A Black-box Router Profiler

Roman Chertov, Sonia Fahmy, Ness B. Shroff
Purdue University

***Abstract*— Simulation, emulation, and wide-area testbeds exhibit different strengths and weaknesses with respect to fidelity, scalability, and manageability. Fidelity is a key concern since simulation or emulation inaccuracies can lead to a dramatic and qualitative impact on the results. For example, high-bandwidth denial of service attack floods of the same rates have very different impact on the different platforms, even if the experimental scenario is supposedly identical. This is because many popular simulation and emulation environments fail to account for realistic commercial router behaviors, and incorrect results have been reported based on experiments conducted in these environments.**

In this paper, we describe the architecture of a black-box router profiling tool which integrates the popular ns-2 simulator with the Click modular router and a modified network driver. We use this profiler to collect measurements on a Cisco router. Our preliminary results demonstrate that routers and other forwarding devices cannot be modeled as simple output port queues, even if correct rate limits are observed. We discuss our future work plans for using our data to create high-fidelity network simulation/emulation models that are not computationally prohibitive.

***Keywords*— simulation, emulation, testbeds, router modeling, router benchmarking**

I. Introduction

Popular network simulators such as ns-2 [5] model any router with no Quality of Service (QoS) support using a single queue for every output port. The input port and switching fabric are assumed to incur no losses and introduce no processing delays. This simple model can significantly impact the fidelity of results when this router is a bottleneck in the simulated network. Discrepancies between the simulated and deployment behaviors can be especially large for security experiments (e.g., denial of service), high bandwidth traffic (e.g., IPTV) scenarios, and network planning/dimensioning experiments (e.g., ISP upgrades). Our previous results with low-rate TCP targeted denial of service attacks (reported in [6]) demonstrate that seemingly identical tests on various testbeds and on the ns-2 simulator produce very 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) [2] that change in complex ways according to the characteristics of the workload they are subjected to.

In commercial simulators such as OPNET [17] and OMNeT++ [1], detailed and complex models of routers, switches, servers, protocols, links, and mainframes are provided. However, the model base needs to be constantly built and validated, and using complicated models significantly increases computational cost, hindering scalability.

With network emulation, setups range from emulating large segments of the network [20], [26] or just artificially shaping a single link [14]. However, current work in emulation is focused on connectivity, delays, and link capacities. Critical properties of Internet forwarding devices such as latencies, maximum packet forwarding rates, policies, and queue sizes are not accurately incorporated, thus reducing the fidelity of the experiments that can be carried out on emulation testbeds. These properties are crucial when dealing with low-to-mid level routers. Compared to core routers, low-to-mid level routers are more performance limited, yet, due to cost, they constitute the majority of the forwarding devices in Internet edges and enterprise networks, and this is where most losses in today's Internet occur. Accurately modeling these devices is especially important in the case of experiments with resource-based attacks, since resource consumption models used in simulators and emulators are not representative of today's commercial routers [6].

To address these fidelity issues in both simulation and emulation, we propose to empirically develop models of real routers and packet forwarding devices (e.g., a variety of Cisco and Juniper routers). The construction and the validation of our models will be concurrently performed in controlled lab experiments to reduce modeling inaccuracies. These models can then be incorporated into simulators such as ns-3 (currently under development), and testbeds such as Emulab [25], DETER [9], and VINI [3].

Previous efforts to understand and profile routers and other devices using black box benchmarking, e.g., [18], [4], [16], [12], have been conducted in limited settings. Our router models will achieve higher fidelity by reflecting the specifics of the devices under diverse conditions. However, a model that is complex and difficult to validate is not useful. Hence, our model must meet the following requirements: (i) the model derivation process is the same regardless of the device; (ii) the model is dynamic, reflecting load changes; (iii) model parameters are derived from actual devices under black box testing; (iv) the model is accurate, but is allowed to miss special cases for the sake of scalability; and (v) the model is not computationally too expensive. However, before we can develop these models, we need a data acquisition system. This paper focuses on the architecture of such a system which we refer to as the *Black-Box Profiler* or BBP. BBP was designed with the following considerations in mind: *simplicity, flexibility,* and *capability of high performance.* Our preliminary experiments with BBP underscore the need for accurate router models, and demonstrate the feasibility of developing such models.

The remainder of this paper is structured as follows. Section II summarizes related work. Section III gives an overview of our BBP system. Sections IV and V present the details of our system. Section VI provides the details of our test setup. Section VII discusses our results. We conclude in Section VIII.

– This research has been sponsored in part by NSF/DHS grant 0335247 and NSF grant 0523249.

– Roman Chertov and Sonia Fahmy are with the Department of Computer Science, 305 N. University St., West Lafayette, IN 47907–2107, USA. Tel: +1-765-494-6183. Fax: +1-765-494-0739, E-mail: {rchertov,fahmy}@purdue.edu. Ness B. Shroff is with the School of Electrical and Computer Engineering, 465 Northwestern Ave., West Lafayette, IN 47907–2035, USA. E-mail: shroff@ecn.purdue.edu

II. Related Work

Traffic generation, emulation, and black box testing are required for black box profiling, and hence we summarize related work on these topics in this section.

A. Traffic Generation

The Harpoon [19] traffic generator uses flow data collected by Cisco routers to create replay flows. The generated flows do not use *live* TCP stacks. Creating highly configurable *live* (i.e., *closed-loop*) traffic is important for our purposes. One of the earliest network simulation-emulation tools was VINT [11] – a part of ns-2. We could not directly use the ns-2 emulation code as it does not support sending/receiving spoofed IPs (required for subnet emulation on a single node), and it is data-rate limited. A recent effort to extend emulation in ns-2 was reported in [15]. However, the system was not built to handle very high data rates and extensive packet logging with micro-second precision, which are important for our measurements. A commercial alternative to generating live TCP traffic is the IXIA-400T traffic generator [13]. IXIA devices use a proprietary OS and do not allow changing the types of the TCP stacks, however.

B. Network Emulation

Testbeds such as Emulab [25] and DETER [9] have the advantage of configuring variable sized network topologies via VLAN capable switches. Such testbeds can be used to create multiple subnets connected to a router. Then, either a PC or a DAG card [10] can be used to log and time-stamp the traffic. We did not use these testbeds because the profiling results when using them would include the delays from these switches connecting testbed nodes. The delays on these switches can also vary due to the load from other testbed users, which would make our results incorrect.

NCTUns [22] is a powerful simulator with sophisticated emulation capabilities, using hooks into the Linux kernel in order to use as much of the OS code as possible to transmit and capture packets. NCTUns relies heavily on *ipfilter* and *tun/tap* devices that are provided with the Linux kernel [23]. Since the NCTUns hooks are tailored to the FedoraCore4 2.6.11 kernel, it is difficult to use another Linux distribution. The use of the default Linux IP stack also adds significant overhead and complexity to the configuration, and is less flexible than the Click modular router [14], which we thus decided to use in our BBP.

C. Black-box Testing

Black-box router testing is described in [18], [4], [16]. In [18], a router was profiled with a focus on measuring its reaction times to OSPF routing messages. RFCs 2544 [4] and 2889 [16] describe the steps to determine the capabilities of a router (e.g., forwarding rate). The RFCs only discuss using homogeneous traffic for profiling, and do not discuss creating models based on measurements.

Derivation of a router model from empirical observations is discussed in [12]. The work derived simple queuing models, but was not designed to handle loss events, and ignored interactions at the input ports. In that work, a production Tier-1 router was used. While this ensures that the router configuration and traffic are highly realistic, repeatability is not possible in a production setup. Times-tamping was performed with GPS synchronized DAG cards [10]. Such devices are very accurate but they increase the setup cost and complexity.

III. System Overview

Figure 1 demonstrates the layout of our BBP infrastructure connected to a 2-port commercial router. A Symmetric Multiprocessing (SMP) multi-NIC PC is used to emulate subnets that multiple flows can traverse. The router that is being profiled, e.g., a commercial Cisco or Juniper router or a programmable router, is configured to provide routing between the subnets, and hence switches every packet that traverses the subnets. To minimize the measurement error, the BBP system is directly connected to the router. As discussed in Section II, it would be possible to use a switch or integrate BBP into an Emulab-type network, but at the cost of an increase in measurement errors.

To create responsive (i.e., closed-loop) traffic, we currently leverage the ns-2 simulator which provides various TCP stacks and traffic workload models. We plan to extend our traffic generation capabilities by reproducing application workloads based on real-life traces as in [21], [24]. Our custom additions to ns-2 allow packets from ns-2 to be injected into the test network and vice versa. Since all the packets originate and terminate on the SMP PC, we can embed arrival/departure time-stamps into packet payloads with micro-second precision, without worrying about clock skew/synchronization. The time-stamping of packets occurs in the network device driver to get an accurate estimate of the delay. Additionally, we can provide very accurate accounting per packet and per flow to determine delay, loss, reordering, and corruption. The router configuration and performance are implicitly captured in the collected data. To derive accurate device models for next generation simulators/testbeds (e.g., ns-3, DETER, VINI), we plan to collect measurements for various Cisco and Juniper routers, and create prediction equations via regression analysis.

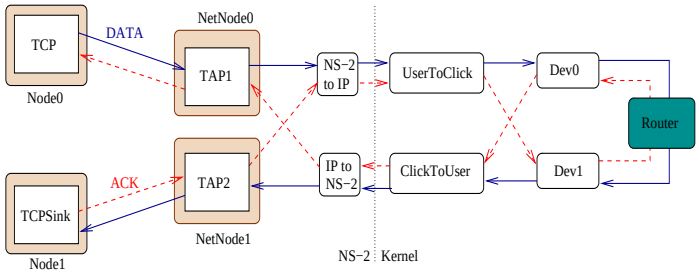


Fig. 1. Example of a single TCP flow from the simulator into the network and vice versa.

IV. ns-2 Integration

We use the ns-2 simulator [5] for traffic generation since it provides several TCP implementations that have been validated by the community. Further, ns-2 provides excellent capabilities for logging and debugging. In order to use ns-2, we had to make a number of changes to the simulator as follows.

A. Emulation

The latest version of ns-2.30 [5] 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 [8]. Second, it is not possible to spoof IP addresses in order 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 *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 remedy the difficulty with spoofing, we have created our own set of emulated objects. Figure 1 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 *UserToClick* device. The incoming network object reads from the *ClickToUser* 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.

B. 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 [15].

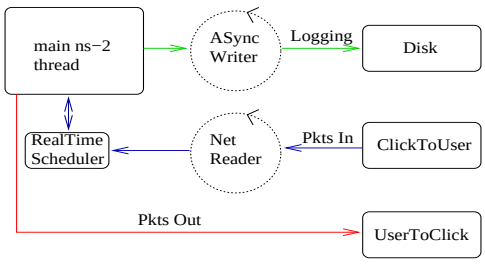


Fig. 2. Relationship between I/O operations and threads in the simulator.

Figure 2 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 very 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 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.

Since we collect information about all packets, every tap object collects information about incoming and outgoing 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.

C. Real-Time Scheduler

The default real-time scheduler was inadequate for our purposes since it is based on a calendar structure and is not MT-safe. Our tests have demonstrated that the Splay scheduler provided with ns-2.30 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. To further increase the performance of the scheduler, we have added 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 [15], we did not use the **RDTSC** assembly instruction to reduce the overhead of calling *gettimeofday*. Since our machine running BBP has 4 CPUs, calling **RDTSC** could have resulted in non-monotonically increasing time-stamps.

Our modified version of ns-2 can now process a 70 Kpps UDP flow that originates and terminates in the simulator. This means that the system manages 140 Kevents per second in real time. Additionally, every packet that leaves and enters the simulator is logged to disk. We believe that this number would be much higher if ns-2 were decoupled from TCL. However, this is a significant undertaking and hence we decided not to proceed with it at this time. The memory footprint of our modified ns-2 is similar to that of a non-modified ns-2, according to the *top* utility.

V. OS Modifications

Generating traffic for collecting measurements is only half of our task. In order to collect measurements with micro-second precision, we had to make a few changes to the operating system.

Linux Configuration. We used Linux 2.6.16.22 kernel and configured the timer to run at 1000 Hz to increase clock resolution. We also selected the option to enable high precision clock reporting. Finally, to avoid measurement problems, we disabled APM/ACPI and CPU scaling.

Device Driver. Since we aim to measure packet delays in the router under test and not in our system, we had to modify the device driver. This is as close as we can get to the point where the packets get transmitted or received without requiring a specialty card. Figure 3 demonstrates the steps we take to time-stamp packets in the device driver.

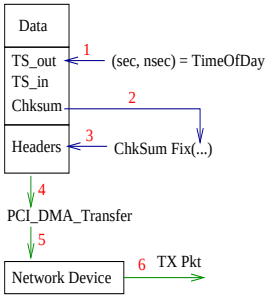


Fig. 3. Time-stamping of packets during a transmit. Time-stamping during a receive is similar, except the flow is reversed with checksum correction being the last step.

When a packet 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 recompute a new checksum. To avoid the overhead of computing an entire checksum from scratch, we embed partial checksums into the payload. This allows us to only compute the checksum of the modified region and then add it to the partial checksum to obtain a correct checksum. Packet reception is done in a similar fashion.

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 kernel module. In Click, it is easy to create a mapper of IPs to real devices as shown in Figure 1. In order to attach virtual subnets to a particular device, we have created a source-based routing element. When packets arrive, we simply direct them into a *ClickToUser* element. In case we need to run multiple *ClickToUser* elements, we can route incoming packets by destination.

VI. Test Network Setup

Figure 4 demonstrates the test setup which we have used for *both* simulations and profiling experiments. In the profiling experiments, *Node0*, *Node1*, *NetNode0*, and *NetNode1* are logical nodes on the same PC, while the "Router" is either a cross-over cable that connects two cards on the PC, or an actual Cisco 3660 router. The Cisco router under test in our profiling experiments has only *two Fast Ethernet ports*. The Cisco router was configured with minimal settings to ensure that forwarding between the ports would happen on a "fast path" without special processing. The cross-over cable configuration is used solely for calibration, in order to determine the latencies due to the network cards. The queue size for all the links has been set to 50 slots; however, in the profiling experiments the queue sizes of links going to and from the router are dictated by the particulars of the hardware. We use a PC with quad 1.8 GHz Xenon CPUs and PCI-E Intel Pro cards to run BBP on.

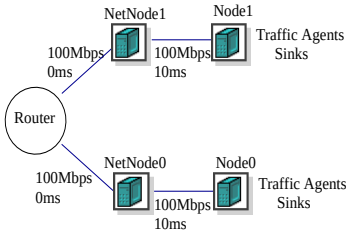


Fig. 4. Test topology with two different subnets.

Calibration. Before we can proceed with data collection, we must determine which network device configuration would give the best performance and induce the least amount of noise into the measurements. This measurement noise results from the network card/bus specifics of our measurement machine. We had an option to configure polling or interrupt based packet send/receive. We can also modify the buffer sizes. During experiments with UDP traffic, we encountered no losses when using polling. When conducting TCP experiments, we noticed higher drops at the cards when using 80-slot buffers compared to using 256-slot buffers. Hence, for the rest of experiments we use 256-slot buffers and polling.

VII. Experimental Results

This section describes the preliminary results we have obtained with a single UDP flow, as well as with 100 TCP flows.

A. Single UDP Flow

Before utilizing complex traffic mixes in router measurements, we conduct a set of baseline experiments with unidirectional UDP constant-rate flows. The experiments were conducted with a cross-over cable or with a Cisco 3660 router (which has 2 ports). For each experiment, we collected statistics for 200 Kpackets. We repeated each experiment 10 times and derived statistics for *packet delay* in nanoseconds (nsec), including the *mean*, *5th* and *95th percentile* delays. The packet delay is computed as the time it takes a packet to go from *NetNode0* to *NetNode1* or vice versa in Figure 4.

Table I shows the results with a UDP flow of 92-byte sized packets. The data indicates that for all rates except 70 Kpps, the cross-over cable gives little variation in the delay. In contrast, Cisco 3660 routers produce much more noticeable *variations for all packet rates.*

Table II gives the delay results when using 1100 and 1400 byte packets at different packet rates. As in the experi-

Test Type		500 pps	10 Kpps	40 Kpps	70 Kpps
Cross-over	mean	21162	21119	22079	46000
	5th	20000	20000	20000	20000
	95th	22000	22000	26000	168000
Cisco 3660	mean	82903	87165	70806	98711
	5th	56000	57000	55000	63000
	95th	109000	111000	99000	228000

TABLE I

PACKET DELAYS FOR 92 BYTE UDP

ments with smaller packets, the variance on the Cisco router is higher. There was no loss observed in any of these experiments. This is because the Cisco 3660 router has a 107 Kpps Maximum Loss Free Forwarding Rate (MLFFR), which is higher than our highest rate of 70 Kpps. We are currently creating a special limited-capability version of our system that can operate at much higher packet rates, in order to induce packet losses.

The large variations in packet delays on the Cisco 3660 can be partially explained by examining its architecture. The Cisco 3600 family has a central CPU with a single bus and interrupt driven I/O [7]. The non-uniform scheduling of interrupts and high CPU load can result in highly varying packet forwarding times.

Test Type		1100 UDP		1400 UDP	
		500 pps	10 Kpps	500 pps	8000 pps
Cross-over	mean	105577	108283	129797	132152
	5th	104000	104000	128000	128000
	95th	107000	114000	131000	134000
Cisco 3660	mean	270272	274343	323447	341869
	5th	243000	249000	297000	328000
	95th	297000	298000	350000	354000

TABLE II

PACKET DELAYS FOR 1100 AND 1400 BYTE UDP

Corresponding experiments with **ns-2** produce *non-varying* delays equal to twice the packet transmission delay (where transmission delay equals the packet size divided by the link bandwidth), regardless of the queue size. This is expected since packets are transmitted twice between the *NetNodes*, and there is no link propagation delay or queuing delay, and ns-2 does not model processing delays.

B. Multiple TCP Flows

We now conduct experiments with 100 long-lived TCP flows. *Node0* and *Node1* in Figure 4 generate 50 TCP flows each, destined to each other. We have chosen *NewReno* with delayed ACKs and *FullTCP* agents in ns-2 to generate TCP traffic in two separate set of experiments. *NewReno* was chosen because it is a well-studied and widely deployed TCP stack. We configured the *NewReno* agents to use close to MTU-sized segments of 1400 bytes. Since *FullTCP* is similar to Reno TCP in BSD4.4, we have chosen it for comparison and kept its configuration at default. Its default segment size is 536 bytes. In both cases, the ACKs were 96 bytes to fit in our measurement payloads. We are currently modifying our system to eliminate this additional payload.

For this set of experiments, we had to slightly modify the topology in Figure 4 by changing the links between *Nodes* and *NetNodes* to 97 Mbps. We needed to do this because the generated traffic was over-running the network cards and leading to a large number of transmission drops. Each TCP experiment was run four times with a cross-over cable, and then with a Cisco 3660 router. For ns-2, we ran the experiment only once per TCP flavor, as there is no non-determinism in the simulation. Each experiment lasted for 240 seconds.

Test Type	New Reno TCP	Full TCP
Cross-over	0.00003553	0.00003004
Cisco 3660	0.0022	0.0000996

TABLE III

AVERAGE TCP LOSS RATIOS

Table III reports the average loss ratios for this set of profiling experiments. The reported values represent the average losses between *NetNode0* and *NetNode1*. In the ns-2 simulations, the drops only occurred at *Node0* and *Node1* for both TCP flavors, as long as the queue sizes for the other links were above 30. No losses were observed at *NetNode0* and *NetNode1* in the simulations. It is also interesting to note that for the Cisco 3660, there was a noticeable change in loss ratios between *NewReno* and *FullTCP*.

To report packet delays, we merge the individual runs and select data from the 150 second to 230 second mark, in order to examine the data after all the flows are past the initial slow start phase. Figure 5 presents the results[1]. Delay results in the case of an **ns-2** router are simply the sum of queuing delays and transmission delays. Clearly, the distributions derived from the physical experiments are quite different in nature. This can be attributed to the complex nature of today's routers and physical devices [2]. Not surprisingly, the choice of TCP flavor and segment size had a noticeable effect on the simulation and testbed experiments.

The differences in the delay distributions between the cross-over and Cisco 3660 scenarios indicate that it is possible to filter out the noise due to the profiler and represent the router delay distribution more accurately. We therefore plan to use our measurements to develop higher fidelity router models.

VIII. CONCLUSIONS AND FUTURE WORK

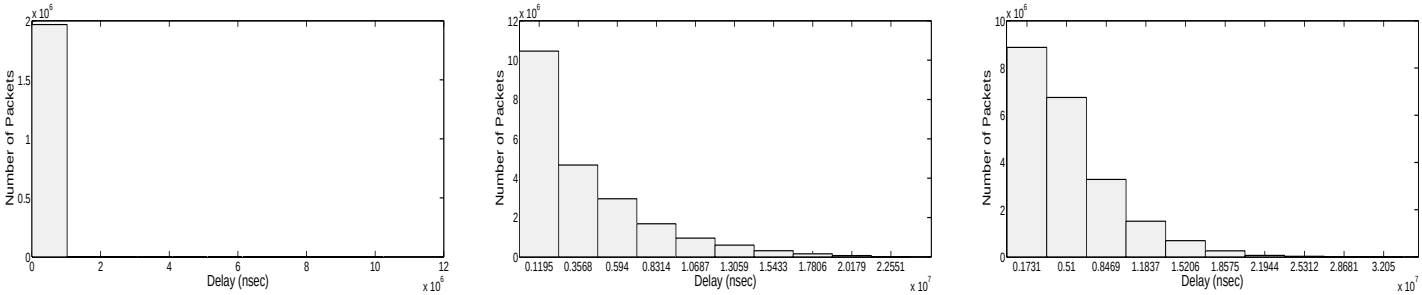
In this paper, we have described the architecture of a router profiling system which we refer to as BBP. Our system is simple, flexible, and capable of high performance. The data generated by our profiler has validated our conjecture that routers are complex devices which cannot be easily modeled as a collection of output queues, without accounting for processing delays and other device-specific bottlenecks.

In the short term, we are conducting profiling experiments with more realistic traffic mixes, more subnets, TCP SACK support, smaller measurement information added to each packet, and incremental instead of partial checksums. In the long term, we plan to use a variety of Cisco router types for experimentation. Based on our results, we plan to derive statistical models of the routers which can be used in simulators/emulators to increase the fidelity of their results. Finally, we plan to integrate our results with the ns-3 and emulation testbed development efforts.

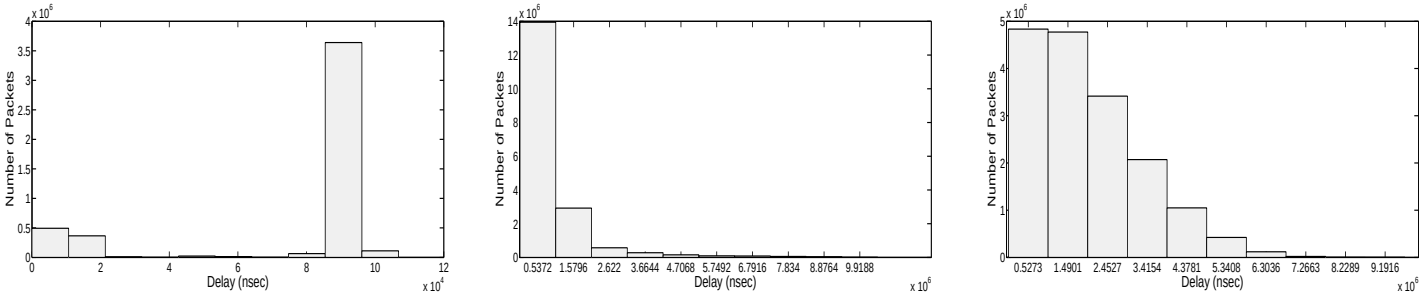
REFERENCES

[1] OMNeT++. http://www.omnetpp.org/.

[1] Please note that scales on the graphs are different.



(a) ns-2 New Reno TCP (b) Cross-over New Reno TCP (c) Cisco 3660 New Reno TCP



(d) ns-2 Full TCP (e) Cross-over Full TCP (f) Cisco 3660 Full TCP

Fig. 5. Packet delays between the NetNodes for 100 TCP flows.

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

[3] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. In *Proc. of SIGCOMM*, pages 3–14, 2006.

[4] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices. RFC 2544, http://www.faqs.org/rfcs/rfc2544.html, March 1999.

[5] 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.

[6] R. Chertov, S. Fahmy, and N. B. Shroff. Emulation versus simulation: A case study of TCP-targeted denial of service attacks. In *Proc. of TridentCom*, February 2006.

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

[8] L. Deri. Improving passive packet capture: Beyond device polling. In *Proc. of SANE*, June 2004.

[9] DETER. A laboratory for security research. http://www.deterlab.net.

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

[11] K. Fall. Network emulation in the vint/ns simulator. In *Proc. of ISCC*, pages 244–250, July 1999.

[12] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot. Bridging router performance and queueing theory. In *Proc. of SIGMETRICS*, pages 355–366, June 2004.

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

[14] 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.

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

[16] R. Mandeville and J. Perser. Benchmarking methodology for LAN switching devices. RFC 2889, http://www.faqs.org/rfcs/rfc2889.html, August 2000.

[17] OPNET. Network modeling and simulation environment. http://www.opnet.com/products/modeler/home.html.

[18] A. Shaikh and A. Greenberg. Experience in black-box OSPF measurement. In *Proc. of IMW*, pages 113–125. ACM Press, 2001.

[19] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proc. of IMC*, pages 68–81. ACM Press, 2004.

[20] 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 *Proc. of OSDI*, December 2002.

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

[22] 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. *Computer Networks*, 42:175–197, June 2003.

[23] S. Y. Wang and K. C. Liao. *Innovative Network Emulations using the NCTUns Tool*. Nova Science, 2006.

[24] 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.

[25] 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 *Proc. of OSDI*, pages 255–270, December 2002.

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