CERIAS Tech Report 2008-23

A DEVICE INDEPENDENT ROUTER MODEL: FROM MEASUREMENTS TO SIMULATIONS

by Roman Chertov

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

A DEVICE INDEPENDENT ROUTER MODEL: FROM MEASUREMENTS TO SIMULATIONS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Roman Chertov

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2008

Purdue University

West Lafayette, Indiana

To my parents and my loving wife.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the tremendous support of my primary adviser Sonia Fahmy and my co–adviser Ness B. Shroff. Their constant encouragement and unwavering support resulted in our publications and ultimately this thesis. During my work, I had to rely on the staff of DETER, Emulab, and WAIL testbeds for my experiments. I am very grateful for their support and patience, in helping me resolve experimental problems.

The second half of this dissertation would not have been possible without the help of Michael Blodgett, Prof. Paul Barford, Ron Ostrenga, Terry Benzel, and Prof. Ray Hansen. With the help of these individuals, I was able to obtain access to four commercial routers necessary for the experiments in this thesis. I want to additionally thank Michael Blodgett and Prof. Ray Hansen for their help in configuring the routers.

Last but not least, I want to thank my family for their constant support and reassurance. I am indebted to my wife, Anastasiya, without her companionship and support, I would have taken much longer to finish this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Chertov, Roman Ph.D., Purdue University, May, 2008. A Device Independent Router Model: From Measurements to Simulations. Major Professors: Sonia Fahmy and Ness B. Shroff.

Simulation, emulation, and wide-area testbeds exhibit different tradeoffs with respect to fidelity, scalability, and manageability. Network security and network planning/dimensioning experiments introduce additional requirements compared to traditional networking and distributed system experiments. For example, high capacity attack or multimedia flows can push packet forwarding devices to the limit and expose unexpected behaviors. Many popular simulation and emulation tools use high-level models of forwarding behavior in switches and routers, and give little guidance on setting model parameters such as buffer sizes. Thus, a myriad of papers report results that are highly sensitive to the forwarding model or buffer size used.

In this work, we first motivate the need for better models by performing an extensive comparison between simulation and emulation environments for the same Denial of Service (DoS) attack experiment. Our results reveal that there are drastic differences between emulated and simulated results and between various emulation testbeds. We then argue that measurement-based models for routers and other forwarding devices are crucial. We devise such a model and validate it with measurements from three types of Cisco routers and one Juniper router, under varying traffic conditions. The structure of our model is device-independent, but requires device-specific parameters. The compactness of the parameter tables and simplicity of the model make it versatile for high-fidelity simulations that preserve simulation scalability. We construct a black box profiler to infer parameter tables within a few hours. Our results indicate that our model can approximate different types of routers.

Additionally, the results indicate that queue characteristics vary dramatically among the devices we measure, and that backplane contention must be modeled.

1 INTRODUCTION

Over the past several years, there has been a substantial increase in malicious traffic in the Internet. Denial of Service (DoS) attacks have become increasingly prevalent [1,2]. The DoS attacks in February 2000 brought down Yahoo! and eBay; a more massive attack in October 2002 brought down eight of the thirteen root Domain Name System (DNS) servers. A three-week backscatter analysis observed 12,000 attacks against more than 5,000 distinct targets [2]. More recently, several e-Businesses have been extorted under the threat of prolonged DoS attacks. Significant damage, such as network partitioning, can potentially be caused by attacks that target the Internet infrastructure, such as inter-domain routing protocols, key backbones, and Domain Name System (DNS) servers (which were attacked in 2002 and again in 2007).

Besides malicious traffic, there is a growing need to accommodate extremely high bandwidth user traffic, as a result of e-commerce and entertainment industries growth. Such traffic includes large media files, voice over IP, multiplayer games, streaming video, and emerging IPTV. High–speed connections are required to accommodate these industries. For example, an IPTV broadcast of cable TV quality requires 1 to 1.5 Mbps, while a high definition broadcast requires 6 to 8.5 Mbps, using a H.264/AVC codec [3]. In Asian countries where 10 Mbps and higher broadband access is wide spread, IPTV is already in use. In Hong Kong alone, there are over 500,000 subscribers with neighboring regions adopting the technology at a rapid pace [4]. The computer gaming market is also rapidly expanding; in 2001 alone, it claimed over $9 billion in the US market [5]. The ever improving graphics result in higher quality interactive worlds and expanded features; in multiplayer games this directly translates to higher bandwidth usage to synchronize a multidute of game objects. The combined aggregate of these bandwidth hungry services for many thousands of users

can strain even the best networks. Hence, the success of these industries heavily relies on sufficient network infrastructure to provide ample bandwidth.

Clearly, the substantial increase in malicious and multimedia traffic demands innovative software and hardware solutions. Research of these new technologies faces difficult challenges because of the scale and the sheer amount of traffic involved. For instance, 10,000 viewers of high definition IPTV would require more than 60 Gbps of network bandwidth. Before being released into production, new technologies must be tested on a large scale to make sure that they work as intended, without undesirable side effects that might impede other services. Network providers must also ensure that their networks can manage present and near future loads. To do so, they must perform network planning/dimensioning experiments to decide if network upgrades are required. An incorrect decision can result in poor service and ultimately loss of customers and revenue.

1.1 Network Research Tools and Analytical Models

Currently, testing and research is carried out on simulators, emulators, and testbeds. The considerably aggressive nature of the high–capacity flows requires high–fidelity experimentation as these flows can stress these limits of the software/hardware and knowledge of the limits is critical before deployment on the Internet can proceed.

1.1.1 Simulation

Network simulators must balance a fidelity versus scalability tradeoff [6, 7]. At one end of the spectrum, simulators can choose to sacrifice fidelity, especially at the lower layers of the protocol stack, for scalability. For this reason, Internet forwarding devices, such as switches and routers, are only modeled at a high–level in popular packet–level simulators such as ns-2 [8]. The wide ranges of intra–device latencies and maximum packet forwarding rates in commercial forwarding devices are not incorporated. Networking researchers often find it difficult to set node parameters such as

router buffer size in their experiments with these simulators. Hence, many research papers, e.g., in the congestion control literature, may report results that are highly sensitive to the default forwarding model or the buffer size they selected, which may or may not be representative of today's switches and routers.

The high–level models used to represent routers in simulators such as ns-2 are typically designed to mimic forwarding in core routers, and hence use a default simple output queuing model, abstracting away any processing delay or backplane contention. Compared to these core routers, low–to–mid level routers have switching fabrics with lower performance. Yet, because of cost considerations, they constitute the majority of the forwarding devices in Internet edges and enterprise networks, which is where most packet losses in the Internet of 2008 occur. Accurately modeling a range of devices is especially important in experiments with high resource utilization, as resource consumption models commonly used in simulators and emulators may not be representative of today's commercial routers. Discrepancies between the simulated and deployment behaviors can be large in experiments with denial of service attacks or high bandwidth traffic, and in network dimensioning experiments. For example, results in the motivational chapter 2 with a low-rate TCP targeted denial of service attack demonstrate that seemingly identical tests on various testbeds and on the ns-2 simulator produce dramatically different results. The discrepancies in the results arise because routers and other forwarding devices have complex architectures with multiple queues and multiple bottlenecks (e.g., buses, CPUs) [9] that change in complex ways according to the characteristics of the workload they are subjected to.

Near the other end of the spectrum from highly-scalable simulators lie simulators such as OPNET [10] and OMNeT++ [11]. In OPNET, detailed models of routers, switches, servers, protocols, links, and mainframes are given, solely based on vendor specifications [12]. Using complex models significantly increases computational cost, hindering scalability. Further, the model base needs to be constantly updated. Validation attempts reveal that even these accurate models are sensitive to parameters

such as buffer size and forwarding rate that are difficult to tune to mimic router behavior [12].

1.1.2 Emulation

To conduct experiments with real hardware and software, testbeds are used instead of simulators. Emulation testbeds can range from small-scale lab networks to massive multi–node facilities such as Emulab at http://www.emulab.net/, DETER at http://www.isi.deterlab.net/, Wisconsin Advanced Internet Laboratory (WAIL) at http://www.schooner.wail.wisc.edu, and Open Network Laboratory (ONL) at http://onl.arl.wustl.edu/. These testbeds are aimed at achieving higher experimental fidelity as a variety of physical hardware running real software is available. By leveraging VLANs it is possible to dynamically create a large set of experimental topologies; however, a few things have to be emulated still. Link propagation delays need to be artificially induced as most of the testbed nodes are only a few milli-seconds away from each other. PlanetLab (http://www.planet-lab.org/) solves this problem by having a large number of sites (600+) scattered all over the globe such that propagation delays are real; however, there is a large number of limitations of what research can be carried out, as this testbed is part of the Internet. Few testbeds (e.g., WAIL, DETER, and ONL) have real routers; although the number/type of routers and ports is limited, thus imposing limitations on the experiment topology scale. Because of these limitations, routers have to be emulated by PCs, hence sacrificing fidelity and potentially inducing artifacts.

1.2 A High–Fidelity Router Model

A key component of any large or small network is a forwarding device, which has complex hardware and software. As it has been discussed above, simulators and emulation testbeds do not always faithfully represent router characteristics. As packet loss and delay occurs mainly at the routers during high loads, it is crucial to have

a high–fidelity model when researching highly–loaded networks. Most real routers are pipelined [13, 14] network devices, meaning the router has to decode a packet, make a routing decision, switch the packet to the correct path and then output it on the correct port. In the majority of cases, routers are modeled as output ports only, disregarding any packet interactions that might occur on the path to the output queue. However, the complexity of the packet path in the router coupled with high network load can make the interactions non-negligible, prompting a need for a high–fidelity router model. Developing such a model requires to overcome several challenges:

1. The sheer multitude of routers used commercially poses an arduous task of keeping track of all of the available hardware such as: switching fabrics, memories, CPUs, NICs, buses.

2. The software that runs on the routers similar to the Cisco IOS can have various policies and caching schemes that change from to version to version, making it exceedingly difficult to incorporate into the model.

3. The model would require extensive validation for each new device, even if all of the required information is captured in the model. Validation might be required even for an already modeled device, if the software is upgraded or changed.

4. The scalability of a simulator can be inhibited by a highly detailed model, even if it is highly accurate, because of significant computational overhead. Decrease in simulator scalability, results in a reduction of network topology size that a simulator can handle.

1.3 A Device Independent Router Model

Above, we have demonstrated a need for a router model that is accurate yet preserves scalability of network simulators. In this work, we argue that it is important to devise a forwarding model that lies between these two extremes, and is well–

founded on extensive device measurements. Such a model must meet the following requirements:

1. It is accurate, but is allowed to miss special cases for the sake of scalability.
2. It is not computationally expensive.
3. Its parameter inference process is the same regardless of the device being modeled.
4. Its parameters are inferred without assuming any knowledge of device internals.
5. The model reflects behavior under changing workloads.

To our knowledge, the only recent study that has modeled a router based on empirical observations was [15]. The authors created a Virtual Output Queuing (VOQ)-based model that added delays to the packets prior to placing them into an infinitely large FIFO output queue. The delays were based on empirical observations of a production Tier-1 access router. However, the model ignores any backplane interactions and uses infinite sized queues. We will show that this model is reasonably accurate for lightly loaded core routers with a sophisticated switching fabric, but it does not generalize to lower-end devices or heavy loads.

1.4 Contribution

The contribution of this work is fourfold:

1. We motivate this work by addressing the question of *when simulation,* and *emulation* are inadequate for studying DoS attacks or high capacity flows. A key component of the answer to this question is the *sensitivity* of simulation and emulation results to parameter settings and testbed capabilities. Experimental fidelity is crucial once the limits of the network are reached.
2. We propose a model that differs from the VOQ-based model [15] in several key aspects – most importantly the queue size and number of servers. Our model

generalizes to different router and switch types, but the model *parameters* make it unique for a specific type. Additionally, we design a parameter inference procedure based on simple measurements.

3. We create a custom low-cost measurement and profiling system to eliminate the need for expensive measurement specialty cards. The tool can also function as a high speed traffic generator, while providing micro-second precision packet delay measurements. We have successfully used the tool for profiling and traffic generation at 100 Mbps and 1 Gbps speeds.

4. We leverage our measurements to model two low-to-mid end Cisco routers: 3660 and 7206VXR, a Cisco high end router, 12410, and a Juniper M7i router. Our preliminary experiments with UDP, FTP and HTTP traffic reveal that our model is capable of capturing different router behaviors.

The results indicate that queue and performance characteristics vary dramatically among the devices we measure, and that backplane contention can be a significant source of delay. When compared to the observed results, our model performs much closer than ns-2 in output queue and backplane contention scenarios. We believe this to be a significant step toward creating high-fidelity yet scalable simulations.

1.5 Thesis Organization

The remainder of this work is organized as follows. Chapter 2 provides a detailed motivation for this work, by performing a detailed comparison between simulation and emulation environments for the same DoS experiment. Readers who are not interested in the detailed comparison can skip this chapter. Chapter 3 surveys the related work on simulation/emulation tools and router modeling. Chapter 4 provides an overview of commercial routers, gives a detailed overview of our device independent model, and describes active probing methods necessary to derive the model parameters. Chapter 5 describes in detail the architecture of our profiling tool, called

the Black Box Profiler (BBP), which is used in our router experiments and model parameter inference. Chapter 6 discusses our results with four different commercial routers which represent a wide cross-section of the router market. Finally, chapter 7 concludes this work and discusses possible future directions.

2 COMPARATIVE STUDY OF NETWORK SIMULATION AND EMULATION FIDELITY

In this chapter, we will describe in detail the motivation behind the need for more accurate simulator router models. The motivation is based on an extensive study of a low–rate TCP Denial of Service (DoS) attack in simulation and emulation experiments. The results of the study indicate that significant differences can occur between simulation and emulation experiments with seemingly identical experimental setups.

2.1 Introduction

Denial of Service attacks have become increasingly prevalent [1, 2], prompting a myriad of network security research papers. The DoS attacks in February 2000 brought down Yahoo! and eBay; a more massive attack in October 2002 brought down eight of the thirteen root Domain Name System (DNS) servers. A three-week backscatter analysis observed 12,000 attacks against more than 5,000 distinct targets [2]. More recently, e-Businesses have been extorted under the threat of prolonged DoS attacks. Significant damage, such as network partitioning, can potentially be caused by attacks that target the Internet infrastructure, such as inter-domain routing protocols, key backbones, and Domain Name System (DNS) servers (which were attacked in 2007 and in 2002). In this work, we address the question of *when simulation* and *emulation* are inadequate for studying DoS attacks. A key component of the answer to this question is the *sensitivity* of simulation and emulation results to parameter settings and testbed capabilities. As a case study, we take an in-depth look at *low-rate TCP-targeted attacks* [16, 17]. In particular, we consider a scenario where an attacker transmits short pulses at an arbitrary frequency. This attack exploits the TCP Additive Increase Multiplicative Decrease (AIMD) mechanism to cause per-

formance degradation to TCP flows traversing the same routers. We use a simple analytical model for predicting the average size of the congestion window of a TCP flow under attack, as a function of the attack frequency. This model gives a lower bound on the window size (for the case with no timeouts) because each pulse causes loss. We compare results from a simple analytical model of TCP goodput under this attack scenario to results from the popular ns-2 simulator, and the DETER, Emulab, and WAIL testbeds under the same scenario.

TCP-targeted attacks are an interesting case study because they are a major cause for concern (they are easy to launch, stealthy, and may be extremely damaging). Further, in a deterministic simulation environment, traffic oscillations induced by these attacks can lead to phase synchronization effects with slight parameter changes [18]. Such effects might appear interesting, but they are not representative of real systems. Additionally, experiments with this attack on testbeds may yield large variations in the results depending on the parameter settings and testbed capabilities.

A number of recent studies [16, 19–21] have experimented with TCP-targeted attacks as well as proposed defenses against them. Among these, only [16,19] conducted testbed experiments, but these experiments were conducted with (*tc*, *iproute2*), NISTnet, or DummyNet [22] for link shaping and queue management, without investigating system parameters, or relating the results to simulations. In contrast, our work investigates emulation environments via a more careful sensitivity analysis, and highlights the danger of default system parameter settings, especially in regards to router nodes.

The primary contribution of this work is a careful comparison of results from simulation and emulation experiments with different DoS attack *and* system parameters. Additionally, we compare PC routers to commercial Cisco routers. We explore key problems that arise because of differences in testbed capabilities and default parameter settings. We design constructs for the emulation testbeds to achieve a level of control comparable to simulation tools. Our results reveal that significant differences can exist between simulation and emulation experiments even if the experimental setups are almost identical.

The remainder of this chapter is organized as follows. Section 2.2 provides background on the attack we use as our case study. Section 2.3 gives a simple analytical model of the performance degradation caused by TCP-targeted attacks. Section 2.4 describes the emulation environment we use, and the tools that we have developed for the DETER emulation testbed. Section 2.5 explains our experimental setup. Section 2.6 discuss our results from ns-2, Emulab, and DETER experiments. Section 2.7 describes our experiences with the Click router. Section 2.8 discusses our experiments with commercial Cisco routers on the WAIL testbed. Finally, Section 2.9 summarizes our findings.

2.2 TCP-Targeted Attacks

TCP-targeted DoS attacks are an ideal case study for understanding the pros and cons of different evaluation methods and testbeds. Most well-publicized DoS attacks have utilized a large number of compromised nodes to create constant high-rate flows towards the victims. Such "flooding attacks" are effective, but have major shortcomings from the attacker's perspective. First, the attacks are easy to detect as a result of the high volume of uniform traffic, e.g., UDP or ICMP. Several defense mechanisms against these (and more sophisticated) DoS attacks have been proposed in the literature [23–27]. Second, the attacks can self-congest at some bottleneck and not reach the intended destination. Finally, users of the compromised machines typically notice a performance degradation, prompting these machines to be examined by system administrators, who can then eliminate the vulnerabilities that caused the machines to be compromised in the first place.

An attack that is less susceptible to these limitations is the low-rate TCP-targeted attack, introduced in [17][1]. This attack has generated significant interest because of its potential to do great harm, go undetected, and the ease by which it can be gener-

[1]We do not consider other types of application-specific, protocol-specific, or implementation-specific DoS attacks, such as SYN attacks, BGP attacks, LAND, or TEARDROP, in this work. We only focus on attacks against TCP congestion control.

ated. The basic idea of low-rate TCP-targeted attacks is that an attacker transmits short pulses, i.e., square waves, with periodicity close to the Retransmission-Timeout (RTO) interval [28] of ongoing TCP connections. These short pulses induce sufficient packet loss to force the TCP flows under attack to time out, and to continually incur loss as they attempt to begin TCP slow start. Therefore, the goodput of these TCP flows virtually goes to zero. Such an attack can be used to strategically target key routers or servers in the network, thus causing wide-spread degradation of TCP performance.

A key feature of this attack is that it is *stealthy*, i.e., it does not continuously generate significant traffic, and thus cannot be easily distinguished from other legitimate flows (e.g., video and other bursty traffic). Moreover, an attacker does not have to be highly sophisticated to generate these attacks. It is straightforward to generate UDP pulses, or use raw sockets to bypass the TCP congestion avoidance mechanism altogether.

A study in [16] has considered a more general class of low-rate TCP attacks, referred to as the Reduction of Quality (RoQ) class of attacks. In a RoQ (pronounced "rock") attack, the attacker sends pulses at arbitrary frequencies, rather than trying to precisely match the RTO periodicity. The attack exploits the TCP Additive Increase Multiplicative Decrease (AIMD) mechanism to cause TCP goodput degradation, rather than focusing on timeouts. The premise is that during the *congestion avoidance phase*, when packet losses occur because of attack pulses, TCP halves its congestion window, but when a successful transmission occurs, it only linearly increases its window size. The motivation behind RoQ attacks is that they *need not* be precisely tuned to the RTO frequency, as RTO may be difficult to ascertain, and can be changed for different TCP sessions. Moreover, these attacks may be even more difficult to detect, as they do not operate at a known frequency. While RoQ attacks may not cause TCP goodput to virtually go to zero, as in the case of [17], they can still significantly degrade service quality. Table 2.1 demonstrates the bandwidth[2] of

[2]The bandwidth calculation includes the Ethernet, IP, and UDP headers.

a TCP-targeted attack averaged over a one second interval. The table demonstrates that the average bandwidth that the attack flow consumes is a small portion of a 100 Mbps Ethernet link. Therefore, we use these attacks as a case study in our work.

Table 2.1
Average bandwidth of a TCP-targeted attack

Packet rate	Packet size	Sleep (ms)	Pulse (ms)	Bandwidth (Mbps)
16 Kpackets/s	64	500	160	1.99
16 Kpackets/s	64	2000	160	0.61
85 Kpackets/s	64	500	160	10.55
85 Kpackets/s	64	2000	160	3.22
140 Kpackets/s	64	500	160	17.38
140 Kpackets/s	64	2000	160	5.31

2.3 Simple Analytical Model

In this section, we give a simple analytical model that we use in our comparisons to gauge the effectiveness of an attack. The model can be considered as a special case of the model given in [19]. The model characterizes TCP performance degradation as a function of the TCP-targeted attack frequency. In prior work, e.g., [29], models of TCP throughput as a function of the round-trip time and loss event rate were developed. These models, however, did not consider the presence of periodic attacks. In contrast, we compute the average TCP window size as a function of the TCP-targeted attack parameters. The analysis assumes that TCP Reno [30] in the congestion avoidance phase is being employed for a single flow under attack.[3] As discussed in Section 2.2, the objective of this attack is to exploit the TCP AIMD

[3]We have generalized our model to multiple flows under attack, but, for simplicity of illustration, present only the single-flow case here.

mechanism and not to cause RTOs. As Reno can typically recover from a single packet loss without an RTO, it is assumed that *every* attack pulse will induce a packet loss, resulting in Cwnd being cut in half but not inducing an RTO. A loss of a single data packet will cause a reduction of the congestion window by half in TCP Reno, after which additive increase will be employed. For simplicity of the analysis, the short fast recovery phase is ignored. The resulting TCP congestion window saw-tooth pattern is depicted in Figure 2.1 for a fixed attack frequency. Observe that the model also gives a close approximation of the behavior of TCP New Reno [31] and TCP SACK [32] *even* with a few packet losses with every pulse, because these TCP flavors can typically recover from multiple packet losses without RTOs.

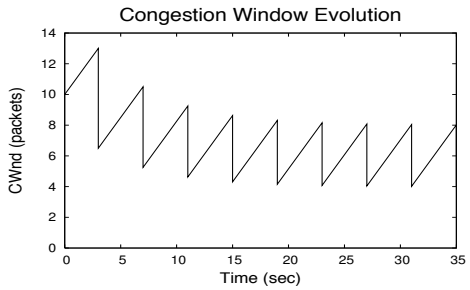


Figure 2.1. Saw-tooth pattern of congestion window evolution because of periodic loss every 4 seconds.

Let W_i be the size of the congestion window (Cwnd) right before the reduction caused by pulse i, $i \geq 0$. Let rtt be the flow round trip time (RTT). Let α be the growth in Cwnd size during the attack *sleep time* t between pulses i and $i+1$. Then, $W_{i+1} = \frac{W_i}{2} + \alpha$, where α (the growth of the window during t) is equal to $\frac{t}{2rtt}$ (assuming the default TCP behavior where every *other* packet is acked [33]; $\frac{t}{rtt}$ if every packet is acked).

Let W_I be the initial Cwnd size before the attack starts. We need to compute W_{max}, the maximum congestion window size after the attack reaches steady state,

as well as the average window size W_{avg}. From the above equation, one can easily compute W_1, W_2, $\cdots$. For example, W_3 can be expressed as:

$$W_3 = \frac{\frac{\frac{W_I}{2}+\alpha}{2} + \alpha}{2} + \alpha.$$

Therefore, W_{max} (assuming the limit on the receiver window size is not reached) can be expressed as:

$$W_{max} = \lim_{i \to \infty} (2^{-i} W_I + \alpha (\sum_{j=0}^{i-1} 2^{-j})) = 2\alpha.$$

The steady state minimum window size is simply $W_{max}/2 = \alpha$. Since $\alpha = \frac{t}{2rtt}$, therefore, $W_{avg} = \frac{\alpha+2\alpha}{2} = \frac{3t}{4rtt}$.

From the computed W_{avg}, we can approximate the average throughput as: $Rate = W_{avg} \times MSS \times 1/rtt$ [33], where MSS is the Maximum Segment Size and rtt is the connection RTT.

2.4 Emulation Environment and Tools

To experiment with this DoS attack in a high–fidelity setting, we leverage the following testbeds: DETER (www.isi.deterlab.net), Emulab (www.emulab.net), and WAIL (www.schooner.wail.wisc.edu). Emulab is a time- and space-shared network emulator located at the University of Utah [34]. The system comprises hundreds of linked PCs that can be connected in any specified topology, and a suite of software tools that manage them. When an experiment gets allocated, the management system allocates PCs and then connects them via VLANs, creating a desired topology. A key advantage of such emulation testbeds is that results are reproducible, allowing for detailed comparisons and careful sensitivity analysis.

The Cyber Defense Technology Experimental Research Network (DETER) is an emulation testbed – based on Emulab – that allows researchers to evaluate Internet security technologies [35]. DETER can be accessed remotely, but is quarantined from the Internet. This allows for experimentation with live malware without infecting real Internet hosts. The Evaluation Methods for Internet Security Technology (EMIST)

project, in which we have participated, is a companion project that designed testing methodologies and benchmarks for the DETER testbed. The Wisconsin Advanced Internet Laboratory (WAIL) is another testbed based on Emulab that has a variety of commercial Cisco routers available to the researchers.

The primary advantage of using a network *emulator* – as opposed to a *simulator* – for security experiments is that an emulation environment affords much higher fidelity, provided that it is correctly configured to avoid artifacts. Further, real security appliances (e.g., off-the-shelf hardware) can be tested on it. This can expose unforeseen implementation vulnerabilities, protocol interactions, and resource constraints. This is because an emulation testbed uses real devices with limited resources, and real applications and operating systems running on them, to faithfully represent every host in an experiment. Flaws and vulnerabilities are not abstracted by a simplified or idealized simulation model.

In addition to emulating link delay and bandwidth, network emulation testbeds may include one component that differs from real Internet components: router nodes. In the current versions of Emulab and DETER, routers are represented by regular PCs that act as forwarding gateways. We refer to these as *PC routers*. Our experiences with WAIL have demonstrated that a regular commodity PC running Linux may outperform low to mid-range Cisco routers. Specialty PC routers such as the ones from ImageStream are usually created from high-end PCs that have SMP, multiple buses, fast memory, and industrial network cards [36, 37]. The performance of specially configured PC routers can challenge that of Cisco 7000 series as well as Juniper M-5/M-10 routers according to [38,39]. Therefore, it is important that we understand the behavior of PC routers under attack. We also use Cisco 3600/7000-series routers on WAIL to compare the results.

Developing efficient PC routers has been the subject of significant research, e.g., [40–42]. In these studies, polling and/or Direct Memory Access (DMA) are used as an alternative to packet receive interrupts to eliminate receive livelock at high packet rates. This is because interrupts can consume much of the CPU and bus capacity

of mid-range machines (i.e., Pentium III and below) at 100 Mbps+ speeds. In this work, we experiment with interrupt– and polling–based packet processing to increase the breadth of the study.

Event control system In network simulators such as ns-2 [8] and iSSF/iSSFNet [43], it is easy to create a topology, assign tasks to the nodes, and monitor every single packet. A basic testbed – without any software support that mirrors some of these capabilities – is limited in its usefulness, as it requires experimenters to be experts in system-level programming. Achieving the same level of control provided by a simulator on physical testbed machines is a significant undertaking. Basic topology creation capabilities are provided by emulation testbeds, but an experimenter only acquires bare machines that form the desired topology, without any tools running on them.

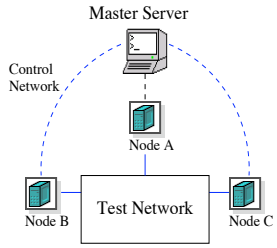


Figure 2.2. Master/Zombie control network.

A natural approach to describe the tasks that must be performed on the testbed nodes is to use event scripts, akin to events in an event-driven simulator. The Emulab software implements a few event types such as link failures; however, most of the interaction with the nodes must be performed via a secure shell (SSH) session. We have designed a flexible mechanism to control all test machines from a central location. We have developed a multi-threaded utility, which we refer to as a *Scriptable Event System,* to parse a script of timed events and execute it on the test machines (communicating with them on the *control network*). Our tool is capable of receiving *callbacks* for event synchronization. Figure 2.2 depicts the architecture of our system.

Measurement tools Instrumentation and measurement on a testbed pose a significant challenge. The capability to log and correlate different types of activities and events in the test network is essential. Not only are packet traces important, but also system statistics must be measured for DoS attacks. We log statistics such as CPU utilization, packets per second, and memory utilization to the local disk for later manipulation. Scripts for measuring, merging, and plotting system data are also available for download.

2.5 Experimental Setup

The topology used for both simulation and testbed experiments is depicted in Figure 2.3.[4] This is a simple dumb-bell topology with four end-systems and two routers. Unlike traditional dumb-bell topologies, the backbone link has no delay. The reason for this is that the current setup on the WAIL testbed makes it impossible to introduce any propagation delay on a router-to-router link. To make accurate testbed comparisons, we have moved all delays to the access links. The links for the attack nodes have no delays because the attack traffic is delay-agnostic. The attacker and the attack sink are varied from one side of the topology to the other. *The same* basic ns-2 script is used for both simulations and testbed experiments. All testbed nodes run the zombie process that forms the basis of our *Scriptable Event System*.

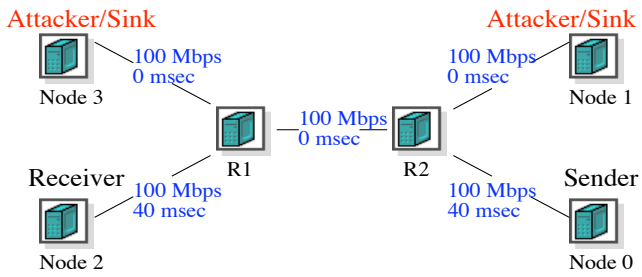


Figure 2.3. Simple dumb-bell topology with 160 ms round-trip-time and 100 Mbps links.

[4]This simple topology is not representative of the Internet, but we have selected it to be able to analyze the results in depth.

2.5.1 Testbed Setup

Machine specifics On the DETER testbed, we use dual Pentium III (P3) 733 MHz with 1024 MB of RAM and dual P4 Xeon 2.8 GHz with 2048 MB of RAM. Both machine types have Intel Pro/1000 cards. P3s are used for *Node0*, *Node2*, *R1*, and *R2*, while P4s are used for *Node1* and *Node3*. On Emulab, *Node0* and *Node2* are P3 600 MHz with 256 MB of RAM; *Node1* and *Node3* are P4 2.0 GHz with 512 MB of RAM; and *R1* and *R2* are P3 850 MHz with 256 MB of RAM. The P3s have Intel EtherExpress Pro 100 cards while the P4s have Intel Pros. On WAIL, all hosts are P4 2.0 GHz with 1024 MB of RAM with Intel Pro cards, while *R1* and *R2* are varied between Cisco 3640s and a 7206VXR-7500 pair. The choice of the specific machine types on the testbeds was driven by the need to minimize cross-switch hops: as the emulation testbeds contain a considerable number of machines, multiple switches are used to connect them together, potentially leading to loss at cross-switch hops.

In our first set of experiments (Section 2.6), all nodes ran Linux 2.4.26 with IRQ-driven packet processing, thus being susceptible to receive livelock [41]. The default NIC driver is `e1000-5.2.30.1-k1` on DETER, and `eepro100.c:v1.09j-t9/29/99` on Emulab. We used TCP SACK [32] with delayed acknowledgments in ns-2, as well as on the testbed machines. The ns-2 (drop tail) buffer sizes for routers were set to 50 packets (except in experiments when we vary the queue size). The buffer size of 50 was chosen because, according to DummyNet documentation [22], it is a typical queue size for Ethernet devices. Also the buffer size of 50 is the default ns-2 queue size. We found that by default the receive and transmit device driver buffers on DETER can hold 256 descriptors (64 on Emulab), according to *ethtool*.

Link delays Shaping methods such as DummyNet [22] or *tc* have been found to induce artifacts, as a result of bursty behavior and not being always true to the desired properties [44, 45]. These issues can be avoided, by emulating link delays on the DETER/Emulab/WAIL testbeds with the Click router. Special care was taken to manually select and configure the delay nodes to be at least as fast as the rest of

the test nodes, so that no packet loss occurs at the delay nodes. We accomplished this task by selecting Dual Pentium 4 Xeon 2.8 GHz machines with PCI-X, running SMP and *polling-enabled* Linux 2.4.26 to act as delay nodes on DETER, and 2.0 GHz uniprocessor nodes on Emulab/WAIL. To avoid bursty behavior, we configured the kernel clock to run at a higher frequency of 1000 Hz. Unlike the default Emulab-type delay nodes, each of our delay nodes are assigned to shape a single link instead of two.

2.5.2 Attack Generation

On the testbeds, the attack packet sizes (as well as all header fields and transmission rates) can be easily configured in our event script. As most queues are composed of *Maximum Transmission Unit (MTU)-sized slots*, we use *small* UDP packets for most of our DoS attacks (unless we are experimenting with the attack packet size). Attacks with smaller packets (and hence higher packet rates) may be more damaging because each packet requires a certain amount of header processing at routers and packet processing at end systems. Although our attack generator is easily able to use IP source and destination address spoofing, we do not employ spoofing in our experiments, to avoid the additional Address Resolution Protocol (ARP) overhead, and avoid backscatter [2] overhead on PC routers.

In our ns-2 experiments, the attack is created by a CBR agent that pulses at regular intervals. To achieve the same behavior on the testbed nodes running Linux, we use raw sockets for packet transmission. System timers are used to measure the duration of the attack pulse, and the sleep time when no packets are sent by the attacker. Using real clock timers is crucial when sub-second granularity is required. We use the real clock to generate desired packet rates and space packets equally in time as much as possible. However, the attack pulse on Linux is less precise than its ns-2 counterpart, as CPU scheduling, etc., affect the pulse precision. This produces small variations in the experimental data during identical runs on the testbeds. In

the case of ns-2, it has been observed that deterministic simulations can lead to synchronization artifacts [18]. To break this synchronization, we have added random jitter to the attack pulse and sleep periods that can extend them by $U(0, 10)$ ms, approximating the coarseness of a system clock. Thus, we repeat each simulation or testbed experiment ten times and report mean values as well as 95-percent confidence intervals.

Our attack generator, *flood*, is capable of generating packets at variable rates. Additionally, the tool can rely on system timers to alternate between on/off periods. To ensure that the desired rate is achieved, the packets are evenly spaced in time, and a high-end machine is required to generate rates of over 140 Kpackets/s.[5] The flooding agent used in the experiments was compared with Click's kernel element *udpgen* to ensure that our generator is on par with that of Click. Attack packets take 64 bytes at the data link layer, as 64 bytes is the smallest possible Ethernet frame. Since ns-2 does not explicitly increase packet size as the packet traverses the protocol stack (i.e., packet size does not increase as headers are added), we have explicitly fixed the size of ns-2 attack packets to 64 bytes.

2.5.3 Traffic Generation and Measurement

To gauge the impact of the attack on the three testbeds, we use *iperf* [46] to create a single TCP flow for 250 seconds. We chose to have a single "good" flow as this creates the worst case for the attacker, as the attacker must send enough traffic to fill the queues and induce losses. Such a long-lived TCP flow is not unrealistic, because several Internet sites offer large downloads, e.g., DVD-quality movies. In addition, this scenario simplifies the comparison between simulation and testbed results. The *iperf* tool reports statistics such as total time and transfer rate. In our case, the transfer rate is a good indicator of the attack potency. We have configured the experimental nodes as suggested in [47], and used an 11 MB receive window in *iperf* to

[5]The limit of 140 Kpackets/s is because of the limitation of the hardware on the network card itself.

allow full link utilization. TCP traces collected by *tcpdump* are processed by *tcptrace* to derive the following statistics: number of sent/received data packets, number of sent/received ACKs, number of retransmits, and congestion window estimates. As we are interested in the effects of the attack during the TCP *congestion avoidance phase*, we have developed a tool *cwnd_track* that records the window for a specific connection from */proc/net/tcp* several times per second (once per RTT).

On the testbeds, we first start the measurement tools as well as *tcpdump*. Then, we trigger the attack agent and, later, the file transfer. The sending node, *Node0*, is instructed to send a TCP stream to *Node2* via *iperf* for 250 seconds. Upon successful completion of the task, the attacker ceases the attack, and the measurement and *tcpdump* files are transferred to the project account for analysis. The ns-2 simulations and the testbed experiments use *the same* basic *tcl* topology script, and we log the attributes of the ns-2 TCP agent directly. The simulation time is 250 seconds and Cwnd values are collected 40 seconds after the simulation start to discount the effects of slow start. We report Cwnd as the number of outstanding segments (instead of bytes) to correspond to the analytical model. To conduct accurate transfer rate comparisons between ns-2 and the testbeds, we set the ns-2 TCP packet size to 1505 bytes. The average payload size reported by *tcptrace* when it analyzes the tcpdump files is 1447. This means that the average packet size is 1505 bytes if Ethernet (14+4 bytes), IP (20 bytes), and TCP (20 bytes) overhead is included.

We encountered two problems in our initial experiments. First, we noticed a large number of hardware duplicates in the *tcpdump* files when we captured traces at the sending node. We suspect the problem is because of *libpcap* as the duplicates were never sent on the wire. To resolve this problem, we use the delay nodes on the testbeds to collect TCP traces via *tcpdump* in a similar fashion as was described in [48]. However, we have combined the tasks of delaying and monitoring into a single node as the number of available testbed nodes is usually limited. As the delay/capture nodes only deal with TCP traffic, they can easily capture 100 Mbps TCP traffic as the packet rates are less than 9000 Kpackets/s [48].

Second, we experienced cases where the window results from *tcptrace* and *cwnd_track* appeared erroneous, and did not correspond to the observed goodput. To make comparisons among the analysis, simulations, and testbed experiments possible, we use goodput instead, applying a simple transformation to the analytical model that gives an approximate analytical goodput, as described in Section 2.3.

2.5.4 Experimental Design

We investigate the impact of varying the following parameters:

(i) The attack sleep time from 500 ms to 4000 ms in 500 ms increments, and from 2000 ms to 20000 ms in 2000 ms increments;

(ii) The attack pulse length to be 20, 40, 60, 80, 120, 160, or 200 ms;

(iii) The attack packet size to be 64, 1184, or 1400 bytes;

(iv) The attack packet rate to be 8.3, 16, 85, or 140 Kpackets/s;

(v) The round-trip time of the attack flow and the long-lived TCP flow to be 60, 80, 120, 160, 200 or 420 ms;

(vi) Router buffer sizes on ns-2 routers to be 25, 50, 100, 150 or 256 packets;

(vii) Router buffer sizes on Click to be 50 or 250 packets;

(viii) The transmission ring buffer size of the *e1000* driver to be 80 or 256 packets; and

(ix) The placement of the attacker to be either *Node1* or *Node3*.

We compute the following metrics:

(i) Average goodput in Kbps (Kbits per second) or Mbps, as reported by *iperf* or computed from ns-2 results;

(ii) Average congestion window size in packets, computed for testbed experiments by taking an average of the congestion window values tracked by our *cwnd_track* utility;

(iii) CPU percentage utilization from */proc/uptime*; and

(iv) Packets per second received and sent on the *test network* interfaces, computed for testbed experiments by using Click's *udpcount* utility.

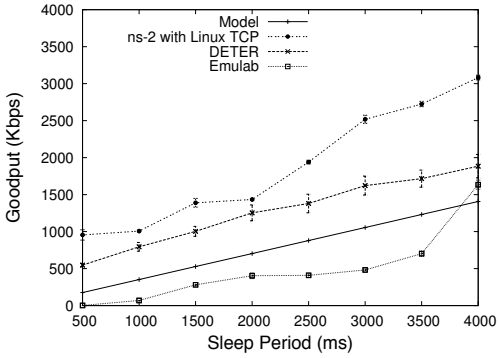
System-level measurements, e.g., CPU utilization, cannot be collected in simulations, because ns-2 does not model host CPUs.

2.6 Emulab and DETER Testbed Results

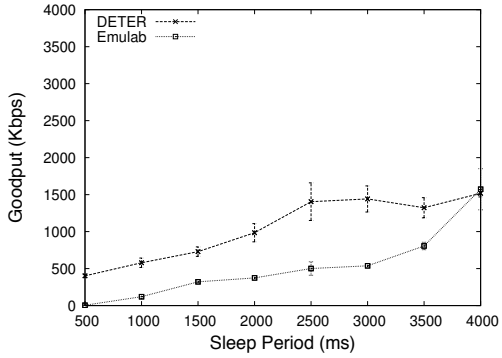
In this section, we undertake a task of comparing results from the analytical model (Section 2.3) and ns-2 simulator to Emulab and DETER testbed results.

TCP without attack We first measure the performance of the TCP flow without any attacks on Emulab and DETER. On Emulab, the average of ten runs was 83.86 Mbps with a standard deviation of 5.871 Mbps. On DETER, we have observed an average transfer rate of 88.28 Mbps with a std. deviation of 0.082 Mbps. In both cases, examining the *tcpdump* files via *tcptrace* revealed that the sender congestion window was close to the bandwidth delay product of the path, and no loss occurred at the delay nodes.

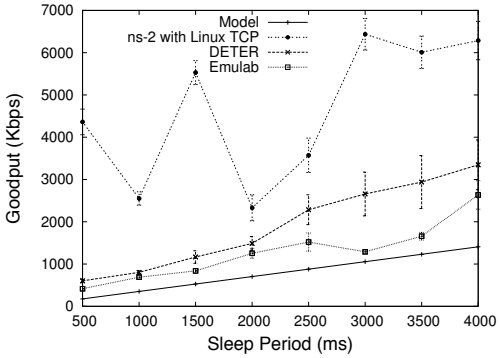
Emulab/DETER versus simulation We initially use the default system settings for the Emulab and DETER nodes, meaning that we do not alter the default Linux configurations. In this set of experiments, the attack packets are 64 bytes. The RTT is set to 160 ms, as depicted in Figure 2.3. We conduct two series of experiments on Emulab and DETER where we vary the attack packet rate from 85,000 to 140,000 packets/s, which is the maximum transmission rate of our attack nodes. Figure 2.4 gives the average goodput for an attacker at node *Node1* (forward direction) or node



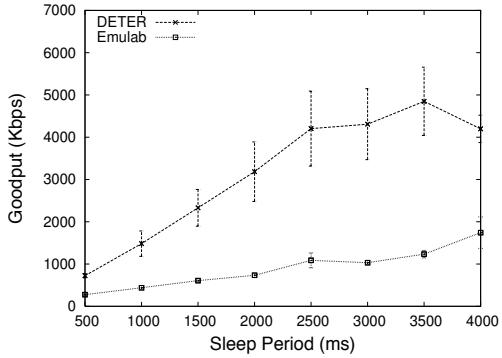
(a) Attack in forward direction at 140 Kpackets/s.



(b) Attack in reverse direction at 140 Kpackets/s.



(c) Attack in forward direction at 85 Kpackets/s.



(d) Attack in reverse direction at 85 Kpackets/s.

Figure 2.4. Comparison of the average goodput from analysis, simulations, DETER and Emulab for different sleep time periods, and an attack pulse of length 160 ms. RTT is 160 ms. Attack packets are 64 bytes. ns-2 results are not plotted in the reverse case because the attack has little impact.

Node3 (reverse direction). The figures show key discrepancies between the simulation, Emulab, and DETER results.

From Figure 2.4(a), we find that for all values of sleep time, the goodput with ns-2 is typically higher than in other scenarios. Results on the Emulab testbed when the packet rate is at 140 Kpackets/s are mostly below the analysis and the DETER

results. This is because the attack creates significant overload on the Emulab PC routers, causing significant packet loss. For the 500 ms sleep time, the goodput on Emulab is almost zero. Examination of *tcptrace* and *netstat* output revealed that only few packets were sent throughout the duration of the connection and there was a considerable number of RTOs with large idle transmit periods. This indicates that the attack behaved as in the original low-rate attack paper [17]. When the attack rate is at 85 Kpackets/s, the results on Emulab are above the analytical results indicating that the attack has lost some of its potency. The simulation results reveal that synchronization effects appear even more pronounced as a result of non-monotonic increases in throughput. Additionally, the simulation results reveal that the attack with these packet rates and Linux TCP has a more significant impact on the testbed nodes compared to the simulated nodes.

To understand the significant differences between Emulab and DETER results, especially with 140 Kpackets/s attacks, we measure CPU utilization at the two PC router nodes *R1* and *R2*. The attackers on both testbeds are similar (2.0 GHz versus 2.8 GHz machines). We find that the load experienced on DETER PC routers is much smaller. This is because DETER PC routers have dual CPUs and an SMP configured OS, a PCI-X bus, and better NIC devices than the Emulab machines used in our experiments. This causes the attack to be more effective on Emulab than on DETER.[6] Therefore, we conclude that testbed measurements can *significantly vary* based on hardware and system software attributes.

In addition, we observe that packet loss in router nodes on the testbed can occur at a number of bottlenecks, as packets traverse multiple components and buffers, and any of these buffers can overflow. In contrast, *packet loss in ns-2 only occurs in case of output buffer overflow*. The ns-2 nodes themselves have "infinite CPU and bus capacity," and are capable of processing any flow without contention. This highly

[6]We have observed that the Emulab and DETER goodputs are similar over this range of sleep times when we induce higher loss on DETER. For example, we used randomly-generated source and destination addresses in attack packets. This results in significant ARP traffic, as well as load from backscatter packets, which reduces goodput on DETER to the values we see on Emulab.

simplified and idealized router model yields a dramatically different behavior when router components such as the CPU, buses, or routing fabric become bottlenecks as in these experiments.

Impact of packet size Our experiments with different attack packet sizes (results not shown here for brevity) have shown that, in case of packets with 1400 byte-payload, there is a less significant goodput degradation, confirming that small packets can cause more damage on PCs and PC routers because of higher packet rates, packet processing overhead, and slot-based queues.

Impact of attacker location Another key observation is that the ns-2 attack flow does not interfere with the TCP flow if it is flowing in the opposite direction (i.e., attacker at *Node3*), as links are full-duplex, port buffers are not shared, and there is no CPU or IRQ (interrupt) overhead per packet in ns-2. As there is no interference and one cumulative ACK loss does not typically cause a multiplicative reduction of Cwnd (just a potentially slower Cwnd increase), ns-2 is unaffected when the attack traffic flows in the opposite direction of the data traffic. We do not plot ns-2 results on Figure 2.4(b) or (d) as the system is only bandwidth limited, and the average goodput consumes the entire link.

Observations from Emulab and DETER experiments tell a different story: the average goodput is clearly affected by an attack in the opposite direction (e.g., Emulab goodput with a 140 Kpackets/s attack rate is almost the same as when the attack was in the forward direction as shown in Figure 2.4(a)). The interference on Emulab and DETER is because the Network Interface Card (NIC) on a PC router which receives a high bandwidth packet flow will consume all available system resources, and other NICs will starve. This results in interference among flows in opposing directions.

2.7 Using the Click Modular Router

In this section, nodes *R1* and *R2* in Figure 2.3 are configured to run the *Click* Linux module post version 1.5.0. Because DETER machines have Intel Pro/1000 Ethernet cards, it was possible to use Click's *e1000-5.7.6* NAPI polling driver to ensure that receive livelock does not occur, and Click has the most direct access to the driver. Additionally, we use dual 2.8 GHz P4 Xeon PC routers to leverage their faster CPUs and buses. Using this more complex IP router configuration caused stability issues with SMP. To overcome this problem, we had to compile the kernel and Click for a single CPU. Further, we changed the kernel version to Linux-2.4.33 to take advantage of patches that may increase stability under heavy load.

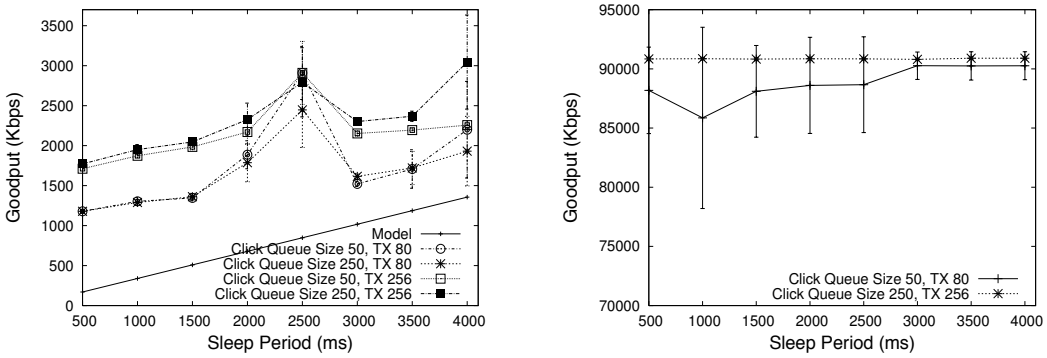
In Click, the entire packet path is easily described, and one can easily configure a simple IP router that bypasses the OS IP stack. Simplification of the packet path yields a performance boost, making the PC router less vulnerable to overload under high packet loads. When the router is configured, each affected network device has to be included into the configuration. It is easy to change the queuing discipline and the queue depth for the queue at each output port. We will, however, show that it is insufficient to change the Click *Queue* element depth. This is because Click (or any software system for that matter) has to go through a device driver when it accepts or outputs a packet. As with any Linux network device driver, the driver for the Intel Pro/1000 card has internal transmit (TX) and receive (RX) buffers. The Click *Queue* elements serve as intermediaries between these. We have created a simplified version of an IP router as described in [42], where the input ports are connected to a routing table. The routing table then sends the packets to the appropriate *Queue* element from which the output port pulls the packets.

Forwarding performance We configure the Click routers with a queue size of 250 packets per output port. The transmit (TX) buffer on the device drivers was left at the default value of 256 MTU-sized packets. Without any attacks, the TCP flow achieved an average goodput of 90.83 Mbps with a std. deviation 0.054 Mbps. With

a non-stop flood of over 140 Kpackets/s at *Node1*, Click's *udpcount* element at *Node3* verified that the receiving rate was the same as the sending rate with minimal packet loss. We confirmed the results with the Emulab's *portstats* utility.

Impact of Click queue and device queue sizes To understand the effect of varying Click (OS intermediates) and device driver buffer sizes, the Click *Queue* element size was set to 50 or 250 slots, while the driver transmit (TX) buffer was set to 80 (minimum setting) or 256 (default). We did not experiment with varying receive (RX) buffer sizes because [49] demonstrated that receiving is much faster than transmitting on PCs, and hence drops do not typically occur because of the RX buffer overflow. The default drop-tail queuing discipline was used. Figure 2.5(a) demonstrates that varying the TX buffer size produces *significant* variation in the results. It is also important to note that the TX buffer size has a *much more profound* impact than the Click queue size. Figure 2.5 clearly shows that a TX of 256 and a Click *Queue* of 50 performs much better than a TX of 80 and a Click *Queue* of 250. This implies that it is *crucial* to be aware of the driver settings. We have also experimented with an attack in the reverse direction as the TCP flow. Figure 2.5(b) clearly indicates the advantage of polling versus interrupts in Click: there is little interference between the opposing flows.

Click versus simulation As we know the exact size of the default TX output queue (256 slots), we ran a set of ns-2 simulations with router output buffer sizes set to 256 packets. We decided not to include the size of intermediate Click queue as the previous experiments have shown that the TX buffer sizes are more significant. In the simulations, we set the attack packet rate to 140 Kpackets/s (as it is on the testbed). Figure 2.6 depicts the results. The simulation results follow Click's results *after the 10 second mark* for this case. The simulation and Click results indicate that the attack is weaker than the analysis predicts.



(a) Attack in forward direction (b) Attack in reverse direction

Figure 2.5. Impact of varying the Click router queue and the transmit buffer ring of the network device driver on DETER. Attack packets are 64 bytes. The attack pulse length and RTT are set to 160 ms.

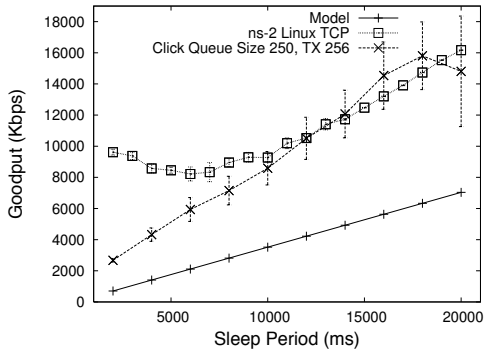


Figure 2.6. Comparison of the average goodput from analysis, simulation, and Click (on DETER) for long sleep periods. Attack packets are 64 bytes. Queue size in ns-2 and output device buffer ring in Click is 256. The attack pulse length and RTT are set to 160 ms.

2.8 WAIL Testbed Results

In this section, we conduct experiments with commercial Cisco routers on the WAIL testbed.

2.8.1 Cisco 3640 Routers

Cisco 3640 is a multi-service access platform for medium and large-sized enterprises and small Internet Service Providers [50]. The Cisco 3640 router is essentially a PC that runs on a 100 MHz RISC-type CPU and uses DRAM and NVRAM for buffer and OS storage. This router uses interrupts for packet processing, making it susceptible to receive livelock. The routers used for our experiments have been configured to use Cisco IOS Version 12.4(5). By default, the input queue size is 75 slots and the output queue size is 40 slots.

Forwarding performance The router supports "Process Switching" mode [51] and "Interrupt Context Switching" [51], as well as several caching options. In process switching mode, the router uses IRQ handlers for receiving and sending packets. Additionally, the router schedules an *ip_input* process that performs the necessary look-ups and makes switching decisions. As the router has to schedule the *ip_input* process for each packet and the CPU is quite underpowered, the router cannot forward a full 100 Mbps flow composed of MTU-sized packets. In our tests, the router was able to forward a maximum of 1248 64-byte packets per second.

The interrupt context switching mode is an optimized version of the process switching mode. The optimization happens when all the switching work is performed via the interrupt handlers and the *ip_input* process does not get scheduled. In this mode, there are three caching methods available for destination lookup. The methods are fast, optimum, and Cisco Express Forwarding [51]. By default, fast switching is used as the caching policy. This method relies on a binary tree for lookup. As this is the simplest method, "Process Switching" has to occur if there is a cache miss. Additionally, the cache has to get invalidated periodically as there is no correlation between the ARP and routing table entries.

The drawback of the fast and optimum switching is in the fact that aging and "Process Switching" occasionally has to happen. To remedy this situation, Cisco IOS versions 12.0 and later support "Cisco Express Forwarding", CEF [51]. Cisco

recommends this switching method and plans to make it the default on most of their router series. CEF caching relies on a 256 way trie data structure with pointers to MAC and outbound interface tables. This removes the need to invalidate the cache as changes to the MAC and outbound interface tables do not require a change in the routing cache. As in our experiments we do not use large routing tables, CEF and default IP lookup tables yield similar performance.

In our experiments, the router was able to process 14.4 K 64-byte packets per second in the interrupt context switching mode with "Cisco Express Forwarding" (CEF) [51], which is a significant improvement over the process switching mode.

TCP without attack Cisco IOS offers the ability to install IP filters on the input and output ports to provide egress/ingress filtering capability. Using IP filters is advised to prevent IP spoofing, thus we ran our experiments with and without filters. Figure 2.3 depicts the topology that we have used, but *R1* and *R2* are now Cisco 3640 routers. The experiment was run in exactly the same manner as was described in Section 2.5. We first ran ten experiments without the attack to determine TCP performance under normal conditions. The filters only allowed the packets with IPs that were part of the test network to be forwarded. Without the filters, the TCP flow was able to achieve an average transfer rate of 83.3 Mbps with a std. deviation of 0.0213 Mbps; however, with the filters the goodput dramatically dropped to an average of 7.3 Mbps with a std. deviation of 0.111 Mbps. This is not surprising, as enabling IP filters forces the packets to be "process" instead of "fast path" switched. The difference in switching paths can result in a performance of difference of an order of magnitude [52].

3640 versus simulation As we know the exact size of the router output queues (40 slots), we ran a set of simulations with this size. In the simulations, we set the attack packet rate to 140 Kpackets/s exactly as on the testbed. We disabled the IP filters on the routers and varied the sleep times from 2 seconds to 20 seconds in 2 second increments to capture a variety of attack scenarios. Figure 2.7 depicts the

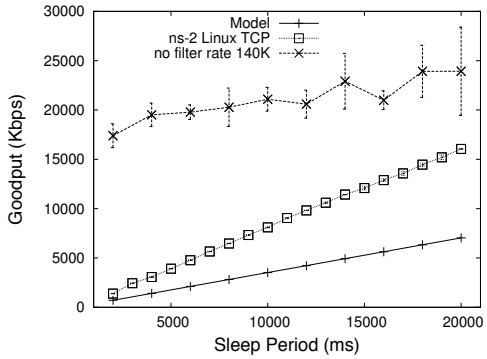


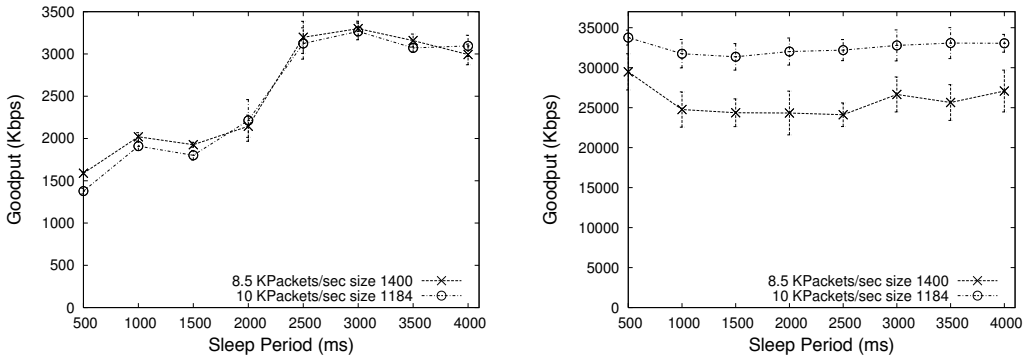
Figure 2.7. Comparison of the average goodput from analysis, simulation, and Cisco 3640s when long sleep periods are used. The IP filters on the Cisco routers are disabled. Attack packets are 64 bytes. The attack pulse length and RTT are set to 160 ms.

results. The router results indicate that the attack is much *less effective*, compared to the simulations and analysis. This is because most of the attack traffic is dropped at the input queue of the interface that receives the attack traffic. If the attack traffic is multiplexed with the TCP traffic on the same input port, then the attack potency is significantly higher.[7]

Impact of packet size We now examine the effect of maintaining the same attack bit rate while changing the packet rate, by varying the packet size. We observe that achieving the same bit rate is not always possible by varying packet rates and sizes: for instance, 8500 Kpackets/s with 1400-byte payload creates a 98 Mbps flow (includes UDP/IP/Ethernet headers). The smallest packet size to use to achieve the same bit rate is 1184 bytes. Any smaller size fails to meet the desired bit rate.

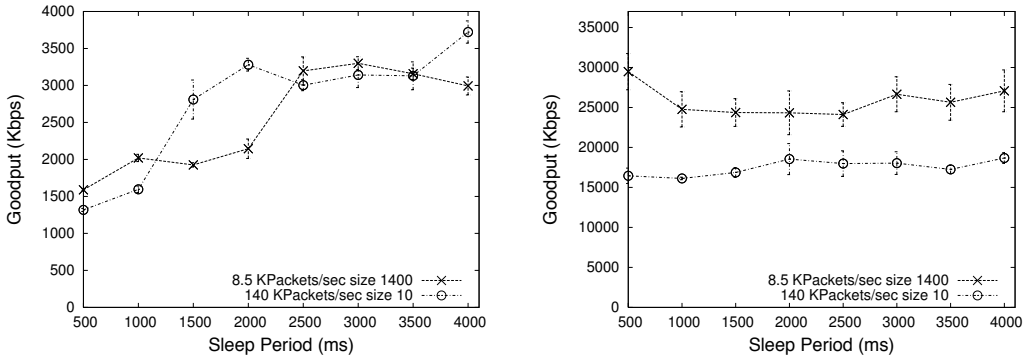
Figure 2.8 demonstrates TCP goodput when the attack rate is 8.5 Kpackets/s with 1400-byte payload versus 10 Kpackets/s with 1184-byte payload packets. The experiment was run with and without IP filters. The results surprisingly indicate that slightly larger packets at a slightly lower packet rate cause more damage to

[7]We have also observed that if we use TCP packets instead of UDP packets for the attack, the TCP goodput degradation is closer to the model. This is because a stream of unsolicited TCP data packets causes the sink to respond with RST packets, causing significant congestion at the router.



(a) With IP filters (b) Without IP filters

Figure 2.8. Effect of varying packet size and packet rate – while maintaining the same bit rate of 98 Mbps – with and without IP filters on Cisco 3640s. The attack is sent at 8.5 Kpackets/s with 1400 byte payload, or at 10 Kpackets/s with 1184 byte payload. The attack pulse length and RTT are set to 160 ms.

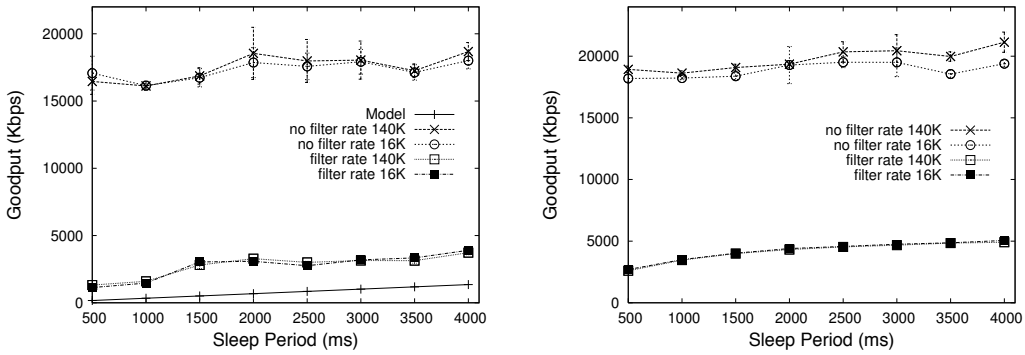


(a) With IP filters (b) Without IP filters

Figure 2.9. Impact of 140 Kpackets/s (64-byte packet size) versus 8.5 Kpackets/s (1400 byte payload) attack flows on Cisco 3640s with and without IP filters. The attack pulse length and RTT are set to 160 ms.

TCP. To investigate the impact of varying packet rates, we have conducted another experiment where the packet rate was 140 Kpackets/s and the packets were 64 bytes and compared it to the 8.5 Kpackets/s with 1400-byte payload results. Figure 2.9

demonstrates the results. It is interesting to note that when IP filters were present, larger packets sometimes induced more damage than smaller packets; however, when IP filters were removed, smaller packets at a higher rate clearly had a more profound impact.



(a) Attack in forward direction

(b) Attack in reverse direction

Figure 2.10. Comparison of the average goodput from analysis, simulations, and WAIL Cisco 3640s for different sleep periods, attack rates, and a pulse length of 160 ms. RTT is 160ms. Attack packets are 64 bytes. The reported results for the Cisco routers are with and without IP filters. ns-2 results are not plotted in the reverse case because the attack has little impact.

Impact of IP filters Figure 2.10 illustrates the results with the TCP-targeted attack in forward and reverse directions. As in Section 2.6, we use UDP packets that create 64-byte frames at the data link (Ethernet) layer. The attack rate is either 140 Kpackets/s or 16 Kpackets/s (14 Kpackets/s is the measured maximum loss free forwarding rate). The results confirm that the *use of IP filters has a profound effect on the impact of the attack*. Figure 2.10(a) and (b) show that an attack with a maximum pulse height is not meaningful if the device can be overloaded at lower attack intensity levels. By observing this fact, an attacker can make the attack much more stealthy and harder to detect. Finally, *the results indicate that the attack has an impact in the reverse direction, even when a commercial router is used.*

2.8.2 Cisco 7000 Series Routers

A few rungs up the ladder from the 3600 series are the 7000 series routers. Such routers are designed as aggregation points for large enterprises or Points of Presence (POP) for small ISPs. The routers support a variety of network modules that include Gigabit Ethernet, Fast Ethernet, SONET, and Serial. Router documentation [53] reveals these routers to be similar to multi-bus interrupt-driven PCs. Similar routers are advertised by ImageStream [38] based on commodity PCs with special network cards running a modified version of the Linux operating system. Unfortunately, the router layout on the WAIL testbed does not allow us to use the same topology as in Section 2.5. The link connecting the two routers is a 155 Mbps POS-3 instead of a 100 Mbps Fast Ethernet. We used a Cisco 7206VXR with IOS Version 12.3(10) as *R2* and a Cisco 7500 with IOS Version 12.3(17b) as *R1* in Figure 2.3.

Forwarding performance Unlike the case of the 3640s, the 7000 series routers are capable of forwarding almost 140 Kpackets/s when IP filters are disabled.

TCP without attack Without the attack and with no filters, the TCP flow achieved an average rate of 24.99 Mbps with 4.53 Mbps std. deviation. With the filters, the average rate decreased to 6.37 Mbps with a std. deviation of 0.179 Mbps. Note that the TCP flow did not achieve a full link utilization in the no-filter run; this was because of occasional packet loss in the routers. However, with multiple flows using the "-P" option in *iperf*, the aggregate of the flows did reach full link utilization. We suspect that some hardware issue is causing occasional packet loss. For high-speed TCP flows, even an occasional loss is detrimental as the congestion window is significantly cut, requiring many consecutive round trips to reach full link capacity.

7000s versus simulation As the link between the routers was a 155 Mbps SONET link and the output queues were of size 40, we ran another set of simulations with these

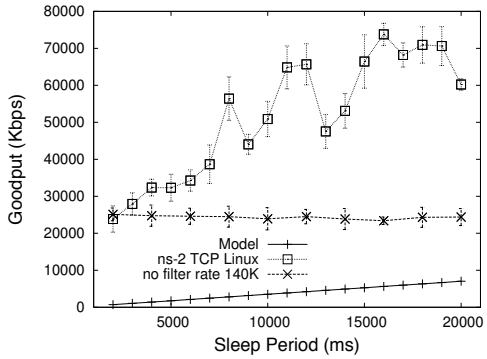
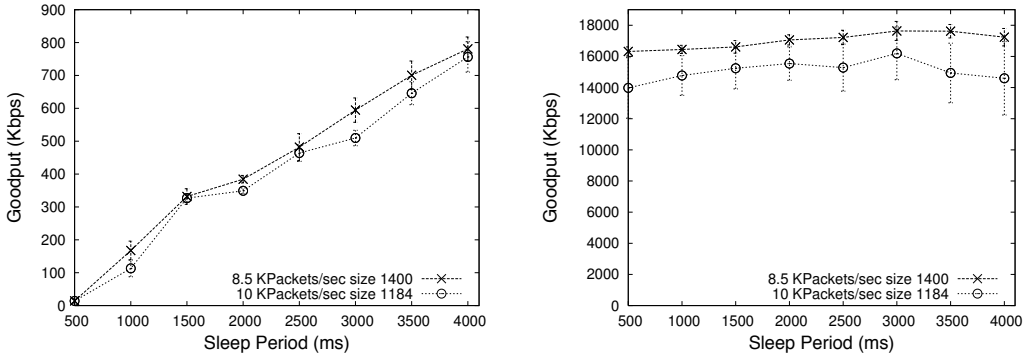


Figure 2.11. Comparison of the average goodput from analysis, simulation, and Cisco 7000s when long sleep periods are used. The IP filters on the Cisco routers are disabled. Attack packets are 64 bytes. The attack pulse length and RTT are set to 160 ms.

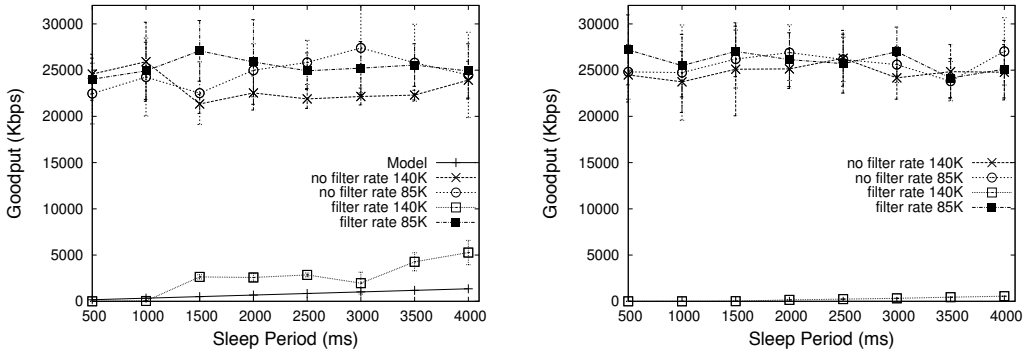
settings. In the simulations, the attack packet rate is set to 140 Kpackets/s as on the testbed. Again, we ran the routers without IP filters and varied the sleep times from 2 seconds to 20 seconds in 2 second increments. Figure 2.11 illustrates the results. In the simulation results for the 140 Kpackets/s attack, there is substantially more damage initially, but then the attack impact is quickly reduced as the throughput climbs to almost 80 Mbps in some cases. The reason for the discrepancy between simulation and the analytical model is because the link speed difference invalidates the assumption that each attack pulse will result in a Cwnd cut. The results on the routers are appreciably close to the results when no attack was present, showing that *the attack is not very effective* in this case.

Impact of packet size As with the 3640 routers, we compare the effect of maintaining the same bit rate while changing the packet rate by varying the packet size. Figure 2.12 demonstrates the results when the attack rate is 8.5 Kpackets/s with 1400-byte payload packets versus a 10 Kpackets/s rate with 1184-byte payload packets. As before, the experiment was run with and without IP filters. Unlike the 3640 results, the 7000 results clearly indicate that even slightly smaller packets at higher packet rates produce more damage.



(a) With IP filters (b) Without IP filters

Figure 2.12. Effect of varying packet size and packet rate while maintaining the same bit rate of 98 Mbps with and without IP filters on Cisco 7000s. The attack is sent at 8.5 Kpackets/s (1400 byte payload) or 10 Kpackets/s (1184 byte payload). The attack pulse length and RTT are set to 160 ms.



(a) Attack in forward direction (b) Attack in reverse direction

Figure 2.13. Comparison of the average goodput from analysis, simulations and WAIL Cisco 7000s for different sleep periods, attack rates, and a pulse length of 160 ms. RTT is 160 ms. Attack packets are 64 bytes. bytes. The reported results for the Cisco routers are with and without IP filters. ns-2 results are not plotted in the reverse case because the attack has little impact.

Impact of IP filters Figure 2.13 shows the results for forward and reverse direction attacks with varying attack packet rates, with and without IP filters. It is interesting to compare the results for forward and reverse directions when IP filters are used. The first router to receive the attack flow drops some of the attack packets; hence, the downstream router receives a reduced attack flow. By changing direction of the attack, we change which router experiences the higher load. The large difference between forward and reverse directions at 140 Kpackets/s when using IP filters is because the Cisco 7206VXR merges the TCP and the attack flows in the forward direction, while the Cisco 7500 merges the TCP ACK and the attack flows in the reverse direction. The Cisco 7206VXR has a R7000 CPU at 350 MHz, while the Cisco 7500 has a R5000 CPU at 200 MHz. It is not surprising that the slower CPU on the 7500 led to higher packet loss when IP filtering was enabled. Additionally, it is interesting to note that the impact of the attack on the TCP goodput is higher when larger packets at low rates are used (Figure 2.12(b)) compared to the attack with smaller packets but higher rates (Figure 2.13(a)) (e.g., 16 Mbps vs. 25 Mbps). The results indicate that *in this case, large attack packets cause more damage than smaller ones.*

The results imply two important conclusions. First, as in all previous experiments except for Click, the attack has an impact in both the forward and *reverse* directions. Second, IP filters dramatically increase the impact of the attack.

2.9 Summary

Our comparisons between simulation and emulation experiments with seemingly identical configurations have revealed *key differences* in the results. Some of these differences occur because simulators abstract a number of system attributes, and make several assumptions about packet handling. For example, because PCs are used on Emulab/DETER and routers on WAIL, their CPUs, buses, devices, and device drivers may create a bottleneck that simulators do not model. Another important

observation from comparing data from the Emulab and DETER emulation testbeds is that even though the hardware and software on both testbeds may appear similar, the nodes on Emulab experience a much higher CPU load than the DETER nodes for the same packet rate. This means that *the same experimental setup (configuration files, software, emulation platform) may produce widely different outcomes*, as results are highly dependent on the details of underlying hardware and software, and their default settings. These different settings do *not* cause widely different outcomes in typical networking and operating systems experiments, but cause dramatic differences under DoS attacks that overload the system. This observation was once again confirmed by running the experiments on Cisco routers at the WAIL testbed. There were major differences between different Cisco routers. We have gone to great lengths to investigate the reasons why PCs, Cisco routers and simulation models vary between each other. Figuring out the minute details is a exceedingly long and time consuming process, highlighting the need for automation and high–fidelity simulation models.

3 RELATED WORK

In the previous chapter, we have used an ns-2 simulator and several emulation testbeds to reveal large discrepancies in the experimental results for the same experiment. The study demonstrated the need for more accurate router models for simulation experiments. In this chapter, we will describe the state of the art simulators and emulators, and demonstrate that router fidelity is currently an unresolved issue. Finally, we discuss the works in the area of router benchmarking and modeling.

3.1 Network Simulation and Emulation

Simulation and emulation are invaluable tools for a networking researcher as they allow experiments to be carried out with much less overhead than with physical networks. Simulators are typically more scalable [6] than emulation tools, allowing for experimentation with considerably large networks and high data rates. However, the scalability comes at the expense of fidelity, as model accuracy has to be sacrificed to reduce computational overhead [7]. Accuracy can be improved by using emulation tools to merge some aspects of the physical network with the simulator. Such an arrangement improves the accuracy at the cost of scalability. In the following sections we will describe the current simulation and emulation tools that are available today.

3.1.1 Network Simulators

One of the most well known simulators is ns-2 [8], maintained by ISI. This discrete event simulator has extremely basic link models and is mainly used for queuing and end-to-end protocol research. Routers are represented as collections of output link queues ignoring any physical layer models. In ns-2, packets are treated as messages

and there is no notion of network layers. This shortcut obviously reduces the running time and the memory requirements but takes away from realism. As ns-2 cannot be run in a parallel or distributed fashion, a parallel/distributed versions PDNS [54] was developed. GTNetS [55] addresses the layer problem of ns-2 by using real packet formats and layers, it also allows distributed simulation just as pdns.

Near the other end of the spectrum, from highly-scalable simulators lie simulators such as OPNET [10] and OMNeT++ [11]. In OPNET, detailed models of routers, switches, servers, protocols, links, and mainframes are given, solely based on vendor specifications [12]. Using complex models significantly increases computational cost, hindering scalability. Further, the model base needs to be constantly updated. Validation attempts reveal that even these accurate models are sensitive to parameters such as buffer size and forwarding rate that are hard to tune to mimic real router behavior [12].

Simulators such as ns-2, are frequently critiqued for lack of network layers. Additionally, the simulator code is not interoperable with any operating system, thus making it difficult to validate the results in a physical experiment. Even though OPNET has network layers, it has the majority of packet handling code written from scratch, raising the issue of credibility as well. NCTUns [56] solves this problem by making hooks into the Linux kernel to use as much of the OS code as possible to make validation easier, as the operating system code is part of the simulator. Even though NCTUns addresses the interoperability issues, it does not provide high-fidelity hardware/software models for wired packet forwarding devices.

There are other simulators such as iSSF/iSSFNet [43], J-sim [57], but they are not as popular as the simulators listed above.

3.1.2 Network Emulation Tools and Emulators

One of the earlier emulators is VINT [58]. VINT integrates the ns-2 [8] simulator with the real world by a “bridge” that translates real packets into simulator represen-

tations and vice versa. This approach offers good scalability and flexibility properties as the user can easily modify the simulator and simulate arbitrary topologies. On the downside, the user has modify the "bridge" to translate new types of packets correctly. The extra packet translation steps can result in sub-realtime performance, hence creating a backlog of events. Additionally, the simulator can induce artifacts into the results because of incomplete/incorrect models.

Modelnet [59] is an emulator capable of large scale topologies. It emulates the network core leaving only the edge nodes to the user. The system has substantial scaling properties because it can aggregate a large number of links on a single high–end machine and be distributed over a set of machines. The system is good at producing end-to-end delay, but is incapable of performing dynamic routing or allowing the user to install his/her own applications at the core. An emulator called EMPOWER [60] is similar to Modelnet but it also adds wireless capability. EMPOWER also suffers from the problem that users cannot run routing software or request a high fidelity router model.

Instead of creating network "clouds", it is possible to use virtualization to achieve a similar effect but with finer control of the "cloud"'s properties. An emulator vBET [61] relies on User Mode Linux (UML) to create virtual machines that run on the same physical machine. The added capability to create emulated switches and hubs allows the virtual nodes to communicate with each other. A single PC can run an experiment of a small-to-medium network with important details such as: routing exchange, real protocols, and real software.

One of the largest emulation testbeds is Emulab. As discussed in Section 2.4 it is a time- and space-shared network emulator located at the University of Utah [34]. There are currently several testbeds in various institutions and labs that are based on Emulab software. The testbed is composed out of hundreds linked PCs that can be connected in any specified topology. During an experiment allocation, the management system reserves PCs and then connects them via VLANs, to form a

desired topology. Emulab [62] also supports virtualization of nodes as well as NICs, allowing for similar capabilities as vBET.

Network emulation can range from emulating large segments of the network as was discussed above, to just artificially shaping a single link. Tools such as *tc*, *iproute2*, NIST-net, DummyNet [22], NetPath [45], Click modular router [42] allow to do just that.

3.1.3 Device Fidelity

All of the work described above focuses on establishing connectivity and providing requested delays and link bandwidth. However, critical properties of real packet forwarding devices such as: queuing delays, maximum packet forwarding rates, policies, and realistic queue sizes are not considered. Failing to consider these properties reduces the fidelity of the simulation/emulation experiments especially in cases when high traffic load is present.

3.2 Router Modeling

Packet forwarding of the router can be modeled in two ways. First, a highly detailed model that captures the hardware and the software can be constructed. Such a model would be extremely difficult to create and validate. Also, using such a model can be a hindrance to scalability because of a large processing overhead. The second approach relies on a less detailed approximation model. Such a model can be constructed either with complete knowledge of the forwarding device, or the router can be treated as a black–box. As it is appreciably difficult to acquire complete knowledge of the system, we will look into the approach where the router is treated as a black–box. In this section, we will cover the related work in the areas of black-box testing, traffic generation, and modeling based on empirical data.

3.2.1 Black-box Testing and Traffic Generation

Black-box testing is an attractive approach to create approximate models. First, no inside knowledge is needed of the device under the test. Second, the model is constructed from empirical observations, thus reducing the need for extensive validation. To conduct black-box testing, it is crucial to generate traffic to act as measurement probes, which are used to infer the router's characteristics.

Black-box Testing Black-box router measurement is described in [63–65]. In [63], a router is profiled with a focus on measuring its reaction times to OSPF routing messages. The router is first configured with an initial OSPF routing table. Then the router is injected with OSPF updates. The researches then measured the reaction times to the updates to produce the results. RFCs 2544 [64] and 2889 [65] describe the steps to determine the capabilities of a network device designed for packet forwarding. The RFCs specify a series of tests where homogeneous traffic is injected into the device under the test. However, there is no discussion regarding heterogeneous traffic tests. Additionally, there is no discussion on how to create models based on the acquired measurements.

Traffic Generation Generating homogeneous traffic is straight forward and does not require sophisticated tools. It is also possible to generate heterogeneous traffic without much effort by replaying *tcpdump* or DAG [66] traffic traces. Creating highly configurable *live* (i.e., *closed-loop*) traffic is important for modeling purposes, as it can expose the router to higher loads than homogeneous traffic can [67].

One of the earliest network simulation-emulation tools was VINT [58] – a part of ns-2. As the ns-2 simulator has a variety of TCP stacks, it is straight forward to configure it to generate *live* traffic via its emulation agents. The problem with ns-2 emulation code is that it does not support sending/receiving spoofed IPs (required for subnet emulation on a single node), and it is considerably data-rate limited. A recent effort to extend emulation in ns-2 was reported in [68]. However, the system was not

built to handle excessive data rates and extensive packet logging with micro-second precision, which are important for high-speed router measurements. A commercial alternative to generating live TCP traffic is the IXIA-400T traffic generator [69]. IXIA devices use a proprietary OS and do not allow changing the types of the TCP stacks, however.

Realism can be increased further, by creating live traffic from models derived from traffic traces. For instance, Harpoon [67] traffic generator uses flow data collected by Cisco routers to generate realistic traffic. Additionally, tools such as *tmix* [70] or Swing [71] generate traffic based on previously measured realistic application workloads. Even though such tools offer accurate traffic generation, they rely on extensive packet traces. Acquiring such traces can sometimes be problematic either as a result of infrastructure or human subject research issues.

3.2.2 Empirical Router Modeling

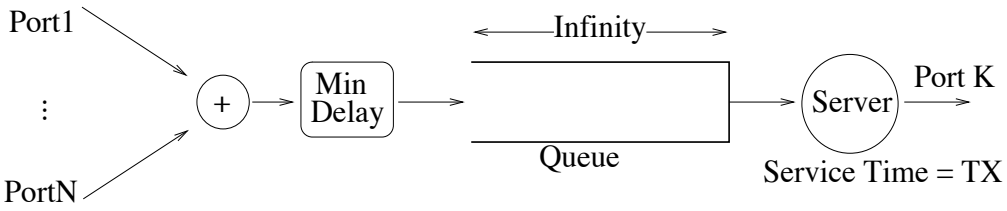


Figure 3.1. Minimum delay queuing model with an unbounded queue *per output port*. The service time is based on the packet transmission (TX) time.

Router modeling based on empirical observations is explored in [15], as discussed in the introduction. The work resulted in a model based on a simple queuing model; however, the model was not designed to handle loss events and ignored interactions at the input ports. A production Tier-1 router was used in that work, as well as in analysis of micro-congestion in [72]. While this ensures that the router configuration and traffic are highly realistic, repeatability is not possible in a production setup.

Time-stamping was performed with GPS synchronized DAG cards. Such devices are highly accurate, but increase the setup cost and complexity significantly. This is because, per each input and output port a DAG card is required. Hence, for a four port router eight DAG devices would be needed. Additionally, a DAG card needs to be installed in a PC so that packet logging to disk can take place.

Fig. 3.1 depicts the Virtual Output Queue (VOQ)-based router model suggested in [15]. The model is considerably similar to the classical output queue model, except there is a constant delay added to each packet based on its packet size. The extra delay signifies additional router overhead required for packet processing. This delay is derived from experimental measurements by finding the lowest delay for a given packet size. The authors choose this method to ensure that queue delay is not a factor. Each output port is modeled in this fashion, ignoring any interactions at the inputs and the backplane. We believe that this model is accurate for core routers which have a sophisticated switching fabric, but it can be *inaccurate* for lower–end devices. For example, there was no loss observed in the core router in [15], and hence the queues have unlimited capacities. The VOQ model is quite attractive because of its simplicity; however, it fails to account for details which can lead to large deviations in the results with other types of forwarding devices as was discussed in chapter 2.

3.3 Summary

In this chapter, we have described the current state of the art in high-fidelity simulation/emulation. As in chapter 2, we have shown that current simulators do not model packet forwarding devices in an accurate fashion, hence reducing the accuracy of the simulations, especially in high–load scenarios. Increasing simulation accuracy can be done with better router models. Using a simple model that uses empirically derived parameters has two primary advantages: model simplicity and improved accuracy. In Section 3.2.2, we have examined the current viable approach to model routers based on empirical measurements. However, the proposed method

ignores any possibility of backplane contention and packet losses. Additionally, the proposed scheme uses DAG cards, which are costly and require extensive setup time, for empirical measurements.

4 BENCHMARKING AND MODELING ROUTERS

As was discussed in Section 3.2.2, the empirical VOQ model has a number of shortcomings which limit its usefulness. To understand how to overcome those shortcomings, we fist conduct an overview of what the commercial routers are composed off and what features they offer. Second, we provide a brief overview of the router software. Third, we describe our empirically based device independent model, its parameters, and finally the parameter derivation steps.

4.1 Commercial Router Overview

Routers come in a multitude of types and configurations. Any device that can execute a routing protocol (e.g., OSPF, BGP, RIP and ISIS) and forward packets is considered a router. Routers made for household use are not technically routers, as they do not run a routing protocol and only forward packets based on DHCP information. However, we still consider such devices as they perform IP packet forwarding. Between the extremely limited household devices and core–level routers, there is a wide range of devices. In the overview, we will focus mainly on Cisco's products, as they are the industry leader and their competitors provide similar offerings.

The Cisco 1600 [73] series routers are full fledged routers designed for small offices and medium enterprises. Routers of this series can be considered as PCs built out of proprietary components. Fig. 4.1 shows the basic layout of such a router. The block diagram except for the video and IDE/SCSI buses is considerably similar to a PC layout. Even though this is a low–end router, it supports the majority of Cisco IOS features. A few levels up the ladder, there are 3600 [52] series routers. These devices are also similar to PCs but have much better hardware compared to the 1600 series. Because the 3600 series are tailored for medium enterprises, branch offices,

and small ISPs, the 3600 series are designed to be highly modular and operate with a large variety of physical access mediums (e.g., SONET, Ethernet, Serial) and support a rich feature set of the Cisco IOS.

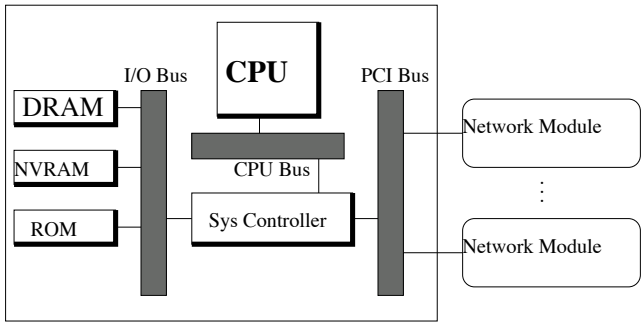


Figure 4.1. Basic layout of an interrupt driven router.

To fulfill the needs of medium enterprises, branch offices, and small and medium ISPs the Cisco 7200 series routers [53] are a good fit. These modular routers support a variety of network modules that include Gigabit Ethernet, Fast Ethernet, SONET, and Serial. Fig. 4.2 demonstrates the layout of such a router. The documentation and the block diagram reveal this router to be similar to a multi-bus PC. Similar routers are built by ImageStream [38], based on commodity PCs with special network cards [37] running a modified version of the Linux operating system. However, routers of this class are inadequate to serve as backbone routers due to their limited capacity. It is interesting to note that the 1600, 3600, 7200 series rely on CPU interrupts to process and forward packets. This implies that the CPU can be a contention point, and flows that do not share the same output queue can interfere with each other.

At the extreme side of the spectrum there are backbone and large data center routers. Typically, such routers rely on highly dedicated and modular hardware to handle OC-192 speeds across multiple ports. Commercial examples would be Cisco 12000 [74] series and Juniper M7 [75] and higher series. Fig. 4.3 demonstrates the layout of a modular router. In contrast to the previously described routers, backbone routers can have multiple switch fabric cards, and dedicated route processing cards.

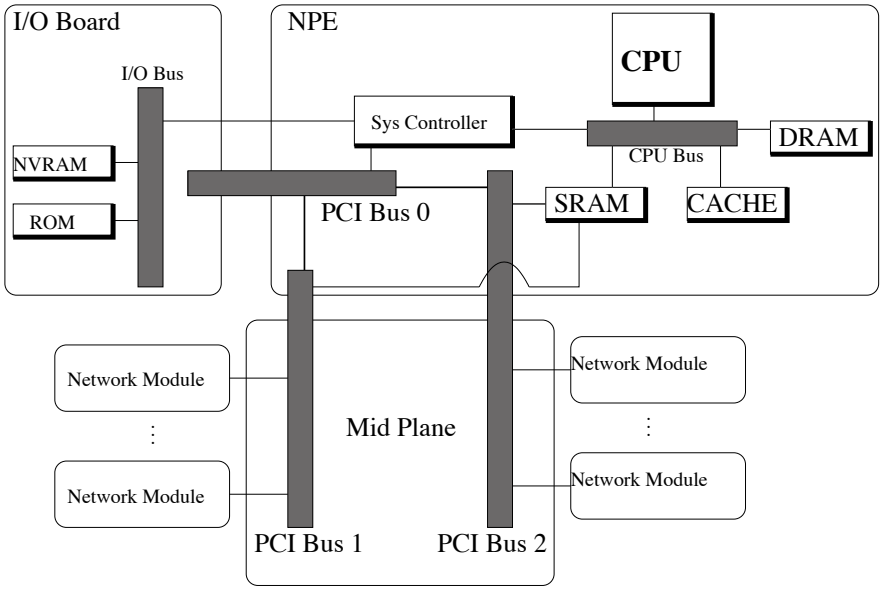


Figure 4.2. Multi-bus router layout block diagram.

The modules typically are connected to the same supervisory module to create a cohesive unit. Additionally, each line card can contain multiple ports as well as its own dedicated switch fabric. This is necessary for cases when switching needs to be performed only between the ports on the same line card. These systems are geared towards massive parallelism to handle OC-192 line rates on multiple interfaces. It is not uncommon for such routers to contain over a million IP addresses in the routing table, requiring an exceedingly sophisticated distributed memory architecture. Such speeds are achieved, by forwarding the majority of packets without the CPU's involvement. A dedicated route processing card runs a routing protocol and then updates the forwarding tables on the switching cards. Only in rare circumstances will the main CPU be involved to deal with a packet. Such cases involve cache misses, IP options, and routing/switching related updates.

Backbone routers typically employ a Virtual Output Queue (VOQ) to eliminate head of line blocking. Head of line blocking occurs when packets destined to one congested port end up delaying packets, which are destined for non-congested ports. Such a situation can result in additional delay or even input buffer overflow. Fig. 4.4

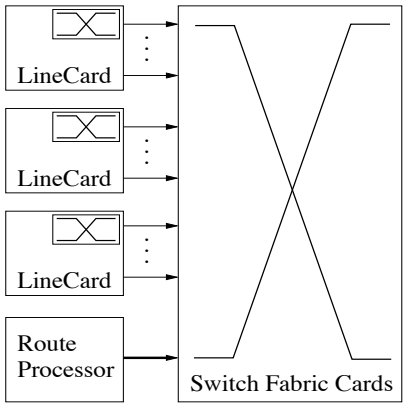


Figure 4.3. High speed modular router layout diagram.

demonstrates how Cisco 12000 series routers solve this problem by implementing the VOQ strategy. When a packet arrives at a line card, it gets classified by its destination and then gets placed into an appropriate virtual output queue. The crossbar switching scheduler examines which virtual output queues are non-empty and then constructs a schedule [76–78] to move the packets from virtual queues to output queues. Once a packet is moved over the crossbar into the output queue, it gets enqued or dropped, if the queue becomes full. As each packet must traverse several modules on its way to the output queue, a packet incurs multiple service delays. The aggregate sum of the delays is non-negligible. In most network simulators such as ns-2, it is most common to approximate routers as a VOQ model and discount any processing delays.

4.2 Router Software Overview

Router software can range from a limited feature sets, such as what is found on tiny home "routers", to considerably large feature sets as found in Cisco IOS, Juniper OS (JUNOS), and Linux/BSD derivatives. The software needs to handle a variety of tasks that include: configuration, hardware interaction, routing protocol exchange, policy execution, packet forwarding, and cache/queue management.

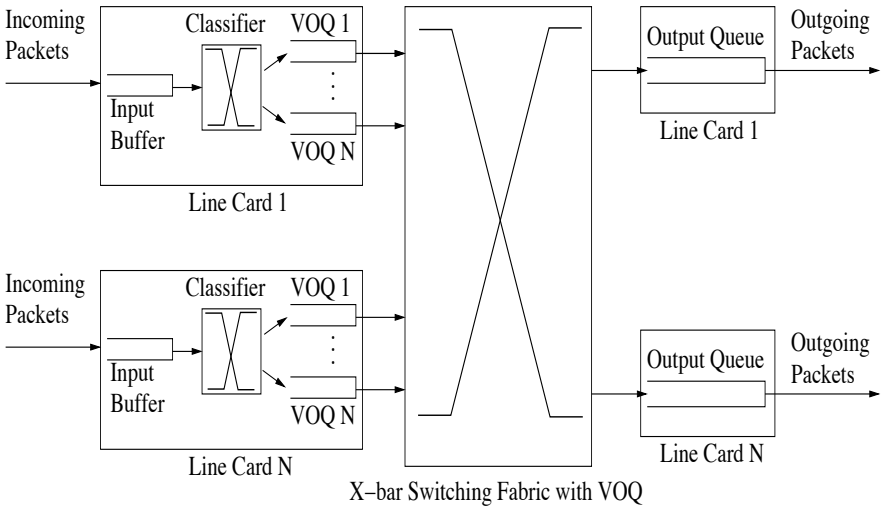


Figure 4.4. VOQ Crossbar layout. Incoming packets get classified in the line card into separate VOQ queues to avoid head of line blocking.

To an outside observer, features such as routing protocol support are not too difficult to determine; however, traffic and queue policies features are much more problematic to deduce. Additionally, polices can describe preferences to VoIP, MPLS, and flow control. The software also influences which traffic takes the "slow" path (involves invoking the CPU and the IP routing process) or the "fast" path (minimal-to-none CPU involvement). Usually the "slow" path is taken by control packets, packets with IP options, and packets whose IP/MAC mapping is not cached. The difference between a "slow" and a "fast" path can be as much as an order of magnitude.

Queue management includes configuration of the sizes and drop conditions for the queues. All routers have a set of input and output queues. This is necessary because the line cards need buffers where to store incoming/outgoing packets. Input queues are drop tail only while the output queues can be configured to run various queuing methods such as: FRED [79], RED-PD [80], SRED [81], Fair Queuing, and drop tail. To complicate the matter further, the router can have intermediary buffers between the line cards in main memory [9]. Even in a VOQ router as in Fig. 4.4, a packet gets broken up into cells and then traverses several buffers before arriving at the final queue destination.

4.3 Modeling a Forwarding Device

As was described previously, a multitude of router types with different architectures and performance exist today. Switching fabrics range from simple shared memory or shared medium designs, to sophisticated highly interconnected designs such as the Knockout switch. Most modern routers fall between these two extremes and use a type of a Multistage Interconnection Network, such as the Banyan, Benes, or Clos ([82] gives a quick overview). Despite these disparate designs, routers share a few critical similarities:

1. Packets may get dropped or delayed within the router.
2. Routers have a number of input and output interfaces.
3. Routers can have intermediate buffers/queues.
4. Packet flow in a router is complex [9], and there can be several queues and servers for each part of the path.
5. Packets can be split into fixed-size units while traveling between the input and output ports (as in many devices that use fixed-size "cells") [74].
6. Packet processing can be pipelined [13].
7. Shared components such as the backplane, caches, and possibly a central processor can lead to interference among flows that do *not* share the same outputs.

The complexities of real router tasks introduce difficulties in developing an accurate and comprehensive model of routers and other forwarding devices. A router must deal with control packets such as ARP and ICMP, as well as routing packets such as BGP, OSPF, and RIP. The control/routing packets can have a profound impact on the forwarding of regular packets. For instance, ARP can lead to a significant delay until mapping between a packet's IP and MAC addresses is established. Routing packets can lead to delays or losses of data packets as routes are removed or added. Routers

can have interfaces with different speeds and hardware (e.g., Ethernet, FastEthernet, SONET). Hence, for simplicity, we make the following assumptions to create a general packet forwarding model:

1. We do not model control traffic (e.g., OSPF, BGP, ARP, ICMP). We use static and small forwarding (address lookup) tables when profiling a device.
2. We assume that all the interfaces have approximately the same performance.
3. We assume that data packets are treated equally (no Quality of Service (QoS)-based packet scheduling, buffer allocation, or early packet discard).
4. We assume full-duplex operation.
5. We do not assume any knowledge of router internals or traffic statistics, however.

4.3.1 General Multi-Server/Multi-Queue Model

To model a range of forwarding devices, we must consider that traffic interactions in forwarding devices can play a significant role in causing packet loss and delay. We need a model that captures such interactions in a range of switching fabrics. Fig. 4.5 demonstrates a simple device-independent model that allows a range of contention behaviors. The additional complexity over the VOQ-based model thus allows modeling devices with limited performance, in addition to the Tier-1 access router modeled in [15].

As previously stated, routers may have multiple queues on the packet path from the input to the output based on the router architecture and type of switching fabric. Modeling the location of all the queues and their respective sizes would require detailed knowledge of each router internals. As this is infeasible, we approximate all the internal queues as a *single aggregate queue* of size Q slots per output port. However, packets can occupy more than one slot in the case of byte-based queues, or queues that use multiple slot sizes. Hence, a table *QueueReq* is used to specify how many slots a given packet size occupies. We infer Q and *QueueReq* from our measurements.

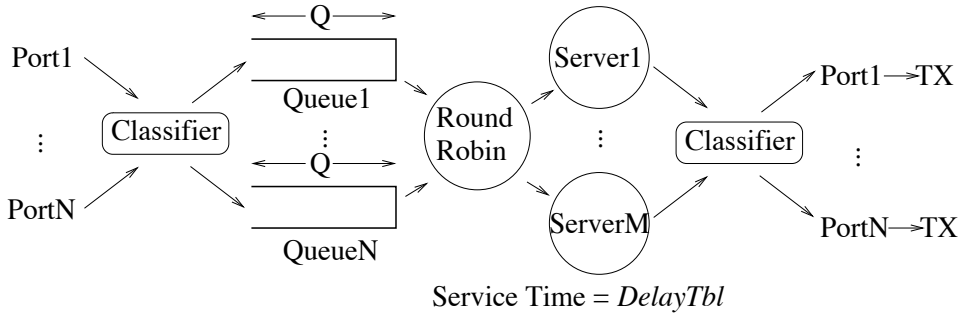


Figure 4.5. N inputs are served by M servers. There is one queue of size Q slots per port. Packets exit the forwarding device through one of the N output ports.

As seen in Fig. 4.5, traffic from N inputs is classified and queued by output port, served by M servers and proceeds to N outputs for transmission. In a forwarding device, input/output queues are served by the processors on the network cards, while intermediate queues might be served by a central processor(s) or specialized switching fabric processors. As it is difficult to determine the exact number of servers in a device, we infer the number of servers, M, based on measurements. Varying M from one to infinity allows us to model the entire range of routers from those with severe contention to those with none at all.

Servers process packets with the measured average processing delay. A table, *DelayTbl*, represents observed *intra-device delays* (excluding transmission delay), as described in chapter 5, for various packet sizes. This is similar to "Min Delay" in Fig. 3.1.

Packets are selected for service in a round-robin fashion. This is a simple but not uncommon switching fabric scheduling policy, e.g., it is used in iSLIP [76], and we plan to explore alternative approaches in our future work. Packets can be served concurrently by different servers, but packet transmissions on the same output link cannot overlap (TX times are serialized for each output port). This allows to approximate packet pipelining [13]. Fig. 4.6 demonstrates this scenario.

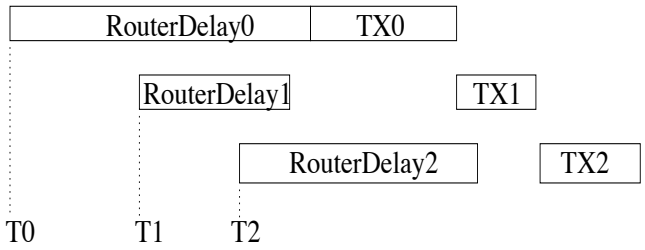


Figure 4.6. Three packets destined to the same output are concurrently served by three servers. A packet is not transmitted on the output link until the previous packet is sent out.

As packets are often split into smaller units (cells) internally within a router [74], packets may need more than one server to process them. Hence, another table, *ServReq*, gives the number of servers required to process packets of different sizes.

In cases when backplane contention results in the slow drain of the input queues in a device, queues in our model overflow causing packet drops. If the contention occurs at the output queues of a device, this contention also propagates from the servers to the queues in our model causing packet drops.

4.3.2 Parameter Inference

The five key model parameters that vary from device to another (M, Q, *DelayTbl*, *QueueReq*, *ServReq*) can be inferred experimentally by subjecting a router to a series of simple tests. Constant Bit Rate (CBR) UDP flows are injected through the router in all of the tests. Table 4.1 gives all the notation used in our parameter inference process, and Fig. 4.7 gives the pseudo-code.

The algorithm consists of four phases. In the first phase, we take an average of the packet delays across different ports when the sending rate is extremely low. This is computed for a variety of packet sizes to construct a comprehensive table. If the delay differences between the interfaces are large, the process terminates because of our assumption that interfaces are approximately similar becomes invalid. Otherwise, we record the minimum delay for each packet size.

1: Compute Packet Delays

Determine $D_{s,r,p}$ for all packet size s and rate $r = LowRate$ on path p

IF $D_{s,r,p1} \approx D_{s,r,p2}$ $\forall s \in S$, and $\forall p1, p2 \in P$

NO → EXIT

YES → $d_s = avg(D_{s,r,p1}, ..., D_{s,r,|P|})$

2: Compute Number of Servers

FOR each $s \in S$ **DO**

$\forall N$, find max. rate s.t. no loss

$ArriveGap_r = \frac{1}{Nr}$

$NumServ_s = \frac{d_s}{ArriveGap_r}$

END FOR

3: Compute Queue Sizes

FOR each $s \in S$ **DO**

find min. rate r s.t. when $N-1$ ports send to one port at rate r each, $rs(N-1) > TX\_Capacity$

$QSize_s = \frac{D_{s,r(N-1),p}}{DepartGap_{s,r(N-1),p}}$

END FOR

4: Record Parameter Values

$DelayTbl =$ set of $d_s, \forall s \in S$

$M = max(NumServ_1, ..., NumServ_S)$

$Q = min(QSize_1, ..., QSize_S)$

$ServReq_s = M/NumServ_s, \forall s \in S$

$QueueReq_s = Q/QSize_s, \forall s \in S$

Figure 4.7. Parameter inference algorithm

Table 4.1
Notation used in parameter inference

Symbol	Meaning
N	Total number of device interfaces
M	Number of servers
Q	Size of the aggregate queue per interface (we assume equivalent interfaces)
DelayTbl	Minimum processing delays for various packet sizes
QueueReq	Number of queue slots occupied by a given packet size
ServReq	Number of required servers for a given packet size
TX_Capacity	Maximum transmission capacity of an interface, measured by sending MTU-sized packet bursts
LowRate	A rate at which queuing delay does not occur but that allows sufficient (say 1000–2000) samples to be collected in a short time (e.g., 50 pps)
p	A packet path between two interfaces
P	Set of all the possible paths p; $P = N(N-1)$ is the number of all paths
S	A set of packet sizes $s = \{64, \ldots 1500\}$ bytes
R	A set of packet rates r (in packets per second), including rates that induce packet loss
$D_{s,r,p}$	Measured average packet delay from input to output for packets of size s at rate r on the path p
d_s	Minimum of $D_{s,r,p}$ values for a specific packet size s
$DepartGap_{s,r,p}$	Measured average gap between the packets when leaving the router
$ArriveGap_r$	Gap between arriving packets for rate r

In the second phase, we compute the maximum number of concurrent servers for each packet size *just before* loss starts occurring. This is done by utilizing all ports to transmit packets, such that flows do not create conflicts on the output ports. For example, suppose N is four, then flows port0-to-port1, port1-to-port0, port2-to-port3, and port3-to-port2 are non-interfering on their output. The number of servers is estimated by dividing the minimum delay observed by the gap between packets arriving into the router.

The third phase considers queue size. To estimate Q and *QueueReq*, we must create a high loss scenario to ensure queue build-up. We send flows from several ports into the output queue of another port. Note that it is important to measure the transmission capacity *TX_Capacity* and not use the nominal capacity (e.g., 100 Mbps for a FastEthernet interface) which can be higher than the actual capacity. The *DepartGap*

between the packets will indicate the maximum drain rate, meaning that the size of the queue can be estimated as the observed delay divided by *DepartGap*. We use the average observed delay here, as we noticed a few outliers that are considerably high, which skew the maximum delay. We attribute these outliers to our custom profiler which is based on commodity hardware, as discussed in chapter 5.

The fourth and final phase simply records our computed values. We record the minimum delays in *DelayTbl*. M is set to the largest number of servers estimated for all packet sizes. *ServReq* can also be constructed based on the resulting number of server estimates. The observed queue size is recorded in Q (we record the minimum size in units of maximum-sized packets) and *QueueReq* is computed for all packet sizes.

4.4 Summary

In this chapter, we described a range of routers from low–end to high–end, which are currently in use. We have then stated a set of similarities which are shared between all routers, so that we can develop a device-independent model. We have then proposed a device-independent model for forwarding devices, such as switches and routers, and outlined a model parameter inference procedure. The proposed model only requires a few parameter tables to mimic a specific packet forwarding device. The tables are compact, which makes the model highly portable and easy to compute. Our device–independent model attempts to replicate not only packet delays due to router processing and switch fabric contention, but also packet losses.

5 PROFILER ARCHITECTURE AND PERFORMANCE

The parameter inference procedure outlined in Section 4.3.2 is based on packet delay and loss data. Acquiring such data at sufficiently high precision requires a high-fidelity measurement system. We have created a custom low–cost measurement and profiling system called the Black Box Profiler (BBP), to eliminate the need for expensive specialty cards. Of course, our system has lower precision than the DAG [66] cards used in prior work, such as [15, 72]. As we will demonstrate later in this chapter, it is adequate for our purpose. Also, we created additional modules for ns-2 which can replay captured traffic data from the BBP, and can execute our device independent model to approximate a profiled router. We give a detailed overview of the BBP and additional ns-2 modules below.

5.1 Profiler Overview

Measuring packet delay through a router with micro-second precision requires speciality equipment similar to a DAG card. This is because of the fact that synchronizing clocks with micro-second precision between several measurement machines is considerably difficult to achieve. DAG cards can be GPS pulse synchronized to avoid this issue. Measurement machines add measurement noise, because some time must pass from when the packet is received to when it can be timestamped. DAG cards avoid this issue by using speciality hardware designed to reduce such overhead. However, installing a DAG card per router port is expensive and requires substantial setup time. Our goal was to build a system out of commodity parts that would be highly configurable, reasonably cheap, and offer a high degree of precision. We have created such a system called the Black Box Profiler (BBP). Fig. 5.1 demonstrates the general layout of our system. A Symmetric Multiprocessing (SMP) multi-NIC PC is used to

emulate subnets that multiple flows can traverse. The router that is being profiled is configured to route between the subnets. We have minimized the measurement error, by connecting the profiler directly connected to the router. As all packets originate and terminate in the BBP there is no need for clock synchronization, as there is only one clock in use.

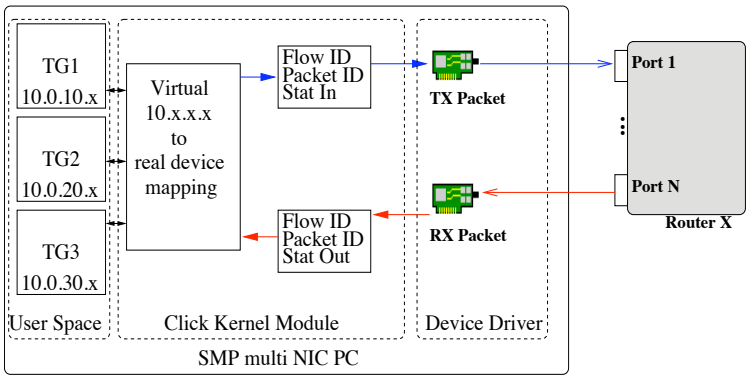


Figure 5.1. Logical view of the profiler's components.

We leverage the ns-2 simulator [8] (version 2.31) for traffic generation. We selected ns-2 because it provides several TCP implementations and application traffic models that have been validated by the research community. Further, ns-2 provides excellent capabilities for logging and debugging.

We modify ns-2 to allow packets to be injected into the test network and vice versa. As all packets originate and terminate on the BBP PC, we can embed arrival/departure time-stamps into the packet payloads with micro-second precision. The time-stamping of packets occurs in the network device driver to obtain an accurate estimate of the delay. Additionally, we can provide highly accurate accounting per-packet and per-flow to determine delay, loss, reordering, and corruption. Fig. 5.2 demonstrates an example of how a TCP flow traverses the key components of the system.

Fig. 5.3 displays the components of a measured packet delay. The measured delay is composed out of the NIC send/receive overheads, two packet transmissions, and the router delay. A calibration phase is required to infer the NIC overheads and the

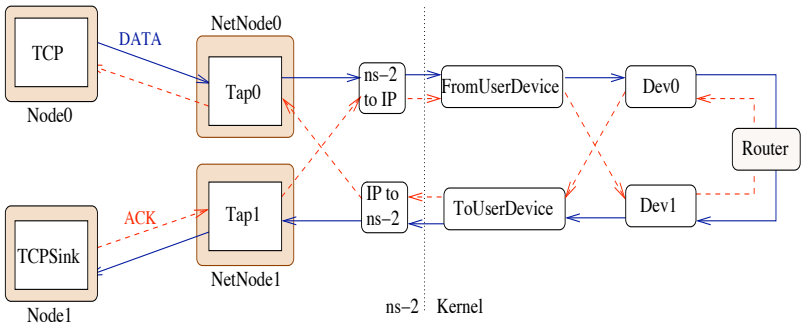


Figure 5.2. Example of a single TCP flow from the simulator into the network and vice versa.

packet transmission delay. Knowing the NIC overheads allows the computation of the router delay. During an initial calibration phase, our setup is similar to Fig. 5.3, except that the NICs are directly connected with a cable and there is no router.

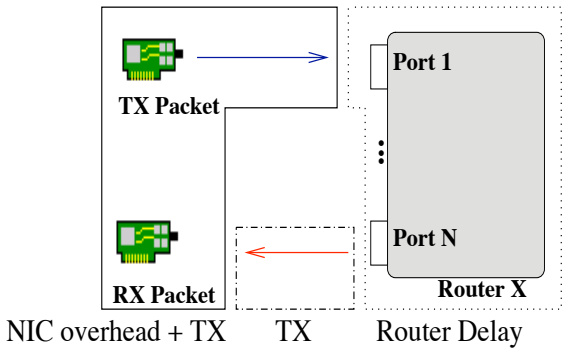


Figure 5.3. Measured packet delay consists of NIC send overhead, NIC receive overhead, router overhead, and two transmit delays.

5.1.1 Device Driver

When a packet arrives, we time-stamp it just before it is sent to the device via a bus transfer. Changing the packet payload will result in a corrupted TCP or UDP checksum, hence we recompute a new checksum. We compute incremental checksums to avoid computing an entire checksum from scratch. Fig. 5.4 demonstrates the process. Packet reception is done in a similar fashion.

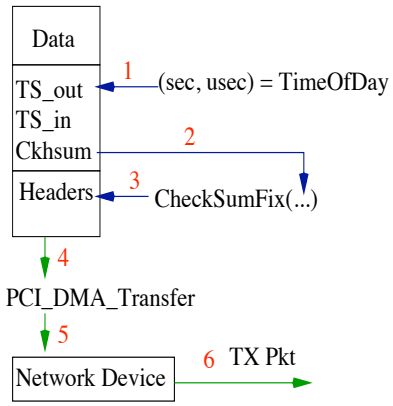


Figure 5.4. Timestamping in the device driver when sending a packet.

As previously mentioned, we have added a capability to replay our captured packet traces into ns-2. The simulator utilizes the time-stamps of when the packet departed the profiler device driver for the first time, as the time when the packet is injected into the simulation. As the packet time-stamp reflects the time when the packet entered the device driver and not the router, we add a packet transmission time to the arrival time. This re-creates the timing of when the packet entered the router or forwarding device.

During initial validation runs, we noticed that loss adversely impacts the accuracy of the trace replay. If the packet is lost, time-stamp accuracy is compromised, because the device driver time-stamp is lost as well. The only available time-stamp is the one from the ns-2 traffic generator which may be a few milliseconds behind. Hence, the packet would appear in the simulation earlier than it would have in the original experiment. This can lead to inaccuracies between the observed and predicted data, as the events do not happen at exactly the same times in both cases.

5.1.2 Click Modular Router

The default Linux IP stack was unsuitable for our purposes for two reasons. First, the default stack was not designed to efficiently handle sending/receiving non-existent

IPs to/from a user-level application. Second, the default stack has several features that we do not need, which add overhead. Hence, we use the Click modular router [42] kernel module.

In Click, packets are processed by various elements. The elements in turn can be linked together to form packet paths. When a packet passes through an element it can be modified, dropped, or the element can update its internal state. The elements pass packets to each other through *push* and *pull* connections. On a *push* connection an element will actively transmit a packet downstream. Conversely, on a *pull* connection an element will actively request packets from upstream elements. The *push* tasks are generated in elements when an unsolicited packet arrives. The *pull* requests are generated by scheduling periodic tasks. As a task has to be scheduled per *push* or *pull* request, it would be excessively inefficient to have all elements generate the tasks themselves. Hence, most of the elements operate in a passive fashion, meaning that they simply receive packets and then send them to the downstream element. This means, that when a *push* or *pull* task is scheduled, a packet might traverse a considerably long list of elements, thus increasing the computational overhead of the task. The list traversal is terminated when the final element either stores the packet (e.g., *Queue*) or destroys it (e.g., *ToDevice, Discard*). See [42] for more details. Hence, it is crucial to be careful when writing a Click configuration, if high level of performance is desired.

To attach virtual subnets of ns-2 to a particular network device, we use a *FromUserDevice* element per network card, and the user application writes IP packets into the correct *FromUserDevice* element depending on the configuration. As a user application is not aware of Click elements, *FromUserDevice* acts as a Linux character device, which can be opened and then written to. In Fig. 5.5 you can see a sample single network card Click configuration that we have used. We had two specific goals when making the BBP configuration. First, *FromUserDevice* must act as a queue and get only drained when *ToDevice* can transmit a packet. Second, the element paths

from *ToDevice* and *PollDevice* must be as short as possible to reduce the amount of processing done per task, hence minimizing measurement noise.

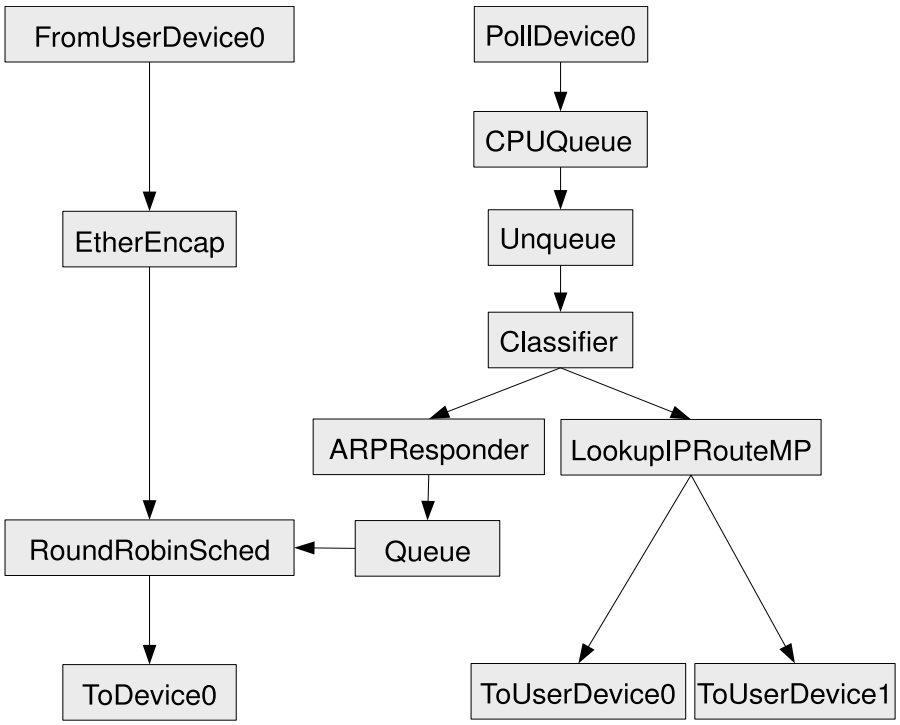


Figure 5.5. Sample Click configuration

To achieve the first goal, we have modified Click's *ToDevice* element to avoid transmit buffer drops. The default Click *ToDevice* element can schedule packets faster than the device can transmit. Instead, we *hold* the packets until the transmit buffer starts draining. We had to connect *ToDevice* to *FromUserDevice* directly, such that if *ToDevice* is unable to maintain a certain transmit rate, the buffer in *FromUserDevice* will fill up and result in a block of a user-level application. However, we could not directly connect both elements to each other. First, the packets have to have a correct Ethernet header. Second, *ToDevice* also has to transmit ARP responses in case the router being profiled generates ARP queries. Hence, we use a *RoundRobinSched* to act as a multiplexing element for IP and ARP packets.

The second goal of minimizing the packet paths was achieved, by placing a *CPUQueue* immediately after the *PollDevice*. This ensures that *push* tasks in *PollDevice* only fetch the packets from the network card and place them into the upstream queue. Unfortunately we could not reduce the element path to *ToDevice* any further, without compromising our first goal. ARP responses are rare, thus in the majority of cases the *pull* task from *ToDevice* gets forwarded from *RoundRobinSched* to *EtherEncap* and then to *FromUserDevice*. This arrangement still produces a desirable level of performance.

We use Click only in cases when extremely high packet rates are required. In these cases, no packet logging is performed, as logging over 200+ Kpps to disk is infeasible without creating a dedicated RAID disc array. To acquire statistics such as arrival rates and packet delays, we use running window averages which are accessible through Click's *proclikefs*.

5.1.3 ns-2 Modifications

We use the ns-2 simulator [8] for traffic generation as it provides several TCP implementations that have been validated by the community. Further, ns-2 provides excellent capabilities for logging and debugging. Additionally, we have modified ns-2 to replay logged traffic traces from physical experiments and incorporated our device independent model to approximate router behavior. To use ns-2, we had to make a number of changes to the simulator as follows.

Emulation The latest version of ns-2.31 [8] has an emulation package which allows outputting packets from the simulator into the network and vice versa. The default emulation objects make extensive use of system calls as well as provide packet translation capabilities from ns-2 to IP and vice versa. The packets from the network are injected into the simulator via reading sockets or by capturing packets with *libpcap*. However, the existing objects introduce two challenges. First, the performance of

libpcap is limited at high packet rates [83]. Second, it is not possible to spoof IP addresses to create an entire subnet with distinct flows on a single PC.

To tackle the performance limitations of libpcap, we have bypassed the Linux IP stack completely and created two devices that we call *FromUserDevice* and *ToUserDevice*. These devices serve as large circular buffers which allow user space applications to write packets to the kernel-level Click module and to receive packets from Click. Such direct access provides several benefits including low overhead and reception of arbitrary IP packets. In a simple test, we have been able to read packets from *ToUserDevice* at over 800 KPackets/s (Kpps). Similarly, *FromUserDevice* can sustain high rates, making the network card a potential bottleneck.

We have created our own set of emulated objects to allow IP spoofing. Fig. 5.2 shows the flow of TCP packets through our objects. As before, the ns-2 agents are connected to tap agents; however, the tap agents do not perform any ns-2 to IP or IP to ns-2 translation. Rather, these agents provide the necessary information such as IP addresses and port numbers. The actual translation is performed by the two network objects (raw-net and raw-pcap) to which all taps point. The outgoing network object converts ns-2 packets to IP and then writes them to the *FromUserDevice* device. The incoming network object reads from the *ToUserDevice* device, converts the IP packets into ns-2 format and then, based on the destination IP to tap object hash, routes the ns-2 packet to the appropriate tap object. This new arrangement makes it possible to have many flows with distinct IPs enter and depart from the simulator.

We embed the measurement payload as a TCP option. This allows creating correct TCP packets that do not have an extended payload when there should be none (e.g., SYN, ACK, FIN). As there is limited space for TCP options, our measurement payload option can only be combined with a time-stamp or a three-block SACK option. For UDP and other IP packets, the measurement payload remains in the data portion.

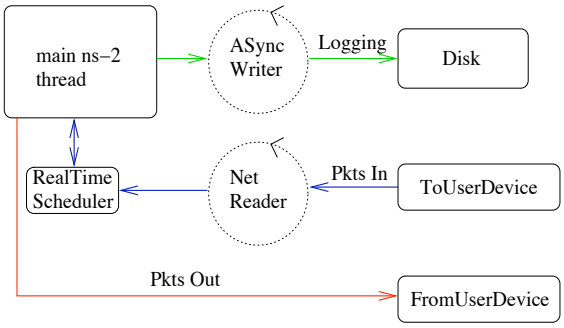


Figure 5.6. Relationship between I/O operations and threads in the simulator.

Asynchronous I/O Currently in ns-2, packet transmission and reception is performed in a synchronous fashion with the help of the TCL subsystem, resulting in less than optimal performance. Further, any logging that results in disk writes is problematic, as it can slow down the main simulation thread, thus reducing real time performance [68].

Fig. 5.6 demonstrates the architecture of asynchronous I/O that we have added to the simulator to boost real time performance. There are now three threads of execution in ns-2: (1) the main simulation thread, (2) the packet reception thread, and (3) the log writer thread. The main simulation thread is quite similar to ns-2.31 with one exception: it does not check if packets have arrived. Instead, there is a separate thread that checks if any packets have arrived and if so, injects them into the main thread. As the default ns-2 is single threaded, we took careful steps to avoid race conditions, while minimizing the number of changes we had to make. First, we modified the "Packet" class to be multi-thread (MT)-safe, as it maintains a global packet free list. Second, we made the scheduler MT-safe. These two changes allow the packet reception thread to simply schedule the newly arrived packets in the near future. When the main simulation thread dispatches the newly arrived packets, these packets are injected into the simulator.

Every tap object collects information about incoming and outgoing packets, thus we collect information about all packets. Storing this information in memory can be

cost prohibitive for long simulation runs. Hence, logging to disk is required. To avoid blocking the main simulation thread during disk writes, each tap object maintains two lists of packet data (in and out). Once a list becomes sufficiently large, the tap agent migrates the list to the disk writer thread and creates a new fresh list. The disk writer thread processes the list writes in the order in which it has received them.

Real-Time Scheduler The default real-time scheduler was inadequate for our purposes because it is based on a calendar structure and is not MT-safe. Our tests have demonstrated that the Splay scheduler provided with ns-2.31 yields a much higher insertion/deletion rate compared to the calendar or heap schedulers. High insert/delete rate is critical for maintaining high packet rates as each packet has to eventually go through the scheduler.

In addition, we have modified the real time aspect of the scheduler to remove any *sleep* calls from the main processing loop. This results in a trade-off between CPU utilization and scheduling accuracy. We have further increased the performance of the scheduler by adding a "catch-up" mode. In the catch-up mode, the scheduler will try to fulfill all the tasks that must occur "now" without invoking the *gettimeofday* system call per event. In the case when the event rate is higher than the scheduler can process, the simulation will become non-realtime as the scheduler tries to catch up. Unlike [68], we did not use the **RDTSC** assembly instruction to reduce the overhead of calling *gettimeofday*. As our machine running BBP has 8 CPUs, calling **RDTSC** could have resulted in non-monotonically increasing time-stamps.

Traffic Generator Performance Our modified ns-2, although capable of rates of more than 100 Kpps (in and out for a total of 200 Kpps), is still inadequate for extremely high load experiments because the scheduler and the TCL subsystem are limiting factors. To overcome this bottleneck, we have created a stand-alone tool called *udp_gen* which is a stripped down version of our modified ns-2. The new tool allows sending/receiving around 200 Kpps, with disk logging now being the limiting

factor. The memory footprint of our modified ns-2 is similar to that of a non-modified ns-2, according to the *top* utility.

Trace Replay and Router Model We have devised two simulation modules that mimic a forwarding device, which we integrated into ns-2.31 [8]. The first module, *RouterState*, executes the Multi-Server model, while the second module, *RouterQ*, represents the Multi-Queue model. The *RouterQ* objects are connected to the *RouterState* object as in Fig. 4.5. The *RouterQ* and *RouterState* objects initialize their capacities to the configured Q and M respectively. When a packet arrives to *RouterQ*, it is enqueued and $QueueReq_s$ is subtracted from the capacity. Conversely, when a packet departs a queue, the capacity is incremented by $QueueReq_s$. If $QueueReq_s$ is greater than the capacity during an enqueue operation, the packet is dropped. *RouterState* operates similarly. However, instead of dropping packets, it rejects them, preventing the *RouterQ* from performing a dequeue operation. Once *RouterState* finishes serving a packet, it pulls packets from one of the upstream queues until its capacity is filled. As there is only one *RouterState* per node, it approximates backplane contention by preventing queues from dequeuing packets, even when the output link is free.

We also create trace agents to replay captured packet traces and compare the delays and losses with the simulator against the observed delays and losses. We utilize the same traffic generation scripts when running simulations (utilizing our model or the default ns-2 model) and physical experiments.

5.2 Profiler Configuration

The profiler must allow several tasks to execute concurrently to achieve the highest precision. We use a PC with two quad 1.8 GHz Xeon CPUs and PCI-E Intel Pro cards to run our profiling system. Fig. 5.7 demonstrates the main tasks that must run concurrently for precise results.

The traffic generation component must have at least two threads of concurrent execution to achieve high packet rates. The main ns-2 thread runs all of the simula-

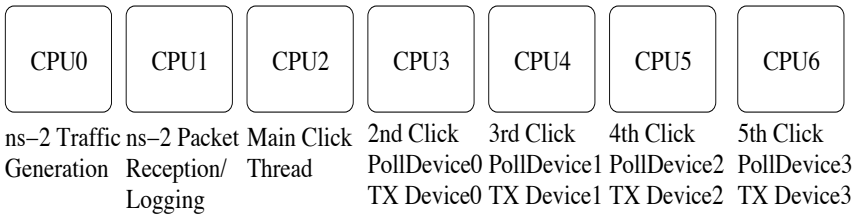


Figure 5.7. Concurrent threads of execution

tion agents and writes packets to *FromUserDevice* elements. Auxiliary ns-2 threads are required for reception of packets and logging data to disk; otherwise, the main simulation thread becomes I/O blocked. It is possible to run *several simulations at once*, by setting routing policies which forward packets into appropriate simulation instances. We use this feature for scenarios when a single simulation instance cannot generate sufficient load.

The Click modular router must also have at least two threads of concurrent execution. The main Click thread is responsible for all the elements in the Click configuration except for packet reception and transmission. If the elements in the main thread are delayed in scheduling, no measurement error will occur. This is because the main elements are not responsible for reading/writing packet timestamps. A problem arises during packet reception and transmission. If there is delay in element scheduling, then the packets will remain in the NIC's send/receive buffer as time goes on. The variance in delay would increase in proportion to the packet rate increase. Hence, it is imperative to have a separate thread per *PollDevice/ToDevice*. As we have four ports in our experiments, we need seven CPUs as depicted in Fig 5.7. We will later demonstrate how the last CPU can be used to execute an additional ns-2 simulation to increase the aggregate load.

5.3 Profiler Calibration

Before proceeding with measurements, we must determine which network device configuration gives the best performance and induces the least amount of noise

into the measurements. This measurement noise results from the network card/bus specifics of our measurement machine. Further, we must determine how fast our logging system performs, as this is crucial for Gigabit speeds. We produced the best results with polling and 64-slot receive and transmit buffers. Figs. 5.8 and 5.9 demonstrate the measured delay between the two NICs compared to pure transmission delay using a transmit speed of 100 Mbps and 1 Gbps. We used a constant low-rate flow of UDP packets to generate the results. In both the 100 Mbps and 1 Gbps scenarios, the difference between the measured delay and a pure packet transmission delay is never more than 14 micro-seconds.

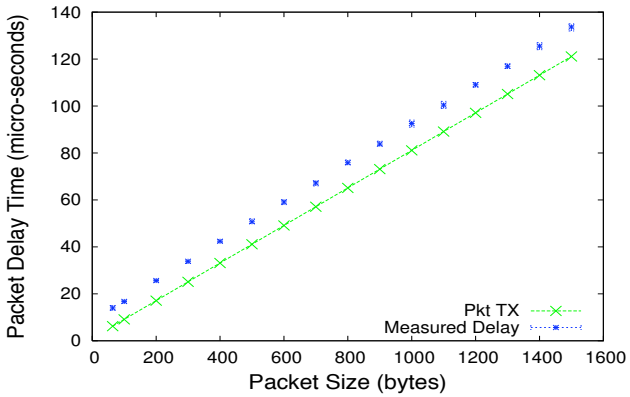


Figure 5.8. NIC–to–NIC (mean, 5 and 95 percentiles) vs. pure 100 Mbps TX delay.

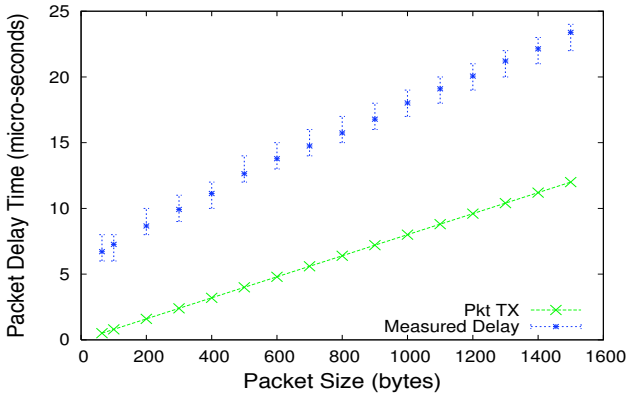


Figure 5.9. NIC–to–NIC (mean, 5 and 95 percentiles) vs. pure 1 Gbps TX delay.

Tables 5.1 and 5.2 demonstrate the mean, 5th, and 95th percentiles for Ethernet frames of sizes of 64, 800, and 1500 at various rates for 100 Mbps and 1 Gbps speeds. We have used our *udp_gen* tool to conduct these measurements over a period of one minute per experiment. The percentiles indicate that variance is relatively small and that our logging system is adequate.

Table 5.1
100 Mbps NIC–to–NIC packet delays for 64-, 800- and 1500-byte Ethernet frames

	64 bytes		800 bytes		1500 bytes	
	4 Kpps	140 Kpps	4 Kpps	14 Kpps	4 Kpps	8 Kpps
mean	13.05	20.42	74.74	77.44	133.13	133.33
5th	13.00	12.00	74.00	74.00	133.00	133.00
95th	14.00	16.00	75.00	76.00	134.00	134.00

Table 5.2
1 Gbps NIC–to–NIC packet delays for 64-, 800- and 1500-byte Ethernet frames

	64 bytes		800 bytes		1500 bytes	
	4 Kpps	200 Kpps	4 Kpps	140 Kpps	4 Kpps	80 Kpps
mean	6.05	7.87	14.94	16.37	22.34	24.73
5th	6.00	6.00	14.00	15.00	22.00	22.00
95th	7.00	16.00	16.00	18.00	23.00	24.00

5.4 Summary

In this chapter, we have described the architecture of our profiling and traffic generation tool, called the Black Box Profiler (BBP). We have leveraged the advances in commodity technology to build a tool that does not rely on specialty hardware, yet offers micro-second level precision. The tool was created by bridging ns-2 with the Click modular router to create a highly configurable traffic generator and logger. We have created agents for ns-2 that can replay previous physical experiments, and can use the model and router specific parameters to approximate a profiled router. The calibration steps that we have undertaken revealed that the system offers an

acceptable level of performance for our purpose with a minimum level of measurement noise. Also, the calibration revealed that BBP is capable of generating high–speed traffic at 100 Mbps and 1 Gbps speeds.

6 ROUTER PROFILING AND VALIDATION RESULTS

This chapter uses the BBP tool developed in chapter 5 and the model from Section 4.3 to obtain model parameters for four routers. We have selected Cisco 3660, Cisco 7206VXR, Cisco 12410, and Juniper M7i routers to represent a variety of routers that are currently found on the Internet. We then compare the observed delays to the delays predicted by our model and the default ns-2 queueing model for a variety of complex traffic scenarios.

6.1 Experimental Setup

We select four router types for a representative cross-section of performance, and a wide range of switching fabrics. The routers range from those for a medium office to Internet carrier grade. The router specifications are as follows:

1. Cisco 3660 with 100 Mbps FastEthernet modules: Multi-service platform for large branch-office multi-service networking; Interrupt-based packet switching on a 225 MHz RISC QED RM5271 CPU, capable of 140-Kpps fast switching; Supports an extensive variety of modules for various media (e.g., Ethernet, FastEthernet, ISDN, T1) [52].

2. Cisco 7206VXR with 100 Mbps FastEthernet modules: Edge device used by enterprises and service providers for services aggregation WAN/MAN; Interrupt-based packet switching on a 262 MHz NPE-300 processor; Supports an extensive variety of high-speed modules for various media (e.g., Ethernet, FastEthernet, ATM) [53].

3. Cisco 12410 chassis with a 4 port 4GE-SFP-LC line card: This router is geared towards carrier IP/MPLS core and edge networks with 23000 units sold; The

router is equipped with multi-gigabit cross-bar switch fabric and each line card runs IOS and uses CEF [51]; The 4GE-SFP-LC card is limited to 4 Mpps and runs Engine 3 (Internet Services Engine); The card has 512 MB of packet memory and splits it in two for TX and RX buffers.

4. Juniper M7i with four SFP Gigabit modules: A multi-service edge device that can be used as an Internet gateway, WAN, campus core, regional backbone, and data center router; Forwarding engine of 4.2 Gbps at full duplex; Internet processor II-based Application-Specific Integrated Circuit (ASIC) for 16-Mbps packet lookup [75].

The lower-end routers (Cisco 3600 and Cisco 7206VXR) we use have *four Fast Ethernet ports*. The Cisco 3660 has two identical cards on the main data plane and a dual port swappable module, while the Cisco 7206VXR has one main card and three swappable modules. On the Juniper, three cards are swappable and the fourth is integrated into the chassis. For the higher-end routers (Cisco 12410 and Juniper M7i) we configured our profiler to use our network cards for *Gigabit* speed.

Fig. 6.1 demonstrates our test setup. In the experiments, *Node0*, *Node1*, *Node2*, and *Node3* are logical nodes on the same PC, while the "Router" node is either a pair of cross-over cables that connect four cards on our profiling system, a Cisco router, or the Juniper router. The cross-over cable configuration is used solely for calibration, to determine the latencies as a result of the network cards. Each logical node has its own dedicated subnet as shown in Fig. 6.1. In the remainder of this paper, we adopt the following naming convention: *portX on the router*, denotes the port connected to *NodeX*.

All routers are configured with minimal settings to ensure that forwarding between the ports occurs on a *fast path* without special processing. Further, we enable the Cisco Express Forwarding (CEF) option [51] on the Cisco 3660 and Cisco 7206VXR as it was not enabled by default. The queue size for all the links in the traffic

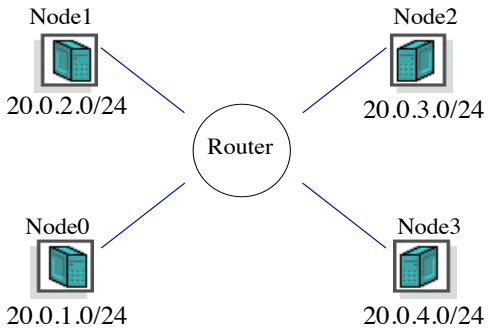
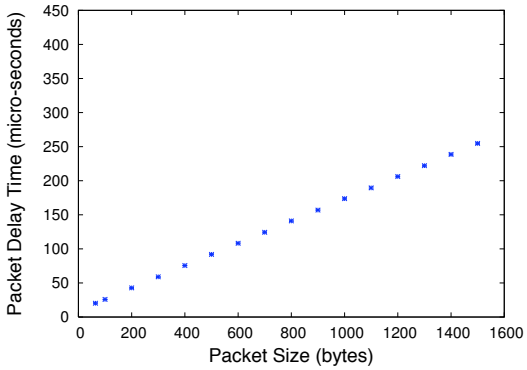


Figure 6.1. Test topology with four subnets

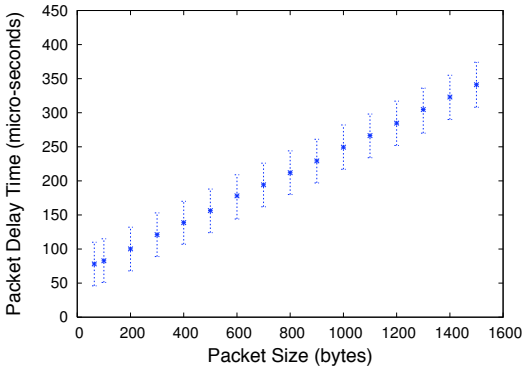
generator was set to 100 slots. The queue sizes on the physical link are dictated by the particulars of the hardware.

6.2 Model Parameters

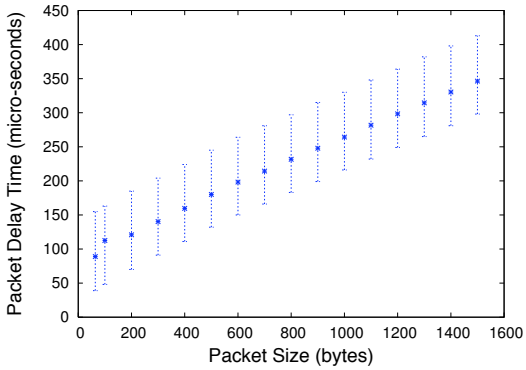
As discussed in Section 4.3.2, we first compute delay tables. We vary the packet size from 64 to 1500 bytes, and keep the rate at a low 50 packets/s. The packet size includes Ethernet/IP/UDP headers. Fig. 6.2 and 6.3 compare the results for a *100 Mbps perfect router*, Cisco 3660, Cisco 7206VXR, *1 Gbps perfect router*, Cisco 12410, and Juniper M7i. The results show the mean, 5, and 95 percentiles. The *perfect router* is a hypothetical router that has zero processing and queuing latency, with packet transmission time being the only delay. We use the data from Fig. 5.8 and Fig. 5.9 to obtain the results for the perfect routers: we add the NIC overhead to one additional packet transmit time. The results indicate that the Cisco 3660 and Cisco 7206VXR routers have significantly higher delays and variance than the 100 Mbps perfect router. The Cisco 12410 showed little variance in delay but added per packet delay over the 1 Gbps perfect router. In contrast, the Juniper M7i exhibited a high degree of variance in measured delays. This can be attributed to the fact that the 4th port of that router was integrated into the chassis and had different delay values than the other ports.



(a) Perfect 100 Mbps Router



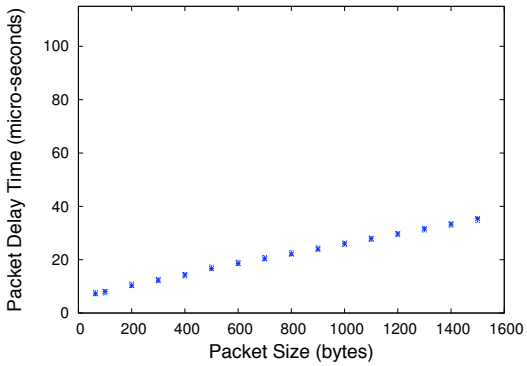
(b) Cisco 3660



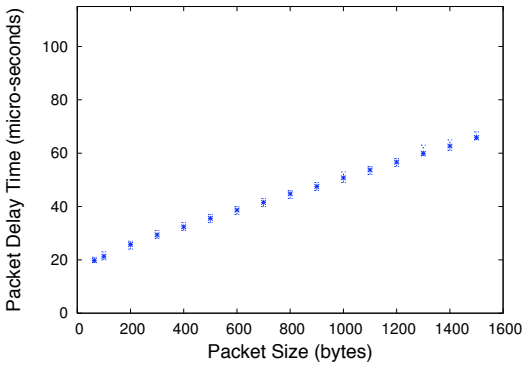
(c) Cisco 7206VXR

Figure 6.2. Observed minimum delays for different packet sizes at 100 Mbps

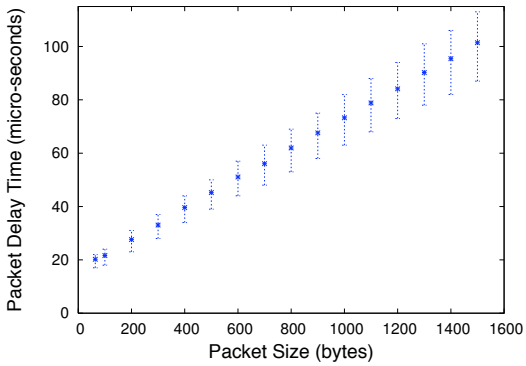
We now use Click to generate the traffic (as discussed in Section 5) to infer the number of servers and queue sizes. It was necessary during this phase to generate rates of 700+ Kpps per interface, to stress the Gigabit routers. For the Cisco 12410 and Juniper M7i, the maximum achievable transmission rate was 986.82 Mbps; hence we set their *TX_Capacity* in algorithm in Fig. 4.7 to 986.82 Mbps. Similarly, the *TX_Capacity* was set to 98.718 Mbps for the Cisco 3660 and 7206VXR. We use these same values as link capacities in all our simulations (with our new modules and with the default ns-2) for a fair comparison.



(a) Perfect 1 Gbps Router



(b) Cisco 3660



(c) Cisco 7206VXR

Figure 6.3. Observed minimum delays for different packet sizes at 1 Gbps

Table 6.1 and Table 6.2 demonstrate the measured values for *QSize* and *NumServ* respectively. The values of M, Q, *ServReq*, and *QueueReq* for the routers are computed from these tables, as given in **Phase 4** of algorithm in Fig. 4.7.

Analysis of Table 6.1 reveals that there are *three different queue sizing strategies*:

1. The Cisco 3660 and Cisco 7206VXR have 3 queue sizes (in terms of number of packets) for this set of packet sizes. This is consistent with the documentation in [84].

Table 6.1
Queue sizes for different packet sizes

Router	64	200	600	800	1000	1400	1500
12410	33654	32891	32712	32706	32647	32159	31583
M7i	249875	72488	27815	21129	16930	14001	9473
3660	1909	674	167	167	167	167	167
7206VXR	272	294	167	167	167	125	125

Table 6.2
Number of servers for different packet sizes

Router	64	200	600	800	1000	1400	1500
12410	36.379	29.115	11.843	10.171	9.006	8.243	7.265
M7i	37.145	39.382	23.866	23.176	22.584	22.101	21.455
3660	6.027	6.080	4.653	3.969	3.389	3.126	2.786
7206VXR	15.489	11.648	5.464	4.318	3.463	2.969	2.439

2. The Cisco 12410 appears to have a slot-based queue which is approximately 32 K packet slots in size. As the router has 4 ports, $4 \times 32\ K \times 1500 \approx 200\ MB$ which is close the line card's 256 MB TX buffer size.

3. The Juniper M7i has a byte-based queue of approximately 16 MB ((packet size-18) $\times QSize \approx 16MB$, as the 18-byte Ethernet header/footer is not queued). This is consistent with its specification, which states that the router is capable of 200 ms buffering [75].

Table 6.2 shows that for most packet sizes, the Juniper router has the highest number of servers, followed by the Cisco 12410, the Cisco 7206VXR, and finally the Cisco 3660. However, the Juniper and Cisco 12410 support comparable rates as a result of differences in *DelayTbl*s. As expected, the number of servers monotonically decreases as packet size increases for each router.

6.3 Model Fidelity

In this section, we compare the performance of our model, the default ns-2, and the observed data under a variety of experimental scenarios. We first conduct the experiment with the physical router and capture packet traces. The packet traces are then fed into the simulator to analyze the accuracy of the two simulation models.

6.3.1 CBR Flows

In our first series of experiments, we replay the simple CBR traces used in the model inference experiments. We use our new ns-2 modules to model the Cisco 3660, Cisco 7206VXR, Cisco 12410, and Juniper M7i routers, and compare packet delay and loss values.

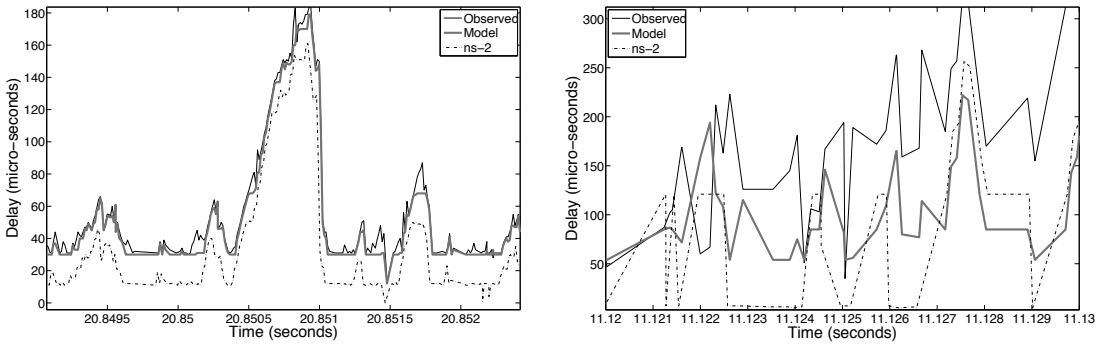
The results indicate that the model can account for backplane contention: the model correctly predicts that the Juniper M7i cannot have all four interfaces forwarding 64–byte frames at more than 715000 pps[1]. We verified the router statistics to ensure that our network cards were not dropping packets upon receiving. Additionally, when multiplexing two flows into a single output port, the model correctly predicts the packet delays because of queue build-ups. The data also confirmed the need for *QueueReq*, as fixed-size slot-based queues are insufficient for correctly modeling the routers, because the two routers had different sized buffer pools depending on packet size and one router used a byte–based queue.

6.3.2 Low-load TCP

In the next series of experiments, we create a medium level of load that does not result in any packet loss, so that queue sizing would not be a factor when replaying the packet traces.

[1] $\frac{d_s}{NumServ_s} \times N = \frac{12.997\mu s}{37.145} \times 4 = 1.397\mu s$
$\frac{1}{1.397\mu s} \times 1e6\frac{\mu s}{s} = 714490$ pps

We use *FullTCP SACK* agents in ns-2. *FullTCP* agents in ns-2 mimic the Reno TCP implementation in BSD 4.4. The links from *Node0*, *Node1*, *Node2*, and *Node3* in Fig. 6.1 were configured to have delays of 20 ms, 30 ms, 40 ms, and 90 ms respectively. For the two Gigabit routers, the speed is 1 Gbps, while for the Cisco 3660 and 7206VXR we set the speed to 100 Mbps. We also limit the bandwidth of TCP to 70 Mbps on each port in the 3660 and 7206VXR experiments. This avoids packet losses.



(a) Cisco 12410 (b) Cisco 3660

Figure 6.4. Low-load TCP: Delays on port2

We create long-lived FTP TCP connection pairs, such that each node has 7 TCP flows to three other nodes for a total of 84 TCP flows. TCP agents are configured to use up to 3 SACK blocks, 1420-byte[2] payloads, and delayed ACKs. Additionally, in the case of Gigabit routers, each node sends two 1 Kpps 1500-byte UDP flows to the other nodes for a total of 24 UDP flows. This was done to inflate bandwidth usage, as our single simulation cannot generate more than 90 TCP flows over Gigabit links in real-time. Certainly, this traffic is not representative of real workloads, but it is capable of generating low-to-medium load and can reveal backplane packet interac-

[2]The 1420-byte payload was chosen so that packets with 3 SACK blocks would not exceed 1518 bytes when all the headers/footers are included.

tions if they exist. The experiment duration was 180 seconds, during which we inject traffic into the router and log all transmitted and received packets.

Table 6.3
Low-load TCP: Mean and COV of packet delays for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR

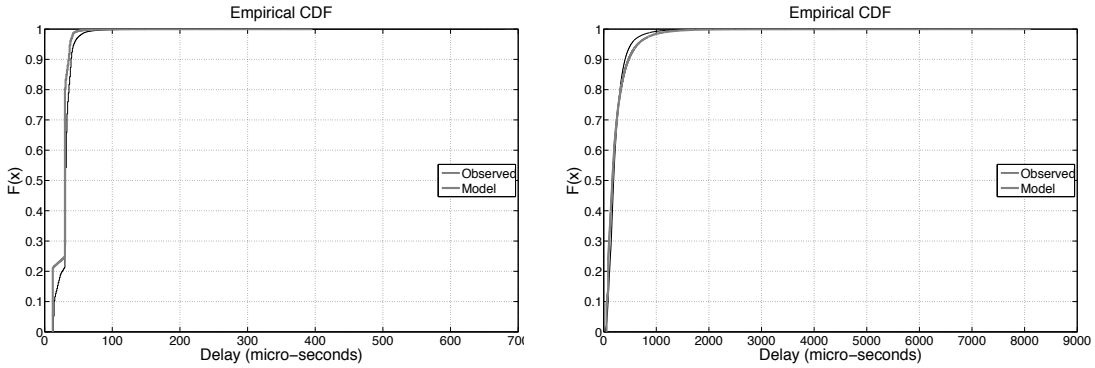
		12410		M7i		3660		7206VXR	
Destination Node	Type	Mean	COV	Mean	COV	Mean	COV	mean	COV
Node0	Model	29.641	0.398	59.371	0.389	535.367	1.209	1070.216	1.231
	Observed	33.038	0.416	67.512	0.393	586.792	1.012	705.387	1.013
Node1	Model	28.115	0.387	57.274	0.393	436.450	1.178	596.630	1.272
	Observed	31.841	0.519	66.045	0.401	362.597	1.123	409.618	1.264
Node2	Model	27.205	0.356	57.738	0.358	234.572	1.066	255.061	1.139
	Observed	30.949	0.354	67.356	0.366	233.135	0.866	193.782	0.958
Node3	Model	28.574	0.433	59.375	0.397	166.958	0.791	183.325	1.023
	Observed	32.264	0.413	52.297	0.394	194.975	0.652	169.927	0.914

Table 6.4
Low-load TCP: Kolmogorov-Smirnov statistic

Destination Node	12410	M7i	3660	7206VXR
Node0	0.365	0.482	0.254	0.150
Node1	0.431	0.520	0.072	0.106
Node2	0.518	0.517	0.133	0.125
Node3	0.453	0.599	0.211	0.136

After completing the experiments, we replayed the traces through our router module and through ns-2 with a large queue of size 200 packets, as we wanted to exclude queue size as a factor in this experiment. Fig. 6.4 demonstrates the detailed results on the Cisco 12410 and 3660 routers over a period of 3.3 ms and 10 ms respectively. We can make two interesting observations. First, adding packet minimum delays improves the predictions over the default ns-2 model which does not consider any processing delay. Second, backplane contention on the 3660 results in a significant level of additional delay. This is also evident in the 7206VXR results in Table 6.3.

Our model attempts to mimic this by relying on the Multi-Server model, but, as the figure shows, it is not always accurate at this high precision (micro-second level), possibly because of our commodity-hardware profiler.



(a) Cisco 12410 (b) Cisco 3660

Figure 6.5. Low-load TCP: CDF plots for port2

Table 6.3 gives the mean (in μs) and Coefficient of Variation (COV) per output port for our model and actual measurements. The COV is a normalized measure of variance in packet delays. For the Cisco 3660 and 7206VXR, the COV values are quite high compared to the Gigabit routers. Overall, the means and the COVs in the table show a satisfactory correspondence. For further analysis of the the model accuracy, we use the Kolmogorov-Smirnov (K-S) statistical test (maximum deviation of CDFs). Table 6.4 demonstrates the results. Although some of the values are greater than 0.5, the CDFs appear to be comparable as Fig. 6.5 demonstrates. Overall, the Multi-Server/Multi-Queue Model shows higher accuracy than the single output queue model.

6.3.3 High-load TCP

In the next set of experiments, we increase the load to induce losses, so that knowledge of queue sizes would be imperative for accurate predictions.

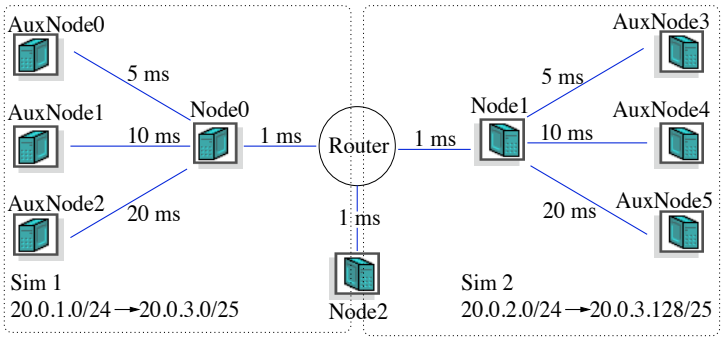


Figure 6.6. High-load TCP topology

Fig. 6.6 demonstrates the topology that we use. We create auxiliary nodes and links to avoid TCP synchronization using three different RTTs of 14 ms, 24 ms, and 44 ms. We induce sufficient load on the Gigabit routers, by executing two separate simulation instances with 28 TCP agents per auxiliary node, for a total of 84 TCP agents per simulation, hence producing 168 TCP in total. As BBP has 8 CPUs and running at least one simulation requires 7 (recall Fig. 5.7), executing two simulation at once can be done in real time.

The TCP agents were driven by infinite source FTP agents. This setup may appear unusual because there are two links to *Node2* (one per simulator). However, as all of the packets will pass through the router, no more than *TX_Capacity* can traverse the links.

This arrangement is achieved, by splitting the address space of *Node2* into two 128 CIDR address blocks. Fig. 6.6 demonstrates that agents from *AuxNode0*, *AuxNode1*, and *AuxNode2* use the lower 128 addresses on *Node2* and that agents from *AuxNode3*, *AuxNode4*, and *AuxNode5* use the upper 128 addresses on *Node2*. Each auxiliary node has 5 UDP flows sending 1500-byte packet at 1500 pps to *Node2*. Finally, to create reverse traffic, we configure 6 UDP flows to send 1500-byte packets at 1500 pps each from *Node2* to *Node0* (to *Node1* in "Sim 2"). For the Cisco 3660 and 7206VXR, we scale down the link speed to 100 Mbps (as before) and also reduce the rate of all

the UDP flows by 10. The experiment duration was 180 seconds as in the previous experiment.

With the unmodified ns-2 forwarding model, we use the *default* 50-slot-sized ns-2 queue, as well as a larger 20000-slot-sized ns-2 queue in the case of Gigabit routers. Table 6.5 demonstrates the loss ratios for the congested output port2. The data indicates that our model provided an accurate approximation of the loss ratios. The default ns-2 queue results for the Gigabit routers are inaccurate for both queue sizes; ns-2 gives accurate loss estimates for the two lower-end routers.

Table 6.5
High-load TCP: Average loss ratios

Router	Observed	Model	ns-2 50	ns-2 20000
Cisco 12410	0.0077	0.0078	0.0342	0.0099
Juniper M7i	0.0175	0.0161	0.0353	0.0125
Cisco 3660	0.1031	0.1028	0.1031	N/A
Cisco 7206VXR	0.0960	0.0953	0.0954	N/A

Table 6.6 compares the means (in μs) and the COV values for the routers for all ports used in the experiment. The results indicate that our model performs well in this high-load scenario. The results further show that our profiling system can successfully scale to Gigabit speeds. It is interesting to note that the COV values for the congested port become considerably small on the Cisco 3660 and Cisco 7206VXR. This is because traffic starts to resemble a single CBR flow, and most packets experience maximum queuing delay, resulting in a decrease of delay variation.

The results from Table 6.7 also confirm that the model performs well especially for the congested ports. The K-S value of 0.292 for Juniper M7i at the $Node2$ output is caused by failing to capture the router behavior in the parameter table. Fig. 6.7 demonstrates the cause of the problem.

In Fig. 6.7(a), our model for Cisco 12410 is accurate, and the average absolute difference is 1492 μs. The plot in Fig. 6.7(b) indicates that our profiling method

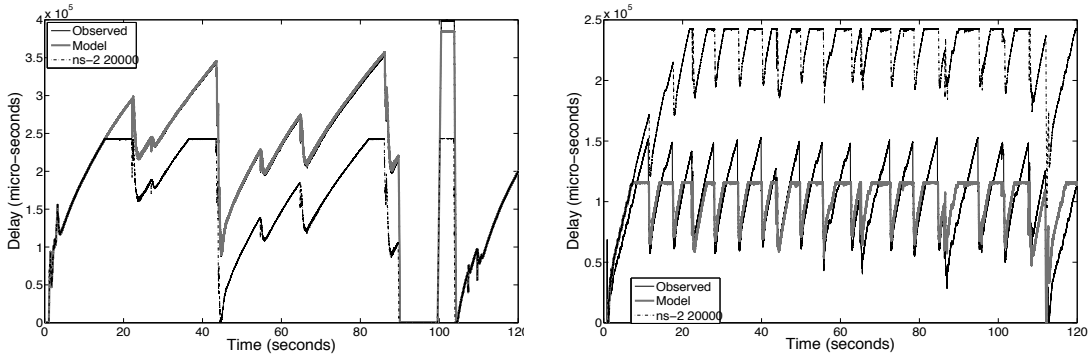
Table 6.6
High-load TCP: Mean and COV packet delays for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR for three destination (Dst) nodes

		12410		M7i		3660		7206VXR	
Dst Node	Type	Mean	COV	Mean	COV	Mean	COV	mean	COV
Node0	Model	30.784	0.470	55.368	0.539	159.044	0.535	132.895	0.559
	Observed	47.279	0.349	74.259	0.470	305.318	0.856	186.287	0.513
Node1	Model	28.153	0.474	59.312	0.515	162.830	0.432	131.163	0.575
	Observed	32.067	0.406	62.524	0.823	243.743	0.547	123.390	0.662
Node2	Model	221454.186	0.387	90374.424	0.402	19823.226	0.043	14609.008	0.047
	Observed	219962.721	0.389	86022.799	0.507	19435.527	0.044	14427.739	0.048

failed to reveal that the M7i router can sometimes delay packets longer than our CBR parameter inference test indicated. The model delays do not exceed 140 K (μs), whereas the delays on the router continue to increase. At first, it may appear that we have simply underestimated the router queue size. However, simulations with larger queues do not resolve the problem, as then the queues do not drain as fast as they do in our observations. This implies that the extra delay is router-induced and not because of additional queuing delay. For both Gigabit routers, the ns-2 predictions with a queue of 50 slots are exceedingly close to the x-axis as the queue cannot have substantial queuing delay. Results with 20000-slot queues are closer. However, the ns-2 results are not entirely correct as they underestimate the delay for the Cisco 12410 and overestimate the delay for the Juniper M7i.

Table 6.7
High-load TCP Kolmogorov-Smirnov statistic

Dst Node	12410	M7i	3660	7206VXR
$Node0$	0.569	0.519	0.388	0.342
$Node1$	0.294	0.241	0.496	0.226
$Node2$	0.019	0.292	0.397	0.268



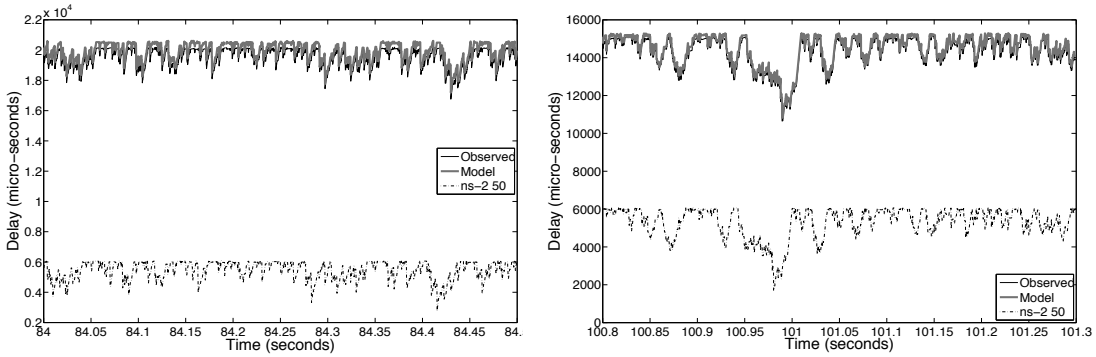
(a) Cisco 12410 (b) Juniper M7i

Figure 6.7. High-load TCP: Delays on port2

The Cisco 3660 and 7206VXR models match the observations well. Interestingly, the data in Table 6.6 for non-congested ports indicates an increase in variance compared to what our model predicts. This is not surprising as both routers reported non-negligible CPU usage. Cisco 3660 reported a 41% CPU utilization while the 7206VXR reported a 30% CPU utilization averaged over a one minute period. As these routers use interrupt driven packet processing (see Section 6.1), they are prone to backplane contention and interference even at moderate loads. Fig. 6.8(a) and Fig. 6.8(b) provide a snapshot of 500 ms and reveal that our model follows the actual data quite closely. The ns-2 model with a 50-slot queue underestimates the delay, but the shapes of the curves are not appreciably different from our model and measurements.

6.3.4 High-load TCP and HTTP

In our previous experiments, we have used purely synthetic traffic and did not induce extremely heavy loads on all router ports. Sommers *et al.* [67] suggest that routers experience higher loads under realistic traffic, compared to synthetic traffic. Hence, we now use the PackMIME HTTP ns-2 module [85]. We made a few



(a) Cisco 3660 (b) Cisco 7206VXR

Figure 6.8. High-load TCP: Delays on port2

modifications to the code to interface the TCP agents with our tap objects (recall Fig 5.2).

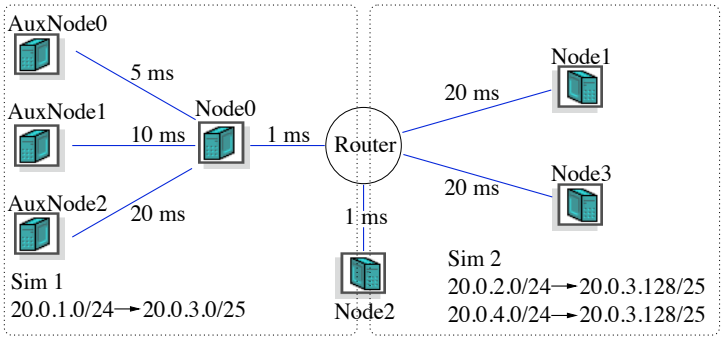


Figure 6.9. High-load TCP and HTTP topology

As in the previous experiment, we ran two simulator instances to create sufficient load for the Gigabit routers. "Sim 2" in Fig. 6.9 is configured to run server and client HTTP PackMIME clouds. The two client clouds are positioned at *Node2* to force server responses to multiplex on the router port2. Each client-to-server cloud pair is configured to generate 60 new connections per second for a total of 120 connections per second. Additionally, we scale up the server response sizes by a factor of 10. This is necessary to create congestion. This scenario can be representative

Table 6.8
High-load TCP and HTTP: Mean and COV of packet delays for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR for four destination (Dst) nodes

		12410		M7i		3660		7206VXR	
Dst Node	Type	Mean	COV	Mean	COV	Mean	COV	mean	COV
Node0	Model	14.166	0.561	17.808	0.269	199.422	0.734	151.114	0.929
	Observed	27.662	0.584	22.226	0.099	431.822	1.517	181.312	0.651
Node1	Model	15.525	0.371	19.295	0.448	133.492	0.588	122.855	0.624
	Observed	18.906	0.248	16.683	0.384	152.611	1.247	185.624	0.816
Node2	Model	287511.650	0.317	106642.239	0.213	19613.124	0.100	14333.229	0.121
	Observed	295902.674	0.298	113576.314	0.268	21682.739	0.167	19540.178	0.183
Node3	Model	15.801	0.394	19.228	0.451	137.552	0.675	128.885	0.737
	Observed	18.617	0.255	12.217	0.391	186.934	1.236	190.487	0.779

of a large campus population downloading considerably image-intensive web pages. Unfortunately, we were unable to use delay-boxes [85] to model the RTTs of the flows because of performance reasons. Hence, we only set the delays on the links. To add load, we create another simulator instance that runs 84 TCP flows driven by FTP agents from auxiliary nodes to *Node2*. Additionally, the auxiliary nodes send 26 UDP flows to *Node2*. The UDP flows amount to 0.5 Gbps to induce heavy load on the congested port. These TCP and UDP flows represent large file downloads and streaming services. For instance, 4300 people listening to 128 Kbps radio broadcasts can use up 0.5 Gbps.

In the case of the Cisco 3660 and Cisco 7260VXR, we use different parameters, as otherwise, the load would be excessive for 100 Mbps. First, as before, we scale down the link speeds to 100 Mbps. Second, we configure the HTTP traffic simulation to use 30 connections per second per client-server cloud, for a total of 60 connections per second. We also remove the server response scaling factor. We remove UDP traffic from the auxiliary nodes to *Node2*, but we create 9 UDP flows of 1500-byte packets at 150 pps from *Node2* to *Node1*.

We set the random seeds in experiments with all routers to the same values. The seeds are required by the random streams when creating request/response sizes as well as server and client "thinking" delays. The experiment duration is set to 180 seconds.

Table 6.9
High-load TCP and HTTP: Average loss ratios

Router	Observed	Model	ns-2 50	ns-2 20000
Cisco 12410	0.0033	0.0021	0.0388	0.0028
Juniper M7i	0.0188	0.0183	0.0402	0.0178
Cisco 3660	0.0528	0.0456	0.0591	N/A
Cisco 7206VXR	0.0615	0.0523	0.0617	N/A

Table 6.9 gives the loss ratios. Our model performs well across all scenarios, especially for the Gigabit routers where it is significantly more accurate than ns-2 with a 50 slot queue. Results of ns-2 with a 20000-slot queue are accurate. Although not shown in the paper, almost half of the packets from *Node1* and *Node3* were lost on the Cisco 3660 and 7206VXR. We suspect this was the result of highly bursty traffic from these nodes, which is consistent with the analysis in [72]. Table 6.8 indicates that our model gives a reasonable match for the mean and COV values; however, the results are not as accurate as in the high-load TCP scenario. This is attributed to backplane interactions which affected the results, except on the Cisco 12410. As in the previous experiment, the COV values for the Cisco 3660 and 7206VXR on the heavily congested port are considerably low.

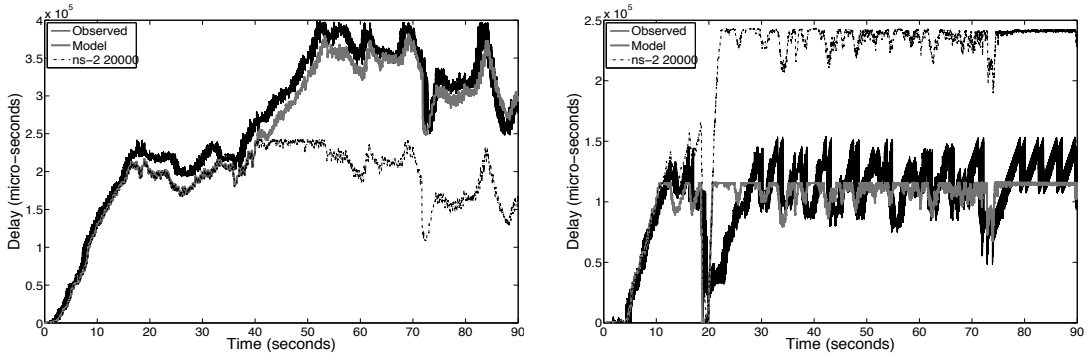
Table 6.10 summarizes the K-S statistics for the routers. Even though, some of the K-S values are as big as 0.909, the CDF in general appear appreciably similar to each other. The CDF plots appear similar to plots in Fig. 6.5 where there is a narrow range of x–values where the discrepancy between the CDFs is large.

Fig. 6.10 depicts the delay data for Cisco 12410 and Juniper M7i for the congested port. Our model follows the observed data quite well. On the Juniper, the model

Table 6.10
High-load TCP and HTTP Kolmogorov-Smirnov statistic

Destination Node	12410	M7i	3660	7206VXR
$Node0$	0.909	0.702	0.274	0.251
$Node1$	0.526	0.375	0.214	0.242
$Node2$	0.145	0.292	0.500	0.830
$Node3$	0.474	0.852	0.128	0.261

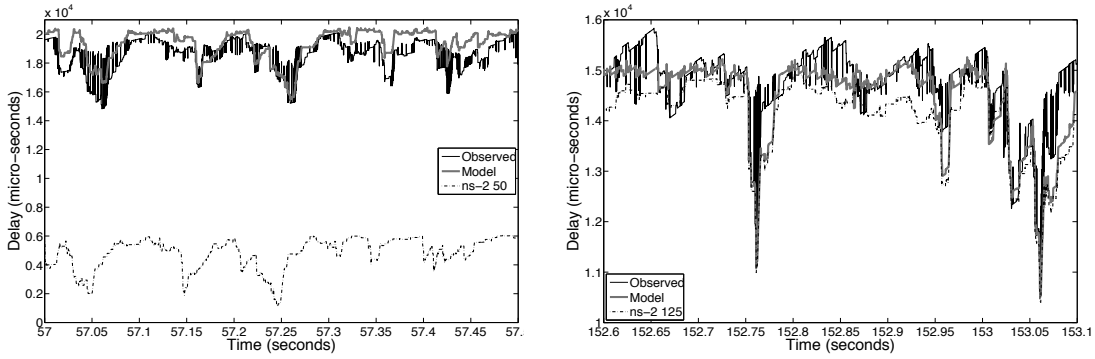
predictions are not as not as precise because some packets experience a much higher delay than the model predicts. This is because our parameter estimates are not entirely accurate, and the router experiences additional delay as a result of heavy load. For both routers, the ns-2 predictions with a 50-slot default queue are close to the x-axis as the queue cannot introduce substantial queuing delay. Although loss ratios for ns-2 with a 20000-slot queue are similar to the observations (Table 6.9), packet delays for ns-2 are quite different.



(a) Cisco 12410 (b) Juniper M7i

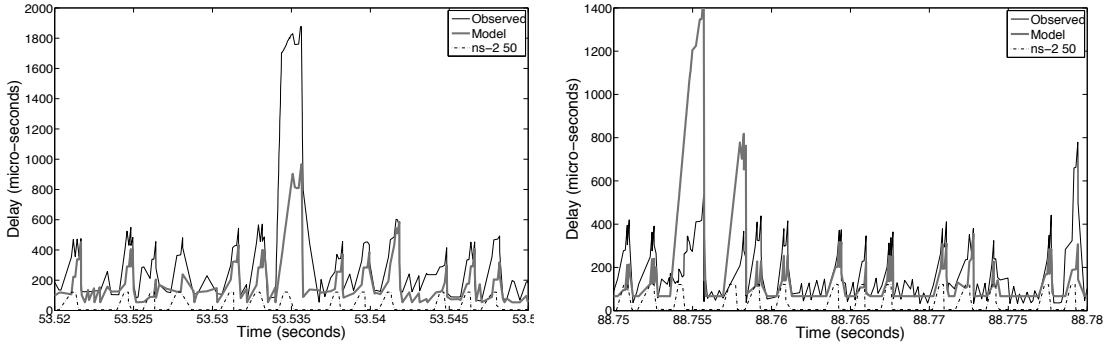
Figure 6.10. High-load TCP and HTTP: Delays on port2

Fig. 6.11 gives the results on the congested port2 for 500 ms, while Fig. 6.12 gives the results on the non-congested port0 for 30 ms for the Cisco 3660 and 7206VXR. For the Cisco 7206VXR, we also execute ns-2 simulations with a 125-slot queue. This is



(a) Cisco 3660 (b) Cisco 7206VXR

Figure 6.11. High-load TCP and HTTP: Delays on port2



(a) Cisco 3660 (b) Cisco 7206VXR

Figure 6.12. Effects of backplane contention on non-congested port0

the number of MTU-sized packets that the 7206VXR can support (see Table 6.1). Our model predictions are close to the observed data. The results on the non-congested ports appear further apart because the scale on the y-axis is fine-grained. Both routers report moderate levels of average CPU usage: 40% for the Cisco 3660 and 33% for the 7206VXR. As these routers use interrupt–driven packet processing (Section 6.1), it is not surprising that packets experience backplane contention. The default ns-2 model does not consider backplane contention, and hence the predicted delays are

lower. Setting the ns-2 queue size to 125 for the 7206VXR increased accuracy over a queue size of 50; however, the results still underestimated the actual delays. This is because there is a wide range of different sized HTTP packets in the experiment, and our results show that the router has separate buffers for different sized packets, meaning that a single 125-slot queue will not suffice.

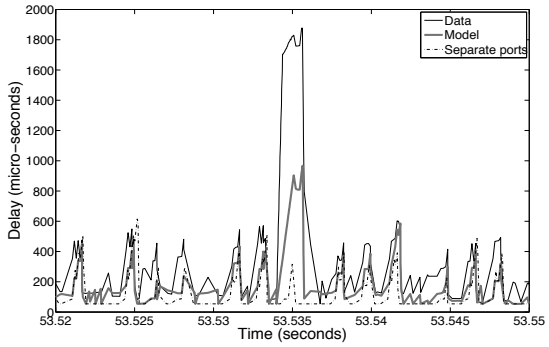


Figure 6.13. Cisco 3660 on port0: Separate ports case

Finally, to demonstrate the effects of backplane contention, we simulate a modified version of our model which decouples the ports from each other, hence removing cross-traffic interactions. Fig. 6.13 presents the result for the same data as in Fig. 6.12(a). The data indicates that significant delay can be caused by cross-traffic, even if the output links are not shared. Failure to capture such behavior can have implications on experiments which rely on RTT estimation, as additional jitter can cause significant deviation between simulation and real life data.

6.4 Summary

In this chapter, we have used the BBP to infer model parameters for three Cisco routers: 3660, 7206VXR, and 12410, and a Juniper router: M7i. We then compared real observations to the model predictions, as well as to the default ns-2 queuing model. The comparisons revealed that the model is able to capture backplane contention, as well as the three queuing strategies we observed in the routers: slot based,

byte based, and separate buffers for differently sized packets. The model was integrated into ns-2 and it has preserved the scalability properties of ns-2, as it did not add significant extra computational overhead. We have also observed cases, when the model failed to completely capture the router behavior, because the complex traffic scenarios have created levels of load not observed in the parameter inference steps. This indicates that more research needs to be carried out to improve the inference steps. We believe that incorporating even the current version of the model into simulators can significantly increase fidelity of network simulations, while preserving scalability. Our inference steps can also be used to only derive queue type and size for use in simpler models such as the VOQ. Knowing only the correct queue type and size can improve simulation results dramatically, and remove the guess work when selecting queue parameters. The inferred parameters can also be used to configure router emulators and software routers, by setting up appropriate rate limits and queueing properties.

7 CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize the main components of this thesis and finish with examination of future research.

7.1 Simulation Versus Emulation

To motivate this thesis, we compared simulation and emulation experiments with seemingly identical parameters and have revealed *key differences* in the results. The majority of differences occur because simulators rely on abstract models, which do not take into account all of the processing required when forwarding packets. PCs used on Emulab/DETER and commercial routers on WAIL can have multiple bottlenecks because of their CPUs, buses, and devices. These bottlenecks are not modeled by simulators. When we compared data from the Emulab and DETER emulation testbeds, the nodes on Emulab experience a much higher CPU load than the DETER nodes for the same packet rate, even though the hardware and software on both testbeds may appear similar. This means the results are highly dependent on the details of underlying hardware and software, and their settings. These different settings are typically irrelevant for typical networking and operating systems experiments, but cause dramatic differences under DoS attacks that overload the system. We have confirmed this observation once again by running the experiments on Cisco routers at the WAIL testbed. There were also major differences between different Cisco routers. It took us great effort to investigate the reasons why PCs, Cisco routers and simulation models vary between each other. This made us consider the feasibility of creating a model that in autonomous fashion can incorporate device specifics to increase the accuracy of network simulations.

7.2 Router Modeling

Motivated by our simulation versus emulation comparison, we have created a device–independent model for forwarding devices, such as switches and routers, and outlined an autonomous model parameter inference procedure. The model was designed with the following specifications in mind: (1) it is accurate, but is allowed to miss special cases for the sake of scalability; (2) it is not computationally expensive; (3) its parameter inference process is the same regardless of the device being modeled; (4) its parameters are inferred without assuming any knowledge of device internals; (5) the model reflects behavior under changing workloads. The model is designed to use only a few parameter tables to mimic a specific router. Since the tables are compact, that makes the model highly portable and easy to compute. The model attempts to replicate not only packet delays as a result of router processing and output queuing but also switching fabric contention.

To derive the device–independent model parameters, we have created a high-performance profiler/traffic generator called the Black Box Profiler (BBP). The purpose of the system was to run on commodity hardware yet provide micro-second level precision measurements. For the hardware to run BBP on, we used an 8 CPU SMP PC with multiple network cards. Having multiple CPUs allows us to run dedicated measurement processes to reduce measurement noise. On the software side, we use Click modular router, a modified network device driver, and a modified ns-2 simulator. Besides acting as a router profiler, BBP was also designed to generate large volumes of closed-loop traffic at 100 Mbps and 1 Gbps speeds. This feature allowed us to perform validation of the model in complex traffic scenarios, by comparing the model predictions to the actual empirical observations.

We have created ns-2 modules that incorporate our model into ns-2 simulator, giving us an option to use our device independent model or the default queuing model. Using BBP, we have inferred model parameters for three Cisco routers: 3660, 7206VXR, and 12410, and a Juniper router: M7i. Then we compared model pre-

dictions to the real observations, as well as to the regular ns-2 queuing model. The comparisons revealed that the model is able to approximate backplane contention, as well as the three queue sizing strategies we observed in the routers. When running our model versus the default queuing model, there was no noticeable overhead, meaning that the scalability property of the simulation was not compromised.

7.3 Future Work

We have presented only one version of a device independent model. The current model is based on a Multi–Queue/Multi–Server model; however, a model might be created based on purely statistical methods. One such method is to use regression analysis to model packet loss and delay. Loss and delay can be modeled as *dependent variables*, which depend on packet arrivals which can be treated as *independent variables*. Hence, it might be possible to create a model for delay and loss based on packet inter–arrival times. Besides using statistical methods, a model can be constructed that uses a notion of "cells" when treating packets to closer approximate the behavior of the switch fabric. A model based on "cells" can have a structure similar to our Multi-Queue/Multi-Server model, although it can conceivably be based on statistical methods as well.

We have investigated the applicability of the model and model parameters in pure simulations. Emulation testbeds are also a critical tool for networking research and can benefit from our efforts. General purpose emulation testbeds similar to Emulab typically rely on large numbers of exactly the same machines; as otherwise, it becomes considerably problematic to support a vast variety of devices. Because of this, researchers are limited in the hardware choices available to them on the testbed. It is possible to add custom devices, but it is time consuming and requires a significant effort from the testbed staff. Ideally, it would be of great use to configure a testbed node to emulate any given device. Realistically, a node cannot emulate a device which is faster than it is. Also, it might not be possible to create the same inter–

device latencies as the target device. This is because the node itself has inter–device latencies which cannot be removed, and achieving micro–second level timer precision to create appropriate delays is extremely difficult. However, despite these limitations it is still possible to use some parameters of our model. First, a testbed node can be configured to maintain the same rate limits as the target device. This applies per port and also for the aggregate forwarding rate. Second, custom queues can be created to mimic the queuing behavior on the target device. Even with a smaller subset of the model features, the experiment accuracy will improve.

LIST OF REFERENCES

LIST OF REFERENCES

[1] J. Mirkovic and P. Reiher. A taxonomy of DDoS attacks and DDoS defense mechanisms. *ACM Computer Communications Review*, 34(2):39–54, April 2004.

[2] D. Moore, G. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proceedings of USENIX*, 2001.

[3] Micronas Inc. Preparing for the IPTV future. `http://www.gnbm.org/uploads/IPTV_the_future.pdf`, 2006.

[4] J. Barrett. The IPTV conundrum in Asia. `http://www.parksassociates.com/free_data/downloads/parks-IPTV_in_Asia.pdf`, 2006.

[5] Alcatel Lucent. Broadband applications fueling consumer demand. `http://www1.alcatel-lucent.com/doctypes/articlepaperlibrary/pdf/Broadband_Apps_swp.pdf`, 2004.

[6] D. M. Nicol. Scalability of network simulators revisited. In *Proceedings of the Communications Networking and Distributed Systems Modeling and Simulation Conference*, February 2003.

[7] D. M. Nicol. Utility analysis of network simulators. *International journal of simulation: Systems, Science, and Technology*, 2003.

[8] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[9] F. Baker. Re: [e2e] extracting no. of packets or bytes in a router buffer. Message to "end2end" mailing list, December 2006.

[10] The worlds leading network modeling and simulation environment. `http://www.opnet.com/products/modeler/home.html`, 2005.

[11] OMNeT++. `http://www.omnetpp.org/`.

[12] B. Van den Broeck, P. Leys, J. Potemans, J. Theunis, E. Van Lil, and A. Van de Capelle. Validation of router models in OPNET. In *Proceedings of OPNETWORK*, 2002.

[13] Peh Li-Shiuan and W. J. Dally. A delay model for router microarchitectures. In *IEEE Micro*, volume 21, pages 26–34, Jan 2001.

[14] K. Yum, E. Kim, and C.R. Das. QoS provisioning in clusters: an investigation of router and NIC design. In *Proceedings. 28th Annual International Symposium on Computer Architecture, 2001*, pages 120–129, June–July 2001.

[15] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot. Bridging router performance and queuing theory. In *Proceedings of SIGMETRICS*, 2004.

[16] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the transients of adaptation for RoQ attacks on internet resources. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Oct 2004.

[17] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants). In *Proceedings of ACM SIGCOMM*, August 2003.

[18] S. Floyd and E. Kohler. Internet research needs better models. *SIGCOMM Computer Communications Review*, 33(1):29–34, 2003.

[19] X. Luo and R. K.-C. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Procceedings of Network and Distributed System Security Symposium (NDSS)*, February 2005.

[20] H. Sun, J. Lui, and D. Yau. Defending against low-rate TCP attacks: Dynamic detection and protection. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Oct 2004.

[21] Y. Chen, K. Hwang, and Y.-K. Kwok. Collaborative defense against periodic shrew DDoS attacks in frequency domain. *Submitted*, 2005. Available at: http://gridsec.usc.edu/files/TR/ACMTISSEC-LowRateAttack-May3-05.pdf.

[22] L. Rizzo. DummyNet. `http://info.iet.unipi.it/~luigi/ip_dummynet/`.

[23] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proceedings of ACM SIGCOMM*, August 2001.

[24] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, May 2004.

[25] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, Feb 2002.

[26] D. G. Andersen. Mayday: Distributed filtering for internet services. In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[27] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: An effective defense against spoofed DoS traffic. In *Proceedings of ACM Computer and Communications Security (CCS)*, pages 30–41, Oct 2003.

[28] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM*, volume 18, pages 314–329, August 1988.

[29] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM*, volume 28, pages 303–314, September 1998.

[30] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[31] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.

[32] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. RFC 2018, October 1996.

[33] J. F. Kurose and K. W. Ross. *Computer Networking – A top-down approach featuring the Internet*. Addison Wesley, 2001.

[34] 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 Operating Systems Design and Implementation (OSDI)*, pages 255–270, December 2002.

[35] R. Bajcsy and et al. Cyber defense technology networking and evaluation. *Communications of the ACM*, 47(3):58–61, March 2004.

[36] PC300 – The Router Killer. `http://www.cyclades.com/resources/?wp=6`.

[37] `http://www.sangoma.com`.

[38] ImageStream and Cicso Comparison. `http://www.imagestream.com/Cisco_Comparison.html`.

[39] OSS network routers. `http://www.siriusit.co.uk/docs/doc.php?pageID=8&typeID=3`, December 2004.

[40] P. Druschel, L. Peterson, and B. Davie. Experiences with a high-speed network adaptor: A software perspective. In *Proceedings of the ACM SIGCOMM Conference*, pages 2–13, August 1994.

[41] J. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, August 1997.

[42] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[43] MOSES Project. iSSF and iSSFNet network simulators. `http://www.linklings.net/MOSES/?page=software`, 2005.

[44] Classful Queueing Disciplines. `http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.qdisc.classful.html`.

[45] S. Agarwal, J. Sommers, and P. Barford. Scalable network path emulation. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2005.

[46] A. Tirumala and et al. Iperf - the TCP/UDP bandwidth measurement tool. `http://dast.nlanr.net/Projects/Iperf/`, May 2005.

[47] M. Mathis and R. Reddy. Enabling high performance data transfers. `http://www.psc.edu/networking/projects/tcptune/`, July 2006.

[48] R. Chertov. Performance of a software link monitor. http://www.cs.purdue.edu/homes/rchertov/reports/click.pdf, August 2005.

[49] A. Bianco, R. Birke, D. Bolognesi, J. Finochietto, G. Galante, M. Mellia, M. Prashant, and F. Neri. Click vs. Linux: Two efficient open-source IP network stacks for software routers. In *IEEE Workshop on High Performance Switching and Routing*, May 2005.

[50] Cisco. Cisco routers. http://www.cisco.com/warp/public/cc/pd/rt/index.shtml.

[51] Cisco. How to choose the best router switching path for your network. http://www.cisco.com/en/US/tech/tk827/tk831/technologies_white_paper09186a00800a62d9.shtml, August 2005.

[52] Cisco Systems. Cisco 3600 series router architecture. http://www.cisco.com/en/US/products/hw/routers/ps274/products_tech_note09186a00801e1155.shtml, 2006.

[53] Cisco Systems. Cisco 7200 series router architecture. http://www.cisco.com/en/US/products/hw/routers/ps341/products_tech_note09186a0080094ea3.shtml, 2007.

[54] PDNS – Parallel/Distributed NS. http://www-static.cc.gatech.edu/computing/compass/pdns/.

[55] GTNetS. http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[56] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou, and C.C. Lin. The design and implementation of the NCTUns 1.0 network simulator. In *Computer Networks 2003*, volume 42, pages 175–197, June 2003.

[57] J-Sim. http://www.j-sim.org.

[58] K. Fall. Network emulation in the Vint/NS simulator. In *Proceedings. IEEE International Symposium on Computers and Communications, 1999.*, pages 244–250, July 1999.

[59] 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)*, Dec 2002.

[60] P. Zheng and L.M. Ni. EMPOWER: A network emulator for wireline and wireless networks. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1933–1942, March–April 2003.

[61] D. Xu J. Xuxian. vBET: a VM-Based emulation testbed. In *Proceedings of ACM Workshop on Models, Methods and Tools for Reproducible Network Research*, Aug 2003.

[62] Emulab. Emulab - network emulation testbed. http://www.emulab.net.

[63] A. Shaikh and A. Greenberg. Experience in black-box OSPF measurement. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 113–125, New York, NY, USA, 2001. ACM Press.

[64] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices. Request for Comments 2544, http://www.rfc-archive.org/getrfc.php?rfc=2544, march 1999.

[65] R. Mandeville and J. Perser. Benchmarking methodology for LAN switching devices. Request for Comments 2889, http://www.faqs.org/rfcs/rfc2889.html, august 2000.

[66] Endace. http://www.endace.com/.

[67] J. Sommers and P. Barford. Self-configuring network traffic generation. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81, New York, NY, USA, 2004. ACM Press.

[68] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. In *Proceedings of DS-RT*, pages 29–36, October 2004.

[69] IXIA. http://www.ixiacom.com.

[70] M. C. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. D. Smith. Tmix: A tool for generating realistic application workloads in ns-2. *ACM Computer Communication Review*, 36:67–76, July 2006.

[71] K. V. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. In *Proceedings of ACM SIGCOMM*, 2006.

[72] K. Papagiannaki, D. Veitch, and N. Hohn. Origins of microcongestion in an access router. In *Proceedings of Passive and Active Measurement (PAM)*, 2004.

[73] Cisco. Cisco 1600 series router architecture. http://www.cisco.com/en/US/products/hw/routers/ps214/products_tech_note09186a0080094eb4.shtml.

[74] Cisco Systems. Cisco 12000 series internet router architecture: Packet switching. http://www.cisco.com/en/US/products/hw/routers/ps167/products_tech_note09186a00801e1dc1.shtml.

[75] Juniper Networks. Juniper networks m-series multiservice edge routing portfolio. http://www.juniper.net/products_and_services/m_series_routing_portfolio/.

[76] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, April 1999.

[77] T.P. Troudet and S. M. Walters. Hopfield neural network architecture for crossbar switch control. *IEEE Transactions on Circuit Systems*, 38:42–57, January 1991.

[78] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, 11(4):319–352, November 1993.

[79] D. Lin and R. Morris. Dynamics of random early dectection. In *Proceedings of the ACM SIGCOMM*, volume 27, pages 127–136, September 1997.

[80] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, November 2001.

[81] T. J. Ott, T. V. Lakshman, and L. H. Wong. SRED: Stabilized RED. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1346–1355, March 1999.

[82] G. Varghese. *Network Algorithmics*. Morgan-Kaufmann, 2005.

[83] L. Deri. Improving passive packet capture: Beyond device polling. In *Proceedings of System Administration and Network Engineering (SANE)*, June 2004.

[84] Cisco Systems. Basic system management. `http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration_guide_chapter09186a008030c799.html#wp1009032`.

[85] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle. Stochastic models for generating synthetic HTTP source traffic. In *Proceedings IEEE INFOCOM*, pages 1547–1558, March 2004.

VITA

VITA

Education

Purdue University

- Ph.D. in Computer Science, Spring 2008.
- Master of Science in Computer Science, Spring 2004.

University of Maryland at College Park

- Bachelor of Science in Computer Science, Spring 2002
- Bachelor of Arts in Economics, Spring 2002

Research interests

High fidelity emulation and simulation, network security/planning, networking, statistical router modeling, traffic generation, distributed systems

Publications

Refereed Conferences and Workshops

- **Roman Chertov**, Sonia Fahmy, and Ness B. Shroff, "A Device-Independent Router Model," *In Proceedings of IEEE* **INFOCOM** *(the conference on computer communications)*, (to appear in April 2008).
- **Roman Chertov**, Sonia Fahmy, and Ness B. Shroff, "A Black-box Router Profiler," *In Proceedings of the IEEE Global Internet Symposium* (GI), May 2007.
- J. Mirkovic, S. Wei, A. Hussain, B. Wilson, R. Thomas, S. Schwab, S. Fahmy, **R. Chertov**, P. Reiher, "DDoS Benchmarks and Experimenter's Workbench for the DETER testbed," *In Proceedings of 3rd International IEEE/CreateNet Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (TridentCom), May 2007.

- **Roman Chertov** and Sonia Fahmy, "Optimistic Load Balancing in a Distributed Virtual Environment," *In Proceedings of the 16th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV), pp. 74-79, May 2006.
- **Roman Chertov**, Sonia Fahmy, and Ness B. Shroff, "Emulation versus Simulation: A Case Study of TCP-Targeted Denial of Service Attacks," *In Proceedings of 2nd International IEEE/CreateNet Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (TridentCom), March 2006.

Refereed Journals

- **Roman Chertov**, Sonia Fahmy, and Ness B. Shroff, "Fidelity of Network Simulation and Emulation: A Case Study of TCP-Targeted Denial of Service Attacks", *Transactions on Modeling and Computer Simulation (TOMACS)* (to appear in 2008)

Technical Reports

- **Roman Chertov**, "Performance of a Software Link Monitor", *Information Science Institute*, 2006
- **Roman Chertov** and Sonia Fahmy, "Design and Validation of a Software Link Monitor", *Purdue University*, 2006

Conference Workshop Presentations

- *IEEE GI*, Anchorage, May 2007
- *DETER Workshop*, Arlington, June 2006
- *ACM NOSSDAV*, New Port, May 2006
- *IEEE TridentCom*, Barcelona, March 2006

Software

Black Box Profiler, a traffic generation/measurement system based on ns-2 simulator, modified Linux network driver, and Click modular router. The system is

capable of creating arbitrary traffic flow scenarios with multiple unique IPs, as well as measuring packet loss, corruption, and delay with microsecond precision. The tool is planned to be released in early 2008.

EMIST Tool Suite, a collection of tools designed to control, measure, and analyze experiments on testbeds. The tools can be downloaded at
http://www.cs.purdue.edu/homes/fahmy/software/emist/