Forwarding Devices: From Measurements to Simulations

ROMAN CHERTOV
Aerospace Corporation
and
SONIA FAHMY
Purdue University

Most 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. Incorrect conclusions are often drawn from these results about transport or application protocol performance, service provisioning, or vulnerability to attacks. In this paper, we argue that measurement-based models for routers and other forwarding devices are necessary. 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 the model uses device-specific parameters. The compactness of the parameters and simplicity of the model make it versatile for high-fidelity simulations that preserve simulation scalability. We construct a profiler to infer the parameters within a few hours. Our results indicate that our model approximates different types of routers significantly better than the default ns-2 simulator models. The results also indicate that queue characteristics vary dramatically among the devices we measure, and that backplane contention can be a factor.

Categories and Subject Descriptors: C.2.3 [**Network Operations**]: Network monitoring; C.4 [**Performance of Systems**]: Performance attributes; I.6.4 [**Simulation and Modeling**]: Model validation and analysis

General Terms: Measurement, Performance

Additional Key Words and Phrases: Emulation, router benchmarking, router modeling, simulation

1. INTRODUCTION

Network simulators must balance a fidelity versus scalability tradeoff [Nicol 2003a; 2003b]. 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 [Breslau et al. 2000]. The ranges of intra-device latencies and maximum packet forwarding rates in commercial forwarding devices are not incorpo-

This research has been sponsored in part by NSF grants 0523249 and 0831353. The authors can be reached at: Roman Chertov, 2310 E. El Segundo Blvd., El Segundo, CA 90245–4609, USA. Tel: +1-310-336-2799. E-mail: Roman.O.Chertov@aero.org. Sonia Fahmy, 305 N. University St., West Lafayette, IN 47907–2107, USA. Tel: +1-765-494-6183. E-mail: fahmy@acm.org.

rated. Networking researchers often find it difficult to select appropriate parameters, such as router buffer sizes and forwarding rate limits, 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 not be realistic [Floyd and Kohler 2002].

The high-level models used to represent routers in simulators like ns-2 typically model 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, switching fabrics in low-to-mid level routers have much lower performance. Yet, due to cost considerations, they constitute the majority of the forwarding devices in Internet edges and enterprise networks, which is where most packet losses in today's Internet occur [Hohn et al. 2004]. Accurately modeling a range of devices is especially important in experiments with high resource utilization, since resource consumption models commonly used in simulators and emulators are not always 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, our prior results with low-rate TCP targeted denial of service attacks [Chertov et al. 2008b] demonstrate that seemingly identical tests on various testbeds and on the ns-2 simulator produce dramatically different results (the attack being completely ineffective versus highly damaging). 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) [Baker 2006] 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 (http://www.opnet.com/products/modeler/home.html) and OMNeT++ (http://www.omnetpp.org/). In OPNET, detailed models of routers, switches, servers, protocols, links, and mainframes are given, based solely on vendor specifications [Van den Broeck et al. 2002]. 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 router behavior [Van den Broeck et al. 2002].

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) the model accurately reflects behavior under a broad range of workloads, but can miss special cases in order to be computationally inexpensive, and (2) its parameters are inferred without assuming any knowledge of device internals, and the parameter inference procedure is independent of the device being modeled.

An accurate model of forwarding behavior is critical for experiments that investigate the performance of transport layer protocols, application layer protocols and services, and network dimensioning. For instance, previous work [Hirabaru 2006; Mathis et al. 1997] has shown that queue size has a significant impact on the performance of TCP across congested links. Our prior work [Mirkovic et al. 2007] has shown that user-perceived application performance can range from acceptable to completely unacceptable depending on the packet delay, loss, or jitter values introduced at an intermediate router. These values can significantly deviate from realistic values if the router forwarding model is inaccurate. Our work

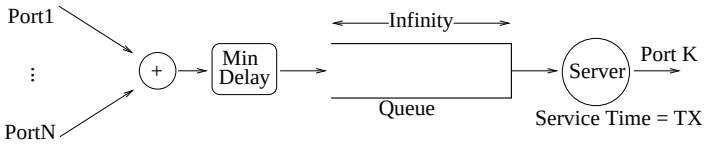


Fig. 1. Hohn *et al.*'s minimum delay queuing model with an unbounded queue *per output port*. The service time is based on the packet transmission (TX) time.

gives methods to ascertain and to model router bottlenecks and router buffer sizes. For example, we can use our results to configure tools such as DummyNet [Rizzo 2010] or NetPath [Agarwal et al. 2005] – used in emulation testbeds such as Emulab (www.emulab.net) and DETER (www.deterlab.net) – for accurate packet delay, loss, and jitter, and hence more realistic performance.

To our knowledge, the most recent study on modeling a router based on empirical observations is that by Hohn et al. [2004]. 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 computed from empirical observations of a production Tier-1 access router. Fig. 1 depicts the model. As the figure shows, the model is substantially similar to classical output queuing models used as a default in simulators like ns-2, but there is a constant minimum delay added to each packet based on its packet size. Each output port is modeled in this fashion, thus abstracting away interactions at the inputs and the backplane. 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 heavier loads, since, among other reasons, loss cannot be modeled.

In this paper, we extend our prior work [Chertov et al. 2008a] with a new model, simulation modules, and experiments. Our proposed models differ from the VOQ-based model [Hohn et al. 2004] in several key aspects – most importantly the queue size and number of servers. Our models generalize to different router and switch types, but *parameters* of a model customize it for a specific type. We design a system for automatically inferring these model parameters. 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 modules can be downloaded from
http://www.cs.purdue.edu/homes/fahmy/software/BBP.tgz. Our experiments with UDP, FTP, and HTTP traffic reveal that our model captures different router behaviors. The results also indicate that queue and performance characteristics vary dramatically among the devices we measure, and that backplane contention should be modeled. We believe this to be a significant step toward creating higher fidelity yet scalable simulations.

The remainder of this paper is organized as follows. Section 2 explains our models and the model parameter inference process (an additional simplified model is given in Section A.3 of the online appendix). Section 3 gives the details of our profiler and experimental setup (details on the profiler can be found in the online appendix). Section 4 discusses our results. Section 5 summarizes related work. We conclude in Section 6.

2. MODELING A FORWARDING DEVICE

A multitude of router architectures exists today. Their switching fabrics range from simple shared memory or shared medium designs, to sophisticated highly interconnected designs

like 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. Despite these disparate designs, routers share critical similarities: (1) A router has a number of input and output interfaces; (2) Packets may be dropped or delayed within a router; (3) A router has intermediate buffers/queues; (4) Packet flow in a router is complex [Baker 2006], 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" [Cisco Systems 2010b]); and (6) Shared components such as the backplane, caches, and possibly a central processor can lead to interference among flows that do *not* share the same output queues.

The complexity of router tasks introduces difficulties in developing accurate and comprehensive models of routers and other forwarding devices. A router must deal with control packets such as ARP (Address Resolution Protocol) and ICMP (Internet Control Message Protocol), as well as routing packets such as BGP (Border Gateway Protocol), OSPF (Open Shortest Path Protocol), and RIP (Routing Information Protocol). These control packets can have a profound impact on the forwarding of regular packets [Shaikh and Greenberg 2001; Chertov et al. 2008b]. For instance, ARP can lead to a significant delay until mapping between a packet's layer three (IP) and layer two (MAC) addresses is established. Routing packets can lead to delays or losses of data packets as routes are added and removed. Routers can also have interfaces with different speeds (e.g., Ethernet, FastEthernet, SONET). For simplicity, we make the following assumptions in order 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. However, we do not assume any knowledge of router internals or traffic characteristics.

Violation of these assumptions can lead to inaccuracies. For instance, if a router is configured with a given QoS policy and the model fails to capture this, then the model predicts packet delays which can significantly deviate from measured values.

2.1 General Multi-Server/Multi-Queue Model

To model a broad range of forwarding devices, we must consider that traffic interactions can cause packet loss and delay. Our model must capture such interactions in a variety of switching fabrics. Fig. 2 demonstrates a simple device-independent model for a range of contention behaviors. The additional complexity over the VOQ-based model allows modeling devices with limited performance, in addition to the Tier-1 access router modeled by Hohn et al. [2004].

As previously stated, forwarding devices may have multiple queues on the packet path from the input to the output based on their architecture and type of switching fabric. Modeling the location of all the queues and their respective sizes would require detailed knowledge of each device internals. Since 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 the number of slots a given packet size occupies. We automatically infer Q and *QueueReq* from our measurements.

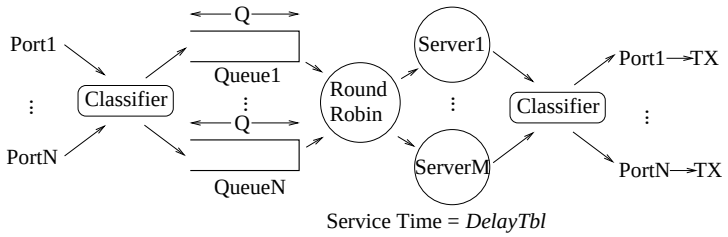


Fig. 2. General model where N inputs are served by M servers. There is one queue of size Q slots per port. Packets exit the forwarding device through one of the N output ports.

As seen in Fig. 2, 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 may be served by a central processor(s) or specialized switching fabric processors. Since it is difficult to determine the exact number of servers (i.e., speedup) in a device, we infer the number of servers, M, based on measurements. Varying M from one to infinity allows modeling the entire range of devices from those with severe contention to those with no contention.

Servers process packets with the measured average processing delay. A table, *DelayTbl*, represents observed *intra-device delays* (excluding transmission delay (TX in Fig. 2)), as described in the online appendix, for various packet sizes. This is similar to "Min Delay" in Fig. 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 [McKeown 1999], 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). Since packets are often split into smaller units (cells) internally within a router [Cisco Systems 2010b], 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.

2.2 Automated Parameter Inference

The five key model parameters that vary from one device to another (M, Q, *DelayTbl*, *QueueReq*, *ServReq*) can be inferred by subjecting a router to a series of simple tests. Constant Bit Rate (CBR) UDP flows are injected through the router in all tests. Table 2.1 gives all the notation used in our parameter inference process, and Algorithm 1 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 set of packet sizes to construct a comprehensive table. If the delay difference between any two interfaces is large (e.g., more than 10%), the algorithm terminates since our assumption that interfaces are approximately similar would be invalid. The experimenter can examine the difference and decide what level of error is acceptable. If error is acceptable, we record the minimum delay for each packet size.

In the second phase, we compute an approximate value for the maximum number of

Table I. 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., at 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, . . . 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$_r$	Time between packet arrivals for rate r

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 an approximate value for 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 in the numerator since we noticed a few outliers that skew the maximum delay. We attribute these outliers to our custom profiler which is based on commodity hardware, as discussed in Section 3.

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.

2.3 Simulation Modules

We have devised two simulation modules that together mimic a forwarding device using the general multi-server/multi-queue model, which we integrated into ns-2.31 [Breslau et al. 2000]. The models can also be integrated with ns-3. The first module, *RouterS*, executes the Multi-Server model, while the second module, *RouterQ*, represents the Multi-Queue

Algorithm 1 Parameter inference

Input: Any forwarding device with N interfaces.
Output: *DelayTbl*, M, Q, *QueueReq*, *ServReq*.

Phase 1: Compute packet delays

1: Determine $D_{s,r,p}$ for all packet size s and rate $r = LowRate$ on path p
2: **if** $D_{s,r,p1} \approx D_{s,r,p2}$ $\forall s \in S$, and $\forall p1, p2 \in P$ **then**
3: $d_s = avg(D_{s,r,1}, ..., D_{s,r,|P|})$
4: **else**
5: exit
6: **end if**

Phase 2: Compute number of servers

1: **for** each $s \in S$ **do**
2: $\forall N$, find max. rate r s.t. no loss
3: $ArriveGap_r = \frac{1}{Nr}$
4: $NumServ_s = \frac{d_s}{ArriveGap_r}$
5: **end for**

Phase 3: Compute queue sizes

1: **for** each $s \in S$ **do**
2: find min. rate r s.t. when $N-1$ ports send to one port at rate r each, $rs(N-1) > TX\_Capacity$
3: $QSize_s = \frac{D_{s,r(N-1),p}}{DepartGap_{s,r(N-1),p}}$
4: **end for**

Phase 4: Record parameter values

1: $DelayTbl =$ set of $d_s, \forall s \in S$
2: $M = max(NumServ_1, \ldots, NumServ_{|S|})$
3: $Q = min(QSize_1, \ldots, QSize_{|S|})$
4: $ServReq_s = M/NumServ_s, \forall s \in S$
5: $QueueReq_s = Q/QSize_s, \forall s \in S$

model. The *RouterQ* objects are connected to the *RouterS* object as depicted in Fig. 3. The *RouterQ* and *RouterS* objects initialize their capacities to the configured Q and M, respectively. When a packet of size s 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. *RouterS* operates similarly. However, instead of dropping packets, it rejects them, preventing the *RouterQ* from performing a dequeue operation. Once *RouterS* finishes serving a packet, it pulls packets from one of the upstream queues until its capacity is filled. Since a single *RouterS* object exists, it approximates backplane contention by preventing queues from dequeuing packets, even when the output link is free.

Finally, we have created trace agents to replay captured packet traces and compare the delays and losses with the simulator against the observed delays and losses, similar to Hohn et al. [2004]. We utilize the same traffic generation scripts when conducting simulations (utilizing our own models or the default ns-2 model) and experiments with the four routers.

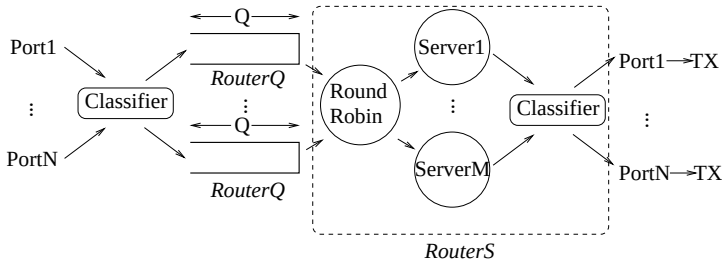


Fig. 3. Our general model implemented in ns-2 using the *RouterS* and *RouterQ* modules.

3. EXPERIMENTAL SETUP

This section explains the setup for our experiments.

3.1 Router Types and Router Configuration

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 120-Kpps fast switching; Supports an extensive variety of modules for various media (e.g., Ethernet, FastEthernet, ISDN, T1) [Cisco Systems 2010c].

(2) Cisco 7206VXR with 100 Mbps FastEthernet modules: Edge device used by enterprises and service providers for service aggregation WAN/MAN; Interrupt-based packet switching on a 262 MHz NPE-300 processor; Supports an extensive variety of high-speed modules for various technologies (e.g., Ethernet, FastEthernet, ATM) [Cisco Systems 2010d].

(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; The router is equipped with multi-gigabit cross-bar switch fabric and each line card runs IOS and uses CEF [Cisco Systems 2010e]; 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) [Juniper Networks 2010].

The lower-end routers (Cisco 3600 and Cisco 7206VXR) we use have *four Fast Ethernet ports*. The Cisco 3660 has two identical ports on the main board and a dual port swappable module, while the Cisco 7206VXR has one main card and three swappable modules. On the Juniper M7i, three cards are swappable and the fourth is integrated into the chassis. The 4 port 4GE-SFP-LC line card on the Cisco 12410 chassis does not utilize the router backplane. For both higher-end routers (Cisco 12410 and Juniper M7i), we configured our profiler to use its network cards at 1 Gbps speed. Clearly, we are measuring a *particular*

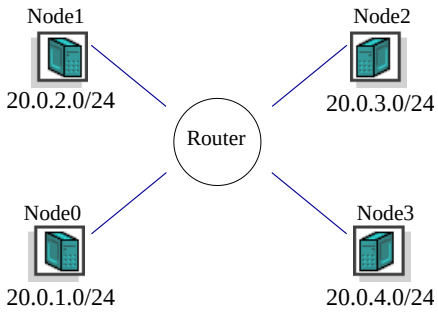


Fig. 4. Test topology with four subnets

configuration for each of the four routers, and it is expected that routers with different configurations will have variable performance levels.

Fig. 4 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 the four network cards on our profiling system, a Cisco router, or a Juniper router. The construction of the logical nodes is explained in the online appendix. The cross-over cable configuration is used solely for calibration, in order to determine the latencies due to the network cards. Each logical node has its own dedicated subnet as shown in Fig. 4. In the remainder of this paper, we adopt the following naming convention: *portX on the router*, denotes the port connected to *NodeX*.

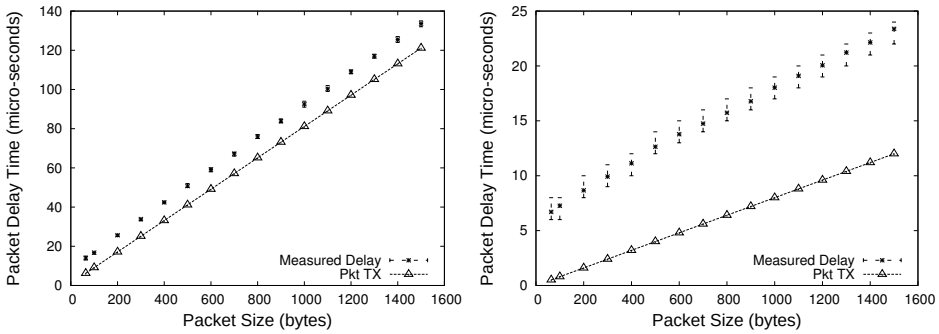
All routers are configured with minimal settings to ensure that forwarding between the ports occurs on a *fast path* without special processing. Further, we enable the Cisco Express Forwarding (CEF) option [Cisco Systems 2010e] on the Cisco 3660 and Cisco 7206VXR as it was not enabled by default. All queue sizes in the traffic generator module in our profiling system were set to 100 slots.

3.2 Calibration

Before proceeding with measurements, we must determine which network device configuration on the measurement machine 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. Fig. 5 demonstrates the measured delay between the two NICs compared to pure transmission delay using a transmit speed of 100 Mbps and 1 Gbps, respectively. We used a constant 50 pps flow of UDP packets to generate these results. In both the 100 Mbps and 1 Gbps scenarios, the difference between the measured delay and pure packet transmission delay never exceeds 14 micro-seconds.

Table II and III demonstrate the mean, 5th, and 95th percentiles of delays of Ethernet frames of sizes of 64, 800, and 1500 at various rates for 100 Mbps and 1 Gbps speed. We have used our *udp_gen* tool to collect these measurements over a period of one minute per experiment. The table indicates that our logging system is adequate.

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



(a) 100 Mbps TX delay

(b) 1 Gbps TX delay

Fig. 5. NIC–to–NIC (mean, 5 and 95 percentiles) vs. pure TX delay.

Table II. 100 Mbps NIC–to–NIC Packet Delays (μs) 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 III. 1 Gbps NIC–to–NIC Packet Delays (μs) 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

simulation. Since 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, since 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. To make the rest of the comparisons accurate, we have removed the events when the actual packet or a predicted packet is dropped. In our scenarios, packet losses never exceed seven percent, hence discarding loss events does not significantly impact the results.

Table IV. Queue Sizes (in Slots) for Different Packet Sizes

Router	64	200	600	800	1000	1200	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 V. Number of Servers for Different Packet Sizes

Router	64	200	600	800	1000	1200	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

4. EXPERIMENTAL RESULTS

This section gives the model parameters inferred for the Cisco 3660, Cisco 7206VXR, Cisco 12410, and Juniper M7i routers, and compares our measurements to results from our simulation modules, as well as ns-2 results using the default ns-2 queuing model.

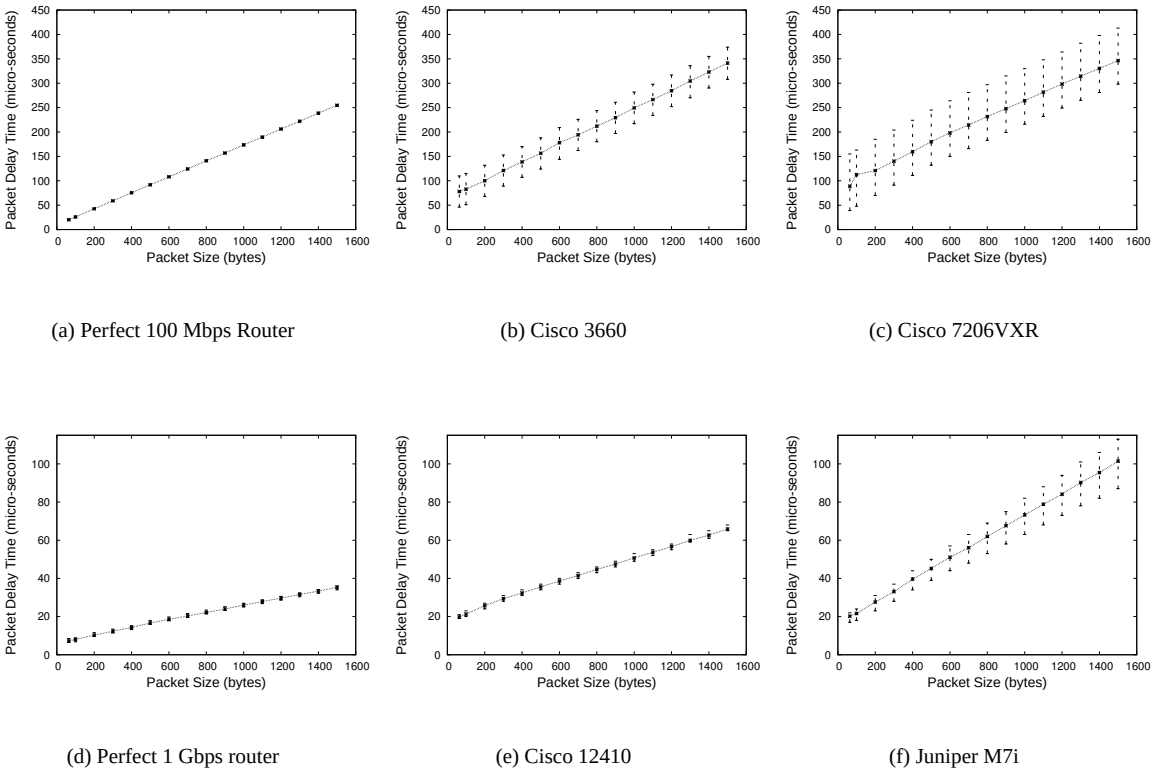
4.1 Model Parameters

As discussed in Section 2.2, we first compute delay tables. We vary the packet size from 64 to 1500 bytes, and maintain the rate at a low 50 packets/s. The packet size includes Ethernet/IP/UDP headers. Fig. 6 compares 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 to obtain the results for the perfect routers: we add the NIC overheads 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 fourth port of that router was integrated into the chassis and had different delay values than the other ports. Unfortunately, since we are limited by the small number of available ports, we were forced to utilize this port in our experiments. Determining the causes of internal router delays requires proprietary details of router hardware and software.

We now use Click, disabling per-packet logging for high speed operation, to infer the number of servers and queue sizes. It is necessary during this phase to use rates of 700+ Kpps per interface, in order to stress the particular Gigabit router interfaces we have. For the Cisco 12410 and Juniper M7i, the maximum achievable transmission rate was 986.82 Mbps; hence we set their *TX_Capacity* in Algorithm 1 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. The measured speed is consistent with the Ethernet specifications, as inter-frame gaps and MAC preambles reduce the usable bandwidth.

Table IV and Table V 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 1.

Analysis of Table IV reveals that there are *three different queue sizing strategies*: (1) The



(a) Perfect 100 Mbps Router (b) Cisco 3660 (c) Cisco 7206VXR

(d) Perfect 1 Gbps router (e) Cisco 12410 (f) Juniper M7i

Fig. 6. Observed minimum delays for different packet sizes

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 [Cisco Systems 2010a]. (2) The Cisco 12410 appears to have a slot-based queue which is approximately 32 K packet slots in size. Since the router has 4 ports, $4 \times 32\ K \times 1500 \approx 200\ MB$ which is close to 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/trailer is not queued). This is consistent with its specification, which states that the router is capable of 200 ms buffering [Juniper Networks 2010].

Table V 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 due to differences in *DelayTbl*s. As expected, the number of servers monotonically decreases as packet size increases for each router.

4.2 Model Fidelity

In this section, we compare the predictions of our general model and the default ns-2 model, with our measurements under a variety of experimental scenarios. We first conduct

each experiment with the physical router and capture packet traces. The traces include packet size and packet arrival information, which is used to create packets in the ns-2 simulator, in order to analyze the accuracy of the simulation models. The same methodology was used by Hohn et al. [2004].

4.2.1 *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. Due to space considerations, we only summarize the results here.

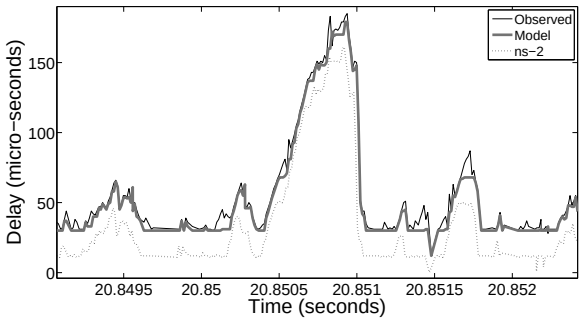
The results indicate that the model can account for backplane contention: the model correctly predicts that the Juniper M7i cannot have all four interfaces forwarding 64-byte frames at more than 715000 pps. This is because $\frac{d_s}{NumServ_s} \times N = \frac{12.997\mu s}{37.145} \times 4 = 1.397\mu s$ and $\frac{1}{1.397\mu s} \times 1e6\frac{\mu s}{s} = 714490$ pps. We checked the router statistics to ensure that 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 due to queue build-ups. Our measurements also confirmed the need for *QueueReq*, as fixed-size slot-based queues are inaccurate.

4.2.2 *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.

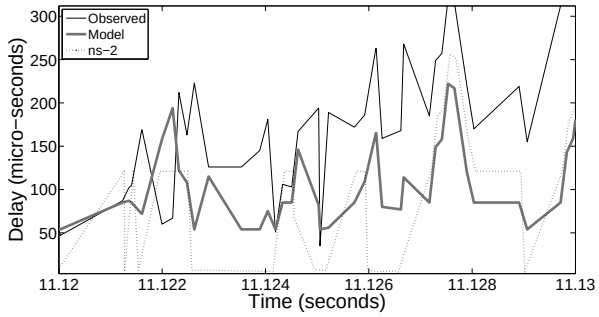
We use *FullTCP SACK* agents in ns-2. *FullTCP* agents in ns-2 mimic the Reno TCP implementation in BSD 4.4. The links from *Node0*, *Node1*, *Node2*, and *Node3* in Fig. 4 were configured to have delays of 20 ms, 30 ms, 40 ms, and 90 ms, respectively. For the two Gigabit routers, the speed is 1 Gbps, while for the Cisco 3660 and 7206VXR we set the speed to 100 Mbps. We also rate limit the link bandwidth to 70 Mbps for each port in the 3660 and 7206VXR experiments. This avoids any TCP packet loss in this series of experiments.

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 payloads, and delayed ACKs. The 1420-byte payload was chosen so that packets with 3 SACK blocks would not exceed 1518 bytes when all the headers/trailers are included, in cases with bi-directional traffic. 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 a single simulation instance 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 interactions if they exist. The experiment duration was 180 seconds, during which we inject traffic into the router and log all transmitted and received packets.

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. 7 gives typical results of the Cisco 12410 and 3660 routers over a period of 3.3 ms and 10 ms, respectively. In the figure, “ns-2” refers to predictions from the default ns-2 model, while “Model” refers to predictions from our general multi-server/multi-queue model. “Observed” is the measured value with the physical router. We can make two interesting observations. First, adding minimum packet delays improves the



(a) Cisco 12410



(b) Cisco 3660

Fig. 7. Low-load TCP: Delays on port2

Table VI. Low-load TCP: Mean and COV of Packet Delays (μs) for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR

		12410		M7i		3660		7206VXR	
Dst 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

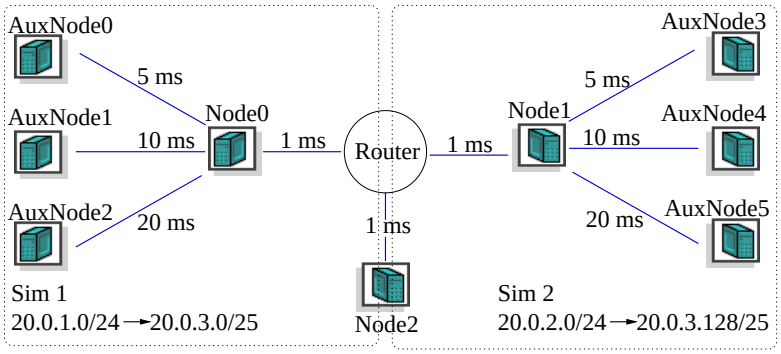


Fig. 8. High-load TCP topology

predictions over the default ns-2 model, which does not consider any processing delay. Second, backplane contention on the 3660 can result in a significant level of additional delay. This is also evident in the 7206VXR results in Table VI. Our model attempts to mimic this by relying on a multi-server model, but, as the figure shows, it is not always accurate at this high precision (micro-second level), possibly due to our commodity-hardware profiler.

Table VI 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.

4.2.3 *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.

Fig. 8 demonstrates the logical 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. To induce sufficient load on the Gigabit routers, we execute two separate simulation instances with 28 TCP agents per auxiliary node, for a total of 84 TCP agents per simulation. The TCP agents were driven by infinite source FTP agents. This setup may appear unusual since there are two links to *Node2* (one per simulator). However, since all of the packets will pass through the router, no more than *TX_Capacity* can traverse the links.

To make this arrangement work, we split the address space of *Node2* into two 128 CIDR address blocks. Fig. 8 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 packets 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 VII demonstrates the loss ratios for the congested output port2. The data indicates that our model provides an accurate approximation of the loss ratios. The default ns-2 queue results

Table VII. 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 VIII. High-load TCP: Mean and COV of Packet Delays (μs) for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR

		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

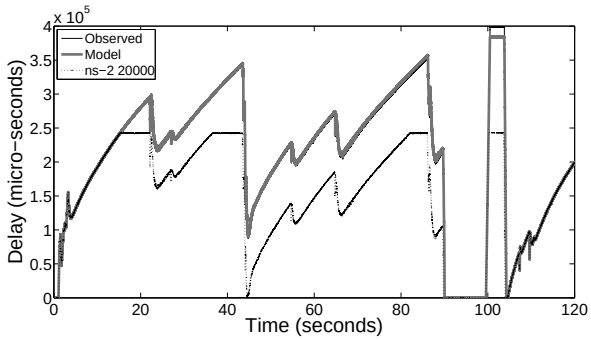
for the Gigabit routers are incorrect for both queue sizes; ns-2 gives accurate loss estimates for the two lower-end routers. These results emphasize the importance of tuning buffer sizes.

Table VIII compares the mean delays (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 very 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.

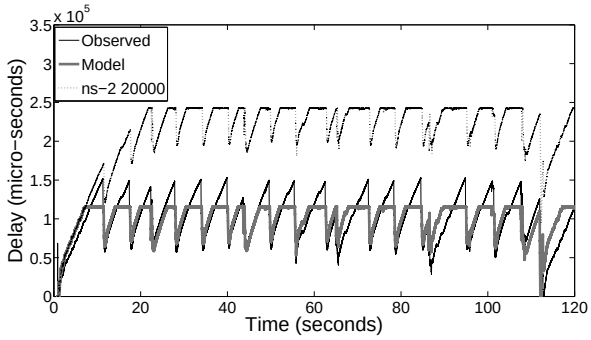
In Fig. 9(a), our model for Cisco 12410 is accurate, and the average absolute difference is 1492 μs. The plot in Fig. 9(b) indicates that our profiling method failed to reveal that the M7i router can sometimes delay packets longer than our simple (CBR) parameter inference indicated. The model delays do not exceed 120 ms, 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 queue sizes do not resolve this problem, as then the queues do not drain as rapidly as they do in our observations. This implies that the extra delay is not additional queuing delay. It is difficult to ascertain the reason for this delay without knowledge of router internals. For both Gigabit routers, the ns-2 results with a queue of 50 slots are orders of magnitude off (very close to the x-axis) as the queue cannot have substantial queuing delay. Results with 20000-slot queues are closer, but still *inaccurate*. Note that the scale on the y-axis is quite coarse-grained in these plots.

Our Cisco 3660 and 7206VXR models match the observations well. Interestingly, the data in Table VIII 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. Fig. 10(a) and Fig. 10(b) provide a snapshot of 500 ms and reveal that our model follows the actual measurements quite closely. The ns-2 model with a 50-slot queue severely underestimates the delay, but the shapes of the curves are not very different from our model and measurements.

4.2.4 *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 and Barford [2004] suggest that routers experience higher loads under realistic traffic, com-



(a) Cisco 12410



(b) Juniper M7i

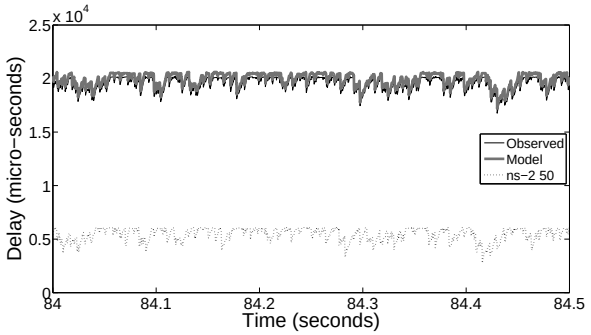
Fig. 9. High-load TCP: Delays on port2

Table IX. High-load TCP and HTTP: Mean and COV of Packet Delays (μs) for Cisco 12410, Juniper M7i, Cisco 3660, and Cisco 7206VXR

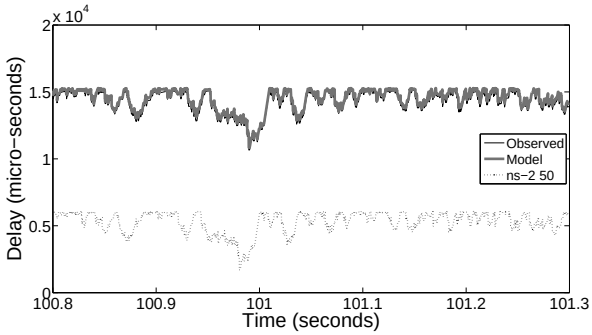
		12410		M7i		3660		7206VXR	
Dst Node	Type	Mean	COV	Mean	COV	Mean	COV	Mean	COV
Node0	Model	14.166	0.561	17.808	0.269	199.422	0.734	151.114	0.929
	Observed	27.662	0.584	22.226	0.099	431.822	1.517	181.312	0.651
Node1	Model	15.525	0.371	19.295	0.448	133.492	0.588	122.855	0.624
	Observed	18.906	0.248	16.683	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

pared to synthetic traffic. Hence, we now use the PackMIME HTTP ns-2 module [Cao et al. 2004]. We made a few modifications to the code to interface the TCP agents with our tap objects (recall Fig. 16).

As in the previous experiment, we ran two simulator instances to create sufficient load for the Gigabit routers. “Sim 2” in Fig. 11 is configured to run server and client HTTP



(a) Cisco 3660



(b) Cisco 7206VXR

Fig. 10. High-Load TCP: Delays on port2

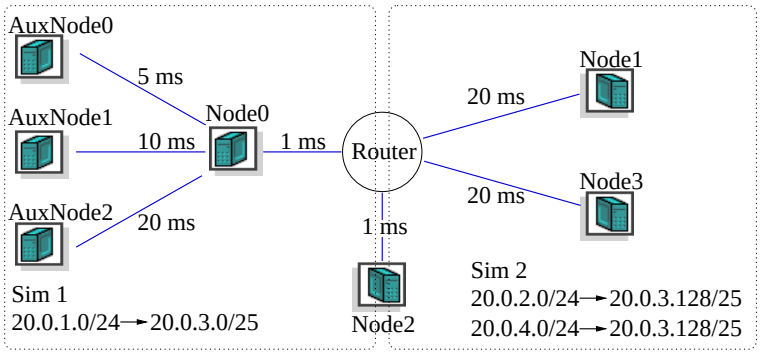


Fig. 11. High-load TCP and HTTP topology

Table X. 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

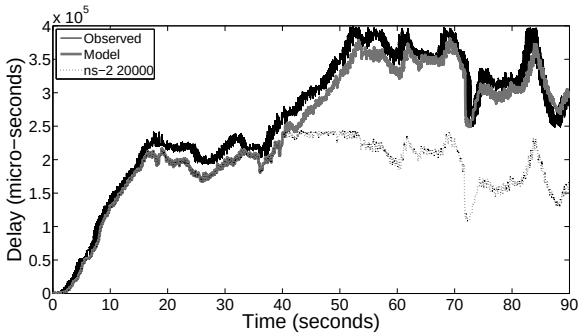
PackMIME clouds. The two client clouds are positioned at *Node2* to force server responses to multiplex on the router port2. Each client-to-server cloud pair is configured to generate 60 new connections per second for a total of 120 connections per second. Additionally, we scale up the server response sizes by a factor of 10. This is necessary to create congestion. This scenario can be representative of a large campus population downloading very image-intensive web pages. Unfortunately, it was not possible to use delay-boxes [Cao et al. 2004] to model flow RTTs, due to the number of packets per second our generator handles. 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 to 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*.

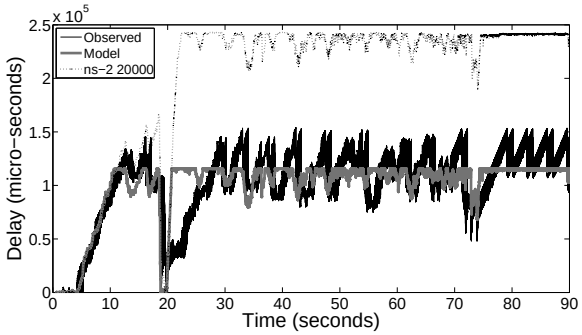
We set the random seeds in experiments with all routers to the same values. The seeds are required by the random streams when creating request/response sizes as well as server and client "thinking" delays. The experiment duration is set to 180 seconds.

Table X 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 reasonably 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 by Papagiannaki *et al.* [Papagiannaki et al. 2004]. Table IX indicates that our model gives a reasonable match for the mean and COV delay values; however, the results are not as accurate as in the high-load TCP scenario. This can be 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 very low.

Fig. 12 depicts a typical window of the packet delays for Cisco 12410 and Juniper M7i for the congested port. Our model follows the observed data quite well. On the M7i router, the model predictions are not as not as precise since some packets experience higher delay than the model predicts. For both routers, the ns-2 results with a 50-slot default queue are close to the x-axis as the queue cannot introduce substantial queuing delay. Therefore, any congestion control, service provisioning, denial of service, or application protocol experiment in this case would have yielded incorrect results. Although loss ratios for ns-2



(a) Cisco 12410

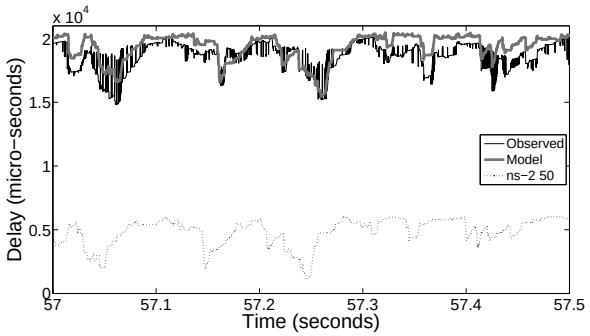


(b) Juniper M7i

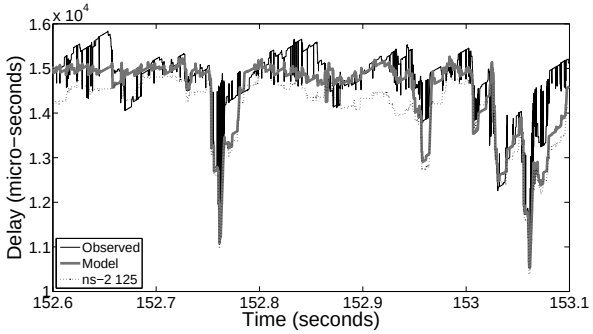
Fig. 12. High-load TCP and HTTP: Delays on port2

with a 20000-slot queue are similar to the observations (Table X), packet delays for ns-2 are *significantly different* (note the scale on the y-axis).

Fig. 13 gives typical results for the congested port2 for 500 ms, while Fig. 14 gives typical results for the non-congested port0 for 30 ms for the Cisco 3660 and 7206VXR. For the Cisco 7206VXR, we also conduct default ns-2 simulations with a 125-slot queue. This is the number of MTU-sized packets that the 7206VXR can support (see Table IV). 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. Since these routers use interrupt-driven packet processing (Section 3), 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 much 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



(a) Cisco 3660



(b) Cisco 7206VXR

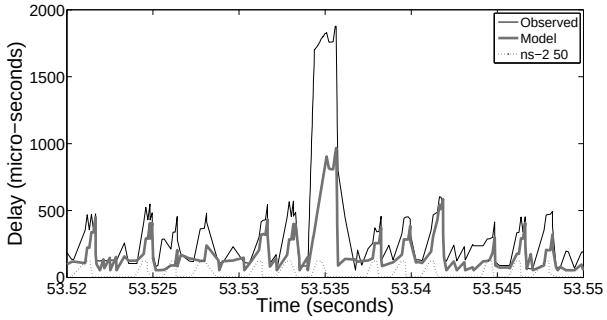
Fig. 13. High-load TCP and HTTP: Delays on port2

router has *separate buffers* for different sized packets, meaning that a single 125-slot queue does not suffice.

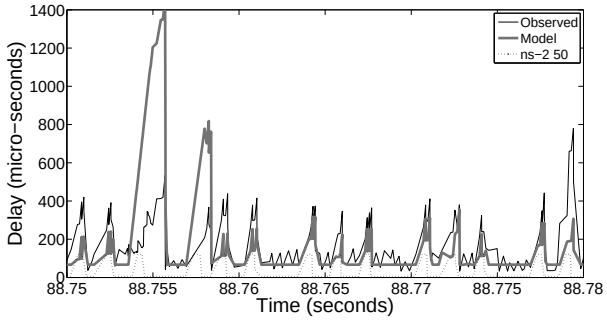
5. RELATED WORK

In this section, we give a brief overview of related work on network simulators, device measurement and modeling, and traffic generation.

Network Simulation. A variety of network simulators make different choices with respect to the fidelity versus scalability tradeoff (e.g., packet-based versus flow-based). Popular simulators today include ns-2, J-Sim, OPNET, OMNET++, iSSF/SSFNet (Java and C++ versions), NCTUns, and simulators from NIST. Some of these simulators also provide simulation/emulation hybrids. Notable emulation tools also include DummyNet, NetPath, ModelNet, and EMPOWER. Simulators that specialize in accurate wireless or sensor network modeling are also available. Nicol compares the running times and memory requirements of ns-2, J-Sim, and the Java and C++ versions of SSFNet [Nicol 2003a], and uses this comparison to analyze their utility to the user [Nicol 2003b]. Our goal is com-



(a) Cisco 3660



(b) Cisco 7206VXR

Fig. 14. Effects of backplane contention on non-congested port0

plementary to these systems: we aim to devise simple yet accurate models for forwarding devices based on real measurements, in order to balance the fidelity/scalability tradeoff.

Device Measurement. Router modeling based on empirical observations is explored by Hohn et al. [2004], as discussed in the introduction. Their work inferred a simple queuing model, but was not designed to handle loss events, or to model interactions at the input ports. A production Tier-1 router was used in that work, as well as in analysis of micro-congestion [Papagiannaki et al. 2004]. 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 [Endace 2010]. Such devices are highly accurate, but increase the setup cost, so we opted for a low-cost custom profiler.

Black-box router measurement has also been investigated. In [Shaikh and Greenberg 2001], a router is profiled with a focus on measuring its reaction times to OSPF routing messages. RFC 2544 [Bradner and McQuaid 1999] and RFC 2889 [Mandeville and Perser 2000] describe the steps to determine the capabilities of a router (e.g., forwarding rate). The RFCs do not discuss creating simulation models based on measurements.

Traffic Generation. The Harpoon traffic generator [Sommers and Barford 2004] uses flow data collected by Cisco routers to generate realistic traffic. One of the earliest network simulation-emulation tools was VINT [Fall 1999] – a part of ns-2. We did not use this for traffic generation as it does not support sending/receiving spoofed IP addresses and is data-rate limited. Recent work to extend emulation in ns-2 has been conducted by Mahrenholz and Ivanov [2004]. Their system does not aim to handle high data rates or perform extensive packet logging with micro-second precision, which are important for our measurements. A commercial alternative for generating live TCP traffic is the IXIA-400T traffic generator (http://www.ixiacom.com). IXIA devices use a proprietary OS, and do not allow changing the types of the TCP stacks, however.

6. CONCLUSIONS AND FUTURE WORK

In this work, we have developed two device-agnostic models for forwarding devices, such as switches and routers, and designed a completely automated model parameter inference process. Both models rely on a few compact parameter tables, with the second model being a simplified version of the first (the simplified model is given in the online appendix). The small size of the parameter tables makes the models highly portable and easy to compute. The running time difference between simulations with the generic ns-2 model and our model is negligible. Our models replicate not only packet delays due to queuing, router processing, and switching fabric contention, but also packet losses, which can significantly impact transport and application protocol performance.

We have inferred model parameters for three Cisco routers: 3660, 7206VXR, and 12410, and a Juniper router: M7i, and compared real observations to the predictions of the models, as well as to the default ns-2 queuing model. Our experiments with different traffic models revealed that our general model is able to capture backplane contention, as well as the three queue sizing strategies we observed in the routers. Results from ns-2 were orders of magnitude off when queue sizes were not correctly tuned, but *even if queue sizes were similar, accurate results for both delay and loss could not be simultaneously obtained with default ns-2 models*. Our simplified model performed equally well to the more sophisticated model in cases where queuing delay was the dominant factor, but it was less accurate in modeling backplane contention. We believe that incorporating the new models in simulators can significantly increase fidelity of network simulations, while preserving scalability. The inferred parameters can also be used to configure router emulators and software routers.

We plan to experiment with a wider variety of forwarding devices (e.g., those implementing active queue management (AQM)), experiment with devices that have varying output port speeds, and further investigate our modeling choices and assumptions, e.g., round-robin scheduling and queue estimation functions. Clearly, our current setup will not extend to faster routers, and we will need more sophisticated tools. We also plan to utilize *tmix* [Weigle et al. 2006] or Swing [Vishwanath and Vahdat 2006] so that generated traffic can be based on realistic application workloads. Finally, we plan to integrate our models with the ns-3 simulator and current testbed development efforts.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tomac/20YY-V-N/p1-Che`

Acknowledgments

The routers profiled were part of the USC-ISI DETER, Wisconsin WAIL, and Purdue TNT testbeds. We would like to thank Paul Barford, Terry Benzel, Michael Blodgett, Ray Hansen, and Ron Ostrenga for facilitating our access to the routers, and Michael Blodgett and Ray Hansen for their help in router configuration. We would like to thank Ness B. Shroff (OSU) for many insightful discussions on earlier versions of this work.

REFERENCES

AGARWAL, S., SOMMERS, J., AND BARFORD, P. 2005. Scalable network path emulation. In *Proc. of MASCOTS*. 219–228.

BAKER, F. 2006. [e2e] extracting no. of packets or bytes in a router buffer. Message thread to "end2end" mailing list, http://www.postel.org/pipermail/end2end-interest/2006-December.txt.

BRADNER, S. AND MCQUAID, J. 1999. Benchmarking methodology for network interconnect devices. RFC 2544, http://www.faqs.org/rfcs/rfc2544.html.

BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., HEIDEMANN, J., HELMY, A., HUANG, P., MCCANNE, S., VARADHAN, K., XU, Y., AND YU, H. 2000. Advances in network simulation. *IEEE Computer 33*, 5 (May), 59–67.

CAO, J., CLEVELAND, W., GAO, Y., JEFFAY, K., SMITH, F., AND WEIGLE, M. 2004. Stochastic models for generating synthetic HTTP source traffic. In *Proc. of IEEE INFOCOM*. 1547–1558.

CHERTOV, R., FAHMY, S., AND SHROFF, N. B. 2007. A black-box router profiler. In *Proc. of Global Internet*. 6.

CHERTOV, R., FAHMY, S., AND SHROFF, N. B. 2008a. A device-independent router model. In *Proc. of IEEE INFOCOM*. 9.

CHERTOV, R., FAHMY, S., AND SHROFF, N. B. 2008b. Fidelity of network simulation and emulation: A case study of TCP-targeted denial of service attacks. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 19*, 1 (December), 4:1–4:29.

CISCO SYSTEMS. 2010a. Basic system management. http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configu%ration_guide_chapter09186a008030c799.html#wp1009032.

CISCO SYSTEMS. 2010b. Cisco 12000 series internet router architecture: Packet switching. http://www.cisco.com/en/US/products/hw/routers/ps167/products_tech_note%09186a00801e1dc1.shtml.

CISCO SYSTEMS. 2010c. Cisco 3600 series router architecture. http://www.cisco.com/en/US/products/hw/routers/ps274/products_tech_note%09186a00801e1155.shtml.

CISCO SYSTEMS. 2010d. Cisco 7200 series router architecture. http://www.cisco.com/en/US/products/hw/routers/ps341/products_tech_note%09186a0080094ea3.shtml.

CISCO SYSTEMS. 2010e. How to choose the best router switching path for your network. http://www.cisco.com/en/US/tech/tk827/tk831/technologies_white_paper091%86a00800a62d9.shtml.

DERI, L. 2004. Improving passive packet capture: Beyond device polling. In *Proc. of SANE*.

ENDACE. 2010. http://www.endace.com/.

FALL, K. 1999. Network emulation in the vint/ns simulator. In *Proc. of ISCC*. 244–250.

FLOYD, S. AND KOHLER, E. 2002. Internet research needs better models. In *Proceedings of ACM HotNets*. 29–34.

HIRABARU, M. 2006. Impact of bottleneck queue size on TCP protocols and its measurement. *IEICE - Trans. Inf. Syst. E89-D*, 1, 132–138.

HOHN, N., VEITCH, D., PAPAGIANNAKI, K., AND DIOT, C. 2004. Bridging router performance and queuing theory. In *Proc. of SIGMETRICS*.

JUNIPER NETWORKS. 2010. Juniper networks m-series multiservice edge routing portfolio. http://www.juniper.net/products_and_services/m_series_routing_portfolio%/.

KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. 2000. The Click modular router. *ACM Transactions on Computer Systems 18,* 3 (August), 263–297.

MAHRENHOLZ, D. AND IVANOV, S. 2004. Real-time network emulation with ns-2. In *Proc. of DS-RT*. 29–36.

MANDEVILLE, R. AND PERSER, J. 2000. Benchmarking methodology for LAN switching devices. RFC 2889, `http://www.faqs.org/rfcs/rfc2889.html`.

MATHIS, M., SEMKE, J., AND MAHDAVI, J. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev. 27,* 3, 67–82.

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

MIRKOVIC, J., HUSSAIN, A., WILSON, B., FAHMY, S., REIHER, P., THOMAS, R., YAO, W., AND SCHWAB, S. 2007. Towards User-Centric Metrics for Denial-of-Service Measurement. In *Proceedings of the Workshop on Experimental Computer Science (part of ACM FCRC)*. 14.

NICOL, D. M. 2003a. Scalability of network simulators revisited. In *Proceedings of the Communications Networking and Distributed Systems Modeling and Simulation Conference*.

NICOL, D. M. 2003b. Utility analysis of network simulators. *International journal of simulation: Systems, Science, and Technology*.

PAPAGIANNAKI, K., VEITCH, D., AND HOHN, N. 2004. Origins of microcongestion in an access router. In *Proc. of PAM*.

RIZZO, L. 2010. DummyNet. `http://info.iet.unipi.it/~luigi/ip_dummynet/`.

SHAIKH, A. AND GREENBERG, A. 2001. Experience in black-box OSPF measurement. In *Proc. of IMW*. ACM Press, 113–125.

SOMMERS, J. AND BARFORD, P. 2004. Self-configuring network traffic generation. In *Proc. of IMC*. ACM Press, 68–81.

VAN DEN BROECK, B., LEYS, P., POTEMANS, J., THEUNIS, J., VAN LIL, E., AND VAN DE CAPELLE, A. 2002. Validation of router models in OPNET. In *Proc. of OPNETWORK*.

VISHWANATH, K. V. AND VAHDAT, A. 2006. Realistic and responsive network traffic generation. In *Proc. of SIGCOMM*. 111–122.

WEIGLE, M. C., ADURTHI, P., HERNANDEZ-CAMPOS, F., JEFFAY, K., AND SMITH, F. D. 2006. Tmix: A tool for generating realistic application workloads in ns-2. *ACM Computer Communication Review 36*, 67–76.

Received August 2009; revised February and April 2010; accepted May 2010

THIS DOCUMENT IS THE ONLINE-ONLY APPENDIX TO:

Forwarding Devices: From Measurements to Simulations

ROMAN CHERTOV
Aerospace Corporation
and
SONIA FAHMY
Purdue University

ACM Transactions on Modeling and Computer Simulation, Vol. V, No. N, Month 20YY, Pages 1–25.

A.1 Profiler

The parameter inference procedure outlined in Section 2.2 is based on measured packet delay and loss data. Acquiring such data at sufficiently high precision requires a high-fidelity measurement system. We have developed our own custom low-cost measurement and profiling system to eliminate the need for expensive specialty cards. Our system is, however, lower precision than the DAG cards used in some prior work [Hohn et al. 2004; Papagiannaki et al. 2004]. We give a brief overview of our system in this section; complete descriptions of the changes made to ns-2, the Click modular router [Kohler et al. 2000], and the device drivers can be found in Chertov et al. [2007] and Chertov et al. [2008a].

Before describing the profiling system, it is important to consider how packet delay measurements can be made with micro-second precision with commodity hardware. Consider the diagram presented in Fig. 15. The diagram shows a commodity PC connected to a router via two network cards. The PC can create a flow of packets, such that packets originate and then terminate on the PC after they traverse the router. The packet delay is composed of the profiler NIC send/receive overheads, two packet transmissions, and the intra-device delay of the device being profiled (the router in this case). If the PC can generate and receive packets at a high rate and the NIC overheads are nearly constant with low variance, then it is possible to measure packet delays through the router.

Our profiling system comprises a traffic generator, the Click modular router, and a custom network device driver. These components are described in detail below.

Traffic Generator. We have made several changes to ns-2 to allow sending and receiving packets at extremely high rates, as well as sending and receiving spoofed IP addresses. These changes make it possible to have many flows with distinct IP addresses enter and depart from the simulator. From the router point of view, the profiler appears as a collection of various subnets with unique flows between them. We selected ns-2 since 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 de-

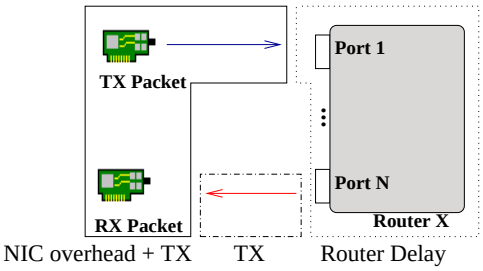


Fig. 15. Packet delay in a system composed out of a commodity PC with two networks cards and a router.

bugging.

The ns-2.30 simulator [Breslau et al. 2000] 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 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 [Deri 2004]. Second, it is not possible to spoof IP addresses in order to create an entire subnet with distinct flows on the same socket.

To tackle these two challenges, we have bypassed the Linux IP stack and created two devices that we call *ClickToUser* and *UserToClick*. 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 *ClickToUser* at over 800 KPackets/s (Kpps). Similarly, *UserToClick* can sustain high rates.

To measure the packet delay, 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). Since 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.

Fig. 16 demonstrates the logical layout of our profiler connected to a 2-port router. A Symmetric Multiprocessing (SMP) multi-NIC PC is used to emulate two subnets via ns-2 and the kernel level Click modular router. The router that is being profiled is configured to route between the two subnets.

To boost real time performance, we have made several changes to allow asynchronous I/O to take advantage of multiple CPUs. Fig. 17 demonstrates the architecture of asynchronous I/O that we have added to the simulator. 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 similar to ns-2.30 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. Since the default ns-2 is single threaded, we took careful steps to avoid race conditions, while minimizing

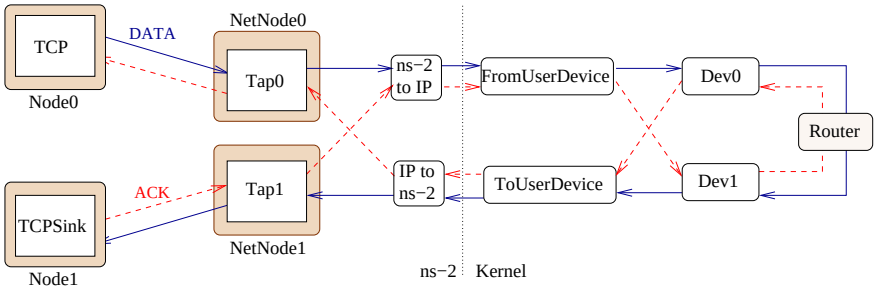


Fig. 16. Example of a single TCP flow from the simulator into the network and back.

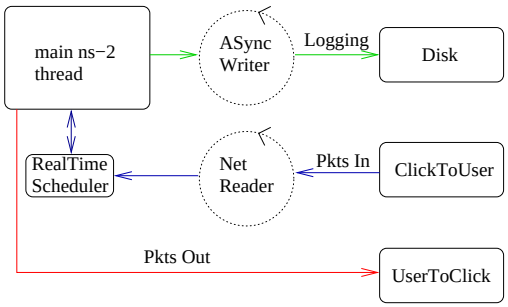


Fig. 17. Relationship between I/O operations and threads in the simulator.

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 newly arrived packets in the near future. When the main simulation thread dispatches the newly arrived packets, these packets are injected into the simulator.

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 very 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.

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 IP addresses 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 [Kohler et al. 2000] kernel module. In Click, it is easy to create a mapping of IP addresses to real devices as shown in Fig. 16. In order to attach virtual subnets to a particular device, we use a *FromUserDevice* element per device, and the user application writes IP packets into the correct *FromUserDevice* element depending on the configuration. We also modified Click's *ToDevice* element to avoid transmit buffer drops. The default Click *ToDevice* element can schedule packets faster than the device can transmit, but we hold the packets until the transmit buffer starts draining.

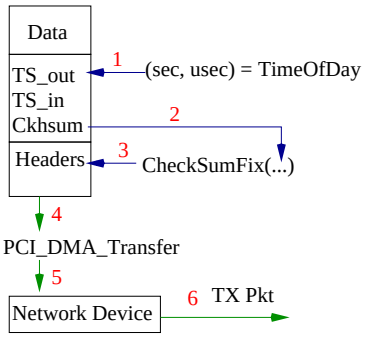


Fig. 18. Time-stamping of packets during a transmit. Time-stamping during a receive is similar, except the flow is reversed with checksum correction being the last step.

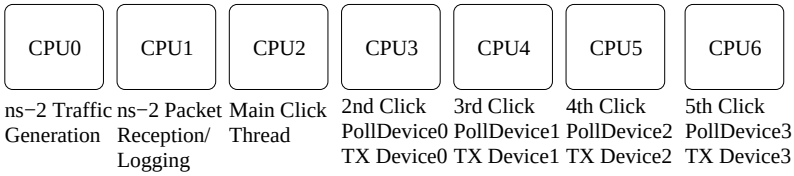


Fig. 19. Concurrent threads of execution

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

Device Driver. When a packet to be transmitted arrives, we time-stamp it just before it is sent to the device via a bus transfer. Since changing the packet payload will result in a corrupted TCP or UDP checksum, we must recompute the checksum. We compute incremental checksums to avoid computing an entire checksum from scratch. Fig. 18 demonstrates this process. Packet reception is performed in a similar fashion.

A.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. We use seven concurrent threads of execution assigned to seven of the CPUs to achieve precise results. Fig. 19 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 simulation agents and writes packets to *FromUserDevice* elements. Auxiliary ns-2 threads are required for packet reception and logging data to disk; otherwise, the main simulation thread becomes I/O blocked.

The Click modular router must also have at least two threads of concurrent execution. The main Click thread is responsible for all elements in the Click configuration except

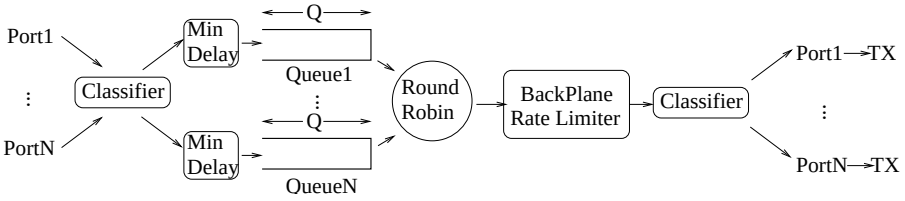


Fig. 20. Simplified model with a queue of size Q slots per port. Before being transmitted, packets go through a backplane rate limiter.

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*. Since we have four ports in our experiments, we need seven CPUs as depicted in Fig 19.

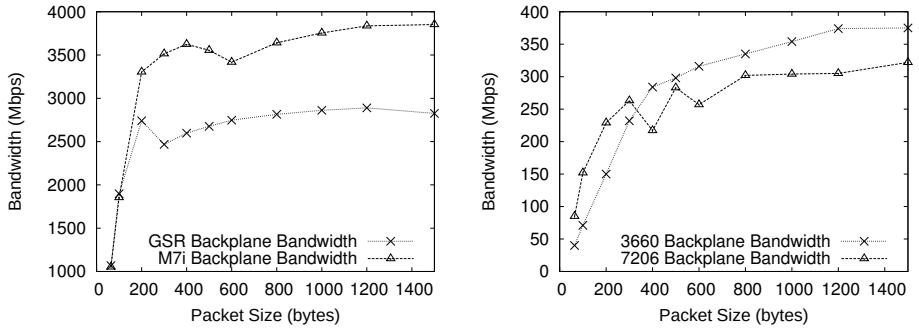
Observe that is possible to run *several* simulation instances simultaneously, by setting routing policies which forward packets into appropriate simulation instances. We use this feature for scenarios when a single simulation instance cannot generate sufficient load.

A.3 Simplified Model

Clearly, the general multi-server/multi-queue model is more complex than the VOQ-based model [Hohn et al. 2004]. To investigate the need for this additional complexity, we also devise a simplified model, which is a hybrid between the VOQ-based model and our general multi-server/multi-queue model. Fig. 20 illustrates this simplified model. As in the VOQ-based model, the packets experience *intra-device delays* prior to entering the queue. These delays are obtained from the *DelayTbl* table. If the device is congested, the packets directly proceed to the queues. The queues have the same structure as in our general multi-server/multi-queue model with the same parameters Q and *QueueReq*. However, packets from t he queues pass through a backplane limiter, which is intended to bound the aggregate packet rate through the device. The backplane limiter mimics a simple link with finite bandwidth capacity. This simple approximation abstracts complex traffic interactions in the switching fabric. Once a packet traverses the backplane limiter, it enters an output port. Contention at the backplane limiter will result in queue build-ups. If the backplane limiter determines that some output port is busy, it will not direct any packets to that port until it becomes free.

The primary difference between the simplified and the general model is the structure of the backplane. The simplified model uses a simple rate limiter, which suffices for modeling high-end routers. Using a simple rate limiter may be insufficient in cases when the packet processing overhead prevents the aggregate rate for some packet size from reaching the bandwidth limit.

For the simplified model, we have created two ns-2 modules: *RouterQ2* and *BackPlane*. *RouterQ2* is derived from *RouterQ* (Section 2.3) with the difference that it ties into the *BackPlane* module instead of *RouterS*. Packets arrive at *RouterQ2* and are queued. The



(a) 1 Gbps routers (b) 100 Mbps routers

Fig. 21. Backplane bandwidth for four routers.

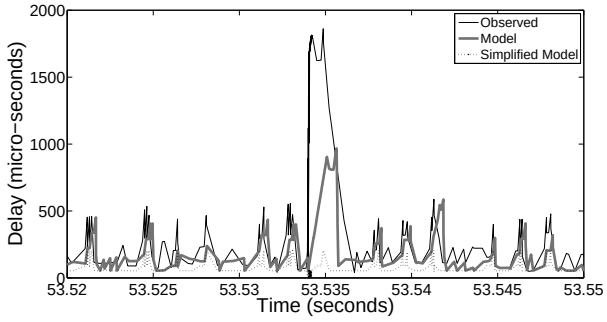


Fig. 22. General model vs. simplified model on non-congested port0 of Cisco 3660

BackPlane module acts as a link with a certain bandwidth, governing the speed at which the queues can direct packets to output ports. If the output ports are busy or the *BackPlane* is blocked, queue build-up occurs.

Simplified Model Parameters. The simplified model uses the same values as those inferred for the general multi-server/multi-queue model for *DelayTbl*, *Q*, and *QueueReq* (Section 4.1). The only new parameter is the backplane bandwidth parameter. We infer this value by observing the maximum aggregate throughput with maximum-sized frames through the device. Fig. 21 gives the values for the four routers. As expected, using a single value to limit the backplane bandwidth is insufficient as the difference between large and small packets increases.

Model Comparison. We repeated all our experiments with the simplified model. Not surprisingly, predictions from both models were similar for the two Gigabit routers. This is because the scenarios we used did not stress the backplane capacity of the routers, and both models have exactly the same queue properties. We observed similar results on the

100 Mbps routers for the ports which were heavily congested. In these situations, queuing delay dominates any delay caused by backplane contention. Since the queues are the same, the loss ratios are similar as well.

The primary difference between the two models arises in cases when backplane contention occurs. Fig. 22 demonstrates the difference between the two models for the Cisco 3660 router on non-congested port0, with the high-load TCP and HTTP experiment. The results indicate that the simplified model underestimates delays due to backplane contention. This is not surprising, as Fig. 21(b) demonstrates a non-constant relationship between the backplane bandwidth and packet size. In such situations, the additional complexity of our general model increases simulation fidelity.