Chapter 4

Routers with Tiny Buffers: Experiments

This chapter describes two sets of experiments with tiny buffers in networks: one in a testbed and the other in a real network over the Internet2 [1] backbone.

4.1 Testbed experiments

In collaboration with the Systems and Networking Group at University of Toronto, we built a testbed network for experimenting with tiny buffers in routers.

This section explains the details of the testbed setup and discusses the results of the experiments.

4.1.1 Setup

Creating an experimental network that is representative of a real backbone network requires significant resources. Among the challenges in building a testbed network are realistic traffic generation, delay emulation, and data measurements. In what follows, we explain how each of these components are implemented in our testbed.

[1] http://www.internet2.edu/

Traffic generation. Traffic in our network is generated using the Harpoon system [46], which is an open-source flow-level traffic generator. Harpoon can create multiple connections on a single physical machine, and sends the traffic through the normal Linux network stack.

We use a closed-loop version of Harpoon [37], where clients perform successive TCP requests from servers. After downloading its requested file, each client stays idle for a thinking period, then makes another request, and the cycle continues as long as the experiment runs. Requested files have sizes drawn from a Pareto distribution with a mean of 80KB and a shape parameter of 1.5. Think periods follow an Exponential distribution with a mean duration of one second. Each TCP connection is immediately closed once the transfer is complete.

Switching and routing. We implement NetFPGA routers [5] in our network. NetFPGA is a PCI board that contains reprogrammable FPGA elements and four Gigabit Ethernet interfaces. Incoming packets can be processed at line rate, possibly modified, and sent out on any of the four interfaces. We program the boards as IPv4 routers in our testbed. Using NetFPGA routers allows us to precisely set the buffer size at byte resolution, and accurately monitor the traffic. More details on the design and architecture of NetFPGA may be found in Appendix A.

Traffic monitoring. NetFPGA is equipped with a queue monitoring module that records an event each time a packet is written to, read from, or dropped by an output queue (Appendix A). Each event includes the packet size and the precise occurrence time with an 8ns granularity. These events are gathered together into event packets, which can be received and analyzed by the host computer or another computer in the network. The event packets contain enough information to reconstruct the exact queue occupancy over time and to determine the packet loss rate and bottleneck link utilization.

Slow access links. All physical links and ports in our testbed run at 1Gb/s. To emulate slower access links, we implement the Precise Software Pacer (PSPacer) [47] package. PSPacer reduces the data rate by injecting gap packets between data

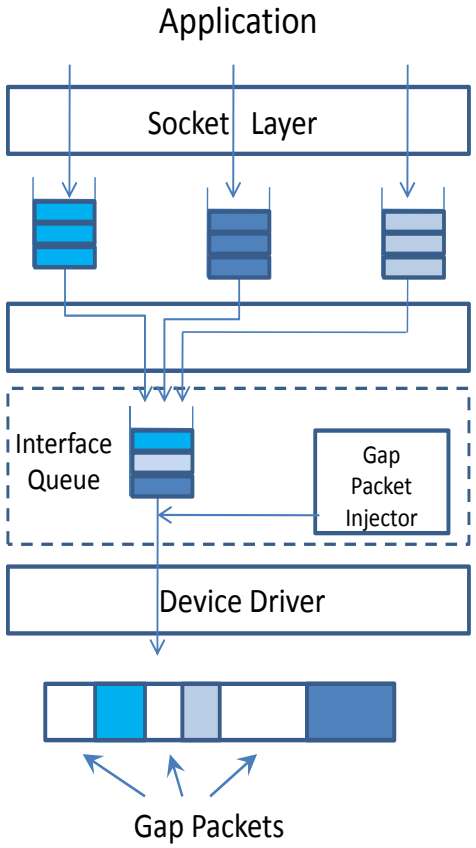


Figure 4.1: Implementation of Precise Software Pacer in the Linux kernel.

packets.

Figure 4.1 shows an overview of the PSPacer implementation in the Linux kernel. After processing of the TCP/IP protocol stack, each transmitted packet is queued in the Interface Queue associated with the output interface. When a dequeue request is received from the device driver, the Gap Packet Injector inserts gap packets based on the target rate. The transmit interval can be controlled accurately by adjusting the number and size of gap packets. If multiple access links are emulated, the size of the gap packet is re-calculated accordingly to emulate merging multiple streams into

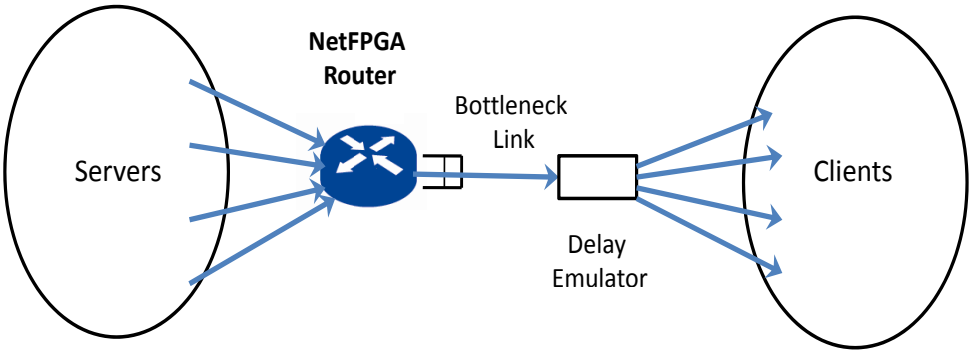


Figure 4.2: Dumbbell topology.

a single stream [47].

The gap frame uses a multicast address that has been reserved by the IEEE 802.3 standard for use in MAC Control PAUSE frames. It is also reserved in the IEEE 802.1D bridging standard as an address that will not be forwarded by bridges. This ensures the frame will not propagate beyond the local link segment.

Packet delay. To emulate the long Internet paths, we implement NIST Net [17], a network emulation package that we run on a Linux machine between the servers and the clients. NIST Net artificially delays packets and can be configured to add different latencies for each pair of source-destination IP addresses. In our experiments, we set the two-way delay between servers and clients to 100ms.

Network topology. Our experiments are conducted in two different topologies. The first, shown in Figure 4.2, has a dumbbell shape, where all TCP flows share a single bottleneck link towards their destinations. In the second topology, illustrated in Figure 4.8, the main traffic (traffic on the bottleneck link) is mixed with some cross traffic, from which it diverges before arriving at the bottleneck link. In Section 4.1.3, we study the effect of this cross traffic on the timings of packets in the main traffic.

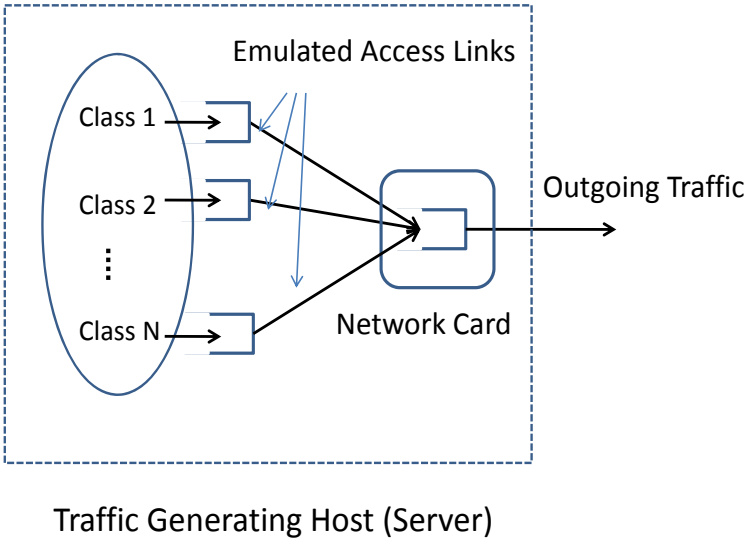


Figure 4.3: Emulating slow access links at end hosts. Flows generated on each physical machine are grouped into multiple classes, where flows in each class share an emulated access link before merging with all other flows into a single stream.

Host setup. Unless otherwise specified, we use TCP New Reno at the end hosts and set the maximum advertised TCP window size to 20MB, so that data transferring is never limited by the window size. The path MTU is 1500B, and the servers send maximum-sized segments.

The end hosts and the delay emulator machines are Dell Power Edge 2950 servers running Debian GNU/Linux 4.0r3 (codename Etch) and use Intel Pro/1000 Dual-port Gigabit network cards.

4.1.2 Results and discussion

The first set of the experiments are run in a network with dumbbell topology (Figure 4.2). We change the size of the output buffer in the NetFPGA router at the bottleneck link, and study how the buffer size affects the utilization and loss rate of the bottleneck link, as well as the flow completion times of individual flows.

The experiments are run twice; first with 1Gb/s access links, and then with emulated 100Mb/s access links. In the former scenario, PSPacer is not activated at the end hosts, and each flow sends its traffic directly to the network interface card. In the latter scenario, flows generated on each physical machine are grouped into multiple classes, where flows in each class share a queue with 100Mb/s service rate (Figure 4.3). The size of these queues are set to be large enough (5000 packets) to prevent packet drops at these queues.

Note that reducing the access link bandwidth to 100Mb/s does not relocate the bottleneck link in the network. Even though the access link bandwidth is reduced, the average per-flow throughput (i.e., the total link bandwidth divided by the number of active flows on the link) is smaller on the shared bottleneck link than on each emulated access link. In other words, the number of flows sharing each of the 100Mb/s emulated links is smaller than one tenth of the total number of flows sharing the 1Gb/s bottleneck link.

What is referred to as the *offered load* in this section is defined similarly as in Chapters 2 and 3. The offered load is the ratio of the aggregate traffic rate to the bottleneck link bandwidth if there are no packed drops and no queueing delay imposed by the network. In our experiments, the flow size distribution causes the average rate of a single flow be slightly more than 1Mb/s if there are no packet drops and no queueing delay in the network. This average rate is limited because most of the generated flows are short and do not send more than a few packets during a round-trip time. We control the offered load by controlling the number of users (clients). At any given time, only about half of the users are active. The rest are idle because they are in their think-time periods. For example, to generate an offered load of 125%, we need to have about 2400 users in the network.

The results shown and analyzed in this section are the average of ten runs. The run time for each experiment is two minutes. To avoid transient effects, we analyze the collected traces after a warm-up period of one minute.

Link utilization. Figure 4.4 shows the average utilization of the bottleneck link versus its buffer size. In the top graph, the bottleneck link is highly congested with

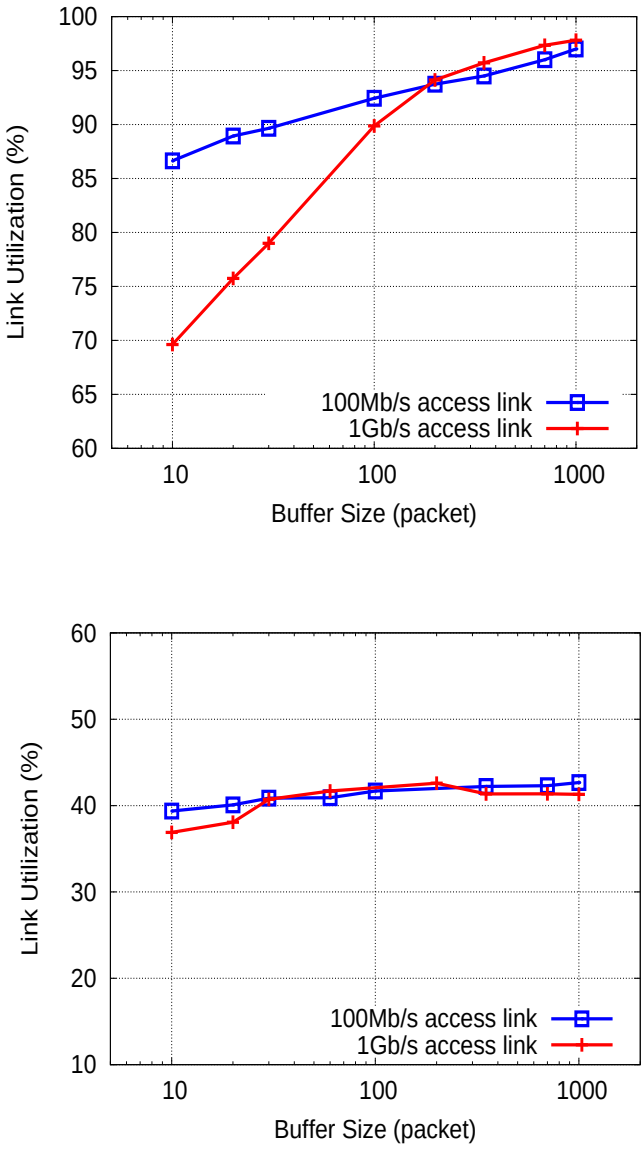


Figure 4.4: Link utilization vs. buffer size. *Top*: offered load = 125%; *bottom*: offered load = 42%.

125% offered load. With this offered load, and with 100Mb/s access links, changing the buffer size from 1000 packets to 10 packets reduces the utilization from 97% to 86%. With 1Gb/s access links, this reduction is from about 98% to 69%.

At buffer sizes larger than 200 packets, Figure 4.4 shows a slightly lower utilization with 100Mb/s compared to 1Gb/s access links. When the buffer size at the bottleneck link becomes large, the traffic load increases. With slower access links emulated at the end hosts, this increased load causes some delay before packets are sent out on output ports. Due to this difference in the round-trip times when the bottleneck link buffer is large, we cannot make an apple-to-apple comparison between the fast and the slow access link scenarios.

In a second experiment, we reduce the load on the bottleneck link to 42% by decreasing the number of users to 800. In this case, we lose only about 5% of the utilization when the buffer size is reduced from 1000 packets to 10 packets (Figure 4.4, bottom).

Drop rate. Figure 4.5 (top) shows the packet drop rate of the bottleneck link versus its buffer size. The offered load on the bottleneck link is 125%, which makes the drop rate high even when large buffers are used. With 100Mb/s access links, the drop rate increases from about 1.2% to 2.1% when we reduce the buffer size from 1000 packets to 10 packets.

The bottom plot in Figure 4.5 shows the packet drop rate versus the offered load. Here, access links run at 100Mb/s and the buffer size is fixed at 50 packets. With loads smaller than 50%, we do not see any packet drops at the bottleneck link during the runtime of the experiment.

Per-flow throughput. To study the effect of smaller buffers on the completion times of individual flows, we measure per-flow throughput (i.e., size of flow divided by its duration) of flows with different sizes. The results are shown in Figure 4.6, where each bar shows the average throughput of all flows in a given flow size bin.

The experiment is first run with a highly congested bottleneck link and very large (20MB) TCP window size (Figure 4.6, top). As expected, smaller flows are less

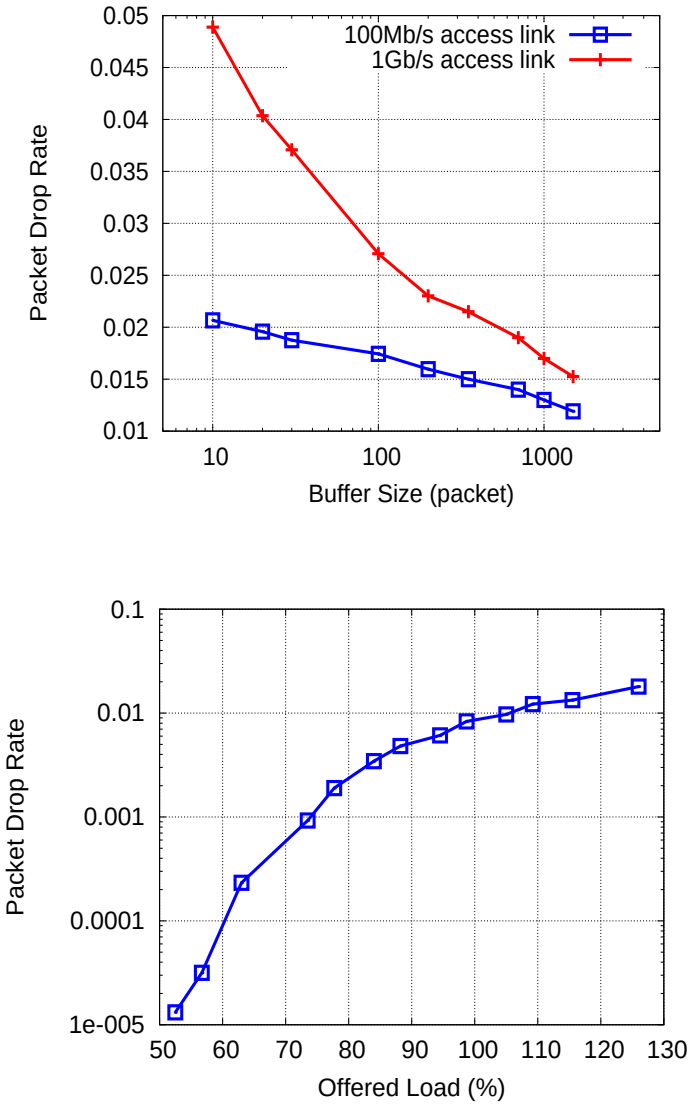


Figure 4.5: *Top*: packet drop rate vs. buffer size. The offered load is 125%; *bottom*: packet drop rate vs. offered load. Access links run at 100Mb/s and the buffer size is 50 packets.

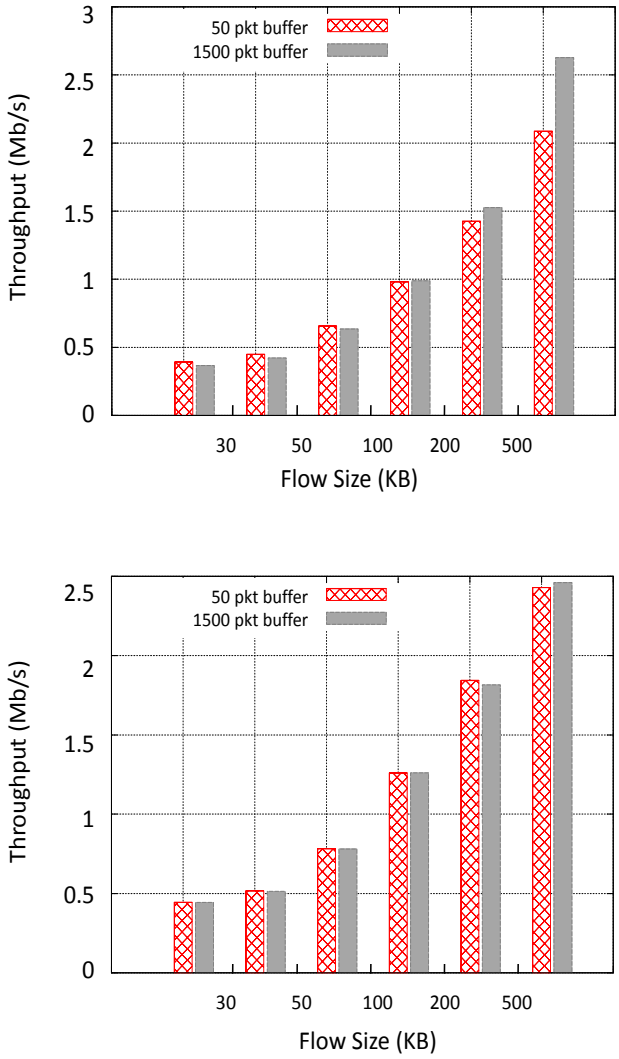


Figure 4.6: Average per-flow throughput vs. flow size. The left-most bars show the average throughput of flows smaller than 30KB. The right-most ones show the average throughput of flows larger than 500KB.
Top: offered load = 125% and $W_{\max}$ = 2MB; *bottom*: offered load = 42% and $W_{\max}$ = 64KB.

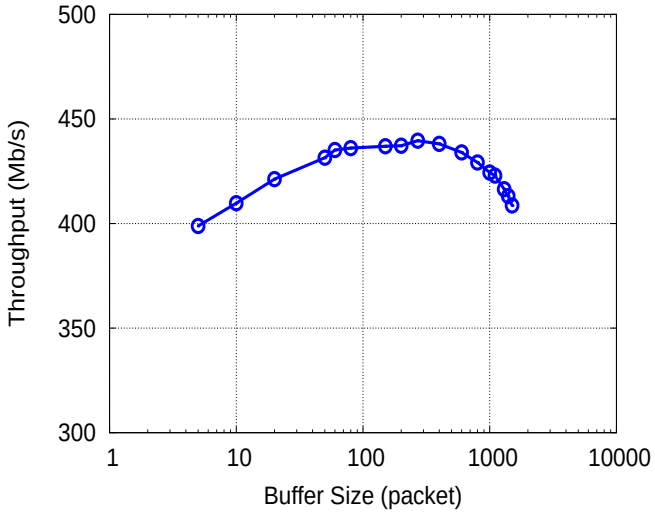


Figure 4.7: Average throughput of small flows ($\leq$ 50KB) vs. buffer size. Very large buffers add to the delay and reduce the throughput.

sensitive to the buffer size because they have smaller window sizes (even when there is no packet drop) and are more aggressive than longer flows; hence, they recover faster in case of packet drop. As the graph shows, flows smaller than 200KB do not see improved throughput with larger buffers of size 1500 packets. In fact, a closer look at the average throughput of these flows shows some throughput loss when the buffer size becomes very large (Figure 4.7), which is the effect of increased round-trip time.

The experiment is then repeated with a less congested bottleneck link and 64KB TCP window size (Figure 4.6, bottom). In this case, a smaller fraction of flows benefit from the increased buffer size. With 42% offered load, increasing the buffer size from 50 packets to 1500 packets only improves the throughput of flows larger than 500KB, and the improvement is slight. Out of all generated flows in these experiments, less than 1% fall in this category [2].

[2]Data collected from real backbone links suggests that this fraction may be even smaller in commercial networks' backbone traffic. Samples from two backbone links show that only $0.2 - 0.4\%$

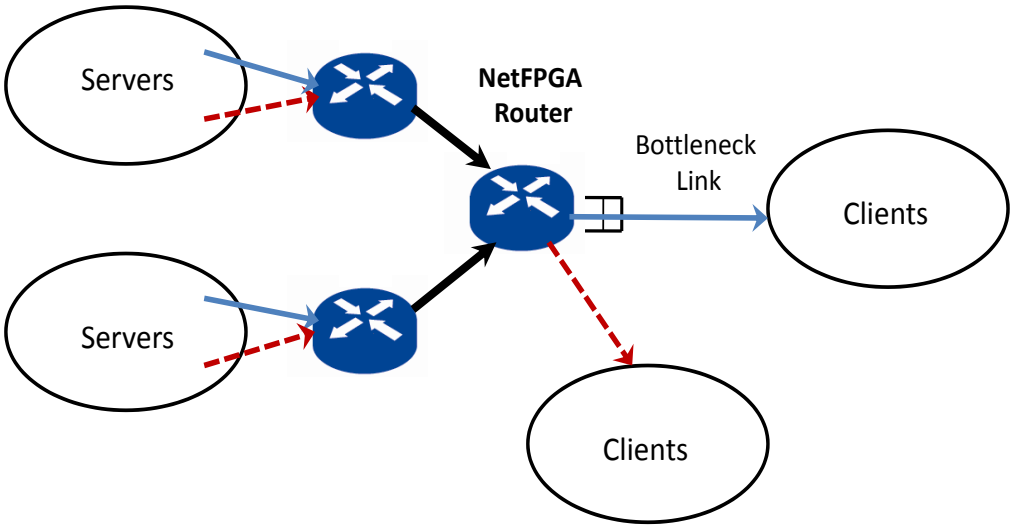


Figure 4.8: Network topology with cross traffic.

In summary, in both congested and uncongested scenarios, while large flows benefit from increased buffer size, for the majority of flows the performance is either unchanged or worse when very large buffers are used.

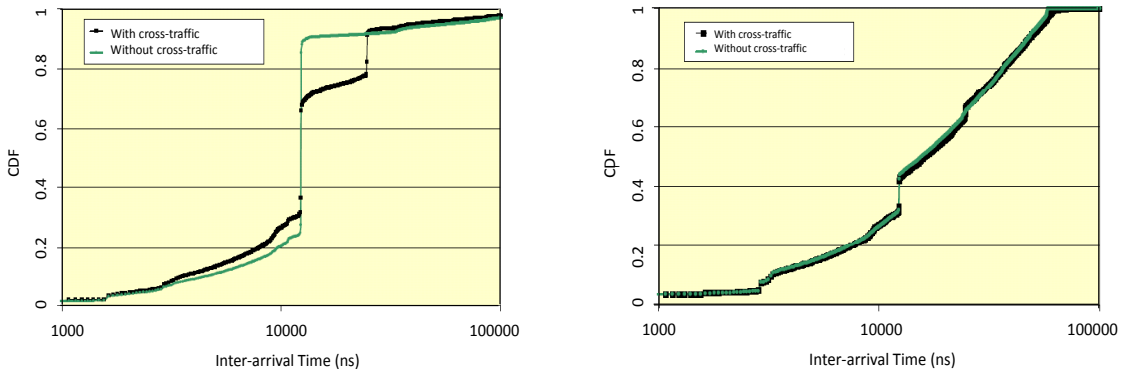
4.1.3 Cross traffic

The second set of experiments are run using the topology depicted in Figure 4.8. We examine the effect of cross traffic on packet inter-arrival times, and hence the required buffer size at the bottleneck link.

The solid arrows in Figure 4.8 show the main traffic and the dotted ones show the cross traffic, which is separated from the main traffic before the main traffic reaches the bottleneck link. There is an equal number of flows in the main traffic and cross traffic. The experiments are run first with 1Gb/s and then with 200Mb/s access links.

Figure 4.9 shows the cumulative distribution of the packet inter-arrival time on the bottleneck link, as reported by the NetFPGA router. Only packets that are stored

of flows are larger than 500KB.



(a) (b)

Figure 4.9: Cumulative distribution of packet inter-arrival time on the bottleneck link. *Left*: 1Gb/s access link; *right*: 200Mb/s access link.

in the queue are included in the statistics; dropped packets are ignored. The buffer size at the bottleneck link is set to 30 packets.

With 1Gb/s access links (Figure 4.9, left) and no cross traffic, most of the inter-arrival times are 12μs. This is roughly the transfer time of MTU-sized packets at 1Gb/s. The 30% of the inter-arrival times that are less than 12μs correspond to packets arriving nearly simultaneously at the two input links. With these fast access links, the addition of cross traffic produces a pronounced second step in the CDF.

When the main traffic comes from 200Mb/s access links (Figure 4.9, right), adding cross traffic does not have a noticeable effect on the distribution of the inter-arrival times and hence the required buffer size. The effect of network topology on traffic pattern of individual flows is theoretically analyzed in Chapter 5.

4.2 Traffic generation in a testbed

In a testbed network, a handful of computers generate traffic that represents the communication of hundreds or thousands of actual computers. Therefore, small changes to the software or hardware can have a large impact on the generated traffic and the outcomes of the experiments [10]. For the buffer sizing experiments, the validity of the traffic is paramount. The two parameters that require tuning in the buffer sizing

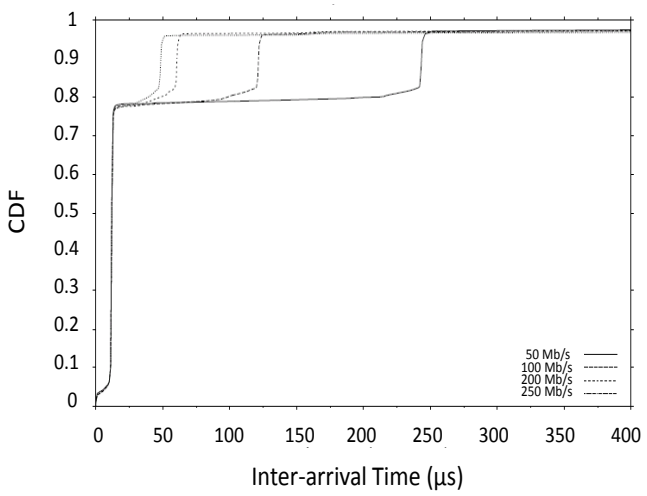


Figure 4.10: Cumulative distribution of packet inter-arrival time with TSO enabled. Even though PSPacer is implemented, the gap only appears between groups of back-to-back packets.

experiments are the following.

TCP Segmentation Offload (TSO). With TSO enabled, the task of chopping big segments of data into packets is done on the network card, rather than in software by the operating system. The card sends out the group of packets that it receives from the kernel back to back, creating bursty and un-mixed traffic.

If packets are being paced in software, TSO must be disabled; otherwise, the gap packets injected by PSPacer (described in Section 4.1.1) are only added between the large groups of packets sent to the network card (Figure 4.10). This would be different from the intended traffic pattern.

Interrupt Coalescing (IC). To lower the interrupt servicing overhead of the CPU, network cards can coalesce the interrupts caused by multiple events into a single interrupt.

With receiver IC enabled, the inter-arrival times of packets are changed. The network card will delay delivering packets to the operating system while waiting for

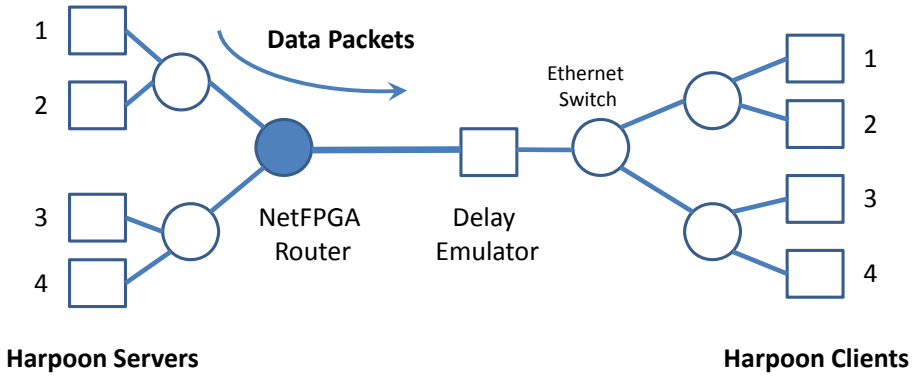


Figure 4.11: A network topology to compare Harpoon traffic generated by one pair versus four pairs of physical machines.

subsequent packets to arrive. Not only does this affect packet timing measurements, but due to the feedback in network protocols like TCP, it can also change the shape of the traffic [38].

4.2.1 Harpoon traffic

To compare the traffic generated by Harpoon to real traffic which comes from a large number of individual sources, we run a set of experiments as described below. In particular, we want to know how the number of physical machines that generate the traffic affects the mixing of flows. The results show that the aggregate traffic becomes less mixed as the number of physical machines becomes smaller.

Figure 4.11 shows the topology of the experiments. In the first experiment, we create a total number of flows between four pairs of source-destination machines. Then, we repeat this experiment by creating the same total number of flows on only one pair of source-destination machines (numbered 1 in Figure 4.11), which requires quadrupling the number of flows generated by a single machine. Multiple TCP servers on each physical machine send their traffic via emulated 50Mb/s access links (Figure 4.3) to the interface card, and from there to the NetFPGA router. In

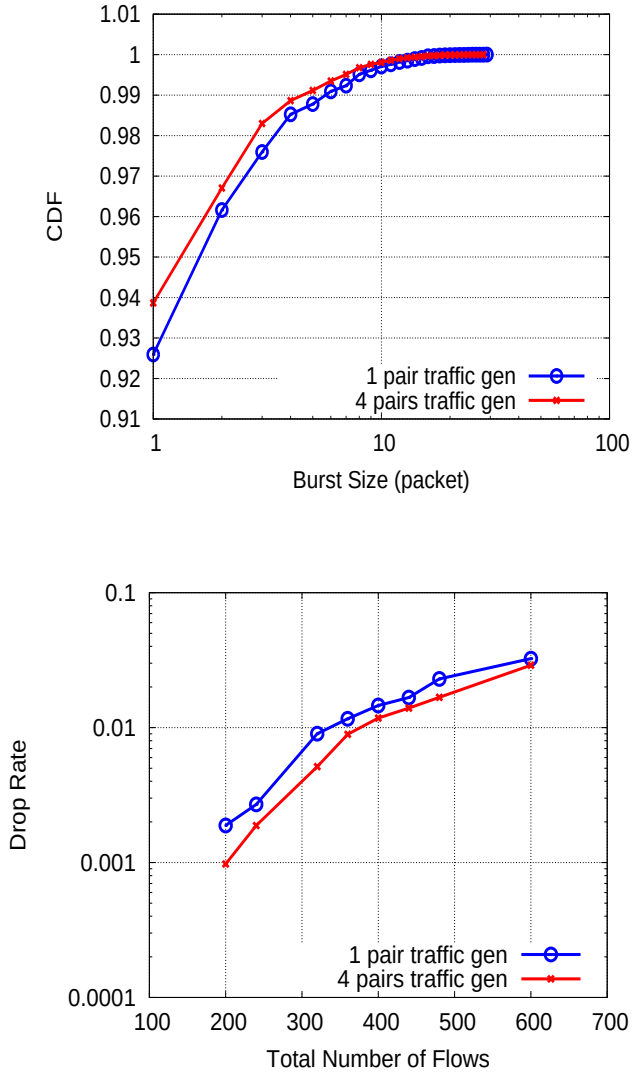


Figure 4.12: Comparison of traffic generated on two versus four hosts. *Top*: cumulative distribution of burst size in the aggregate traffic; *bottom*: packet drop rate at the bottleneck link.

both experiments, the total number of emulated access links is 16 (i.e., 4 access links per machine in the first experiment, and 16 access links per machine in the second experiment), through which the traffic is sent to the interface cards. All flows are mixed on the single link connecting the NetFPGA router to the delay emulator, which runs at 250Mb/s. We do our measurements on this shared link. The round-trip time of flows is set to 100ms, which is added by the delay emulator to the ACK packets.

Figure 4.12 (top) compares the cumulative distribution of burst size in the aggregate traffic, where a burst is defined as a sequence of successive packets belonging to a single flow. The blue curve with circular markers corresponds to the single-machine experiment. The red one corresponds to the four-machine experiment. The plot shows that when traffic is generated on four machines, packets of individual flows are less likely to appear successively in the aggregate traffic, and the burst size is smaller. The results shown in this figure are from an experiment with 400 total flows and 20-packet buffer size at the bottleneck link.

Similar results hold when we change the round-trip time (20ms-200ms), the bottleneck link buffer size (5-1000 packets), and the total number of flows (4-800); traffic is always better mixed if it comes from a larger number of physical machines. In all cases, a better mixed traffic corresponds to a smaller drop rate at the bottleneck link. Figure 4.12 (bottom) shows the drop rate versus the number of generated flows in both experiments.

4.3 Internet2 experiments

We run another set of experiments in a real network with real end-to-end delay. This is done by deploying the NetFPGA routers in four Points of Presence (PoP) in the backbone of the Internet2 network (Figure 4.13). These routers are interconnected by a full mesh of dedicated links. In these experiments, we generate traffic (using Harpoon) between end hosts at Stanford University and Rice University, and route the traffic through the bottleneck link between the Los Angles and Houston routers. We do our measurements on this bottleneck link.

Real-time buffer sizing. To determine the required buffer size in real time, we

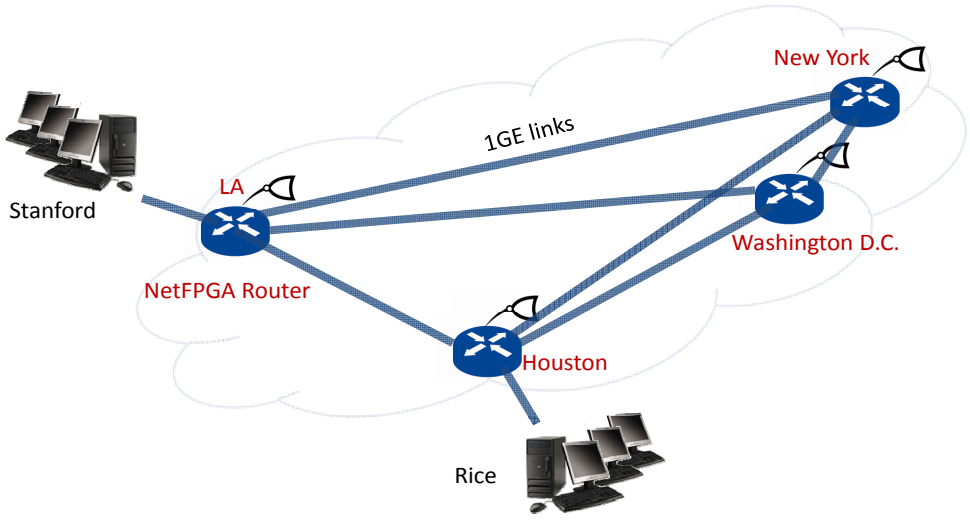


Figure 4.13: Experimental NetFPGA-based network over Internet2's backbone network.

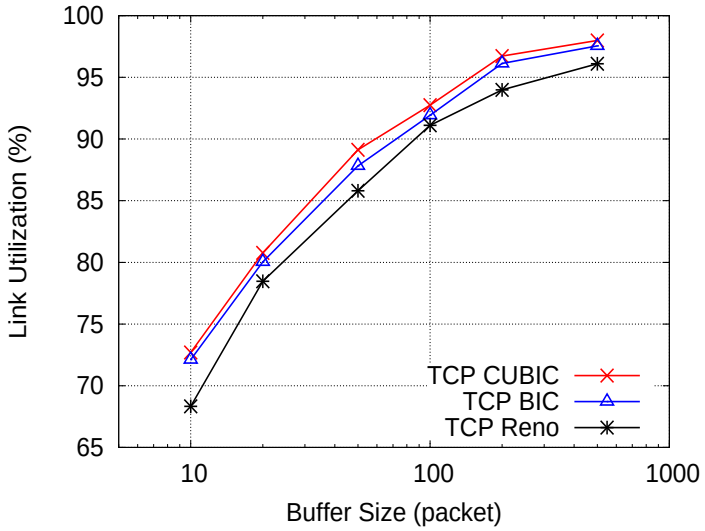


Figure 4.14: Utilization vs. buffer size with TCP Reno, BIC, and CUBIC.

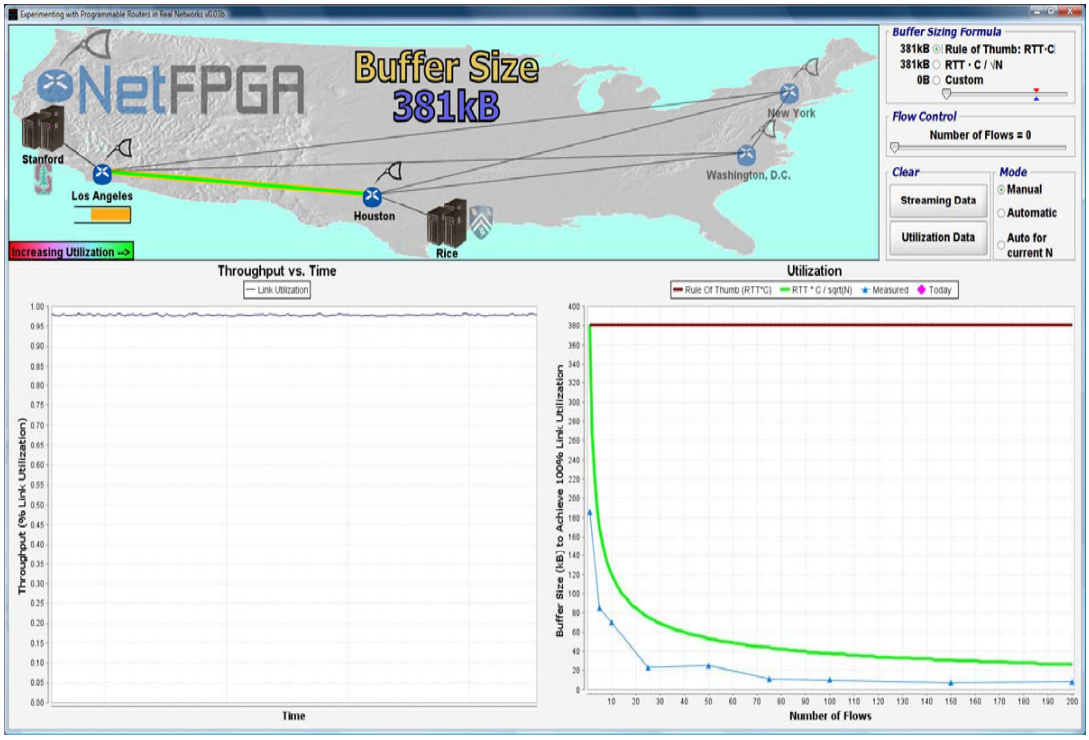


Figure 4.15: A screenshot of the real-time buffer sizing experiments.

have developed an automatic tool, which finds the minimum buffer size needed to achieve any desired link utilization. This is done through a binary search. The buffer size is set to an initial large value. In each step, after measuring the utilization over a period of three seconds, the buffer size is decreased if the utilization is within 99.5% of the desired throughout, and is increased otherwise. Figure 4.15 shows a screenshot of this real-time measurement. A control panel on the screen allows us to set the number of TCP connections between servers and clients and the buffer size at the bottleneck link. The throughput on the link, as well as the buffer occupancy and the drop rate can be monitored in real time.

Experimental results with TCP Reno implemented at the end hosts in this network are consistent with the testbed results, and will not be repeated here. Figure 4.14

compares the utilization of the bottleneck link achieved under TCP Reno to that achieved under TCP BIC and TCP CUBIC. As explained in Chapter 3, these high-speed variations of TCP are designed to improve the performance of TCP in large delay-bandwidth networks, and are more widely implemented in newer versions of the Linux kernel (since version 2.6.8). Both TCP BIC and CUBIC achieve above 80% utilization with a buffer size of 20 packets.

In these experiments, the bottleneck link runs at 62Mb/s (a rate-limiter module in NetFPGA controls the output bandwidth), and there are no slow access links emulated at the end hosts; the generated traffic is directly sent out on 1Gb/s output interfaces.