from the standpoint of a network provider for monitoring compliance with SLAs. Also important are the dynamic properties of an active measurement estimator, *i.e.*, how well the method adapts to changing network conditions and how quickly the estimator converges to the average path state. We now examine the time varying nature of the BADABING estimates for packet loss rate.

Figure 7.8 shows the true loss rate and BADABING-estimated loss rate over the duration of experiments using constant bit rate traffic (top), long-lived TCP traffic (middle) and self-similar traffic at 60% offered load (bottom). As above, true loss rate estimates are shown for 10 second intervals and estimates for BADABING are shown for 30 second intervals. The upper and lower bars for BADABING indicate estimates of one standard deviation above and below the mean using the variance estimates derived using the methodology of [191]. We see the narrowing of variance bounds as an experiment progresses, and that the true loss rate is, with few exceptions, within these bounds. We also see that BADABING tracks the loss rate over time quite well, with its estimated mean closely following the true loss mean.

7.8 Using BADABING in Practice

There are a number of important practical issues which must be considered when using BADABING in the wide area:

- The tool requires the user to select values for p and N. Assume for now that the number of loss events is stationary over time. (Note that we allow the duration of the loss events to vary in an almost arbitrary way, and to change over time. One should keep in mind that in our current formulation we estimate the *average* duration and not the distribution of the durations.) Let B_0 be the mean number of loss events that occur over a unit period of time. For example, if an average of 12 loss events occur every minute, and our discretization



7.8: Comparison of true loss rate with BADABING estimates over time. True loss rates are plotted using 10 second intervals. BADABING estimates are plotted using 30 second intervals. Plots shown for CBR (top), long-lived TCP (middle), and self-similar traffic at 60% offered load (bottom) traffic scenarios.

unit is 5 milliseconds, then $B_0 = 12/(60 \times 200) = .001$ (this is, of course, an estimate of the true the value of B_0). With the stationarity assumption on B_0, we expect the accuracy of our estimators to depend on the product pNB_0, but not on the individual values of p, N or B_0.[4]. Thus, the individual choice of p and N allows a trade off between timeliness of results and impact that the user is willing to have on the link. Prior empirical studies can provide initial estimates of B_0. An alternate design is to take measurements continuously, and to report an estimate when our validation techniques confirm that the estimation is robust. This can be particularly useful in situations where p is set at low level. In this case, while the measurement stream can be expected to have little impact on other traffic, it may have to run for some time until a reliable estimate is obtained.

- Our estimation of duration is critically based on correct estimation of the ratio B/M (*cf.* Section 7.5). We estimate this ratio by counting the occurrence rate of $y_i = 01$, as well as the occurrence rate of $y_i = 10$. The number B/M can be estimated as the average of these two rates. The *validation* is done by measuring the *difference* between these two rates. This difference is directly proportional to the expected standard deviation of the above estimation. Similar remarks apply to other validation tests we mention in both estimation algorithms.

- The recent study on packet loss via passive measurement reported in [163] indicates that loss episodes in backbone links can be very short-lived (*e.g.,* on the order of several microseconds). The only condition for our tool to successfully detect and estimate such short durations is for our discretization of time to be finer than the order of duration we attempt to estimate. Such a requirement may imply that commodity workstations cannot be used for accurate active measurement of end-to-end loss characteristics in some circumstances. A corollary to this is that active measurements for loss in high bandwidth networks may require high-performance, specialized systems that support small time discretizations.

- Our classification of whether a probe traversed a congested path concerns not only whether the probe was lost, but how long it was delayed. While an appropriate τ parameter appears to be dictated primarily by the value of p, it is not yet clear how best to set α for an arbitrary path, when characteristics such as the level of statistical multiplexing or the physical path configuration are unknown. Examination of the sensitivity of τ and α in more complex environments is a subject for future work.

- To accurately calculate end-to-end delay for inferring congestion requires time synchronization of end hosts. While we can trivially eliminate offset, clock skew is still a concern. New on-line synchronization techniques such as reported in [167, 204] or even off line methods such as [216] could be used effectively to address this issue.

[4]Note that estimators that average individual estimations of the duration of each loss episode are not likely to perform that well at low values of p.

Chapter 8

Calibration Case Study 3: Router Buffer Sizing

> "If I weren't interested in having my ideas proven real, I'd be a mathematician."
> –S. Traweek, quoting an anonymous experimental physicist [199].

8.1 Overview

Modern routers are complex systems with many features and capabilities intended to improve performance of basic packet switching tasks. At the core of any router architecture is a series of buffers that absorb bursts of packets when the aggregate demand on ingress links exceeds the capacity of an egress link. These memories can be large and, as a result, can comprise a non-negligible portion of a router's overall cost. While the implementation of buffers on router line cards varies significantly from system to system, the objective in their design and configuration is to enable the system, and by extension the network, to meet specified performance targets.

The problem of determining how to configure and size buffers in routers has received significant attention from the research community. A recent example is the work by Appenzeller *et al.* in [51] that argues that a buffer size B equal to the product of the capacity C of the link and round trip time T divided by the square root of the number N of long-lived TCP flows results in full utilization of the link and challenges the conventional wisdom that router buffers should be sized to be (at least) the bandwidth-delay product (BDP) of a link (*e.g.*, [120, 205]). The practical significance of the $B = CT/\sqrt{N}$ formula advocated in [51] is that it suggests that buffers can be configured significantly smaller than commonly thought, to the point of eliminating the need for up to 99% of output buffers for line speeds of 10 Gb/s and above.

However, routers are part of the physical infrastructures of ISPs, and are managed and configured for the purpose of guaranteeing their customers a certain level of service. These performance objectives are spelled out in detail in contracts called Service Level Agreements (SLAs) that provide critical context for many configuration and provisioning decisions. Through the case study in this chapter, we argue that buffer sizing and configuration decisions should explicitly account for the tussle space defined by ISP economics, router hardware design, and network performance

measures [80]. In particular, for any ISP, SLAs involve trading off risk and expense. On one end of the spectrum, risk-averse ISPs tend to build expensive infrastructures to satisfy a wide range of SLA-related performance objectives with high probability despite genuine uncertainties in user behavior, traffic mix and network conditions. At the other end of the spectrum are more risk-tolerant ISPs that build less expensive network infrastructures to meet similar SLA objectives. Such providers are more likely to incur additional expenses in the form of credits to customers whose SLAs cannot consistently be met. A similar tradeoff applies to customers who have a choice of buying more or less expensive (*i.e.*, stringent) SLAs depending on their willingness to tolerate sub-par performance in the presence of a range of network-related uncertainties.

Our motivation in this chapter is to illustrate the benefits and capabilities of our calibration framework in a different context than the previous two chapters. The focus of our case studies in chapters 6 and 7 was to apply our calibration framework and methodology to evaluation of the performance and accuracy of existing active measurement tools, and in the design of new ones. In this case study, we apply our framework and methodology to the study of Internet routers. More specifically, we leverage the realism, transparency, and control afforded by our calibration environment to study the impact of router buffer size on SLA performance objectives.

The primary goal in this chapter is to illustrate the tussle space defined by ISP economics, router hardware design, and network performance measures. To this end, we proceed in two steps. We first revisit the original router buffer sizing problem considered in [51], examining thoroughly the assumptions and premises underlying this work. We report on a detailed empirical evaluation of the tradeoffs that result from considering different performance metrics, different traffic scenarios, different router architectures and different queuing mechanisms. In particular, we consider performance measures that include delay, loss, goodput, throughput and jitter computed for the aggregate traffic and on a per-flow basis; our traffic scenarios range from homogeneous (*i.e.,* long-lived TCP flows) to highly heterogeneous (*i.e.,* mixtures of self-similar TCP traffic and multimedia-type UDP traffic), with a spectrum of realistic aggregate demands. In contrast to most buffer sizing studies to date (a noticeable exception is the empirical evaluation in [51]), instead of relying on idealized router models and abstract queuing simulations, we use three different popular commodity router setups (two Cisco and one Juniper) and configure them for different sizes of both drop tail and RED queues. A systematic exploration of the resulting high-dimensional parameter space enables us to broadly assess and compare prior methods for sizing buffers.

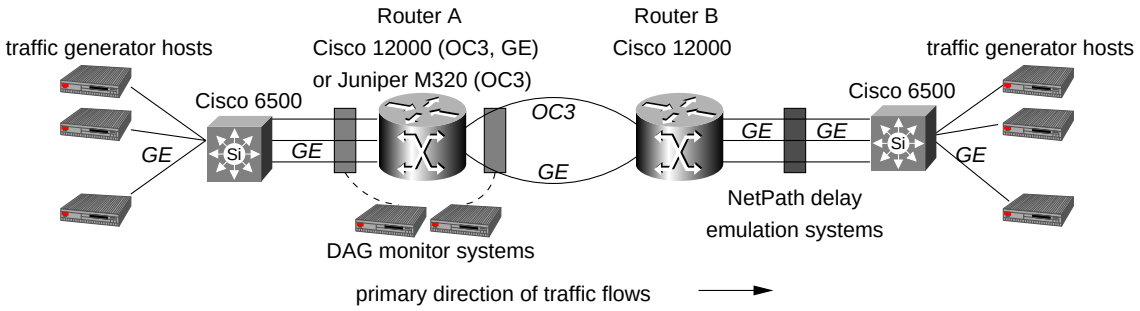
We find that while throughput is relatively insensitive to differences in router architecture, buffer size, and traffic mix, other performance metrics such as loss and delay by and large tend to be much more sensitive. In particular, our findings shed light on the performance risks of the $B = CT/\sqrt{N}$ method and the performance gains that result from other methods that advocate somewhat larger buffers. We also observe that RED queues can provide improvements in both aggregate and per flow performance profiles in restricted buffer configurations.

Next, we present initial results that show how ISP economics in the form of SLAs can impact router buffer sizing decisions. In particular, by experimenting with a set of "toy" SLAs that resemble real-world SLAs, we fix a set of

canonical performance requirements and illustrate how an SLA-centric perspective exposes new factors that contribute to a more informed decision making process for router buffer sizing, especially at the edge of the network. These results suggest that since networks are managed to provide a level of service specified by these guarantees (*i.e.,* this is what network providers and customers care about the most), SLAs should play an important role in configuring router buffers. They also motivate studying new problem areas such as measuring SLA (non-)compliance, engineering for robust SLA compliance, or quantifying the risk of SLA non-compliance. For example, we find that for the same SLA, the service provider's risk of SLA non-compliance is typically greater with fine-grained SLA compliance measurement/reporting than with coarse-grained ones. Moreover, we observe that this risk can be quantified in terms of the degree of sensitivity of SLA compliance to uncertainties in traffic mix and volume. At the same time, an analytic treatment of these problems poses formidable technical challenges and is beyond the scope of our work.

8.2 Laboratory Calibration Testbed

Our laboratory testbed, shown in Figure 8.1, was organized as a dumbbell topology. At its core was a bottleneck OC-3 (155 Mb/s nominal) or Gigabit Ethernet (GE) link connected between either two Cisco GSR routers, or between a Juniper M320 and a Cisco GSR (routers A and B in the figure). Both OC-3 and GE bottlenecks were used with the two-GSR configuration, and only the OC-3 bottleneck was used with the M320-GSR configuration. Some detailed differences between these architectures are discussed below. In general, the primary flow of traffic was in the direction of router A to router B (left to right in the figure). Synchronized Endace DAG cards were connected via optical splitters to the links on either side of router A (either Cisco GSR or Juniper M320) for the purpose of ground truth, comprehensive packet-level measurement. DAG 4.3 cards were used for GE and a DAG 3.8 card was used for OC-3[1].



8.1: Laboratory testbed. Multiple Gigabit Ethernet (GE) links connected Cisco 6500 routers to two routers separated by a bottleneck link of either OC-3 or GE. Router A was either a Cisco GSR or a Juniper M320. Synchronized Endace DAG cards captured traffic on either side of this bottleneck router. Linux hosts running NETPATH were interposed in the testbed to perform propagation delay emulation.

At each end of the topology were 14 hosts (28 hosts total) running HARPOON to generate a variety of traffic scenarios. Traffic from these workstations was aggregated via two Cisco 6500 routers. These hosts ran either Linux

[1]The DAG software version used was 2.5.5.

2.6 or FreeBSD 5.4 and were equipped with either one or two Intel Pentium 4 processors, at least 1 GB RAM, and an Intel Pro/1000 network interface card. Each host had another network interface for management traffic, which flowed over a separate physical network (not shown in Figure 8.1). The TCP stacks on each host were configured with 64 KB receive windows and were configured to be SACK-capable.

Linux hosts running NETPATH were interposed in the testbed to perform propagation delay emulation. Two round-trip time distributions were used. The first distribution ("intra-continental") was uniformly distributed between 20 and 80 milliseconds with a mean of 50 milliseconds. The second distribution ("inter-continental") was uniformly distributed between 140 and 260 milliseconds with a mean of 200 milliseconds. We monitored these systems during experiments and periodically reconfigured our DAG cards to run calibration tests on the NETPATH systems to ensure that load was well-distributed and that they were able to handle maximum offered loads.

8.2.1 Router Architectures

The three router configurations (router A in Figure 8.1) used in our tests were as follows:

Cisco GSR / OC-3. This is a low-end line card (termed "engine 0") with features (*e.g.*, RED) primarily implemented in software running on a general purpose processor (MIPS R5000)[2]. The entire board contains a total of 128 MB DRAM packet memory (64 MB available for transmit, 64 MB for receive), shared across four physical interfaces. In addition to DRAM, there is a 128 KB fixed FIFO transmit buffer on each interface. Maximum output queue limit is 65535 packets for each physical interface.

Cisco GSR / GE. This is a more sophisticated line card (termed "engine 3") with RED and other features implemented in special-purpose ASICs[3]. There are four physical ports on this line card and a total of 512 MB DRAM packet memory available (256 MB transmit, 256 MB receive). In addition to DRAM, there is a 512 KB fixed FIFO transmit buffer on each interface. Maximum output queue limit is 262143 packets for each physical interface.

Juniper M320 / OC-3. This physical interface card (PIC) housed in a flexible PIC concentrator (FPC) contains four physical OC-3 ports[4]. As with the GSR, packets are buffered both on the ingress FPC/line card and the egress FPC/line card [15], and, like the GSR engine 3 line card, RED is implemented in a hardware ASIC. It is important to note that JUNOS imposes a hard upper limit of 50 milliseconds on buffers configured for a given interface [14][5].

[2] http://www.cisco.com/en/US/products/hw/modules/ps2710/products_data_sheet09186a00800920aa.html.

[3] http://www.cisco.com/en/US/products/sw/iosswrel/ps1829/products_feature_guide09186a0080154e7a.html.

[4] http://www.juniper.net/techpubs/hardware/m320/m320-pic/sonet-sdh-oc-3.html.

[5] This limit was confirmed with Juniper technical support. For core-class routers like the M320 and T640, a relatively low upper limit is set on the maximum buffer size. For edge routers like the M20, there are higher limits.

Each of these router configurations has significantly different capabilities with respect to the specific line/interface card attached to the bottleneck link, the amount of memory available for packet buffers, and how particular features are implemented (*e.g.*, in software or specialized ASICs). Even between the two Cisco GSR line cards, there are many significant differences [4]. While there are certainly other architectural differences, notably between the Cisco GSR and Juniper M320 chassis, our focus here is on the packet forwarding path specifically as it relates to a single (potentially) congested egress interface. Finally, there are implementation differences in how each line card above handles the division of local buffer space among multiple physical interfaces. Since our focus is on a single congested egress interface, we leave the problem of understanding the impact of multiple, simultaneously congested interfaces on a single line card for future work.

8.2.2 Traffic Scenarios

We used four traffic mix scenarios in our tests and a range of offered loads for each mix, as described below:

Long-lived TCP sources. This setup consisted solely of long-lived TCP flows. Offered load was varied by changing the number of flows. At minimum, we used enough flows to be able to saturate the bottleneck link. The maximum load was an overload scenario where each flow had effectively one packet worth of bandwidth available per round-trip time. For example, with the OC-3 bottleneck and 50 milliseconds mean RTT setup, we used between 30 and 1200 sources.

Self-similar TCP sources. In this setup, files drawn from a heavy-tailed distribution were transferred across the testbed to create self-similar traffic approximating a mix of web-like and peer-to-peer traffic commonly seen in today's networks. We used average offered loads of 60%, 90%, 100%, and 110% of the bottleneck capacity. For the self-similar TCP sources, as well as the long-lived TCP sources in the first traffic scenario, we configured 90% of the flows to have maximum segment sizes of 1448 bytes, and 10% of the flows to use 512 bytes.

Self-similar TCP with low-rate UDP sources. In this scenario, we used self-similar traffic at an offered load of 50%, and created UDP constant bit rate (CBR) flows of 64 kb/s to produce overall offered loads of 60%, 90%, 100%, and 110% of the bottleneck capacity. In addition, we experimented with a fixed ratio between traffic and UDP CBR traffic of 90%/10%, respectively, and tuned overall offered loads to be 60%, 90%, 100%, and 110% of the bottleneck capacity.

Self-similar TCP with high-rate UDP sources. In this scenario, we again used self-similar traffic at an offered load of 50%, and created UDP CBR flows of 1 Mb/s to produce overall offered loads of 60%, 90%, 100%, and 110% of the bottleneck capacity.

For each of these traffic mixes, we used two configurations of traffic direction. In the first, all data traffic flowed in the direction of router A to router B (left to right) in Figure 8.1. In this setup, the reverse (right to left) direction consists

8.1: Quasi-logarithmic drop-tail queue settings, in number of packets. Bandwidth-delay product (assuming 1500 byte packets) appears in boldface.

Bottleneck	Mean RTT	Queue Sizes					
OC-3	50 milliseconds		1	39	156	**624**	2496
OC-3	200 milliseconds		1	156	624	**2496**	9984
GE	50 milliseconds	1	65	261	1044	**4167**	16668

8.2: RED configuration settings for 50 millisecond round-trip time tests, in number of packets.

	small		medium		large	
	min_{th}	max_{th}	min_{th}	max_{th}	min_{th}	max_{th}
OC-3	50	150	225	675	400	1200
GE	333	1000	1500	4500	2667	8000

entirely of TCP ACK traffic. In the second configuration, data traffic flowed in both directions in equal proportions, on average, according to the desired offered load.

8.2.3 Buffer Configurations

The key additional dimension in our tests was to experiment over a broad set of queue configurations. For each of the traffic mixes and offered loads, traffic directions, and round-trip time distributions, we ran experiments using both drop-tail and RED queues, over a range of sizes. We set the output queues on the bottleneck interfaces on both routers A and B identically for each experiment. No other buffer or queue in our testbed was modified from its default setting.

Table 8.1 shows drop-tail queue lengths in numbers of packets used for three bottleneck and mean round-trip time configurations. Our settings follow a quasi-logarithmic distribution, which we created by starting with the bandwidth-delay product and dividing by successive factors of four and multiplying by one factor of four, with a size of 1 explicitly selected as a special case. Output buffer sizes on Juniper routers are configured in terms of microseconds. To arrive at a buffer size in microseconds, we multiplied the number of packets by the time taken to transmit a 1500 byte packet at OC-3 or GE speed. Note that we explicitly selected a queue size of 1 as a special case to investigate.

Table 8.2 shows the RED configurations used in our 50 millisecond round-trip time experiments. These configurations were drawn from Cisco's default recommendations. By convention, we used the Cisco recommended setting (appropriately translated) for the OC-3 Juniper tests. The small, medium and large settings indicate the three different RED queue thresholds used in these experiments. While there is a large literature on RED tuning, experiments with additional configurations were beyond the scope of our study.

8.2.4 Testing and Analysis Protocol

For each experiment, we calculated aggregate and per-flow throughput, goodput, loss, delay, and delay variation. We processed the DAG traces taken on either side of router A to extract delay and drop information. We also compared

the drop information with router interface counters as a further calibration. We then threw away the first and last 30 seconds of the traces before calculating aggregate and per-flow statistics.

Tests using long-lived TCP sources were run for 3 minutes and all other tests were run for 10 minutes. Between each experiment, the DAG traces were moved to a separate host for offline analysis. In total, we ran approximately 1200 experiments, producing about 1 terabyte of *compressed* packet header data.

8.3 Buffer Sizing: Sensitivity Properties

Performance measures that are robust or insensitive to a wide range of networking-related uncertainties are appealing for service providers and customers alike. For service providers, they offer the hope of coping with many of the unknowns associated with user-generated traffic demands and operating the network in an economically sound manner. From the customer perspective, the existence of robust performance measures avoids the need to specify a detailed traffic profile or application mix and still obtain acceptable service from the network.

In the following, we discuss in more detail the robustness of performance metrics from the customer perspective. After that, we look at the same issue from a provider's point of view. The design space within which we explore sensitivity issues related to the buffer sizing problem accounts for the different traffic scenarios and different router architectures and buffer configurations detailed above, as well as for a variety of different performance metrics. To illustrate our main empirical findings, we limit our discussion to throughput, delay, and loss. We note that the results for goodput are qualitatively consistent with results for loss. We leave detailed investigation of the impact on jitter (delay variation) for future work.

Furthermore, we show results of only a subset of the full range of experiments that we ran for all possible combinations of performance metric, traffic scenario, offered load, router architecture and buffer configuration. Results of experiments that we do not explicitly discuss are consistent with the results shown below. In particular, we do not show results of experiments with bidirectional traffic and 200 millisecond average round-trip times or for experiments using the Gigabit Ethernet bottleneck link, however they are consistent with the results shown below. The selected plots are meant to be representative for the discussions at hand, but may differ in detail from comparable plots. Our focus is on qualitative comparisons and less on quantifying particular differences.

8.3.1 Performance Profiles: Aggregate Traffic Statistics

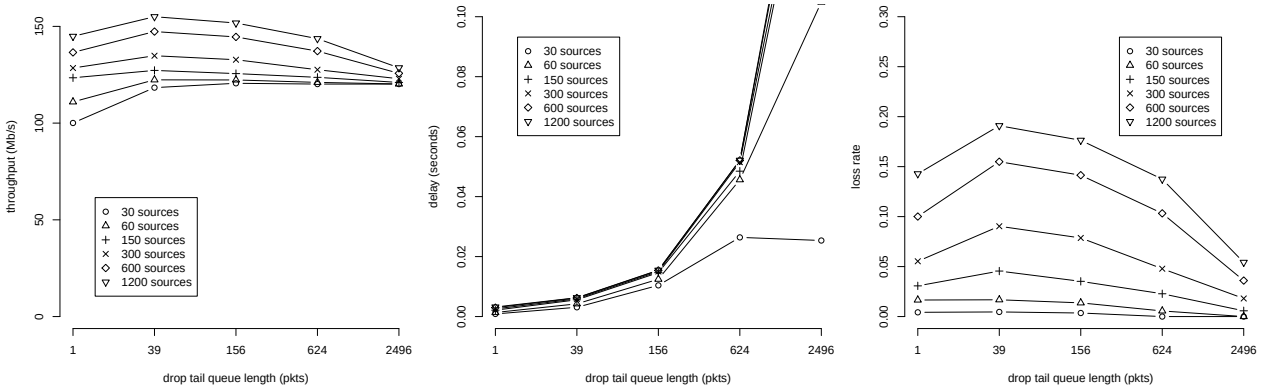
We first consider the case where router A can be viewed as an access router (Cisco GSR/OC-3 with drop-tail queue) that is fed by traffic generated by long-lived TCP sources. In Figure 8.2a (top row), we show two-dimensional performance profiles that result from running this setup for 5x6=30 different buffer size/traffic load combinations. A separate curve is plotted for each source configuration. The buffer size is on the x-axis and the metric of interest is on the y-axis. The three plots show average throughput (left), delay (middle) and loss (right). Another way to view these same results is shown in the three-dimensional perspective plots in Figure 8.2b (middle row). For these plots,

the buffer size is on the x-axis, the offered load (number of TCP sources) is on the y-axis, and the metric of interest is on the z-axis. We show these plots in part to reveal relationships among buffer size, offered load, and our metrics in a more visually striking way. As with the 2D plots, these 3D plots show average throughput (left), delay (middle) and loss (right). (We describe the additional annotations on these plots, below.)

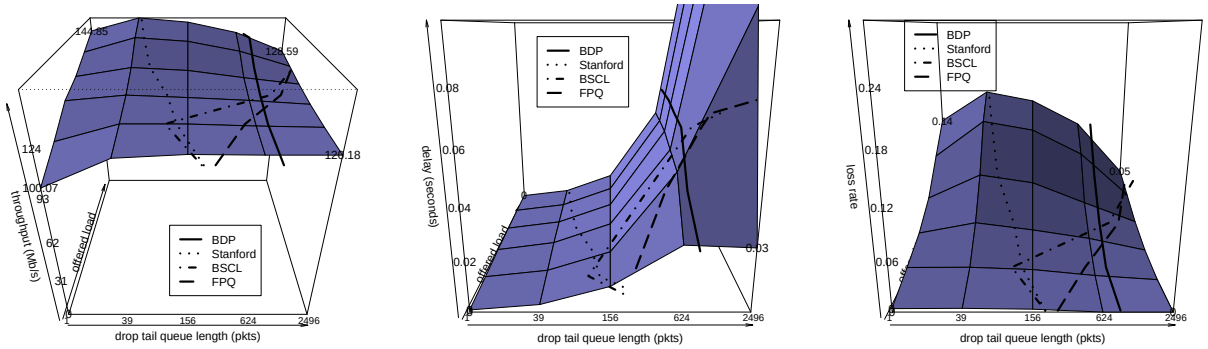
To gauge the variability of these average-based performance profiles, we depict in Figure 8.2c (bottom row) the sets of 30 cumulative distribution functions (CDFs) associated with these profiles. For example, the plot on the left shows the 30 throughput CDFs resulting from running this setup for the 30 pairs of buffer size and traffic load combinations, with specific corner cases labeled. The CDFs in the middle (delay) and on the right (loss) are similarly constructed. A common feature of these CDFs is that with the exception of some of the corner cases, they are tightly concentrated around their means (*i.e.*, they are almost step functions). This implies that almost identical performance profiles would result when using percentiles rather than averages; that is, the percentile-based profiles would deviate only minimally from the average-based ones shown in the top two rows.

Upon closer examination, Figure 8.2 reveals a surprising degree of insensitivity of the throughput performance profile (left-hand plots of Figures 8.2a and 8.2b) to a wide range of changes in buffer size and offered load. In contrast, the delay profile (middle plots of Figures 8.2a and 8.2b) shows the expected increase in delay with larger buffers, while the loss profile (right-hand plots of Figures 8.2a and 8.2b) reflects the common wisdom that losses generally increase with higher traffic loads. To demonstrate that these features are by and large insensitive to the underlying traffic model and/or router architecture, we show the 2D and 3D performance profiles resulting from running the same experiments, but with the long-lived TCP sources replaced with heterogeneous self-similar TCP sources in Figure 8.3, and with the Cisco GSR/OC-3 router replaced by a Juniper M320/OC-3 system, also fed by self-similar TCP sources in Figure 8.4. Plots for the two remaining traffic scenarios and with a Cisco GSR/GE system in place for router A are not shown but have similar characteristics to those found in Figures 8.3 and 8.4. Together, Figures 8.2–8.4 confirm and provide additional support for the concerns expressed in [87] regarding the exclusive reliance on the throughput metric in [51] when advocating the $B = CT/\sqrt{N}$ result. In fact, the performance profiles make it clear why throughput is not a very useful metric for buffer sizing, and that other metrics such as loss and/or delay are better for making a more informed decision.

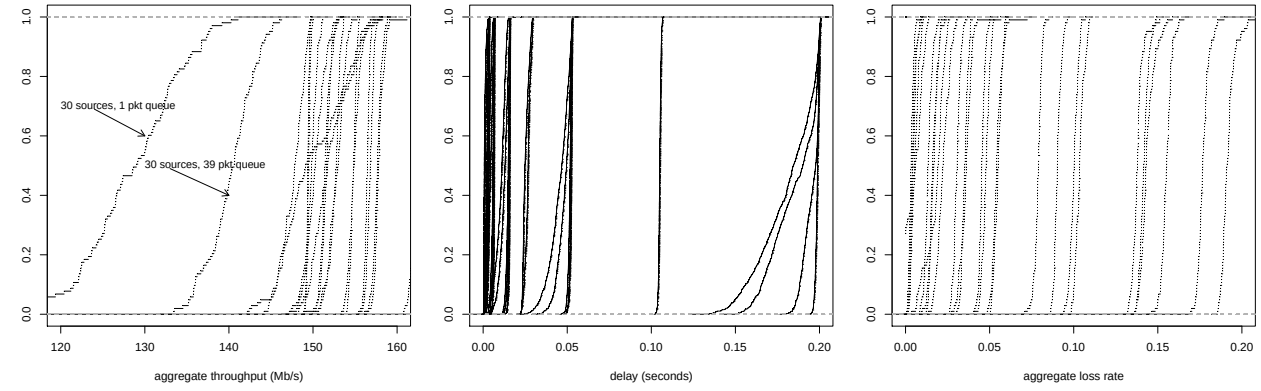
To put some of the previously proposed buffer sizing techniques into perspective, Table 8.3 shows buffer sizes in numbers of packets derived from four different proposed formulas. In particular, Table 8.3 shows the values corresponding to *(i)* the traditional bandwidth-delay product (BDP), *(ii)* the $B = CT/\sqrt{N}$ formula advocated in [51] (Stanford), *(iii)* the BSCL (buffer sizing for congested Internet links) scheme proposed by Dhamdhere *et al.* [88], and *(iv)* Morris's flow-proportional queuing method (FPQ) [156]. To derive these values, we calculated the number of flows long enough to have exited slow start and used this value to parameterize the $CT/\sqrt{N}$ and BSCL models. We used the total number of flows for the FPQ model. Additionally, we followed the methodology in [88] to empirically derive other parameters required for BSCL. For a more visually compelling comparison, we annotate Figures 8.2b,



(a)

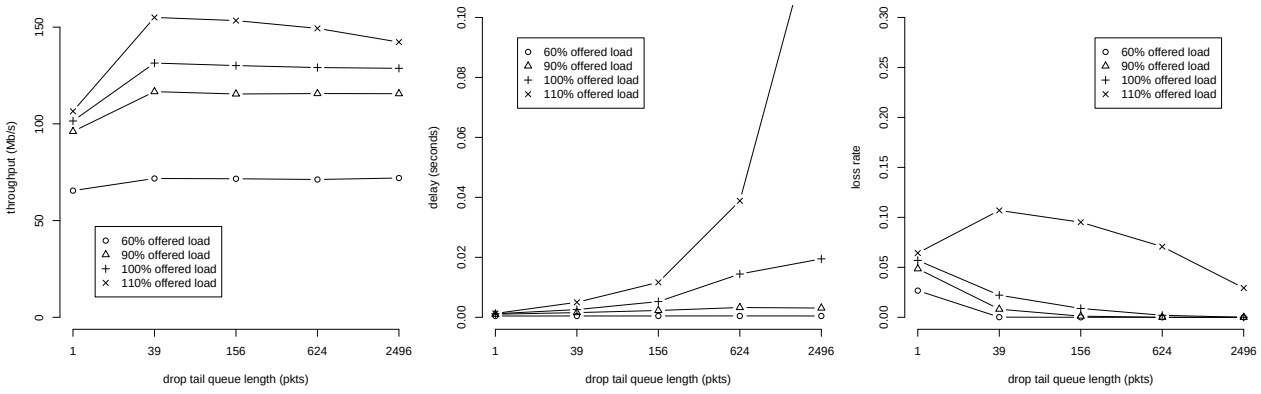


(b)

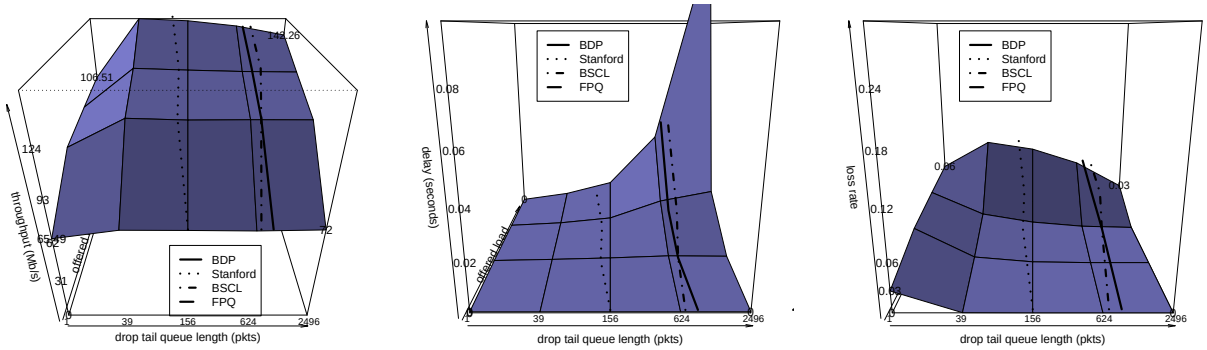


(c)

8.2: Aggregate results for Cisco OC-3 with infinite TCP sources and drop-tail queuing discipline. 2D and 3D profiles of mean throughput, delay, and loss shown in (a) and (b), respectively, and corresponding CDFs shown in (c).

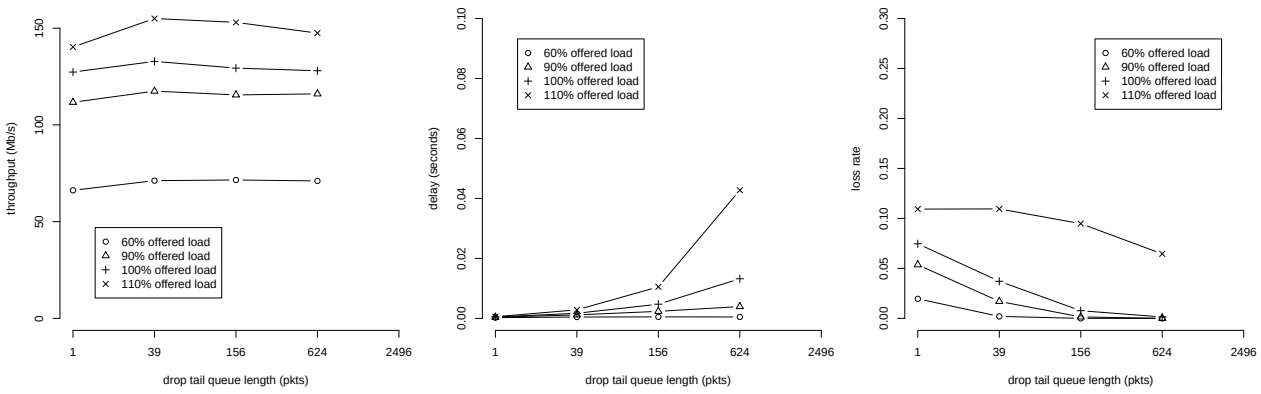


(a)

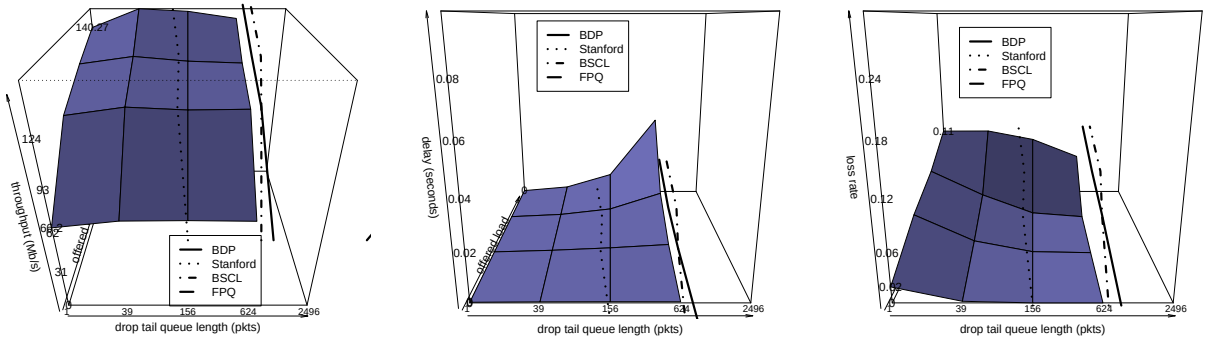


(b)

8.3: 2D (a) and 3D (b) aggregate profiles of mean throughput, delay, and loss for Cisco OC-3 with self-similar sources and drop-tail queuing discipline.



(a)



(b)

8.4: 2D (a) and 3D (b) aggregate profiles of mean throughput, delay, and loss for Juniper OC-3 with self-similar sources and drop-tail queuing discipline. As noted in Section 8.2.1, the Juniper M320 OC-3 interface has a hard upper limit of 50 milliseconds (≈624 packets of 1500 bytes) on buffer size.

8.3: Comparison of buffer sizing schemes for infinite TCP source and self-similar traffic setups for OC-3 bottleneck and 50 millisecond mean RTT. Buffer sizing formulas shown are the traditional bandwidth-delay product (BDP), the $B = CT/\sqrt{N}$ formula advocated in [51] (Stanford), the BSCL (buffer sizing for congested Internet links) scheme proposed by Dhamdhere *et al.* [88], and Morris's flow-proportional queuing method (FPQ) [156]. Values indicate buffer size in packets.

Infinite source load (flows)	**BDP**	**Stanford**	**BSCL**	**FPQ**
30	624	113	121	180
60	624	80	59	360
150	624	50	780	900
300	624	36	2085	1800
600	624	25	4695	3600
1200	624	18	9915	7200
Self-similar traffic offered load	**BDP**	**Stanford**	**BSCL**	**FPQ**
60%	624	84	414	2767
90%	624	68	651	4140
100%	624	65	725	4565
110%	624	60	872	4958

8.3b, and 8.4b to include curves for each of these four buffer sizing methodologies. Note that in Figures 8.3b and 8.4b that the buffer sizes for FPQ are too large to be shown on the plots. Note also that the physical buffer limit of the Juniper M320/OC-3 configuration (Figure 8.4b) is often smaller than the buffer sizes required for the BSCL and FPQ methodologies.

From the perspective of the throughput performance profiles, all 4 methods would do well in achieving high throughput and, judging from the shape of the profiles, so would many other methods. However, some differences become clear when considering the delay and loss performance profiles. For one, considering the buffer sizes proposed by the Stanford model along with the loss performance profiles of Figures 8.2b, 8.3b, and 8.4b, there is a clear tradeoff between choosing small buffers and risking poor performance in terms of high loss rates. Moreover, using the BSCL formula tends to yield much larger buffers which, considering the delay performance profiles, is at the cost of incurring significant delay. Likewise, the FPQ scheme has a tendency to keep losses low even when it comes at the expense of unreasonably large delays. In short, these results demonstrate that buffer sizing could benefit from a new perspective that provides critical context for configuration and provisioning decisions in general.

8.3.2 Performance Profiles: Per-Flow Traffic Statistics

In addition to "what" performance metric(s) to consider for the buffer sizing problem, there is also the issue of "how" the metric(s) in question should be computed. In Section 8.3.1, the metrics were computed based on the *aggregate* customer traffic. In the following, we present our empirical findings for the same three performance metrics, but now computed on a *per-flow* basis. Per-flow characteristics can also be considered as related to metrics such as flow

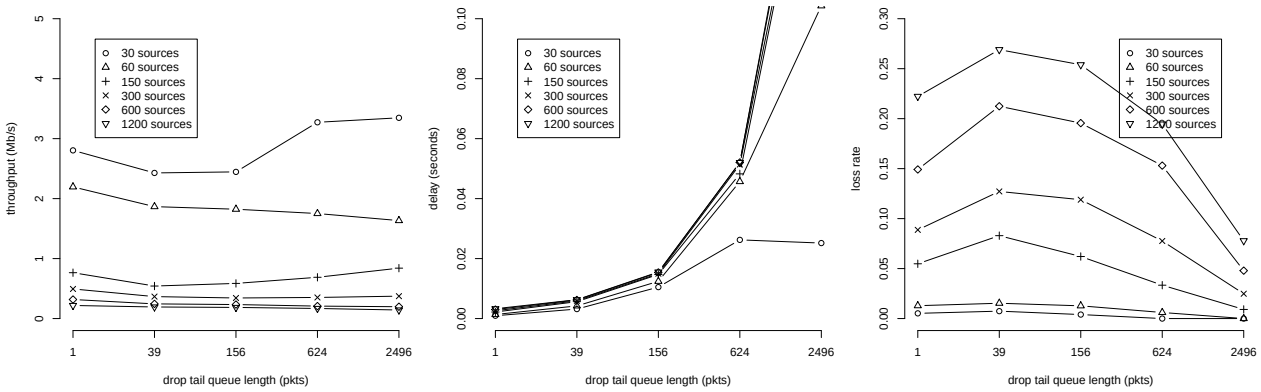
completion time. Using the same set of experiments as in Section 8.3.1, we computed in each case throughput, delay, and loss for each individual flow encountered during the experiment, and plot in Figures 8.5–8.7 the resulting averages to obtain the throughput, delay, and loss performance profiles. Results for additional configurations are similar to those shown in Figure 8.6. As in Figure 8.2, Figure 8.5 also depicts the individual CDFs of the three performance metrics for the 5x6=30 pairs of buffer size and offered load combinations for the setup that deploys a Cisco GSR/OC-3 router with drop-tail queue fed by long-lived TCP sources.

Comparing Figures 8.2 and 8.5, we note that the variability of the per-flow based performance profiles around the depicted averages in Figure 8.5 is much more pronounced than for the aggregate-based profiles in Figure 8.2. The evidence for this is provided by the CDF plots in Figure 8.5c (bottom row) that show in general a wide spread for the different values of a given performance metric, and typically not just for the corner cases. The practical implication is that performance profiles based on, say, 90th percentiles would deviate significantly from their average-based counterparts shown in Figures 8.5a and 8.5b (top and middle rows) and look quite different. Keeping this feature in mind when interpreting average- and per-flow based performance profiles, the main feature in Figures 8.5–8.7 is that the throughput performance profiles, when computed on a per-flow basis, are no longer insensitive to changes in buffer size and/or offered load. In fact, per-flow throughput tends to decrease as offered load increases, which in turn causes an increase in per-flow loss rates. Not surprisingly, we typically also observe an increase in per-flow delay as buffer size increases. These and other observations (not shown) for certain subclasses of flows (*e.g.*, classified by RTT or flow size) confirm the per flow-based findings reported in [87] and complement them by using our large design space to validate these main characteristics for a wider range of possible traffic, load, router, and buffer scenarios. As mentioned in [87], gaining a better understanding of the observed tradeoffs between per-flow throughput, delay, and loss is an important open issue for assessing application-layer performance. The per-flow perspective also brings up questions of fairness (*e.g.*, see [211]), but we defer those issues for future work.

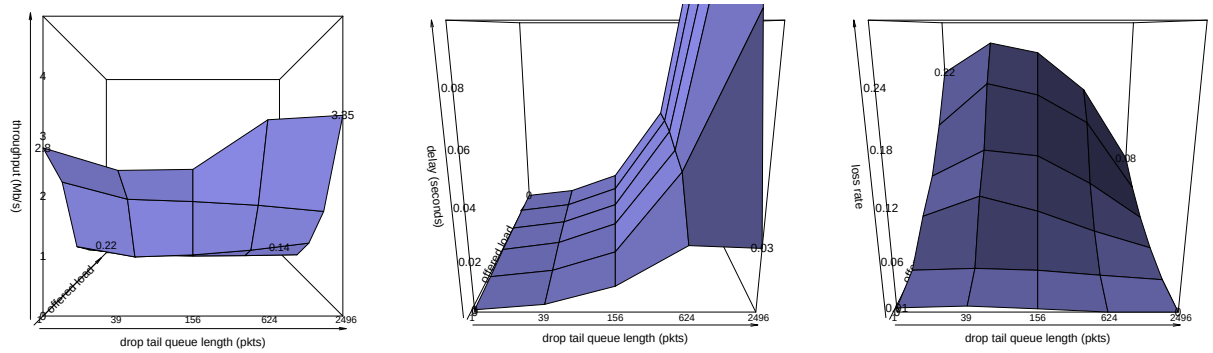
8.3.3 Performance Profiles: Impact of RED

In a final set of experiments that explore sensitivity aspects of the buffer sizing problem, we consider the same experimental setups as in Figure 8.2, but with routers that have RED queues instead of drop-tail queues, with the RED configuration settings given in Table 8.2. Figures 8.8 and 8.9 should be compared to Figures 8.2 and 8.5, respectively, and show the resulting RED-induced performance profile (with corresponding CDFs), computed for the aggregate (Figure 8.8) and on a per-flow basis (Figure 8.9). We observe that the resulting aggregate-based performance profiles exhibit in general only small variability around the plotted averages (as evidenced by the step-function like CDFs) and show typically a higher degree of insensitivity to differences in buffer size and/or offered traffic load for throughput and delay than their drop-tail counterparts.

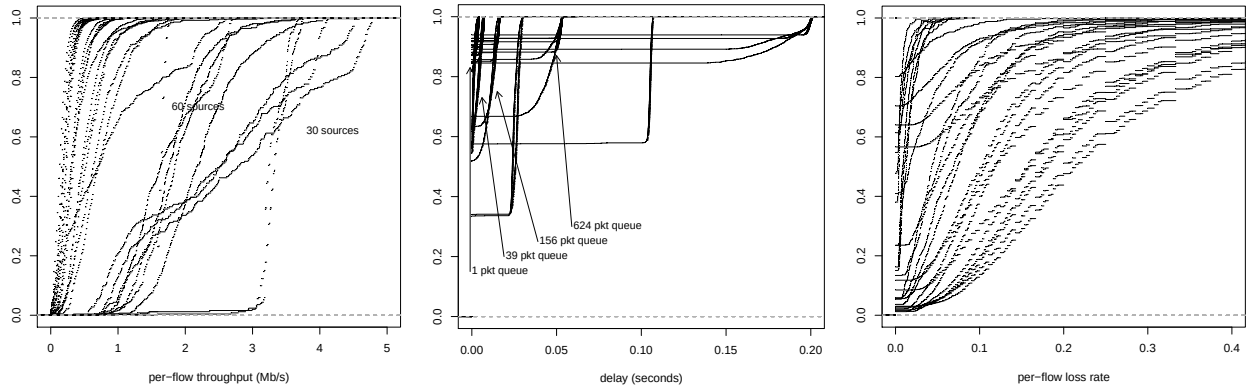
In addition, RED-induced performance tends to be better than drop-tail based performance when performance is measured in terms of throughput or delay and about the same in terms of loss-based performance. With respect to the



(a)

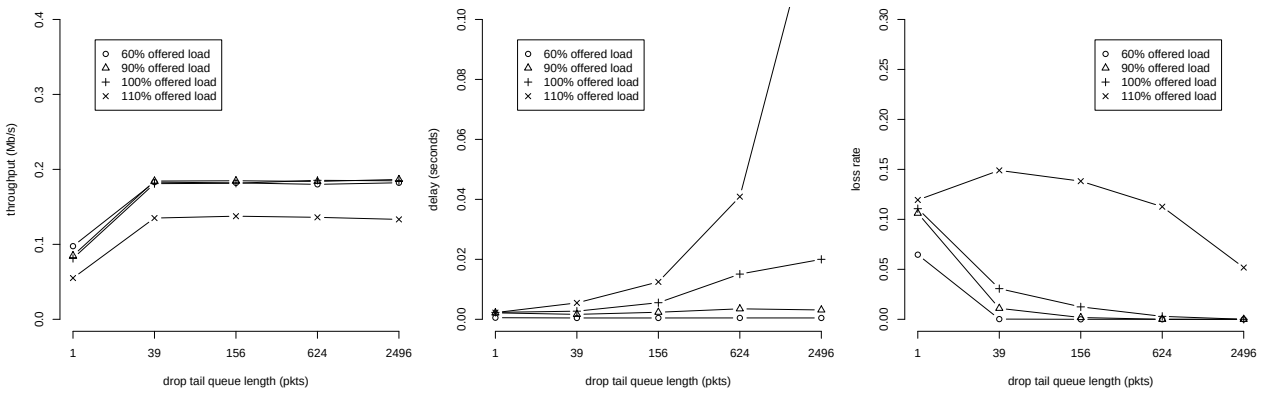


(b)

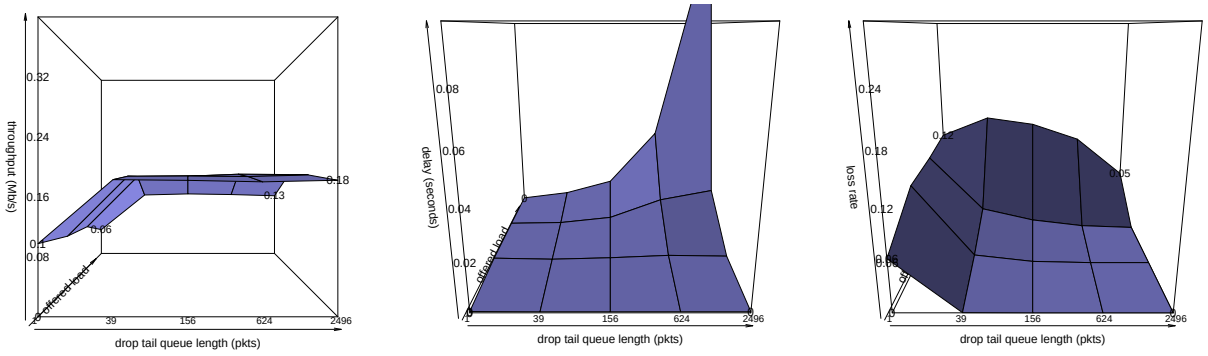


(c)

8.5: Per-flow results for Cisco OC-3 with infinite TCP sources and drop-tail queuing discipline. 2D and 3D profiles of mean throughput, delay, and loss shown in (a) and (b), respectively, and corresponding CDFs shown in (c).

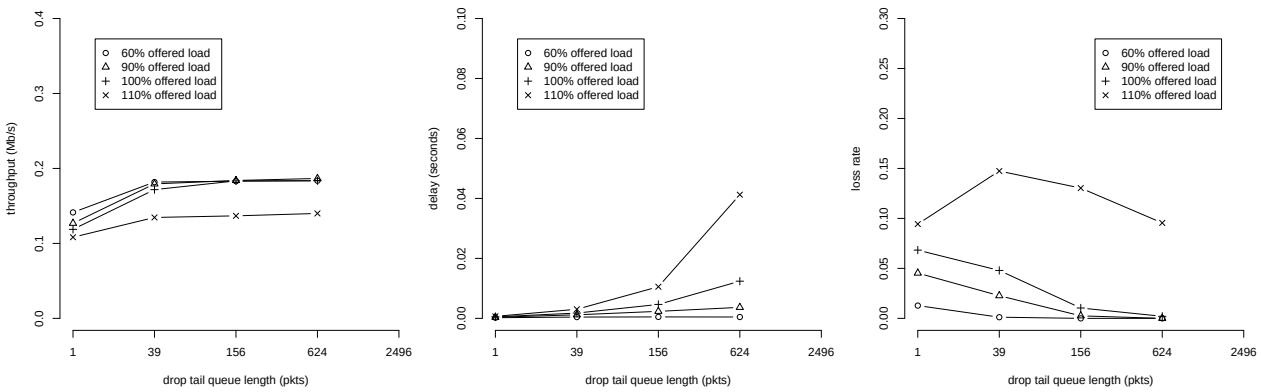


(a)

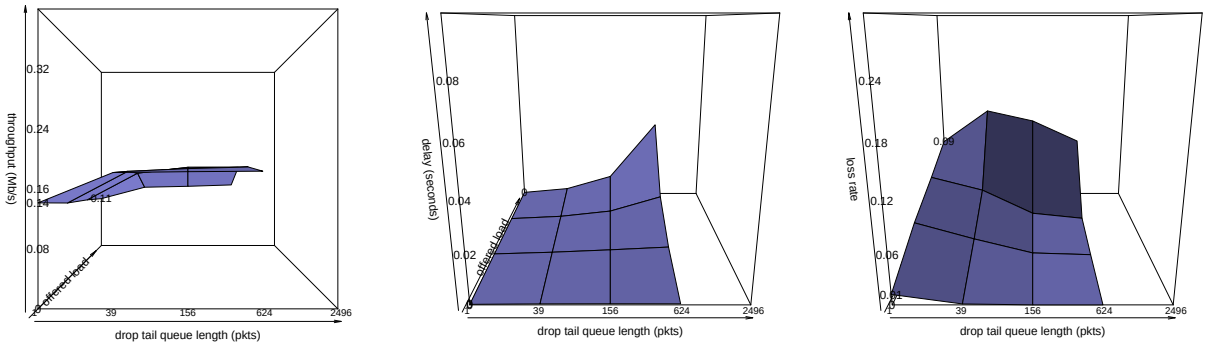


(b)

8.6: 2D (a) and 3D (b) per-flow profiles of mean throughput, delay, and loss for Cisco OC-3 with self-similar sources and drop-tail queuing discipline.

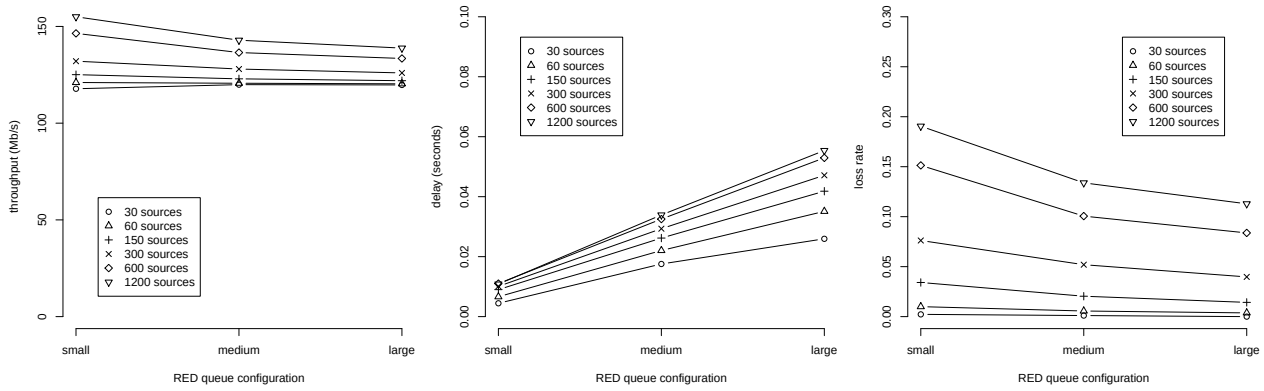


(a)

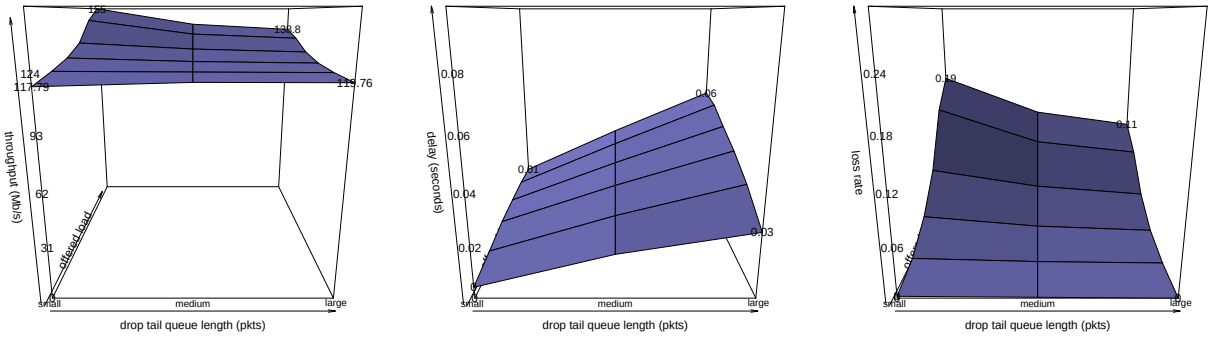


(b)

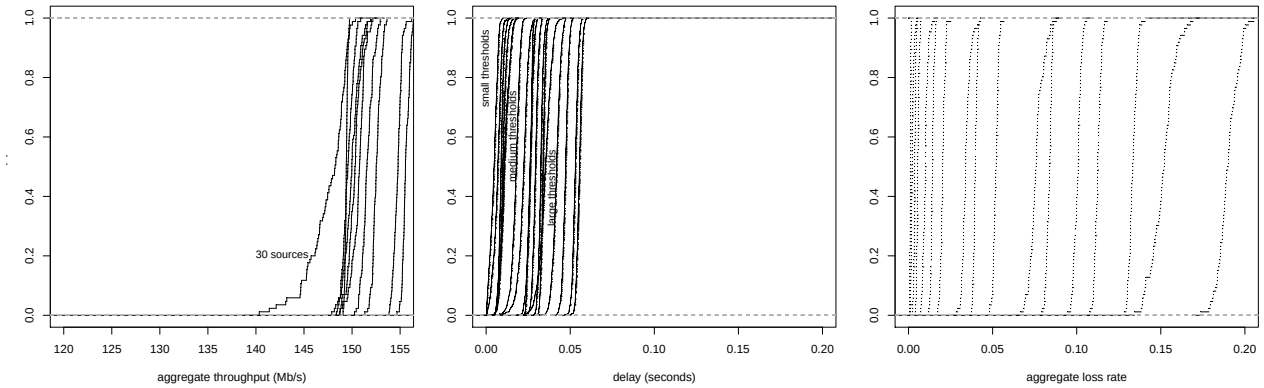
8.7: 2D (a) and 3D (b) per-flow profiles of mean throughput, delay, and loss for Juniper OC-3 with self-similar sources and drop-tail queuing discipline. As noted in Section 8.2.1, the Juniper M320 OC-3 interface has a hard upper limit of 50 milliseconds (≈624 packets of 1500 bytes) on buffer size.



(a)

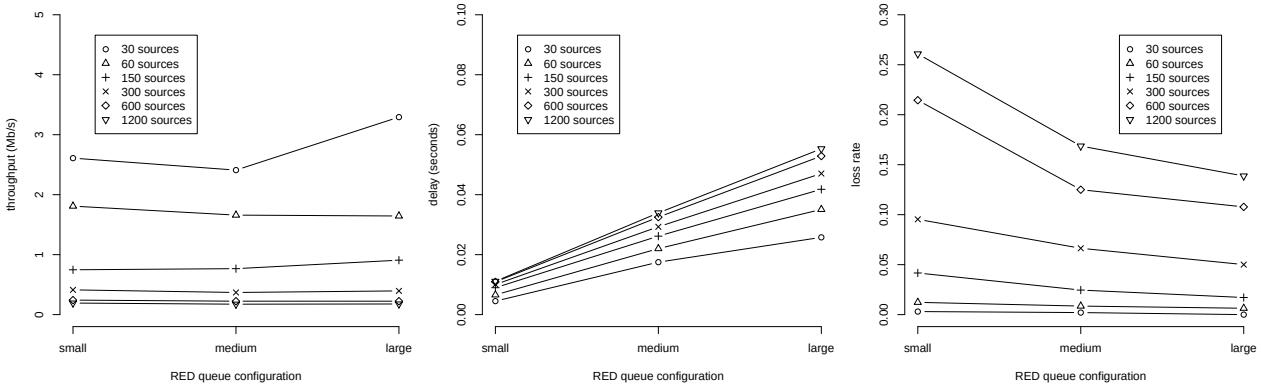


(b)

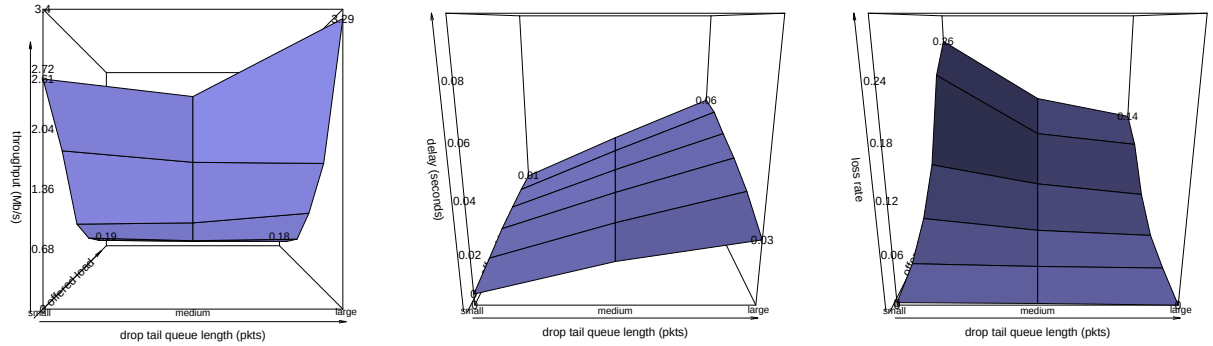


(c)

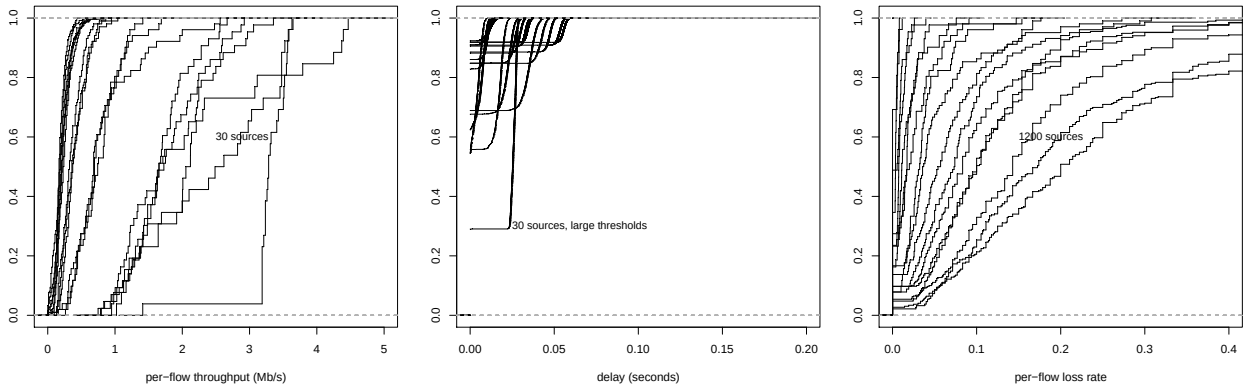
8.8: Aggregate profiles of mean throughput, delay, and loss shown for Cisco OC-3 with infinite TCP sources and RED queuing discipline. 2D and 3D profiles of mean throughput, delay, and loss shown in (a) and (b), respectively, and corresponding CDFs shown in (c).



(a)



(b)



(c)

8.9: Per-flow profiles of mean throughput, delay, and loss shown for Cisco OC-3 with infinite TCP sources and RED queuing discipline. 2D and 3D profiles of mean throughput, delay, and loss shown in (a) and (b), respectively, and corresponding CDFs shown in (c).

per-flow based performance profiles shown in Figure 8.9, RED-induced performance appears overall comparable to drop-tail based performance, with the exception of delay, with RED's delay performance profile being less sensitive and better than its drop-tail counterpart (which is to be expected, as reducing delay is one of the goals of RED). In short, without special tuning of the RED parameters and simply by relying on a set of default configuration settings, RED queues appear to result in somewhat more robust and slightly improved aggregate and per-flow performance profiles when compared to their drop-tail counterparts, even in rather constrained buffer configurations. These stronger insensitivity properties of RED queues are appealing in view of SLA-related efforts to meet certain levels of performance when carrying a customer's traffic, largely irrespective of the volume and type of traffic the customer generates.

8.4 Buffer Sizing: An SLA Perspective

The empirical findings discussed above illustrate that the "black art" of buffer sizing [87] could benefit from a new perspective, especially one that provides critical context for a range of traffic engineering issues, including configuration and provisioning decisions.

8.4.1 SLAs and Buffer Sizing: Illustrative Examples

In the following, we seek to illuminate the extent to which providers can engineer networks for robust SLA compliance, and help to quantify the risk (for a given buffer size, a given traffic mix, and a given router configuration) of SLA non-compliance. We consider a range of "toy" SLAs that provide assurances for network performance expressed in terms of delay and loss. While the SLAs we consider resemble real-world SLAs, they are necessarily much simpler than the latter, but we use them here mainly for illustrative purposes (for ISP-specific SLA-related information, see *e.g.*, [2, 22, 26]). In particular, our SLAs cover a range of target loss rates and target delay bounds, from reasonably tight (*i.e.*, 0.1% loss, 5 millisecond delay) to rather loose (*i.e.*, 2% loss, 50 millisecond delay), and we set the link utilization threshold beyond which an SLA becomes non-binding to 70%. As a result, some of our traffic scenarios with high offered loads are eliminated, since they are intended to saturate the bottleneck link and, in turn, tend to make the SLA non-binding for most of the time.

For a given buffer size-traffic scenario pair (X,Y), to measure SLA compliance, we obtain the "ground truth" by relying on the passive measurements of the traffic seen on the access link that is generated by the particular traffic scenario Y and fed into router A (see Figure 1) with buffer size X. We imitate actual SLA reporting by slicing the passive trace data into smaller intervals (here we use 10 second segments; actual intervals are typically 5-10 minutes). For each interval, we check whether or not the utilization during that interval was below the 70% threshold. If so, we compute loss rate and (average) delay, and if not, we simply discard the interval. To obtain the final score, we compute the average of all (valid) 10 second scores across the entire trace and report SLA compliance/non-compliance depending on whether the final scores are within the target delay and loss rate thresholds. In Figure 8.10a, each of the sixteen 2D plots corresponds to a particular SLA. For a given buffer size X (x-axis) and traffic/load scenario Y

(y-axis), we plot a particular letter in the corresponding (X,Y) coordinate if the SLA is violated in some way. A blank indicates that the SLA is met.

Coarse-grained reporting of SLA compliance is preferred by service providers because it allows for temporary SLA violations which smooth out when averaging over a large time interval (*e.g.*, week or month). Nevertheless, Figure 8.10a shows that from an SLA perspective, small buffer configurations are to be avoided across the whole spectrum of traffic types and loads. Moreover, for a tight SLA (top left plot), the buffer size needs to be substantial, in which case delay is almost certain to cause problems. This unavoidable tradeoff is evident and to some degree quantified in Figure 8.10a. Complying with a tight SLA is costly for the provider because it can easily be violated by a number of realistic traffic scenarios, despite configuring buffers to be large. In turn, the provider will charge the customer to compensate for the loss of revenues due to providing large buffers and crediting the customer when the SLA is not met. On the other hand, since a loose SLA (bottom right plot) is relatively easy to satisfy for a wide range of possible traffic mixtures, even with moderate buffer sizes, they are less taxing for the provider and hence cheaper for the customer. SLAs with mixed constraints (top right or bottom left plot) have their own economics and their tradeoffs can be read from Figure 8.10a. To show the impact that the choice of time scale has on reporting SLA compliance, and by extension, on buffer sizing, Figure 8.10b shows the results of a fine-grained reporting of SLA compliance. Instead of averaging the scores of the (valid) 10 second slices across the entire trace, we take the individual 10 second slices and associated raw scores (*i.e.*, no averaging) and plot a particular letter in the corresponding (X,Y) coordinate if the SLA is violated one way or another during at least one 10 second segment. A blank indicates that the SLA is met in each (valid) 10 second interval across the whole trace. The letter coding is explained in the caption of the figure. While fine-grained SLA compliance reporting is favored by customers because it relates more directly to customer-perceived quality of service, Figures 8.10a and 8.10b depict one aspect of how this tension between what the provider prefers and what the customer desires materializes when making configuration and provisioning decisions. For one and the same SLA, the risk of SLA non-compliance is typically greater with fine-grained SLA compliance reporting than with coarse-grained and can be quantified to some degree by the increase in buffer size needed to achieve roughly the same degree of robustness (to uncertainties in traffic type and volume) of SLA compliance.

8.4.2 Quantifying Risk of SLA Non-Compliance

A common way for customers and ISPs to limit risk of non-compliance with an SLA is through a utilization threshold: the SLA becomes non-binding if the threshold is exceeded over a given monitoring interval. A lower threshold effectively shields an ISP from uncertainties in the traffic mix while a higher threshold may be used by more risk-tolerant ISPs. Thus, we consider the tradeoffs between risk of non-compliance and expense, *i.e.*, buffer size. While the notion of cost is complex, we make the simplifying assumption that it can be equated with buffer size for illustrative purposes. For given delay and loss rate thresholds (*i.e.*, one of the sixteen subplots of Figure 8.10b), we check actual utilization levels in each interval. We consider the 90th percentile of the utilization measures over

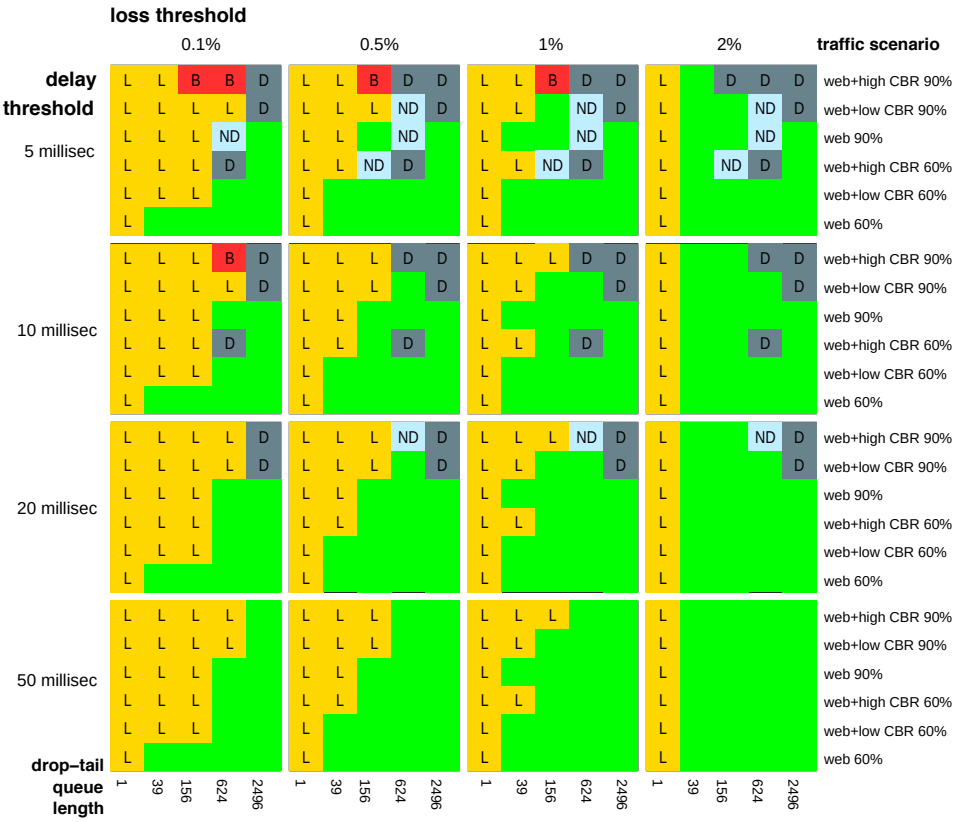
intervals where the thresholds are *not* exceeded as giving an indication of how much "wiggle room" remains before becoming non-compliant, *i.e.*, we use the 90th percentile utilization as a proxy for risk.

Considering now the results from Figure 8.10b, we make the following observations. First, there may be buffer configurations for which no realistic SLA can be supported. Indeed, the 1 packet buffer is one such case, since the loss rate threshold is routinely violated even with low overall utilization. Similarly for the 39 packet buffer case, the loss rate threshold is frequently violated. For very small buffer configurations like these, it is unlikely that an ISP will accept such risk. Referring to Table 8.3, we note that there are a number of buffer sizes predicted by the Stanford scheme which fall in this category. Second, for a given buffer size, as traffic becomes more heterogeneous (*i.e.*, from self-similar traffic, to self-similar traffic with different types of UDP flows) risk increases. Not surprisingly, there is also an increase in risk as the offered load increases from 60% to 90%. For 60% load average scenarios, the level of wiggle room tends to mirror the headroom in utilization, *i.e.*, about 40%. For high load averages, the level of wiggle room shrinks to between 20% and 5% of the link capacity. Lastly, for a given threshold pair and traffic mix, risk related to changes in configured buffer size tends to decrease as buffer size increases. The specific level of risk depends on the thresholds and traffic scenario. Referring again to Table 8.3, the lowest levels of risk but highest levels of cost can generally be associated with the BSCL and FPQ buffer sizing schemes.

The SLA perspective captured in Figure 8.10 coupled with observations above illuminates how a desire to engineer for robust SLA compliance can influence buffer sizing at the CEs and the PEs, and how buffer sizing decisions for edge routers can help to quantify the risk of SLA non-compliance. Clearly, an analytical treatment of how to engineer for robust SLA compliance and how to more generally quantify the risk of SLA non-compliance looms as a promising open problem, but looks very daunting at this point.

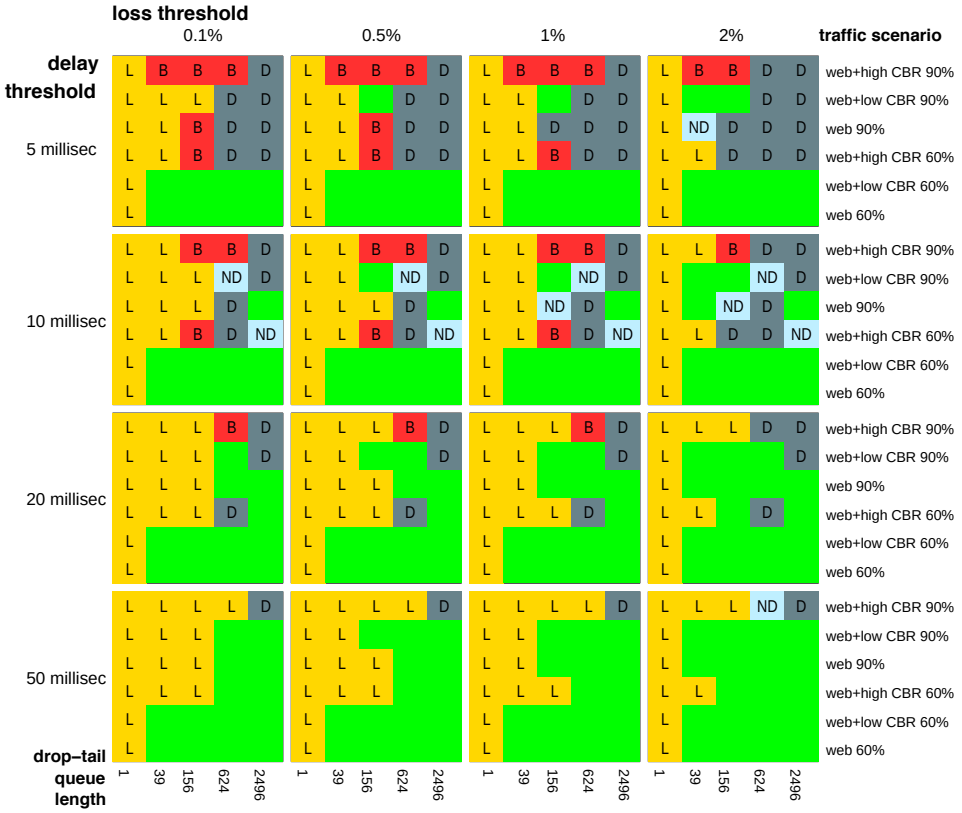
8.5 Discussion

An issue that has received attention in buffer sizing studies is synchronization among sources. For example, the lack of synchronization is critical to the central limit theorem arguments of Appenzeller *et al.* [51]. We examined whether there was evidence of synchronization in our experiments by using the methodology of Floyd *et al.* [102], *i.e.*, by analyzing the coefficient of variation (CoV; standard deviation divided by mean) of source sending rates over a wide range of time scales. Lack of synchronization should yield low values of CoV, especially as the time scale gets large. Regardless of buffer size, we found some indication of synchronization for low numbers of sources for the long-lived TCP scenarios (*e.g.*, 30 and 60 sources) but little evidence for other scenarios. Also, there was little correlation between buffer size and CoV for a given traffic scenario. Furthermore, CoV values were consistently lower for scenarios using RED queues versus drop-tail queues, confirming the analysis in [104]. These results suggest that focusing on the issue of synchronization with respect to buffer sizing may be of less importance, although further analysis is needed.



(a) Scores from 16 SLAs using coarse-grained (entire trace) SLA compliance reporting. Blank squares indicate SLA compliance.

8.10: 4x4 matrices of the scores from 16 SLAs (OC-3, 50 millisecond round-trip time) using (a) coarse- and (b) fine-grained SLA reporting. Each SLA has a delay and loss rate threshold. Four delay thresholds (top to bottom) are used and four loss rate thresholds (left to right) are used. Each of the 16 SLAs corresponds to one of the 4x4 matrix elements, and each of them is a 6x5 block, with columns representing 5 different drop-tail queue sizes and with rows representing 6 different traffic scenarios. Scores from an individual traffic scenario/queue length combination are coded as L=loss violation, NL=near loss violation, D=delay violation, ND=near delay violation, B=both delay and loss thresholds violated, and NB=near violation of both delay and loss thresholds. "Near" means that the score is within 10% of the threshold. Blank squares indicate SLA compliance.



(b) Scores from 16 SLAs using fine-grained (10 second time scale) SLA compliance reporting. Blank squares indicate SLA compliance in every (valid) 10 second interval across the entire trace.

8.10: 4x4 matrices of the scores from 16 SLAs (OC-3, 50 millisecond round-trip time) using (a) coarse- and (b) fine-grained SLA reporting. Each SLA has a delay and loss rate threshold. Four delay thresholds (top to bottom) are used and four loss rate thresholds (left to right) are used. Each of the 16 SLAs corresponds to one of the 4x4 matrix elements, and each of them is a 6x5 block, with columns representing 5 different drop-tail queue sizes and with rows representing 6 different traffic scenarios. Scores from an individual traffic scenario/queue length combination are coded as L=loss violation, NL=near loss violation, D=delay violation, ND=near delay violation, B=both delay and loss thresholds violated, and NB=near violation of both delay and loss thresholds. “Near” means that the score is within 10% of the threshold. Blank squares indicate SLA compliance.

Another interesting aspect of our study concerns Figure 8.2 (and to a lesser extent 8.3). The aggregate loss rates in the experiments with a buffer size of 1 packet are lower than the loss rates with the buffer sizes of 39 and 156 packets. Note that in these experiments, data traffic is unidirectional and buffers in the data and ACK direction are symmetric. In our initial investigation of this phenomenon, we found that in the 1 packet configuration, there was a large amount of packet loss in the ACK direction. We conjecture that the ACK loss results in a more drastic reduction or even some kind of pacing of the data traffic resulting in lower overall loss. We ran an additional test with a 1 packet buffer in the data direction and large enough buffer in the ACK direction to reduce ACK loss to zero. The result was that the loss rate in the data direction then rose above all of the tests with larger data buffers. We plan to investigate these phenomena in greater detail in future work.

While our study is designed to address a wide range of aspects of router buffer sizing, several caveats remain. First is the issue of representativeness of our experimental results. We argue that the use of commercial routers and the broad consideration of traffic and performance in our study provides an improved perspective on buffer sizing. However, this does not obviate the need to refine our experimental environment to consider, *e.g.*, more complex topologies, or for future analytical, simulation and live deployment tests which we believe also provide useful perspectives. Indeed, we believe that our empirical results will be useful for calibration of simulation environments and analytical models. Furthermore, many aspects of our study could be adopted by router vendors and service providers who already conduct extensive tests on live systems. An appealing aspect of [51] was a closed-form expression for specifying buffer sizes. Our results demonstrate the need for expressions that include a broader range of considerations, which is a focus for future work. Another critical issue is that traffic characteristics are known to be quite different in the core versus the edge of the Internet. Traffic in the core tends to be relatively smooth, while traffic in the edge tends to be more bursty. It is clear that the target of the Stanford study [51] is core routers that are therefore less susceptible to traffic bursts. However, there is a continuum between core and edge, so it is important to understand the conditions under which small buffers pose a risk. All of our tests were conducted with a two versions of TCP (from the FreeBSD and Linux kernels). While there are obviously a wide variety of TCP variants in the Internet today, it is not clear that the details of differences between the dominant versions of TCP would meaningfully alter our results. However, we plan to examine this issue in future work. Perhaps more interesting is the question of how specific versions of TCP behave with small buffers. A first step in this direction was taken in [113] by running simulations that included both NewReno and Vegas. However, more tests are needed, including some with high-speed TCP variants. Fairness is another consideration in assessing the impact of buffer sizing. Wischik addresses the issue of fairness in [211] and encourages consideration of other mechanisms (such as AQM) to address loss (and by extension fairness) in order to preserve the benefits of smaller buffers. While we do not address the issue of fairness directly in our study, the ability of RED to improve performance in our experiments would lend support to that argument. We plan to explore fairness in greater detail in future work. There are also the arguments by Dukkipati and McKeown in [94] in favor of flow completion time as the

"right" metric for congestion control. This raises the following important question: Should SLAs be designed to relate to FCT, and if so, how?

Chapter 9

Summary, Conclusions, and Future Directions

"It is what it is."
—Origin unknown.

In this dissertation, we advocate calibration as a key component in the design, development and rigorous testing of active measurement methodologies and networked systems. We argue for the use of controlled laboratory experiments as a means for addressing the limitations associated with realism that are inherent in standard network simulators, and limitations related to control and instrumentation that are associated with experiments in the live Internet. While this type of *in vitro*-like testing and experimentation is unlikely to fully replace experiments *in situ*, it offers complete control and repeatability, transparency and instrumentation, and with contributions from this dissertation, important aspects of realism. As we have asserted, each of these aspects are critical to network measurement experimentation and calibration. Through a series of three case studies, we illustrated the benefits and capabilities of our approach. In this chapter, we summarize our work, discuss conclusions, and outline future research directions.

9.1 A Framework for Calibrated Network Measurement

In the first part of this dissertation we described our calibration framework and approach, including crucial aspects of our laboratory environment. We now summarize these chapters.

9.1.1 Calibration Framework and Environment

We propose a framework for the calibration of active probe-based measurement tools and networked systems. We described a set of key issues to consider in calibration and our basic strategy for calibration. Our approach centers on running experiments in a controlled laboratory setting that enables transparency, control, and realism, which we claimed are required for calibrated network measurement.

We described the use of phase plots as one way to analyze the fine-grained measurements resulting from calibration experiments, and the SPLAT visualization tool that we designed for phase plot and scatter plot analysis. We demonstrated SPLAT's effectiveness as a means for mining different types of data sets and for use in network measurement

calibration. The measurements used by SPLAT can have a temporal component (*e.g.*, ingress/egress spacings), a spatial component (*e.g.*, IP addresses), other network-specific components (*e.g.*, flow attributes), or any combination thereof. Additionally, the measurements can be fine-grained (*e.g.*, packets, IP addresses) or more coarse-grained (*e.g.*, flows, prefixes). SPLAT offers a basic set of useful capabilities for mining and exploring the voluminous and semantically rich data sets that can be collected either in laboratory testbeds or in the live Internet. We illustrated its utility for exploring such diverse and challenging problems as assessing traffic dynamics (examples 1 and 2) and the potential for detecting/identifying network traffic anomalies through spatial fingerprinting (example 3). We also demonstrated its use in calibrating available bandwidth estimation tools in Chapter 6.

9.1.2 Traffic Generation

Creating a realistic laboratory environment for calibration demands the ability to create network traffic conditions that are representative of the live Internet. Previous successful approaches to traffic generation have focussed on generating application-specific (*e.g.*, web or multimedia) traffic features by generating request sequences that have the same statistical properties as live traffic from the modeled application. While these systems are useful for evaluating the behavior and performance of host systems (such as servers and caches), they can be cumbersome for use in router or switch tests and obviously only recreate one kind of application traffic, not the diversity of traffic observed in the Internet.

We described the architecture and implementation of HARPOON, a tool designed to create application-independent and statistically representative network traffic based on eight distributional characteristics of TCP and UDP flows. A key *self-configuration* feature of HARPOON enables parameters for these distributions to be automatically extracted from NetFlow data collected from a live router. We implemented HARPOON as a client-server application that can be used in testbed environments. We parameterized HARPOON using data collected from a NetFlow trace and from a set of packet traces and verified in controlled laboratory tests that HARPOON generates traffic that is qualitatively the same as the input data.

We demonstrated HARPOON's utility beyond simple background traffic generation through a series of throughput tests conducted on a Cisco 6509 router. We compared and contrasted the workload generated by HARPOON with the constant bit rate workloads recommended for standard tests. We found that HARPOON generated similar results for overall throughput, but that the stresses placed on router subsystems by HARPOON during these tests were significantly different. These results suggest that (in addition to the background traffic generation) HARPOON could be useful as a tool for providing network hardware designers and network operators insight into how systems might behave under realistic traffic conditions.

9.1.3 Path Emulation

In addition to realistic traffic conditions, experiments conducted in laboratory-based network research testbeds often require the capability to recreate characteristics of the wide area Internet paths accurately. We described the design and implementation of NETPATH, a new tool that provides scalable, precise path emulation capabilities. We also described VICT, a tool that facilitates the use of NETPATH in laboratory environments by automatically configuring the paths to and from NETPATH hosts. NETPATH is based on the Click modular router platform and was developed by significantly extending the capabilities of the basic elements offered in the Click software. In particular, we enable all of main memory to be used by NETPATH, and provide auxiliary queuing capability on disk. We found that implementing the resource management and scheduling components of NETPATH (particularly in the case of disk access) were challenging due to the tight design requirements for network path emulation systems in modern high-speed environments (*e.g.*, microsecond-level accuracy on Gigabit links). Our changes greatly improve the scalability of the system, enabling the emulation of path characteristics on Gigabit links.

Our evaluation of the performance of NETPATH considers throughput, ability to generate precise delays, and scalability. We show with that NETPATH, precise path emulation for high throughput links is now possible using a software-based system. NETPATH provides throughput and precision that are well beyond the capabilities of other software-based network emulators and approach those of an expensive hardware-based system. We also show that NETPATH can be effectively multiplexed to provide path emulation across several links simultaneously.

There are several key implications of our work on NETPATH. First, the ability to provide higher throughput and better precision than other software-based network emulation systems means that evaluation of delay sensitive protocols and systems can be conducted under more realistic conditions. Second, the scalability of NETPATH and its ability to accommodate high-throughput multi-link configurations means that resources in research labs can be used more efficiently. Third, the automated network configuration capability offered by VICT should enable NETPATH to be used widely in a variety of laboratory environments.

9.2 Case Studies in Calibrated Network Measurement

In the second part of this dissertation, we described three case studies that leveraged the control, transparency, and realism available in our laboratory calibration environment. We now summarize these chapters.

9.2.1 Available Bandwidth Estimation

In our first case study, we examined two canonical methods for estimating end-to-end available bandwidth. Our case study exposes potential biases and inaccuracies in these methodologies due to the use of commodity systems for high fidelity measurement and/or inaccurate assumptions about network system behavior and traffic dynamics.

As a result of these observations, we developed a calibrated Pathload-like tool called YAZ, which is consistently more accurate than prior ABETs. For example, in a 30 minute-long experiment in a traffic scenario using Internet-like bursty cross traffic, 81% of YAZ AB estimates are within 10% of the true value, while only 57% of Pathload estimates and 41% of Spruce estimates fall within the same window of accuracy. For this significantly higher level of accuracy, YAZ uses packets roughly of the same order of magnitude as Spruce, but at least an order of magnitude fewer packets than Pathload. If Pathload and Spruce represent a tradeoff between measurement accuracy and overhead, our results for YAZ suggest that this tradeoff is not fundamental. We believe that YAZ is representative of the type of active measurement tool that can be expected as a result of insisting on more stringent calibration.

SPLAT and the use of phase plots were instrumental in exposing existing ABET bias and errors, and the qualitative insight rendered by them was key to the resulting design of YAZ. We do not claim that such plots are a panacea, exposing all sources of bias and error for all active measurement tools. However, we believe that there is an outstanding need for new flexible analysis and visualization tools capable of more fully exposing the enormous quantity of high fidelity measurement data that can be collected in the calibration framework we advocate in this dissertation. We believe SPLAT represents a positive step in that direction.

9.2.2 Measurement of Packet Loss

The purpose of our second case study was to understand how to measure end-to-end packet loss characteristics accurately with probes and in a way that enables us to specify the impact on the bottleneck queue. We began by evaluating the capabilities of the standard Poisson-modulated probing methodology of RFC 2680 [47]. Our laboratory calibration strategy enabled us to evaluate Poisson probing in a laboratory setting with a range of repeatable traffic scenarios, and in which we had access to detailed ground truth measurements. Our initial tests indicate that standard Poisson probing is ineffective at measuring loss episode frequency or measuring loss episode duration, especially when subjected to TCP (reactive) cross traffic.

These experimental results led to our development of a geometrically distributed probe process that provides significantly more accurate estimation of loss characteristics than simple Poisson probing. The experimental design is constructed in such a way that the performance of the accompanying estimators relies on the total number of probes that are sent, but not on their sending rate. Moreover, simple techniques to validate the measurement output are introduced, enabling users to have confidence that the measurements correctly report the state of the network. We implemented this method in a new tool, BADABING, which we tested extensively in our laboratory. Our tests demonstrate that BADABING, in most cases, accurately estimates loss frequencies and durations as well as loss rates over a range of cross traffic conditions. For the same overall packet rate, our results show that BADABING is significantly more accurate than the Poisson probing methodology of RFC 2680 for measuring loss episode characteristics.

9.2.3 Router Buffer Sizing

In our third case study we examined the problem of router buffer sizing through our calibration framework. Our goal was to broadly evaluate the tradeoffs between buffer configuration, traffic mix, and performance. Our experiments show that all performance metrics are, by and large, insensitive to router architecture, and that aggregate throughput (the performance metric used in Appenzeller *et al.* [51]) is also insensitive to buffer size and traffic mix. However, all other performance metrics show clear dependencies on buffer size, traffic mix, traffic volume and round trip time delay.

By mapping four different known buffer sizing policies into the design space established by our experiments, we are able to compare them and assess their advantages and disadvantages. In particular, our mapping highlights the performance risks of the buffer sizes specified by [51], and shows that the policies proposed in [88, 156] either result in relatively large buffers or in buffers similar to [51], depending on traffic mix. This illustrates the need for a new perspective in support of making more informed buffer sizing decisions. Thus, the second goal of our case study is to put forward a novel perspective: that of the ISP and by extension the SLAs that drive their networks. That is, we argue that buffer sizing decisions must not only grapple with constraints of router design and network performance, but also ISP economics [80]. To this end, we consider a set of toy SLAs and use them to study the buffer sizing problem within the context of the economic incentives behind a marketplace for SLAs. We find that the stringency of the SLA and how SLA compliance is monitored to be contributing factors for making informed buffer configuration decisions at the network edge. For example, tying SLA-specific performance closely to traffic mix and load opens up the possibility that careful traffic engineering may enable smaller router buffers to meet certain SLAs. In this context, the observed benefits of AQM in small buffer configurations suggest that this often maligned technology may play a more prominent role in networks with a thriving SLA business model.

While we believe that the problem of router buffer sizing in a competitive Internet remains largely unsolved, our results serve to illuminate a wide range of known and new issues, and to highlight interesting challenges that remain.

9.3 Future Work

The focus of this dissertation was on developing a calibration framework and environment and illustrating the capabilities of that environment. While we believe that our case studies demonstrate the potential benefits of our approach, there are many issues yet to be examined. Our future goals include the following.

1. How generalizable is the calibration framework advocated in this dissertation, and can the notion of calibration be usefully formalized in this context? What role can and should standardized test suites or canonical bodies of network and traffic setups play? What complementary roles should simulation, laboratory emulation, and wide area experiments play in calibration and testing of network measurement tools, and, more generally, network experiments?

2. Our HARPOON study showed that relatively coarse features could go a long way toward recreating traffic conditions measured in the live Internet. Work by others has since suggested that a number of other elements such as link loss/error rates, topology, and application-level characteristics may be important to consider. However, it is still an open question as to what features are sufficient for creating representative conditions over a wide range of time scales, or whether simple recipes exist. Furthermore, as applications and users continue to evolve, evaluating the representativeness and realism of traffic models requires ongoing study and refinement.

3. While our tool for measuring packet loss, BADABING, enables superior accuracy and a better understanding of link impact versus timeliness of measurement, there is still room for improvement. First, we intend to investigate why p=0.1 does not appear to work well even as N increases. Second, we plan to examine the issue of appropriate parameterization of BADABING, including packet sizes and the α and τ parameters, over a range of realistic operational settings including more complex multihop paths. Finally, we have considered adding adaptivity to our probe process model in a limited sense. We are also considering alternative, parametric methods for inferring loss characteristics from our probe process. Another task is to estimate the variability of the estimates of congestion frequency and duration themselves directly from the measured data, under a minimal set of statistical assumptions on the congestion process.

4. Our buffer sizing case study reveals the need for additional empirical measurements and analytical study to better understand the problem, and to design methodologies that allow service providers to effectively support SLAs. For example, our results concerning performance improvements with AQM warrant additional analysis. While our focus in this case study was on qualitative analysis, statistical comparisons (*e.g.*, tests for significance of characteristics observed in Section 8.3.3) also support AQM as providing somewhat more robust/insensitive performance. Exploring the specific nature of these improvements, and the performance and economic tradeoffs between buffer size and management policy both empirically and analytically are subjects for future work. Our intent remains to develop a model for buffer sizing that takes these issues into account.

5. Systems such as PlanetLab have shown that there is a tremendous need within the networking community for a widely-deployed shared infrastructure to test new applications and to gather a variety of geographically diverse network and application measurements. Unfortunately, PlanetLab's popularity has led to poor performance for users (*i.e.*, the so-called "tragedy of the commons" problem), casting doubt on the quality of some types of network measurements derived from PlanetLab-based studies. What system designs (*e.g.*, in areas such as virtualization, architecture, operating systems, network abstractions) for such an infrastructure can best support a range of measurement activities? Can components to aid in at least limited calibration be incorporated in such a setting? System-level support for robust network measurement would enable shared distributed systems to be applied toward a broader range of measurement-related questions.

6. The focus in this dissertation has been on experimentation in a controlled and realistic laboratory setting. While we have discussed the shortcomings of both simulation and *in situ* environments, namely the lack of realism and the lack of ground truth and control, respectively, these remain valuable experimental settings and will certainly see continued wide use by the research community. Rather than espousing an exclusive approach based in a laboratory, we believe that a *complementary* strategy, leveraging the strengths of simulation, laboratory, and *in situ* environments, will result in the deepest and most valuable insights into network behavior.

   In the particular case of simulation, some of the key limitations with respect to realism were described by Floyd and Kohler in their paper "Internet Research Needs Better Models" [105]. Indeed, current deficiences in simulation environments should not be considered a fatal obstacle. Simulation has several compelling advantages over laboratory and *in situ* environments, including the ability to simulate very large networks and complex scenarios that would be unfeasible in a laboratory because of complexity and cost. In a technical report, we have already examined particular deficiences in the ns-2 simulator related to packet dynamics over short time scales [192]. For example, the delayed acknowledgment mechanism in the ns-2 implementation of TCP introduces artifacts unlike any TCP deployed in the live Internet. We used laboratory experiments as well as traces collected in the Internet as a basis for evaluating aspects of realism, and designed modifications to ns-2 that enabled more realistic packet arrivals over short time scales. We plan to continue this effort of improving network simulation environments.

LIST OF REFERENCES

[1] Abilene backbone network. http://abilene.internet2.edu/.

[2] AT&T Managed Internet Service (MIS). http://new.serviceguide.att.com/mis.htm.

[3] CAIDA visualization tools. http://www.caida.org/tools/visualization/.

[4] Cisco Systems. Cisco 12000 Series Internet Router Architecture: Packet Switching. http://www.cisco.com/warp/public/63/arch12000-packetsw.html.

[5] Cisco Systems Inc. Catalyst 6500 series switches. http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/index.htm.

[6] Cisco's IOS Netflow feature. http://www.cisco.com/warp/public/732/netflow.

[7] Coral: Passive network traffic monitoring and statistics collection. http://www.caida.org/Tools/Coral.

[8] The DETER testbed. http://www.isi.edu/deter/.

[9] Emulab — network emulation testbed. http://www.emulab.net.

[10] Endace DAG Network Monitoring Cards. http://www.endace.com/Default.aspx?pageid=34.

[11] Ethereal: A network protocol analyzer. http://www.etheral.com/.

[12] The eXpat XML parser. http://expat.sourceforge.net.

[13] GloMoSim: Global Mobile Information Systems Simulation Library. http://pcl.cs.ucla.edu/projects/glomosim/.

[14] Juniper Networks. Configuring the Scheduler Buffer Size. http://www.juniper.net/techpubs/software/junos/junos74/swconfig74-cos/html/cos-scheduler-maps3.html.

[15] Juniper Networks. Data Flow Through the M320 Router and T640 Routing Node Packet Forwarding Engine. http://www.juniper.net/techpubs/software/nog/nog-hardware/html/key-components12.html.

[16] Nam: Network animator. http://www.isi.edu/nsnam/nam/.

[17] National laboratory for applied network research. http://moat.nlanr.net/Datacube.

[18] Nessus. http://www.nessus.org/.

[19] Netflow services solutions guide (Netflow white paper). http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm.

[20] NISTnet wide area network emulator. http://www-x.antd.nist.gov/nistnet/.

[21] NLANR Active Measurement Program - AMP. http://amp.nlanr.net/AMP.

[22] NTT Communications Global IP Network Service Level Agreement (SLA). http://www.us.ntt.net/support/sla/network/.

[23] PlanetLab—an open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org.

[24] Secure and Accountable Measurement Infrastructure (SAMI). http://www.psc.edu/networking/projects/sami/.

[25] Spirent Communications Inc. Adtech AX/4000 broadband test system. http://www.spirentcom.com/analysis/product_line.cfm?pl=1&WS=173&wt=2.

[26] Sprint NEXTEL service level agreements. http://www.sprint.com/business/support/serviceLevelAgreements.jsp.

[27] SSFnet network simulator. http://www.ssfnet.org.

[28] The surveyor project. http://www.advanced.org/csg-ippm/.

[29] tcptrace—official homepage. http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html.

[30] The PingER Project. http://www-iepm.slac.stanford.edu/pinger/.

[31] The University of New Hampshire Interoperability Laboratory. http://www.iol.unh.edu/.

[32] The WAN in Lab (WiL) Project. http://wil.cs.caltech.edu/.

[33] The Wisconsin Advanced Internet Laboratory. http://wail.cs.wisc.edu/.

[34] Tobi Oetiker's MRTG - The Multi-Router Traffic Grapher. http://oss.oetiker.ch/mrtg/.

[35] Web Polygraph. http://www.web-polygraph.org/.

[36] XML-RPC home page. http://www.xmlrpc.org.

[37] Internet protocol performance metrics. http://www.advanced.org/IPPM/index.html, 1998.

[38] The IEEE 802.1Q - Virtual LANs. http://www.ieee802.org/1/pages/802.1Q.html, 2001.

[39] Workshop on models, methods and tools for reproducible network research. http://www.acm.org/sigs/sigcomm/sigcomm2003/workshop/mometools, 2003.

[40] J. Abbate. *Inventing the Internet*. MIT Press, 2000.

[41] P. Abry and D. Veitch. Wavelet analysis of long range dependent traffic. *IEEE Transactions on Information Theory*, 44(1):2–15, 1998.

[42] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson. Creating a scalable architecture for Internet measurement. *IEEE Network*, 1998.

[43] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Proceedings of ACM IMC '03*, October 2003.

[44] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, Ohio University, 1997.

[45] M. Allman, W. Eddy, and S. Osterman. Estimating loss rates with TCP. *ACM Performance Evaluation Review*, 31(3), December 2003.

[46] G. Almes, S. Kalidindi, and M. Zekauskas. RFC 2679. A One-way Delay Metric for IPPM. ftp://ftp.rfc-editor.org/in-notes/rfc2679.txt, September 1999.

[47] G. Almes, S. Kalidindi, and M. Zekauskas. RFC 2680. A One Way Packet Loss Metric for IPPM. ftp://ftp.rfc-editor.org/in-notes/rfc2680.txt, September 1999.

[48] S. Alouf, P. Nain, and D. Towsley. Inferring network characteristics via moment-based estimators. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[49] K.G. Anagnostakis and M.B. Greenwald. A hybrid direct-indirect estimator of network internal delays. In *Proceedings of ACM SIGMETRICS '04 (poster session)*, 2004.

[50] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, 2001.

[51] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM '04*, Portland, OR, 2004.

[52] A.C. Arpaci-Dusseau and R.H. Arpaci-Dusseau. Information and control and gray-box systems. In *Proceedings of ACM Symposium on Operating Systems Principles*, 2001.

[53] K. Avrachenkov, U. Ayesta, E. Altman, P. Nain, and C. Barakat. The Effect of Router Buffer Size on TCP Performance. In *Proceedings of the LONIIS Workshop on Telecommunication Networks and Teletraffic Theory*, St. Petersburg, Russia, January 2002.

[54] K. Avrachenkov, U. Ayesta, and A. Piunovskiy. Optimal choice of the buffer size in the internet routers. In *Proceedings of IEEE Conference on Decision and Control*, 2005.

[55] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot. The role of PASTA in network measurement. In *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.

[56] S. Banerjee, T. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Proceedings of Passive and Active Measurement Workshop*, Antibes Juan-les-Pins, France, April 2004.

[57] G. Banga, J. Mogul, and P. Druschel. A Scalable and Explicit Event Delivery Mechanism for UNIX. In *Proceedings of USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[58] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski. Modeling Internet backbone traffic at the flow level. *IEEE Transactions on Signal Processing (Special Issue on Networking)*, August 2003.

[59] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, Madison, WI, June 1998.

[60] P. Barford and M. Crovella. A performance evaluation of hyper text transfer protocols. In *Proceedings of ACM SIGMETRICS '99*, Atlanta, GA, May 1999.

[61] P. Barford and M. Crovella. Critical path analysis of TCP transactions. In *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweden, September 2000.

[62] P. Barford and J. Sommers. A comparison of probe-based and router-based methods for measuring packet loss. Technical report, University of Wisconsin-Madison, 2003.

[63] P. Barford and J. Sommers. Comparing probe- and router-based packet loss measurements. *IEEE Internet Computing*, September/October 2004.

[64] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services,. In *First Symposium on Network Systems Design and Implementation (NSDI'04)*, San Francisco, CA, March 2004.

[65] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. In *Proceedings of ACM SIGCOMM*, August 2006.

[66] P. Benko and A. Veres. A passive method for estimating end-to-end TCP packet loss. In *Proceedings of IEEE Globecom '02*, Taipei, Taiwan, November 2002.

[67] J. Bolot. End-to-end packet delay and loss behavior in the Internet. In *Proceedings of ACM SIGCOMM*, San Francisco, CA, September 1993.

[68] R. Bradford, R. Simmonds, and B. Unger. A parallel discrete event IP network emulator. In *Proceedings of IEEE MASCOTS*, 2000.

[69] S. Bradner. RFC 1242. Benchmarking Terminology for Network Interconnect Devices. ftp://ftp.rfc-editor.org/in-notes/rfc1242.txt, July 1991.

[70] S. Bradner and J. McQuaid. RFC 2544. Benchmarking Methodology for Network Interconnect Devices. ftp://ftp.rfc-editor.org/in-notes/rfc2544.txt, March 1999.

[71] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGMETRICS '96*, Philadelphia, PA, May 1996.

[72] S. Brumelle. On the relationship between customer and time averages in queues. *Journal of Applied Probability*, 8, 1971.

[73] T. Bu and D. Towsley. Fixed point approximation for TCP behavior in an AQM network. In *Proceedings of ACM SIGMETRICS '01*, San Diego, CA, June 2001.

[74] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *Proceedings of IEEE INFOCOM '00*, Tel-Aviv, Israel, March 2000.

[75] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. In *Proceedings of Performance '96*, Lausanne, Switzerland, October 1996.

[76] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (SNMP). ftp://ftp.rfc-editor.org/in-notes/rfc1157.txt, 1990.

[77] Y.-C. Cheng, U. Hölzle, N. Cardwell, S. Savage, and G.M. Voelker. Monkey see, monkey do: A tool for TCP tracing and replaying. In *Proceedings of the USENIX 2004 Conference*, June 2004.

[78] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for Web Traffic. In *Proceedings of ACM SIGCOMM '00*, Stockholm, Sweeden, August 2000.

[79] K. Claffy, G. Polyzos, and H.-W. Braun. Internet traffic flow profiling. Technical Report TR-CS93-328, University of California San Diego, November 1989.

[80] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, 2005.

[81] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson. Design principles for accurate passive measurement. In *Proceedings of PAM '00*, 2000.

[82] W. Cleveland, D. Lin, and D.Sun. IP packet generation: Statistical models for TCP start times based on connection rate superposition. In *Proceedings of ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[83] M. Coates and R. Nowak. Network loss inference using unicast end-to-end measurement. In *Proceedings of ITC Conference on IP Traffic, Measurement and Modeling*, September 2000.

[84] H.M. Collins. *Changing order: replication and induction in scientific practice*. The University of Chicago Press, 1985.

[85] Committee on Research Horizons in Networking. *Looking Over the Fence at Networks: A Neighbor's View of Networking Research*. National Academy Press, Washington, D.C., 2001.

[86] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[87] A. Dhamdhere and C. Dovrolis. Open issues in router buffer sizing. *ACM SIGCOMM Computer Communications Review*, 36(1):87 – 92, January 2006.

[88] A. Dhamdhere, H. Jiang, and C. Dovrolis. Buffer Sizing for Congested Internet Links. *Proceedings of IEEE INFOCOM '05*, 2005.

[89] S. Donnelly. *High Precision Timing in Passive Measurements of Data Networks*. PhD thesis, University of Waikato, 2002.

[90] A. Downey. Using pathchar to estimate Internet link characteristics. In *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, August 1999.

[91] N. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley. Network delay tomography from end-to-end unicast measurements. In *Proceedings of 2001 International Workshop on Digital Communications*, Taormina, Italy, September 2001.

[92] N. Duffield, F. Lo Presti, V. Paxson, and D. Towsley. Inferring link loss using striped unicast probes. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, April 2001.

[93] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003.

[94] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communications Review*, 36(1):59 – 62, January 2006.

[95] M. Enachescu, A. Goel, N. McKeown, and T. Roughgarden. Part III: Routers with Very Small Buffers. *ACM SIGCOMM Computer Communications Review*, 35(3), July 2005.

[96] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, April 1996.

[97] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, 2003.

[98] K. Fall. Network Emulation in the Vint/NS Simulator. In *Proceedings of the Fourth IEEE Symposium on Computers and Communication*, Red Sea, Egypt, July 1999.

[99] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, Boston, MA, August 1999.

[100] A. Feldmann, A. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of internet wan traffic. In *Proceedings of ACM SIGCOMM '98*, August 1998.

[101] S. Floyd. Random early detection resources. http://www.icir.org/floyd/red.html.

[102] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM '00*, 2000.

[103] S. Floyd and T. Henderson. RFC 2582. The NewReno Modification to TCP's Fast Recovery Algorithm. ftp://ftp.rfc-editor.org/in-notes/rfc2582.txt, 1999.

[104] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[105] S. Floyd and E. Kohler. Internet research needs better models. In *Hotnets-I*, Princeton, NJ, October 2002.

[106] S. Floyd and V. Paxson. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[107] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4), August 2001.

[108] M. Fomenkov, K. Keys, D. Moore, and K. Claffy. Longitudinal study of internet traffic from 1998-2001: a view from 20 high performance sites. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2002.

[109] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 2003.

[110] S. Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts. Statistical bandwidth sharing: A study of congestion at flow level. In *Proceedings of ACM SIGCOMM '01*, San Diego, CA, August 2001.

[111] M. Fullmer and S. Romig. The OSU flow-tools package and Cisco NetFlow logs. In *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, New Orleans, LA, December 2000.

[112] M. Garetto and D. Towsley. Modeling, Simulation and Measurements of Queuing Delay under Long-tail Internet Traffic. In *Proceedings of ACM SIGMETRICS '03*, San Diego, CA, June 2003.

[113] S. Gorinsky, A. Kantawala, and J. Turner. Link Buffer Sizing: A New Look at the Old Problem. In *Proceedings of the IEEE Symposium on Computers and Communications*, Cartagena, Spain, June 2005.

[114] I. Hacking. *Representing and Intervening: Introductory topics in the philosophy of natural science*. Cambridge University Press, 1983.

[115] F. Hernández-Campos, F.D. Smith, and K. Jeffay. How real can synthetic network traffic be? In *Proceedings of ACM SIGCOMM '04 (Poster Session)*, 2004.

[116] J. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *Proceedings of ACM SIGCOMM '96*, Palo Alto, CA, August 1996.

[117] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.

[118] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–332, August 1988.

[119] V. Jacobson. traceroute. ftp://ftp.ee.lbl.gov/traceroute.tar.Z, 1989.

[120] V. Jacobson. Modified TCP congestion control algorithm. End-to-end interest mailing list: ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail, April 1990.

[121] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of ACM SIGCOMM '02*, Pittsburgh, Pensylvania, August 2002.

[122] M. Jain and C. Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *Proceedings of ACM IMC '04*, Taormina, Sicily, Italy, October 2004.

[123] M. Jain and C. Dovrolis. End-to-end estimation of the available bandwidth variation range. In *Proceedings of ACM SIGMETRICS '05*, Banff, Alberta, Canada, June 2005.

[124] R. Jain and S.A. Routhier. Packet Trains - Measurements and a New Model for Computer Network Traffic. *IEEE Journal on Selected Areas in Communications*, 4(6):986–995, September 1986.

[125] S. Jansen and A. McGregor. Simulation with real world network stacks. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, 2005.

[126] G. Jin and B. Tierney. System capability effects on algorithms for network bandwidth measurement. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '03*, October 2003.

[127] S. Jin and A. Bestavros. GISMO: Generator of Streaming Media Objects and Workloads. *Performance Evaluation Review*, 29(3), 2001.

[128] Y. Joo, V. Ribeiro, A. Feldmann, A. Gilbert, and W. Willinger. TCP/IP traffic dynamics and network performance: a lesson in workload modeling, flow control, and trace-driven simulations. *ACM SIGCOMM Computer Communications Review*, 31(2), April 2001.

[129] S. Kalidindi and M. Zekauskas. Surveyor: An Infrastructure for Internet Performance Measurements. In *Proceedings of INET '99*, 1999.

[130] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. CapProbe: a simple and accurate capacity estimation technique. In *Proceedings of ACM SIGCOMM*, Portland, OR, August 2004.

[131] T. Karagiannis, M. Faloutsos, M. Molle, and A. Broido. A nonstationary Poisson view of Internet traffic. In *Proceedings of INFOCOM '04*, 2004.

[132] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed structure of addresses in IP traffic. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop '02*, Marseilles, France, October 2002.

[133] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[134] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, 2000 2000.

[135] K. Lakshiminarayanan, V.N. Padmanabhan, and J. Padhye. Bandwidth estimation in broadband access networks. In *Proceedings of ACM IMC '04*, Taormina, Sicily, Italy, 2004.

[136] L. Le, J. Aikat, K. Jeffay, and F.D. Smith. The effects of active queue management on web performance. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, 2003.

[137] B.M. Leiner, V.G. Cerf, D.D. Clark, R.E. Kahn, L. Kleinrock, D.C. Lynch, J. Postel, L.G. Roberts, and S. Wolff. A Brief History of the Internet. http://www.isoc.org/internet/history/brief.shtml.

[138] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 2:1–15, 1994.

[139] J. Liu and M. Crovella. Using loss pairs to discover network properties. In *Proceedings of ACM Internet Measurement Workshop*, San Francisco, CA, October 2001.

[140] X. Liu, K. Ravindran, B. Liu, and D. Loguinov. Single-hop probing asymptotics in available bandwidth estimation: sample-path analysis. In *Proceedings of ACM IMC '04*, Taormina, Sicily, Italy, October 2004.

[141] F. Lo Presti, N.G. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal delay distributions. *IEEE/ACM Transactions on Networking*, 10(6):761–775, 2002.

[142] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *Proceedings of ACM Symposium on Operating Systems Principles*, 2003.

[143] J. Mahdavi, V. Paxson, A. Adams, and M. Mathis. Creating a scalable architecture for Internet measurement. In *Proceedings of INET '98*, Geneva, Switzerland, July 1998.

[144] R. Mandeville. RFC 2285. Benchmarking Terminology for LAN Switching Devices. ftp://ftp.rfc-editor.org/in-notes/rfc2285.txt, February 1998.

[145] R. Mandeville and J. Perser. RFC 2889. Benchmarking Methodology for LAN Switching Devices. ftp://ftp.rfc-editor.org/in-notes/rfc2889.txt, August 2000.

[146] J. Martin and A. Nilsson. On service level agreements for IP networks. In *IEEE INFOCOM 2002*, volume 2, 2002.

[147] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018. TCP Selective Acknowledgement Options. ftp://ftp.rfc-editor.org/in-notes/rfc2018.txt, 1996.

[148] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[149] W. Matthews and L. Cottrell. The PingER project: Active Internet performance monitoring for the HENP community. *IEEE Communications Magazine*, May 2000.

[150] M. May, T. Bonald, C. Diot, and J. Bolot. Reasons not to Deploy RED. In *Proceedings of the Seventh IEEE International Workshop on Quality of Service*, London, England, June 1999.

[151] S. McCanne and S. Floyd. UCB/LBNL/VINT Network Simulator—ns (version 2). http://www.isi.edu/nsnam/ns/.

[152] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proceedings of Global Internet Symposium*, 2000.

[153] D. Mills. A Brief History of NTP Time: Memoirs of an Internet Timekeeper. *ACM SIGCOMM Computer Communications Review*, 33(2):9–21, April 2003.

[154] D.L. Mills. Internet delay experiments. ftp://ftp.rfc-editor.org/in-notes/rfc889.txt, December 1983.

[155] S. Moon, P. Skelley, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[156] R. Morris. TCP Behavior with Many Flows. In *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[157] R. Morris. Scalable TCP Congestion Control. In *Proceedings of IEEE INFOCOM*, April 2000.

[158] T. Munzner. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University, 2000.

[159] D. Mutz, G. Vigna, and R. Kemmerer. An Experience Developing an IDS Simulator for Black-Box Testing of Network Intrusion Detection Systems. In *Proceedings of ACSAC*, 2003.

[160] E. Nahum, M. Rosu, S. Seshan, and J. Almeida. The Effects of Wide-Area Conditions on WWW Server Performance. In *Proceedings of ACM SIGMETRICS '01*, Cambridge, MA, June 2001.

[161] D. Newman, G. Chagnot, and J. Perser. Internet Core Router Test. http://www.lightreading.com/document.asp?site=testing&doc_id=4009, March 2001.

[162] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, Setpember 1998.

[163] D. Papagiannaki, R. Cruz, and C. Diot. Network performance monitoring at small time scales. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '03*, Miami, FL, October 2003.

[164] K. Park, G. Kim, and M. Crovella. On the Effect of Traffic Self-Similarity on Network Performance. In *Proceedings of SPIE International Conference on Performance and Control of Network Systems*, November 1997.

[165] K. Park and W. Willinger. *Self-similar network traffic and performance evaluation*. Wiley Interscience, 2000.

[166] A. Pásztor and D. Veitch. A precision infrastructure for active probing. In *Proceedings of Passive and Active Measurement Workshop*, Amsterdam, Netherlands, 2001.

[167] A. Pásztor and D. Veitch. PC-based Precision Timing without GPS. In *Proceedings of ACM SIGMETRICS*, Marina Del Rey, CA, June 2002.

[168] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of ACM SIGCOMM '96*, Palo Alto, CA, August 1996.

[169] V. Paxson. End-to-end Internet Packet dynamics. In *Proceedings of ACM SIGCOMM*, Cannes, France, October 1997.

[170] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California Berkeley, 1997.

[171] V. Paxson. On calibrating measurements of packet transit times. In *Proceedings of ACM SIGMETRICS '98*, pages 11–21, Madison, WI, June 1998.

[172] V. Paxson. Strategies for sound Internet measurement. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '04*, Taormina, Italy, November 2004.

[173] V. Paxson, A. Adams, and M. Mathis. Experiences with NIMI. In *Proceedings of Passive and Active Measurement Workshop*, 2000.

[174] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[175] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. In *Proceedings of ACM SIGCOMM HotNets–III*, 2004.

[176] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, New Orleans, LA, December 2000.

[177] R. Prasad, M. Jain, and C. Dovrolis. Effects of interrupt coalescence on network measurements. In *Proceedings of PAM '04*, April 2004.

[178] H. Pucha, Y.C. Hu, and Z.M. Mao. On the Impact of Research Network based Testbeds on Wide-Area Experiments. In *Proceedings of ACM Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[179] G. Raina, D. Towsley, and D. Wischik. Part II: Control Theory for Buffer Sizing. *ACM SIGCOMM Computer Communications Review*, 35(3), July 2005.

[180] V. Riberio, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient available bandwidth estimation for network paths. In *Proceedings of Passive and Active Measurement Workshop*, 2003.

[181] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communications Review*, 27, January 1997.

[182] M. Roughan. Fundamental bounds on the accuracy of network performance measurements. In *ACM SIGMETRICS*, June 2005.

[183] S. Spagnola Rutherford and S. Gordon. Using OPNET to simulate an IP network. In *Proceedings of the 36th conference on Winter simulation*, 2004.

[184] K. Salamatian and S. Fdida. A Framework for Interpreting Measurement over Internet. In *Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools)*, August 2003.

[185] S. Savage. Sting: A tool for measuring one way packet loss. In *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, April 2000.

[186] A. Shaikh and A. Greenberg. Operations and Management of IP Networks: What Researchers Should Know. Tutorial Session, ACM SIGCOMM '05. http://www.research.att.com/~ashaikh/talks/tutorial-sigcomm05.ppt, August 2005.

[187] A. Shriram and J. Kaur. Empirical evaluation of techniques for measuring available bandwidth. In *Proceedings of IEEE INFOCOM '07*, 2007.

[188] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and kc claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *Proceedings of PAM '05*, 2005.

[189] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '04*, 2004.

[190] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving Accuracy in End-to-end Packet Loss Measurement. In *Proceedings of ACM SIGCOMM*, Philadelphia, PA, August 2005.

[191] J. Sommers, P. Barford, N. Duffield, and A. Ron. A geometric approach to improving packet loss measurement. *To appear, IEEE/ACM Transactions on Networking*, 2008.

[192] J. Sommers, P. Barford, A. Ron, and W. Willinger. Phase Plot-based Analysis of Internet Packet Traffic Dynamics. Technical report, University of Wisconsin-Madison, 2004.

[193] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '04*, 2004.

[194] J. Sommers, V. Yegneswaran, and P. Barford. Recent advances in network intrusion detection system tuning. In *Proceedings of the 40th IEEE Conference on Information Sciences and Systems*, March 2006.

[195] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility . In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[196] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison Wesley Longman, Inc., third edition, 1999.

[197] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of ACM IMC '03*, Miami, Florida, October 2003.

[198] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf 1.7.0 – the TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf.

[199] S. Traweek. *Beamtimes and Lifetimes: The World of High Energy Physicists*. Harvard University Press, 1992.

[200] J.W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[201] A. Turner. tcpreplay. http://tcpreplay.sourceforge.net/.

[202] S. Uhlig. Simulating interdomain traffic at the flow level. Technical Report Infonet-TR-2001-11, University of Namur, Institut d' Informatique, 2001.

[203] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[204] D. Veitch, S. Babu, and A. Pásztor. Robust synchronization of software clocks across the Internet. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '03*, Taormina, Sicily, Italy, October 2004.

[205] C. Villamizar and C. Song. High Performance TCP in ANSNET. *ACM SIGCOMM Computer Communications Review*, 24(5), October 1994.

[206] K.V. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. In *Proceedings of ACM SIGCOMM '06*, 2006.

[207] G. Vu-Brugier, R. S. Stanojevic, D. J. Leith, and R. N. Shorten. A critique of recently proposed buffer-sizing strategies. *ACM SIGCOMM Computer Communications Review*, 37(1):43–48, 2007.

[208] M. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F.D. Smith. Tmix: a tool for generating realistic TCP application workloads in ns-2. *ACM SIGCOMM Computer Communications Review*, 36(3):65–76, 2006.

[209] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[210] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[211] D. Wischik. Fairness, QoS and Buffer Sizing. *ACM SIGCOMM Computer Communications Review*, 36(1), January 2006.

[212] D. Wischik and N. McKeown. Part I: Buffer Sizes for Core Routers. *ACM SIGCOMM Computer Communications Review*, 35(3), July 2005.

[213] R. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2), March-April 1982.

[214] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of temporal dependence in packet loss. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[215] L. Zhang and D.D. Clark. Oscillating behavior of network traffic: A case study simulation. *Journal of Internetworking: Research and Experience*, 1:101–112, 1990.

[216] L. Zhang, Z. Liu, and C. Xia. Clock Synchronization Algorithms for Network Measurements. In *Proceedings of IEEE Infocom*, New York, NY, June 2002.

[217] L. Zhang, S. Shenker, and D.D. Clark. Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic. In *Proceedings of ACM SIGCOMM '91*, pages 133–147, 1991.

[218] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of Internet path properties. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop '01*, San Francisco, November 2001.