TINY BUFFERS FOR ELECTRONIC AND OPTICAL ROUTERS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Neda Beheshti-Zavareh
December 2009

This dissertation is online at: http://purl.stanford.edu/hj571fp4478

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ashish Goel

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Balaji Prabhakar

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Routers in the Internet are typically required to buffer 250ms worth of data. In high-speed backbone networks, this requirement could translate into the buffering of millions of packets in routers' linecards. This, along with the access time requirement, makes it very challenging to build buffers for backbone routers.

There could be significant advantages in using smaller buffers. Small buffers can fit in fast memory technologies such as on-chip and embedded memories. If very small buffers could be made to work, it might even be possible to use integrated optical buffers in routers. Optical routers, if built, would provide almost unlimited capacity and very low power consumption.

This work is about backbone routers with tiny buffers. Through analysis, simulation, and experiment, we show that when the backbone traffic comes from slow access links (which is the case in a typical network, as the traces collected from backbone links show), then buffers of size 10-50 packets result in over 80% throughput. We address several theoretical and practical issues in implementing tiny buffers in backbone networks—how different network conditions and load parameters affect the required buffer size, how to maintain the traffic pattern of individual flows across a backbone network, and how to build optical buffers with a minimum number of optical switches.

Acknowledgements

First and foremost, I would like to thank my adviser, Nick McKeown. He has provided invaluable guidance and motivation throughout my years at Stanford. His exceptional vision has been a constant source of inspiration for me. Nick, I consider it my good fortune to have been your student. Thank you.

I would like to thank my dissertation committee members, Prof. Ashish Goel, Prof. Balaji Prabhakar, and Prof. Nick Bambos.

I thank Prof. Ashish Goel for the many enlightening discussions I have had with him. Chatting with him has always been a relaxing experience for me.

It has been a privilege for me to work with Prof. Balaji Prabhakar during my first and second years at Stanford. I cannot say how much I have enjoyed sitting in his classes and presentations and learning from his remarkable insight.

I would like to thank Prof. Guru Parulkar, the Executive Director of the Clean Slate Program at Stanford, for his always positive and encouraging presence in our group.

It has been my pleasure to have interacted and collaborated with several researchers. I would like to thank Emily Bursmeister, Daniel Blumenthal, and John Bowers from University of California at Santa Barbara, Yashar Ganjali, Monia Ghobadi, Geoff Salmon, and Amin Tootoonchian from University of Toronto, T. V. Lakshman and Murali Kodialam from Bell Labs, Anja Feldman, Andreas Gladisch, and Hans-Martin Foisel from Deutsche Telekom Labs in Berlin.

The *Mckeown Group* at Stanford made the networking research a richer and more fun experience for me. I would like to thank former and current members of our group: Guido Appenzeller, Sara Bolouki, Martin Casado, Adam Covington, Saurav Das,

Nandita Dukkipati, David Erickson, Yashar Ganjali, Glen Gibb, Nikhil Handigol, Mikio Hara, Brandon Heller, Te-Yuan Huang, Sundar Iyer, Peyman Kazemian, Masa Kobayashi, Isaac Keslassy, John Lockwood, Jianying Luo, Jad Naous, Justin Petit, Srini Seetharaman, Rob Sherwood, Dan Talayco, David Underhill, Tatsuya Yabe, KK Yap, Yiannis Yiakoumis, Rui Zhang, and Jiang Zhu.

I would like to thank Claude Reichard, the Director of the Technical Writing Program at Stanford, who generously and despite his busy schedule helped me edit parts of this thesis. I would also like to thank Adam Covington for proof-reading parts of this thesis.

Our group administrators, Betul Buyukkaya and Hong Clark, have always been patiently helpful. I am grateful for all their help.

I would like to thank my friends in the Information Systems Lab and Computer Systems Lab. Each and every one of the them has made my time at Stanford more enjoyable and worthy.

I thank all my friends in the Persian Student Association for their company and for the joyful times that we have had together.

A very special thanks to Payman for his loving presence despite living far away. My deepest gratitude goes to my family: my brother Kamran, my sisters Roya and Elham, and most specially my parents Fatemeh and Heshmatollah. They have been a constant source of support, encouragement, and enthusiasm for me.

Contents

List of Figures

Chapter 1

Introduction

In a packet-switched network, packets are buffered when they cannot be processed or transmitted at the rate they arrive. There are three main reasons that a router, with generic switching architecture as shown in Figure 1.1, needs buffers: to store packets at times of congestion, to store packets when there is internal contention, and for pipelining and synchronization purposes.

Congestion occurs when packets destined for a switch output arrive faster than the speed of the outgoing line. For example, packets might arrive continuously at two different inputs, all destined to the same output. If a switch output is constantly overloaded, its buffer will eventually overflow, no matter how large it is; it simply cannot transmit the packets as fast as they arrive. Short-term congestion is common, due to the statistical arrival time of packets. Long-term congestion is usually controlled by an external mechanism, such as the end-to-end congestion avoidance mechanisms of TCP, the XON/XOFF mechanisms of Ethernet, or by the end-host application.

Deciding how big to make the congestion buffers depends on the congestion control mechanism; if it responds quickly to reduce congestion, then the buffers can be small; otherwise, they have to be large.

Even when the external links are not congested, most packet switches can experience internal contention because of imperfections in their data paths and arbitration mechanisms. The amount of contention, and therefore the number of buffers

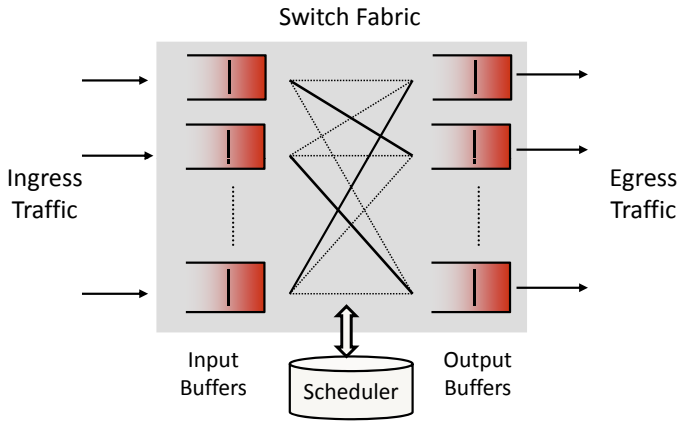


Figure 1.1: Input and output buffers in a CIOQ router. Input buffers store packets when there is internal contention. Output buffers store packets when output links are congested.

needed, is, in part, determined by the switch architecture. For example, output-queued switches have no internal contention and need no contention buffers. At the other extreme, input-queued switches can have lots of internal contention. For 100% throughput, these switches need large internal buffers (theoretically, of infinite depth) to hold packets during times of contention. Some architectures can precisely emulate output queueing [23, 21] through careful arbitration and a combination of input and output queues (CIOQ). These switches still need contention queues (at their inputs) to hold packets while the arbitration algorithm decides when to deliver each to its output queue. Most switches today use CIOQ, or multiple stages of CIOQ.

Packet switches also have staging buffers for pipelining and synchronization. Most designs have hundreds of pipeline stages, each with a small fixed-delay buffer to hold a fixed amount of data. Most designs also have multiple clock-domains, with packets crossing several domains between input and output; each transition requires a small fixed-size FIFO.

In this work, we will not be considering staging buffers; these buffers are of fixed size and delay determined by the router's internal design, not by the network.

1.1 Router buffer size

Network operators and router manufacturers commonly follow a rule-of-thumb to determine the required buffer size in routers. To achieve 100% utilization, the rule dictates that the buffer size must be greater than or equal to $RTT \times C$, also known as the *delay-bandwidth product*. Here, RTT is the average round-trip time of flows passing through the router, and C is the output link's bandwidth. This rule, as will be explained in Chapter 2, is based on the congestion control mechanism of TCP and the way transmission rate is cut off in response to packet drops in the network. The suggested buffer size is devised to ensure that buffers can stay in continual transmission, even when the sender's transmission rate is reduced. In high-speed backbone networks, this requirement could translate into the buffering of millions of packets in routers' linecards. For example, with an average two-way delay of 100ms, a 10Gb/s link requires 1Gb buffers to follow the rule-of-thumb. The buffer size has to grow linearly as the link speed increases.

Why does the buffer size matter? There are two main disadvantages in using million-packet buffers. First, large buffers can degrade network performance by adding extra delay to the travel time of packets. Second, larger buffers imply more architectural complexity, cost, power consumption and board space in routers' linecards. These issues are discussed in Sections 1.2 and 1.3.

The problem of finding the right buffer size in routers has recently been the subject of much discussion, which will be reviewed in Section 1.4. There is general agreement that while the delay-bandwidth-product rule is valid in specific cases (e.g., when one or a few long-lived TCP flows share a bottleneck link), it cannot be applied to determine the buffer size in all Internet routers.

In this dissertation, we consider routers in the backbone of the Internet. Backbone links typically carry tens of thousands of flows. Their traffic is multiplexed and aggregated from several access networks with different bandwidths that are typically much smaller than the core bandwidth. We discuss the conditions under which routers in backbone networks perform well with very small buffers. We will show that if the

core traffic comes from slower access networks (which is the case in a typical network, as the traces collected from backbone links show), then buffering only a few tens of packets can result in high throughput.

1.2 Buffer size and network performance

In a packet-switched network, the end-to-end delay consists of three components: propagation delay, transmission delay, and queueing delay.

While propagation delay and transmission delay are independent of the buffer size, queueing delay varies widely depending on the number of packets in the buffers along the path. Large buffers can potentially result in large delay and delay variations (jitter), and negatively impact the users' perceived performance. Some examples of these problems include:

- Over-buffering increases the end-to-end delay in the presence of congestion. This is especially the case with TCP, as a single TCP flow in the absence of other constraints will completely fill the buffer of a bottleneck link, no matter how large the buffer is. In this case, large buffers cannot satisfy the low-latency requirements of real time applications like video games.

  Consider a 10Gb/s link shared by flows with 100ms average round-trip time. A buffer of size $RTT \times C = 1$Gb, if full, adds 100ms delay to the travel time of packets going through this link, making it twice as large. In online gaming, a latency difference of 50ms can be decisive. This means that a congested router with buffers of size $RTT \times C$ will be unusable for these applications, even though the loss rate of the router is very small because of the huge buffers used.

- Unlike in open-loop systems, larger buffers do not necessarily result in larger throughput (or equivalently smaller flow completion time) under the closed-loop rate control mechanism of TCP. In an open-loop system, the transmission rate is independent of the buffer size, hence the throughput is only a function of the buffer's drop rate. Under the closed-loop mechanism of TCP, the average throughput over a round-trip time RTT, is W/RTT. Both RTT and W vary as

the buffer size changes. Larger buffer size means larger RTT and at the same time smaller drop rate, or equivalently larger window size. Whether we gain or lose throughput by increasing the buffer size depends on how RTT and W change versus the buffer size.

- Large delay and delay variations can negatively affect the feedback loop behavior. It has been shown that large delay makes TCP's congestion control algorithm unstable, and creates large oscillations in the window size and in the traffic rate [32, 41]. This in turn results in throughput loss in the system.

1.3 Buffer size and router design

Buffers in backbone routers are built from commercial memory devices such as dynamic RAM (DRAM) or static RAM (SRAM). SRAMs offer lower (faster) access time, but lower capacity than DRAMs.

With 100Gb/s linecards under development, it has become extremely challenging to design large and fast buffers for routers. The typical buffer size requirement, based on the delay-bandwidth product rule, is 250ms, which is equivalent to 25Gb at 100Gb/s speed. To handle minimum length (40B) packets, a 100Gb/s linecard's memory needs to be fast enough to support one read/write every 1.6ns.

The largest currently available commodity SRAM is approximately 72Mb and has an access time of 2ns [1]. The largest available commodity DRAM today has a capacity of 1Gb and an access time of 50ns [2].

To buffer 25Gb of data, a linecard would need about 350 SRAMs, making the board too large, expensive, and hot. If instead DRAMs are used, about 25 memory chips would be needed to meet the buffer size requirement. But the random access time of a DRAM chip could not satisfy the access time requirement of 1.6ns.

In practice, router line cards use multiple DRAM chips in parallel to obtain the aggregate memory bandwidth they need. This requires using a very wide DRAM bus with a large number of fast data pins. Such wide buses consume large amounts of board space, and the fast data pins consume too much power.

The problem of designing fast memories for routers only becomes more difficult as the line rate increases (usually at the rate of Moore's Law). As the line rate increases, the time it takes for packets to arrive decreases linearly. But the access time of commercial DRAMs decreases by only 1.1 times every 18 months [1][44].

However, memory dimension cannot become very small, since breaking memory into more and more banks results in an unacceptable overhead per memory bank.

There could be significant advantages in using smaller buffers. With buffers a few hundred times smaller, the memory could be placed directly on the chip that processes the packets (a network processor or an ASIC). In this case, very wide and fast access to a single memory would be possible, but the memory size would be limited by the chip size. The largest on-chip SRAM memories available today can buffer about 64-80Mb in a single chip [1]. If memories of this size are acceptable, then a single-chip packet processor would need no external memories.

If very small buffers could be made to work, it might even be possible to use integrated optical buffers in routers. Optical routers, if built, would provide almost unlimited capacity and very low power consumption.

The following section explains more about technological constraints and advances in building optical memories.

1.3.1 Optical buffers

Over the years, there has been much debate about whether it is possible (or sensible) to build all-optical datapaths for routers.

On the one hand, optics promises much higher capacities and potentially lower power consumption. Optical Packet switching decouples power and footprint from bit-rate by eliminating the optoelectronic interfaces. The results are higher capacity and reduced power consumption, and hence increased port density. Over time, this could lead to more compact, high-capacity routers.

[1]The access time of a DRAM is determined by the physical dimensions of the memory array, which do not change much from generation to generation. Recently, fast DRAMs such as Reduced-Latency DRAM (RLDRAM) have been developed for networking and caching applications. The shortest access time provided by RLDRAM today is 2.5ns at 288Mb density [3]. This architectures reduces the physical dimension of each array by splitting the memory into several banks.

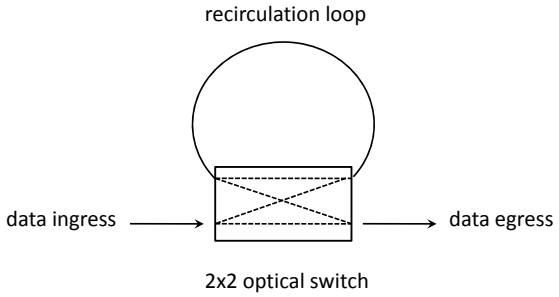


Figure 1.2: Schematic of a feed-back buffer. A 2×2 switch is combined with a waveguide loop to provide variable delay for an optical signal.

On the other hand, most router functions are still beyond optical processing, including header parsing, address lookup, contention resolution and arbitration, and large optical buffers. Optical packet switching technology is limited in part by the functionality of photonics and the maturity of photonic integration. Current photonic integration technology lags behind the equivalent electronic technology by several years [12].

To ease the task of building optical routers, alternative architectural approaches have been proposed. For example, label swapping simplifies header processing and address lookup [13, 16, 49], and some implementations transmit headers slower than the data so they can be processed electronically [35, 36]. Valiant load-balancing (VLB) has been proposed to avoid packet-by-packet switching at routers, which eliminates the need for arbitration [30].

Building random access optical buffers that can handle variable length packets is one of the greatest challenges in realizing optical routers. Storage of optical data is accomplished by delaying the optical signal either by increasing the length of the signal's path or by decreasing the speed of the light. In both cases, the delay must be dynamically controlled to offer variable storage times, i.e., to have a choice in when to read the data from the buffer. Delay paths provide variable storage time by traversing a variable number of short delay lines—either several concatenated delays (feed forward configuration) or by looping repeatedly through one delay (feedback

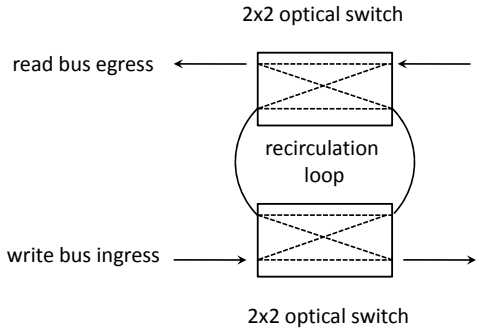


Figure 1.3: Physical implementation of speedup and simultaneous read/write.

configuration). Buffers that store optical data through slowing the speed of light do so by controlling resonances either in the material itself or in the physical structure of the waveguide.

Among the various optical buffering technologies, feedback buffers are beneficial for their low component count and small footprint [14]. The base memory element shown in Figure 1.2 can be built using two photonic chips and cascaded to form a practical optical buffer for many packets. The element is flexible in that it may be used as a recirculating feedback buffer or concatenated to form a feed forward buffer for arbitrary packet lengths. Feedback loops can store packets for a number of recirculations, whereas feed forward configurations require N loops to store a packet for N packet durations. In a feedback buffer, the length of the delay line determines the resolution of possible delays. These buffer elements can also enable easy implementation of simultaneous read/write as well as speedup. The design extension to enable a speedup of 2 and simultaneous read/write is shown in Figure 1.3.

Integrated feedback buffers as developed by Burmeister et al. [15] and Chi et al. [20] show promise of offering a practical solution to optical buffering. These recirculating buffers meet all the necessary requirements for buffering packets at high link bandwidth by providing low optical loss and fast switching time. The integrated optical buffer described in [15] achieves 64ns of packet storage, or 5 circulations, with 98% packet recovery at 40Gb/s link bandwidth.

1.4 Related work

Even though research on router buffer sizing has been done since early 1990s, this problem has only recently attracted wide interest, especially following the work of Appenzeller et al. in 2003 [9]. Since then, there has been much discussion and research on buffer sizing; it has been studied in different scenarios, and under different conditions. Some studies have concluded that the rule-of-thumb excessively overestimates the required size, while others have argued that even more buffering is required under certain conditions. Below is a brief summary of the related work.

Villamizar and Song [48] showed that a router's buffer size must be equal to the capacity of the router's network interface multiplied by the round-trip time of a typical flow that passes through the router. This result was based on experimental measurements of up to eight long-lived TCP flows on a 40Mb/s link.

Appenzeller et al. [9] suggest that the required buffer size can be scaled down by a factor of $\sqrt{N}$, where N is the number of long-lived TCP flows sharing the bottleneck link. These authors show that the buffer size can be reduced to $2T \times C/\sqrt{N}$ without compromising the throughput. For example, with 10,000 flows on the link, the required buffer size is reduced by two orders of magnitude. This follows from the observation that the buffer size is, in part, determined by the saw-tooth window size process of TCP flows. The bigger the saw-tooth, the larger must the buffers be to achieve full utilization. As the number of flows increases, the aggregate window size process (the sum of all the congestion window size processes for each flow) becomes smoother, following the Central Limit Theorem. This result relies on three assumptions: (1) flows are sufficiently independent of each other to be desynchronized (2) the buffer size is dominated by long-lived flows, and (3) there are no other significant, un-modeled reasons for buffering more packets.

In [25], Enachescu et al. show that the buffer size can be further reduced to as small as $O(\log W)$, at the expense of losing only a small fraction of the throughput $(10-15\%)$. The suggested buffer size is about $20-50$ packets, if the traffic is paced, either by implementing paced-TCP [7] at the source or by running the bottleneck link much faster than the access links. We will examine these assumptions more closely

in Chapter 2. Similar results are shown independently by Raina and Wischik [41], who study the stability of closed-loop congestion control mechanisms under different buffer sizes. Using control theory and simulations, the authors show that a system is stable with tiny buffers.

Dhamdhere et al. study a particular network example in [24], and argue that when packet drop rate is considered, much larger buffers are needed, perhaps larger than the buffers in place today. In their work, they study a situation in which a large number of flows share a heavily congested low capacity bottleneck link towards the edge of the network, and show that one might get substantial packet drop rate even if buffers are set based on the rule-of-thumb. In [39], Prasad et al. argue that the output/input capacity ratio at a network link largely determines the required buffer size. If that ratio is larger than one, the loss rate drops exponentially with the buffer size and the optimal buffer size is close to zero. Otherwise, the loss rate follows a power-law reduction with the buffer size and significant buffering is needed.

1.5 Organization of thesis

The rest of this dissertation is organized as follows. Chapter 2 gives an overview of buffer sizing rules and analyzes the utilization of CIOQ routers with tiny buffers at input and output ports. Chapter 3 presents simulation results on the impact of using very small buffers in routers, and studies the effect of various traffic and network conditions on the required buffer size. Chapter 4 describes two sets of buffer sizing experiments, one run in a testbed and another in a real network. Chapter 5 considers a network with multiple routers and explains how traffic can be made buffer-friendly and smooth across the network. Chapter 6 discusses some issues in building optical buffers. Chapter 7 is the conclusion.

Chapter 2

Buffer Sizing Rules: Overview and Analysis

As seen in Chapter 1, based on the rule-of-thumb, buffers need to be at least as large as the delay-bandwidth product of the network, i.e., $RTT \times C$, to achieve full utilization.

In this chapter, we study a CIOQ router model with contention buffers at the input ports and congestion buffers at the output ports (Figure 1.1). We first consider the congestion buffers, explain the origins of the rule-of-thumb for determining the size of these buffers, and briefly review the case with many TCP flows on the bottleneck link. We then give an overview of the *tiny buffers* analysis [25], which shows the feasibility of making the congestion buffer size only a few dozen packets. Finally, we consider the contention buffers in the CIOQ model, and show that the tiny buffers rule could also be applied to the contention buffers at the input side of the router.

2.1 How big should the congestion buffers be?

To understand how large to make the congestion buffers, we first study output-queued routers, which only have congestion buffers, and packets are immediately transferred to the output ports as soon as they arrive. Each output port has one FIFO queue, which is shared by the flows going through that port. The size of the buffer depends

on the arrival traffic: If traffic is light or non-bursty, buffers can be very small; if big bursts are likely to arrive, buffers need to be much larger.

In what follows, we explore how large to make the congestion buffers under three scenarios:

1. When a link carries just one TCP flow. This turns out to be the worst-case, and leads to the rule-of-thumb $B = RTT \times C$.

2. When a link carries many TCP flows, allowing us to reduce the buffer size to $B = \frac{RTT \times C}{\sqrt{N}}$.

3. Finally, when traffic comes from slow access networks, or when the source paces the packets it sends. In this case, we can reduce the buffer size to a few tens of packets.

2.1.1 When a link carries just one TCP flow

To understand why we need $RTT \times C$ buffers with just one TCP flow, we need to understand the dynamics of TCP. The dynamics of a TCP flow are governed by the window-size (the number of outstanding unacknowledged packets). A long-lived flow spends most of its time in the additive-increase and multiplicative-decrease (AIMD) congestion avoidance mode, during which the window size increases additively upon receiving an ACK packet, and is halved when a packet or ACK is lost.

To maximize the throughput of the network, the buffer in a router's output port needs to be big enough to keep the outgoing link busy during times of congestion. If the buffer ever becomes empty, the link goes idle and we waste the link capacity.

On the other hand, TCP's sawtooth congestion control algorithm is designed to fill any buffer, and deliberately causes occasional loss to provide feedback to the sender. No matter how big we make the buffers at a bottleneck link, TCP will occasionally overflow the buffer. Consider the simple topology in Figure 2.1, where a single TCP source sends data packets to a receiver through a router. The sender's access link is faster than the receiver's bottleneck link of capacity C, causing packets to be queued at the router. The sender transmits a packet each time it receives

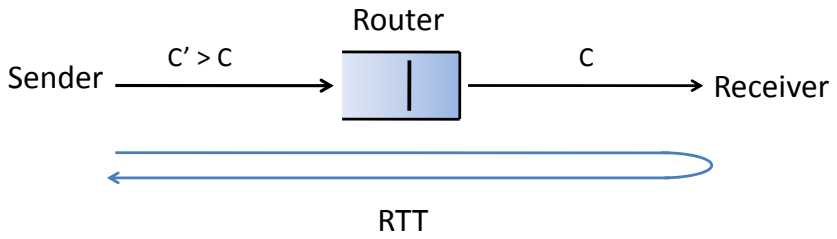


Figure 2.1: Single-bottleneck topology. The sender's access link is faster than the receiver's bottleneck link, causing packet accumulation in the router.

an ACK, and gradually increases the number of outstanding packets (the windows size), which causes the buffer to gradually fill. Eventually a packet is dropped, and the sender does not receive an ACK. It halves the window size and pauses until the number of outstanding packets has fallen to $\frac{W_{\max}}{2}$ (where $W_{\max}$ is the peak window size). Figure 2.2 shows the window size dynamics of a single flow going through a bottleneck link. The key to sizing the buffer is to make sure that while the sender pauses, the router buffer does not go empty and force the bottleneck link to go idle.

The source pauses until it receives $\frac{W_{\max}}{2}$ ACK packets, which arrive in the next $\frac{W_{\max}}{2C}$ seconds (remember that C is the bottleneck rate). During the pause, $\frac{W_{\max}}{2}$ packets leave the buffer; for the bottleneck link to stay busy, the buffer needs to hold at least $\frac{W_{\max}}{2}$ packets when the pause starts. Now we just need to determine $W_{\max}$.

At the instant the pause is over, the source can send $\frac{W_{\max}}{2}$ consecutive packets as $\frac{W_{\max}}{2}$ ACKs arrive. It then pauses until it receives an ACK one RTT later (the first ACK arrives after exactly RTT because the buffer is empty). In other words, the source sends $\frac{W_{\max}}{2}$ packets in RTT seconds, which must be just enough to keep the bottleneck link busy; i.e., $\frac{W_{\max/2}}{RTT} = C$, which means $B = RTT \times C$, the rule-of-thumb for one TCP flow.

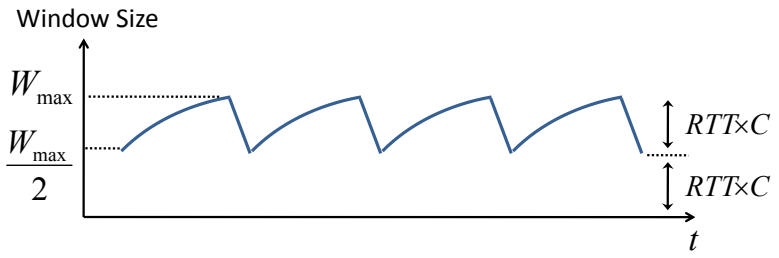


Figure 2.2: Window size dynamics of a TCP flow going through a bottleneck link. To achieve 100% utilization, the buffer size needs to be large enough to store $RTT \times C$ packets.

2.1.2 When many TCP flows share a link

If a small number of flows share a link, the aggregate window size (the sum of their individual window sizes) tends to follow the same TCP sawtooth, and B is the same as for one flow.

If many flows share a link, small variations in round-trip time and processing time desynchronize the flows [40, 29, 26]. Therefore, the aggregate window size gets smoother. This is studied in detail in [9], where it is shown that with N long-lived TCP flows, the variation in the aggregate window size scales down by a factor $\sqrt{N}$. As with one flow, the variation in the aggregate window size dictates the buffer size needed to maintain full utilization of the bottleneck link. Hence $B = \frac{RTT \times C}{\sqrt{N}}$. With 10,000 flows on a link, this suggests the buffer size could be reduced by 99% without any change in performance (i.e., a 1Gb buffer becomes 10Mb). These results have been found to hold broadly in real networks [9, 11].

2.1.3 When traffic comes from slow access networks

In backbone networks, in addition to the above multiplexing effect, each individual flow gets smoother too. This is because a backbone network interconnects many slower networks. When packets from slower networks are multiplexed together onto a fast backbone, the bursts are spread out and smoothed. Hence, in backbone networks

not only is the aggregated TCP's AIMD sawtooth smoothed, but also the underlying traffic arrivals are smoothed. We will see that the smoothing substantially reduces the required buffer size.

To get a feel for how smoothing could help reduce the buffer size, imagine for a moment that the traffic is so smooth that it becomes a Poisson process. The drop rate has an upper bound of ρ^B, where ρ is the load, and B is the buffer size. At 80% load and with only 20-packet buffers, the drop rate will be about 1%, independent of RTT and C. At the other extreme, compare this with the buffer size needed for 100% utilization with a single TCP flow, when RTT is 200ms and C is 10Gb/s; $B = 2$Gb, or about a million average sized packets.

Traffic in backbone networks cannot be modeled as a collection of independent Poisson flows, since a TCP flow can send a whole window of packets at the beginning of a round-trip time, creating significant bursts. But there are two ways the bursts can be broken. We can explicitly break them by using Paced TCP [7], in which packets are spread uniformly over the round-trip time. The rate and behavior of each flow are almost indistinguishable from regular TCP, but as we will see shortly, the amount of required buffering drops dramatically.

Even if we do not modify the TCP source, the burst is naturally broken if the core links are much faster than the access links, as they typically are. As the packets from one flow enter the core, they are spread out, with gaps, or packets from other flows being multiplexed between them.

To see how breaking the bursts reduces the required buffer size, we start by analyzing TCP pacing. Sources follow the AIMD dynamics, but rather than sending out packets in bursts, they spread traffic over a round-trip time.

Assume that N long-lived TCP flows share a bottleneck link. Flow i has a time-varying window size $W_i(t)$ and follows TCP's AIMD dynamics. If the source receives an ACK at time t, it will increase the window size by $1/W_i(t)$, and if the flow detects a packet loss, it will decrease the congestion window by a factor of two. In any time interval $(t, t']$ when the congestion window size is fixed, the source will send packets as a Poisson process at rate $W_i(t)/RTT$. Under this assumption buffering $O(\log W_{\max})$ packets is sufficient to obtain close to peak throughput. More precisely, to achieve

effective utilization of θ, buffer size

$$B \geq \log_{1/\rho} \frac{W_{\max}^2}{2(1-\theta)} \tag{2.1}$$

suffices [25], if the network is over-provisioned by a factor of $1/\rho$. Here, ρ is assumed to be less than or equal to one.

This result assumes that the network is over-provisioned. In other words, it assumes that the maximum traffic rate—with all TCP sources simultaneously transmitting at their maximum rate—is $1/\rho$ times smaller than the bottleneck link bandwidth. Here, θ is the desired effective utilization of the shared link. It represents the fraction we aim to achieve out of the maximum possible utilization ρ (i.e., a fraction $\rho\theta$ of the full link rate). Although this result has not been extended to the under-provisioned case, the simulation results of Chapter 3 indicate that over-provisioning is not a requirement.

The above result suggests that TCP traffic with $W_{\max} = 83$ packets and $\rho = 75\%$ needs a buffer size of 37 packets to achieve link utilization $\theta\rho$ of 70%.

According to Equation 2.1, the buffer size needs to increase only logarithmically as the maximum window size grows larger. In a TCP connection, $W_{\max}$ is the maximum amount of data the transmitter can send over one RTT. This amount is limited by the source transmission rate, even if the operating system does not explicitly limit $W_{\max}$: at a source rate of C_T , at most $C_T \times RTT$ units of data can be sent over a round-trip time. If this amount increases from 10KB to 100MB, then the buffer size only needs to be doubled.

What happens if instead of TCP Pacing we simply rely on the multiplexing of flows from slow access links onto fast backbone links? It is shown in [25] that if access links run at least $\log W_{\max}$ times slower than the bottleneck link, then approximately the same buffer size as in Equation 2.1 is required. In our example above, $\log W_{\max}$ was less than seven, whereas in practice access links are often two orders of magnitude slower than backbone links (for example, a 10Mb/s DSL link multiplexed eventually onto a 10Gb/s backbone link). Under these conditions, the packet loss probability is

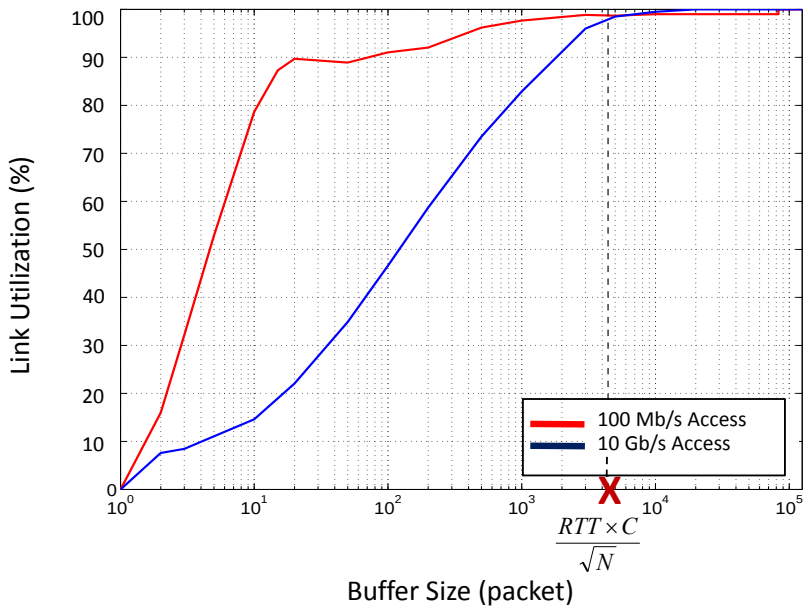


Figure 2.3: Link utilization vs. buffer size. With 800 flows on the link, close to 100% utilization will be achieved if buffer size is $\frac{RTT \times C}{\sqrt{N}}$. If flows come from slower access links, a tiny buffer size of 10 packets suffices for 80% utilization.

comparable to Poisson traffic with the same buffer size.

To compare the results of Sections 2.1.1-2.1.3, we illustrate them through the simulation of a 10Gb/s bottleneck link with 800 long-lived TCP flows sharing the link (Figure 2.3). The average RTT is 100ms. We measure link utilization as we vary the link's buffer size from only one packet to $RTT \times C = 125,000$ packets. As the graph shows, utilization remains almost unchanged (above 99%) with buffer sizes larger than $\frac{RTT \times C}{\sqrt{N}} \approx 4419$ packets. With access links running at 100Mb/s, i.e., 100 times slower than the bottleneck link, we can set the buffer size to only 10 packets and achieve close to 80% utilization.

2.2 How big should the contention buffers be?

Now we turn our attention to the size of the contention buffers.

The size of contention buffers in a CIOQ switch depends on the internal speedup of the switch (i.e., how fast the switch fabric runs compared to the link rate). Larger speedups reduce the average number of packets waiting at the input side, since packets are removed faster from input buffers. We show that when speedup is greater than or equal to two, the occupancy of contention buffers on any port is less than twice the size of congestion buffers. In other words, buffers of size $O(\log W_{\max})$ at input ports lead to the same performance as in an output-queued switch.

Definition: Consider two routers R and S, and assume that the same input traffic is fed to both routers. Router R is said to exactly emulate router S, if it has exactly the same drop sequence and the same departure sequence as router S.

With unlimited buffer size, a CIOQ router with speedup two can exactly emulate an OQ router [21]. In other words, despite the contention at the ingress side, there exists an algorithm that guarantees not to keep packets longer than what an OQ router does. Now assume that S is an OQ router, and R is a CIOQ router, both with output buffers of size B. Consider the scenario where router R drops an arriving packet exactly when there is a drop at router S (i.e., when the total number of packets destined for a given output port exceeds B). With such emulation, we show that the occupancy of the input buffers in router R is limited according to the following theorem.

Theorem 2.1. If router R exactly emulates router S, then at any time t, $Q(i,t) \leq 2B$, where B is the output buffer size in both routers, and $Q(i,t)$ is the buffer occupancy of Router R at input port i.

Proof. Assume the contrary. There must be a time t_0, and an input port i_0 such that $Q(i_0,t_0) > 2B$. With speedup of two, at most two packets are removed from port i_0 at any time slot. Therefore, there is a packet in router R that cannot be sent out in B time slots. This contradicts the exact emulation assumption, since any packet

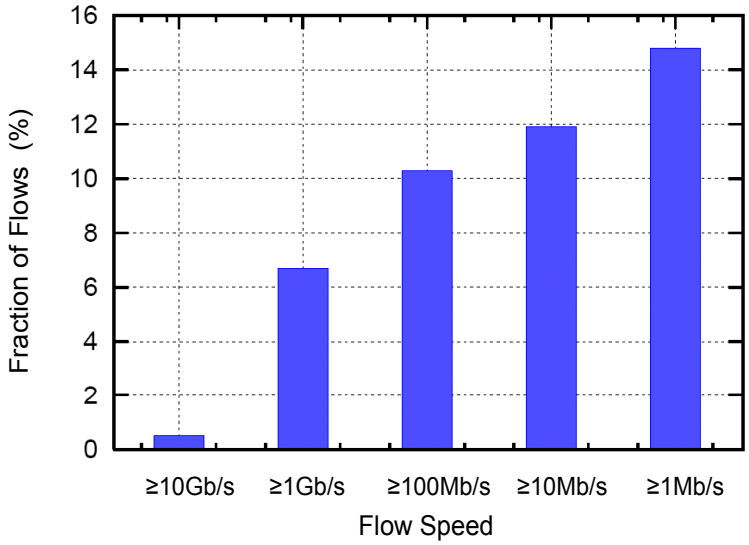


Figure 2.4: Cumulative distribution of access bandwidth in a commercial backbone network. The backbone link runs at 10Gb/s.

in the OQ router should be sent out in at most B time slots. □

Our analysis assumes that a stable marriage algorithm [21] controls the switch configuration. However, the simulation results shown in Chapter 3 suggest that even under more practical algorithms, using very small ingress buffers results in high utilization. For example, in an 8×8 switch, setting the input buffer size to only five packets per virtual output queue (VOQ) gives 80% link utilization.

2.3 Core-to-access bandwidth ratio

Figure 2.4 shows the distribution of access link bandwidth in a commercial backbone network. This data is based on traces collected by CAIDA [4] in May 2008 from an OC192 backbone link running at 10Gb/s between Seattle and Chicago.

For each value on the x axis, the bar shows the percentage of flows that appear on the backbone link faster than that value. For example, about 10% of flows are faster

than 100Mb/s. We measure the speed of a flow by finding the minimum time interval between any two consecutive packets of the flow. Note that this measurement shows the speeds at which the flows appear on the backbone link, and may not reflect their exact access link bandwidths. This is because the time intervals between packets can be changed by upstream buffering.

With the distribution shown in Figure 2.4, our assumption of an average ratio of 100 (made in our simulations and experiments that will be described in the next two chapters) seems conservative. We can see that about half a percent of flows are as fast as the backbone link (10Gb/s). However, the majority of flows (more than 85%) are either slower than 1Mb/s, i.e., 10,000 times slower than the backbone link, or they are shorter than three packets (for which we cannot have an accurate measurement).

Chapter 3

Routers With Tiny Buffers: Simulations

To validate the results of Chapter 2, we perform simulations using the ns-2 network simulator [6]. We have enhanced ns-2 to include an accurate CIOQ router model, and will study how much buffering this router needs at ingress and egress ports to achieve high utilization.

In this chapter, our metric for buffer sizing will be *link utilization*. This metric is operator-centric; if a congested link can keep operating at 100% utilization, then it makes efficient use of the operator's congested resource. This is not necessarily ideal for an individual end-user since the metric doesn't guarantee a short flow completion time (i.e., a quick download). However, if the buffer size is reduced, then the round-trip time will also be reduced, which could lead to higher per-flow throughput for TCP flows. The effect of tiny buffers on user-centric performance metrics will be discussed in Chapter 4.

3.1 Simulation setup

Figure 3.1 shows the topology of the simulated network. TCP flows are generated at separate source nodes (TCP servers), go through individual access links, and are multiplexed onto faster backbone links before reaching the input ports of the switch.

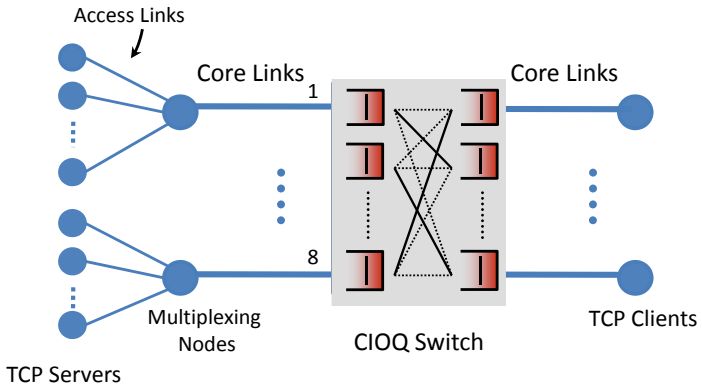


Figure 3.1: Simulated network topology.

Large buffers are used at the multiplexing routers to prevent drops at these nodes. All buffers in the network are drop-tail buffers.

The simulated switch is a CIOQ switch, which maintains virtual output queues (VOQ) at the input to eliminate head-of-line blocking. In each scheduling cycle, a scheduling algorithm configures the switch and matches input and output ports. Based on this configuration, either zero or one packet is removed from every input port, and is sent to the destination output port.

We define M to be the multiplexing factor, which is the ratio of the core link speed to the access link speed. Today, a typical user is connected to the network via a 10Mb/s DSL link, and backbone links often run at 40Gb/s; i.e., $M = 4,000$. In our simulations we conservatively pick M to be 100.

During an initialization phase, different flows start at different times and try to send an infinite amount of data. All data packets are 1000 bytes, and the control packets (SYN, SYN-ACK, FIN) are 40 bytes. The propagation delay between each server-client pair is uniformly picked from the interval 75-125ms (with an average of 100ms).

To measure link utilization, we first wait 30 seconds for the system to stabilize, and then take measurements for 60 seconds.

To study the effects of traffic characteristics and switch parameters in these simulations, we first choose a baseline setting with the following components:

1. The simulated switch is an 8×8 CIOQ switch with its linecards running at 2.5Gb/s.

2. The load is distributed uniformly among input and output ports of the router. In other words, all output ports are equally likely to be the destination port of a given flow.

3. The switch configuration is controlled by the Maximum Weight Matching (MWM) algorithm. MWM is known to deliver 100% utilization for admissible traffic distributions [34, 23], but the algorithm is too complex to be implemented in real routers.

4. We relax the over-provisioning assumption of Section 2.1.3, and offer 100% load to every output link of the router. In other words, we set the number of flows (N) sharing each output link and the TCP maximum window size ($W_{\max}$) such that the maximum aggregate rate of TCP sources sharing the link is equal to the link capacity:

$$\frac{N \times W_{max}}{RTT} = C. \tag{3.1}$$

   With an average RTT of 100ms and $W_{\max} = 64$KB, we need about 490 flows on each core link to fill the link.

5. The end hosts use TCP Reno with the Selective Acknowledgement (SACK) option disabled.

In Section 3.2, we present the results of simulating a router with the above baseline setting. Sections 3.3 and 3.4 examine how changing each of the components of the baseline setting affects link utilization.

3.2 Simulation results – baseline setting

Figure 3.2 shows the average link utilization versus input and output buffer sizes in the baseline setting. To see the effect of the input and output buffer sizes independently, we first set the switch speedup to one, which makes the switch function as an input-queued switch. With speedup one, there is no queueing at egress ports, since the switch fabric runs no faster than the output links. Next, we set the switch speedup equal to the switch size (eight) to eliminate input queueing. With speedup eight, the switch functions as an output-queued switch and needs buffering only at the output side.

In both input-queued and output-queued scenarios, we run the simulations twice: first with $M = 1$, i.e., access links run at 2.5Gb/s, and then with $M = 100$, i.e., access links run at 25Mb/s.

Figure 3.2 shows the benefit of a larger M. Because the network naturally spaces out packets of each flow, much smaller buffer size is enough for high utilization. The plots show that when access links run at 25Mb/s (100 times slower than the core links), then buffering 3 packets in each VOQ and 15 packets at each output port suffices for 80% utilization. These numbers increase to 40 and more than 400 (not shown on this plot), respectively, when access links run as fast as core links.

With speedups between one and eight, we can combine the results shown in Figure 3.2: for each pair of input and output buffer sizes, utilization is not lower than the minimum utilization shown on these two graphs at the given input (top) and output (bottom) buffer sizes. This is because if speedup is greater than one, then packets are removed faster from the input queue, and the required buffer size goes down. If speedup is smaller than eight, then packets reach the output queue later, and hence the backlog is smaller.

Therefore, with any speedup, we can achieve more than 80% utilization with 3-packet VOQs and 15-packet output queues in the baseline setting. Remember that this result is with 100% load on the output links of the router. This suggests that the theoretical results of Section 2.1.3 are conservative in their over-provisioning assumptions.

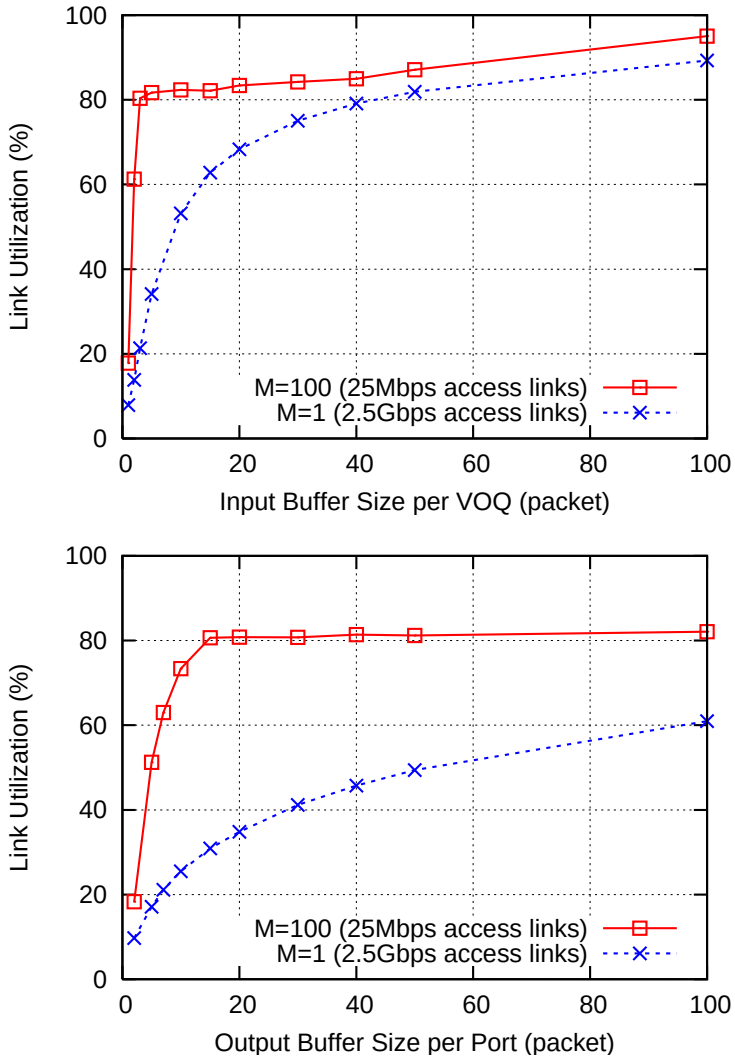


Figure 3.2: Link utilization vs. input and output buffer sizes. *Top*: speedup=1 and all the queueing takes place at the input; *bottom*: speedup=8 and all the queueing takes place at the output. With 25Mb/s access links, 3-packet VOQs, and 15-packet output buffers make the utilization above 80%.

3.3 When switch parameters change

In this section, we will see how the CIOQ switch parameters (scheduling algorithm, load distribution, switch size, and output port bandwidth) affect link utilization when tiny buffers are used. Network conditions and traffic characteristics are the same as in the baseline setting.

3.3.1 Switch scheduling algorithm and load distribution

In the baseline setting, we assumed that the switch was scheduled by the MWM algorithm, and that the load distribution was uniform. The MWM algorithm is very complex to implement and cannot be used in practice, but delivers full throughput for all admissible traffic.

Here, we relax these two assumptions and compare the results of the baseline setting to those obtained under the iSLIP scheduling algorithm [33] and non-uniform traffic.

The widely implemented iSLIP scheduling algorithm achieves 100% throughput for uniform traffic. This iterative round-robin-based algorithm is simple to implement in hardware, but the throughput is less than 100% in the presence of non-uniform bursty traffic.

Among various possible non-uniform distributions of load, we choose the diagonal load distribution. With a diagonal load, 2/3 of the total flows at a given input port i are destined for output port $i+1$, and the remaining 1/3 of the flows are destined for output $i+2$ (modular addition). The diagonal distribution is skewed in the sense that input i has packets only for outputs $i+1$ and $i+2$. This type of traffic is more difficult to schedule than uniformly distributed traffic, because arrivals favor the use of only two matchings out of all possible matchings. Under any scheduling algorithm, the diagonal load makes the router's buffers have the largest average backlog compared to any other distribution [45].

Figure 3.3 shows link utilization versus input buffer size per VOQ. With iSLIP and speedup 1 (top graph), there is no queueing at the output side of the switch. When the speedup is set to 1.2 (bottom graph), the switch fabric runs 1.2 times faster than

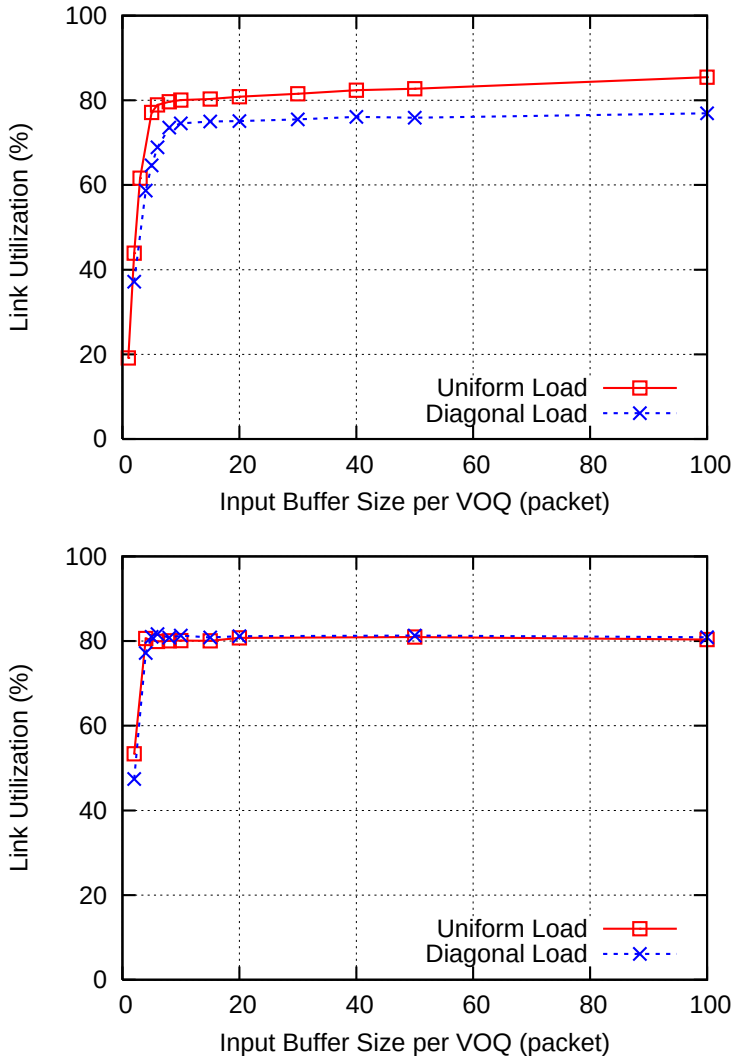


Figure 3.3: Link utilization vs. buffer size with iSLIP scheduling algorithm. *Top*: speedup=1; *bottom*: speedup=1.2.

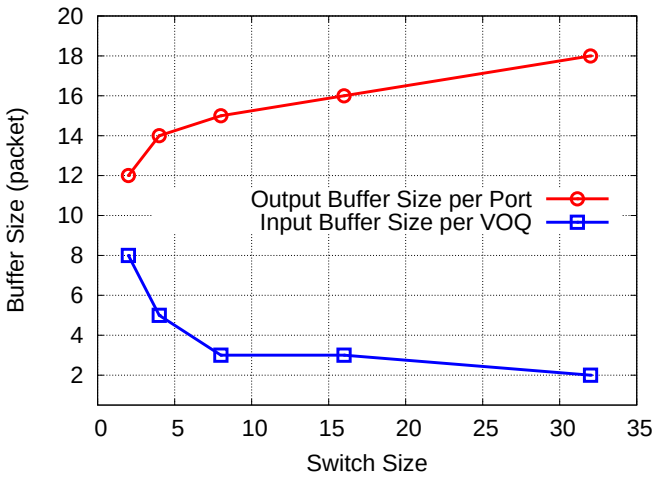


Figure 3.4: Minimum required buffer size for 80% utilization vs. switch size. The switch has a uniform load distribution, which results in more contention and short-term congestion when the number of ports increases.

the line rate, which may cause backlog in output buffers. In this case, we have set the output buffer size to only 20 packets per port. That is why we see some utilization loss when the speedup is increased from 1 to 1.2.

The results show that with speedup 1.2—for all combinations of scheduling algorithm and load distribution—setting the buffer size to 5 packets per VOQ and 20 packets per output port raises the utilization to more than 80%. Larger speedups make the impact of the scheduling algorithm even smaller because the switch behaves more and more like an output-queued switch.

3.3.2 Switch size

The output link utilization of a router depends on its size (number of ports). Increasing the number of ports creates more contention among the input ports, and adds to the short-term congestion (caused by the statistical arrival time of packets from different ingress ports) on the output links. Hence, with the same traffic, the

utilization could be different for different switch sizes, depending on the distribution of the load.

Figure 3.4 shows the minimum required buffer size at input and output ports of a switch for 80% utilization on the output links. The simulation setting follows the baseline, except that the switch size is changed from 2 to 32. The number of flows on each link is kept constant for each switch size to maintain 100% offered load on the core links.

In this set of simulations, the switch has a uniform load distribution. If the traffic at a given output port comes from a fixed subset of the input ports, then we do not expect to see any changes when the switch size is varied. With diagonal load distribution, for example, where the traffic on each output link i comes only from ports $i-1$ and i, the required buffer size for 80% utilization remains constant as we change the number of ports.

We assume that the ingress ports maintain a separate VOQ for each output port. Therefore, despite the decrease in the VOQ size as the switch size grows, the total buffer size per ingress port (i.e., the size of the VOQ times the number of output ports) increases.

3.3.3 Output link bandwidth

The theoretical results of Section 2.1.3 shows that the required buffer size, for achieving high utilization, is independent of the absolute bandwidth of the bottleneck link. This is different from what the rule-of-thumb proposes. According to this rule, the buffer size needs to increase linearly as the bottleneck link bandwidth increases; hence, a 40Gb/s link needs 40 times as many buffers as a 1Gb/s link.

Figure 3.5 shows how link utilization stays unchanged when we increase the core link bandwidth, but keep $M = 100$ by increasing the access link bandwidth proportionally (the dotted curve). The buffer size is constant at 20 packets per port.

The solid curve in Figure 3.5 shows the bottleneck link utilization when the access bandwidth is fixed at 25Mb/s. Increasing the core bandwidth creates more spacing between packets, and reduces burst size; hence, the utilization improves.

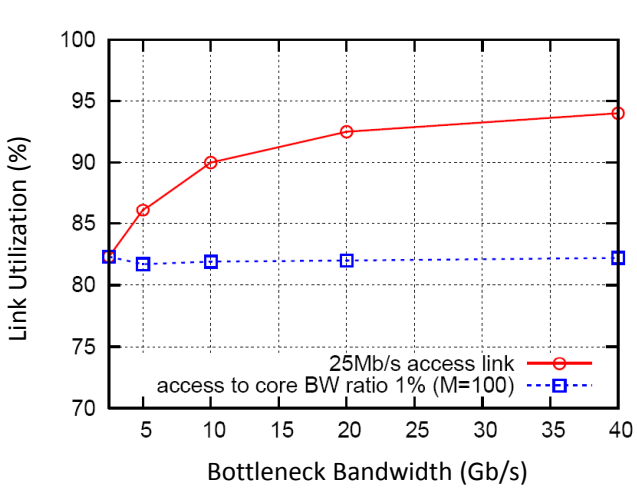


Figure 3.5: Link utilization vs. bottleneck link bandwidth. Utilization is determined by the core-to-access bandwidth ratio, not by the absolute core-link bandwidth. Buffer size at the bottleneck link is fixed at 20 packets.

3.4 When traffic characteristics change

This section considers two properties of the network traffic: the traffic load, and the TCP flavor implemented at the hosts. We will see how tweaking these parameters changes link utilization when tiny buffers are used in the simulated router.

3.4.1 Traffic load

The baseline setting assumes 100% offered load on the output links of the router. In other words, if all servers send traffic at their maximum rate ($W_{\max}/RTT$), then the aggregate traffic rate will be equal to the bottleneck bandwidth.

The offered load on the bottleneck link varies if either the number of flows sharing the link or the amount of the traffic that a single flow could generate changes. The effect on the link utilization in both cases is illustrated in Figure 3.6. The access bandwidth in both sets of simulations is fixed at 25Mb/s and the output buffer size is fixed at 20 packets (the simulated switch is an output-queued switch).

As the number of flows grows, the bandwidth share of each flow, the average

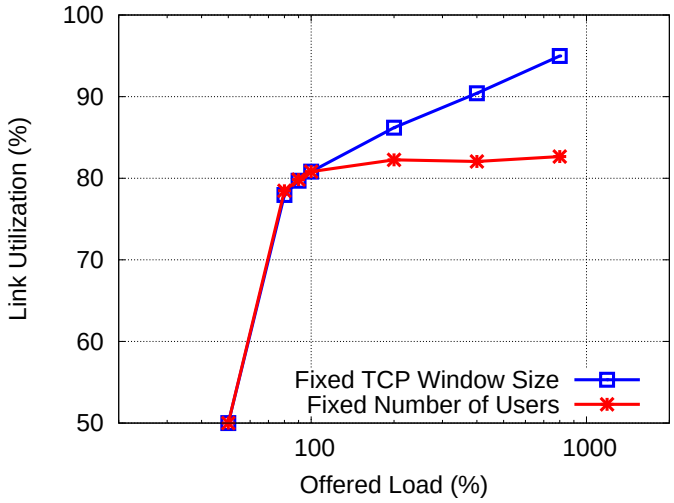


Figure 3.6: Link utilization vs. offered load. The offered load is changed by increasing the number of flows (with a fixed TCP window size) and increasing the maximum TCP window size (with a fixed number of flows).

window size, and the burst size a flow could possibly generate become smaller; hence, the link utilization improves.

Increasing the maximum window size (while keeping the number of flows constant) slightly improves utilization, from 81% to 83%. In this case, packet drops at the bottleneck link keep the average window size almost constant, regardless of how large the maximum allowed window size is.

3.4.2 High-speed TCPs

Figure 3.7 compares the link utilization achieved by two high-speed TCPs, TCP BIC [50](default in Linux kernels 2.6.8 through 2.6.18) and TCP CUBIC [42] (default in Linux Kernel since version 2.6.19), with that achieved by TCP Reno [1]. The plot

[1]The main difference between these high-speed TCPs and TCP Reno is in their window growth functions. When TCP BIC gets a packet loss event, it reduces its window by a multiplicative factor. Then it performs a binary search to find the right window size (rather than additively increasing the window size as in Reno). CUBIC is an enhanced version of BIC; it simplifies the BIC window

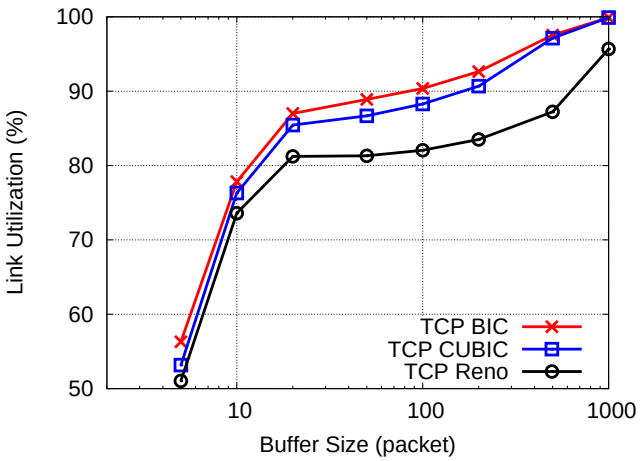


Figure 3.7: Comparison of utilization achieved by TCP Reno, BIC, and CUBIC.

shows that for all buffer sizes, these newer variants of TCP consistently outperform TCP Reno.

High-speed flavors of TCP are designed to improve the performance of long-lived large-bandwidth flows. In finding an individual flow's share of bandwidth, they are more aggressive than the traditional TCP Reno (and hence more responsive to the available bandwidth), but have also a higher packet drop rate compared to TCP Reno.

3.4.3 Packet size

So far, we have assumed that the size of data packets is 1000 bytes, and the unit of our buffer size measurements has been the *number of packets*. However, the buffer size required to achieve a certain link utilization is not independent of the packet size.

Figure 3.8 shows the bottleneck link utilization in a set of simulations with packet sizes 50-1000B. In these simulations, independent of the packet size, the offered load

control and improves its RTT-fairness. The window growth function of CUBIC is governed by a cubic function in terms of the elapsed time since the last loss event.

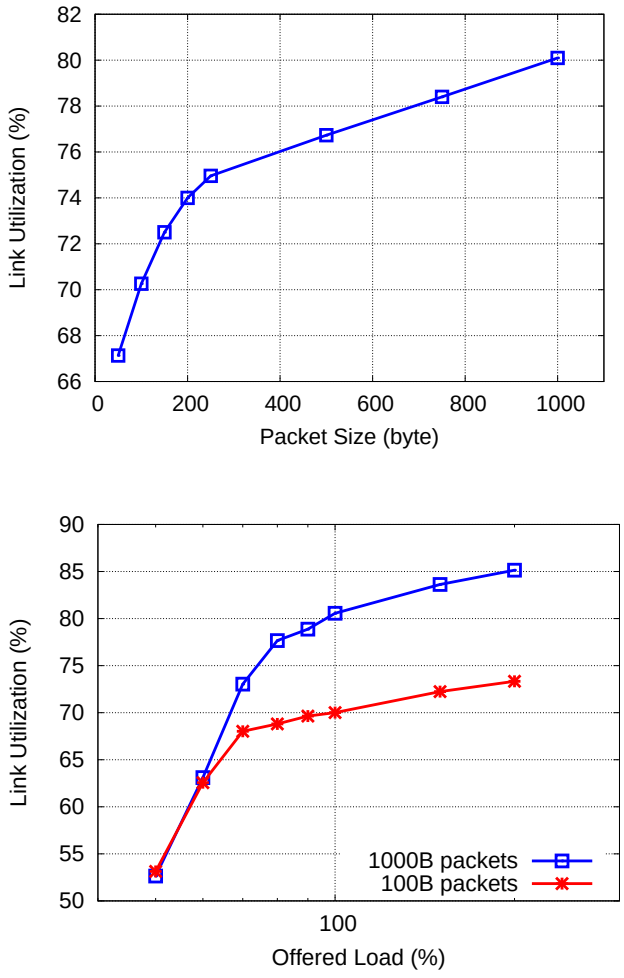


Figure 3.8: Link utilization vs. packet size. *Top*: on a congested link, smaller packets result in a lower utilization; *bottom*: at loads below 60%, same utilization is achieved with 100B and 1000B packets.

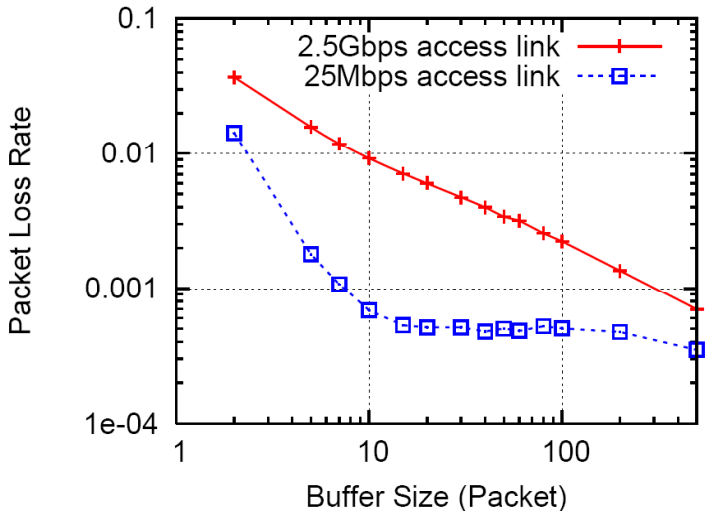


Figure 3.9: Packet drop rate vs. buffer size. The offered load on the bottleneck link is 100%.

is 100%, the buffer size at the bottleneck link is 15 packets, and the maximum TCP window size is 64KB. The window size in number of packets increases as the packet size becomes smaller.

With smaller packet sizes, recovery from packet drops becomes slower. In the congestion avoidance mode, TCP increases the congestion window by one segment every round-trip time; therefore, smaller packets make the ramp-up periods longer and the utilization loss larger. As the load on the bottleneck link (and consequently the drop rate at the buffer) reduces, the difference in utilization becomes smaller (Figure 3.8). At loads below 60%, link utilization is the same with both 100B and 1000B packets.

3.5 Packet drop rate

When a network link is congested, there have to be some drops at the link to notify the TCP senders of the congestion, and make them reduce their transmission rates.

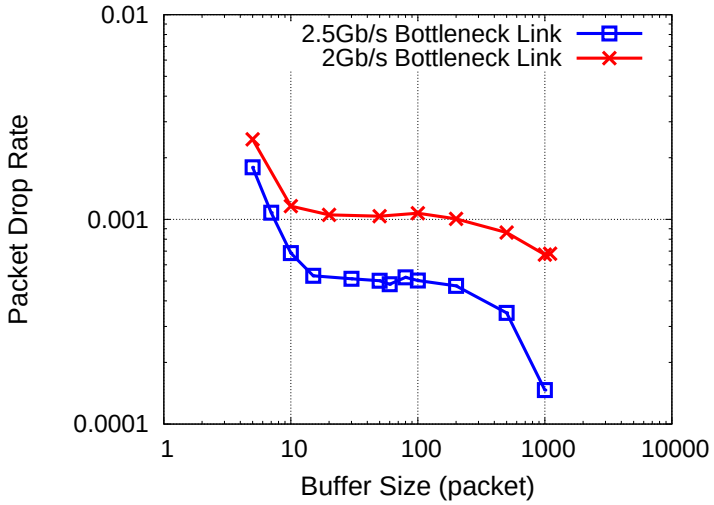


Figure 3.10: Comparison of packet drop rate when tiny buffers are used versus when bottleneck bandwidth is reduced. Under the same load, the 2.5Gb/s link with a tiny 10-packet buffer results in fewer drops than a 20% slower link with much larger buffers.

Figure 3.9 shows the average drop rate at the output ports of the simulated router in the baseline setting.

The offered load in the above simulation is 100%. Repeating the simulation with an offered load below 70% does not show any packet drops at the router, even with 15-packet buffers at output ports. This suggests that under typical conditions, when the core links are not congested, using tiny buffers does not affect the drop rate.

As noted previously, losing about 20% of the link utilization is equivalent to running the backbone link 20% slower. The plot in Figure 3.10 compares the packet drop rate of a 2.5Gb/s link with that of a 2Gb/s link (with 20% reduced bandwidth) under the same offered load. We can see that the drop rate of the faster link at 15-packet buffer is smaller than the drop rate of the slower link at $\frac{RTT \times C}{\sqrt{N}} \approx 1100$-packet buffer. In other words, the fast link with tiny buffers drops fewer packets than the link with 20% reduced bandwidth, but with much larger buffers.

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

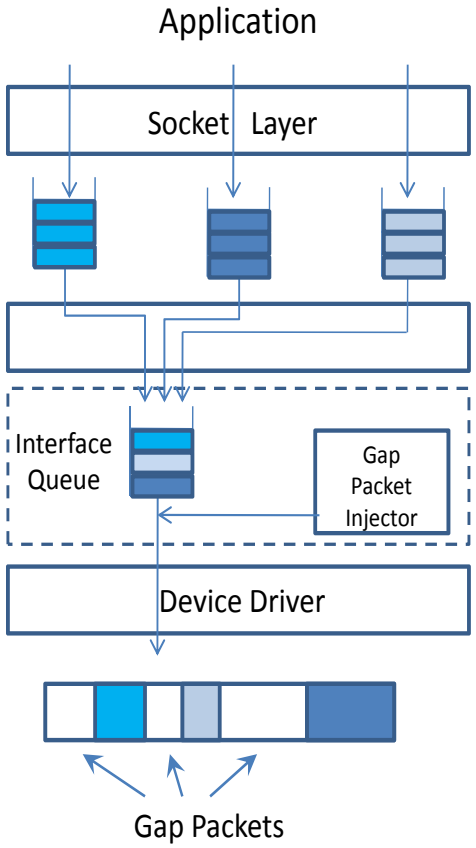


Figure 4.1: Implementation of Precise Software Pacer in the Linux kernel.

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

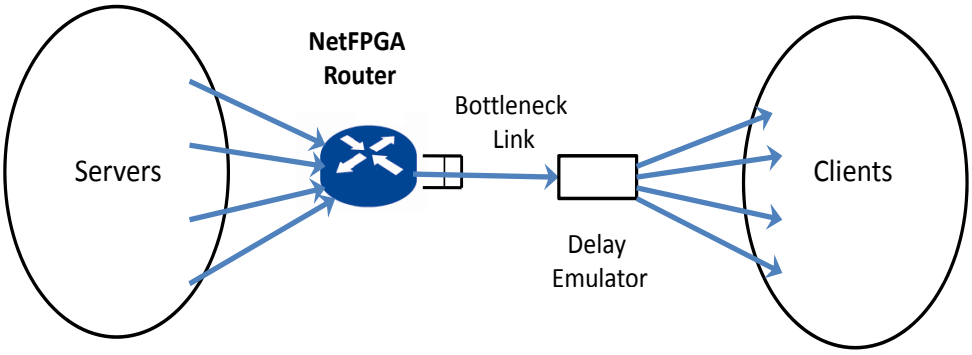


Figure 4.2: Dumbbell topology.

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.

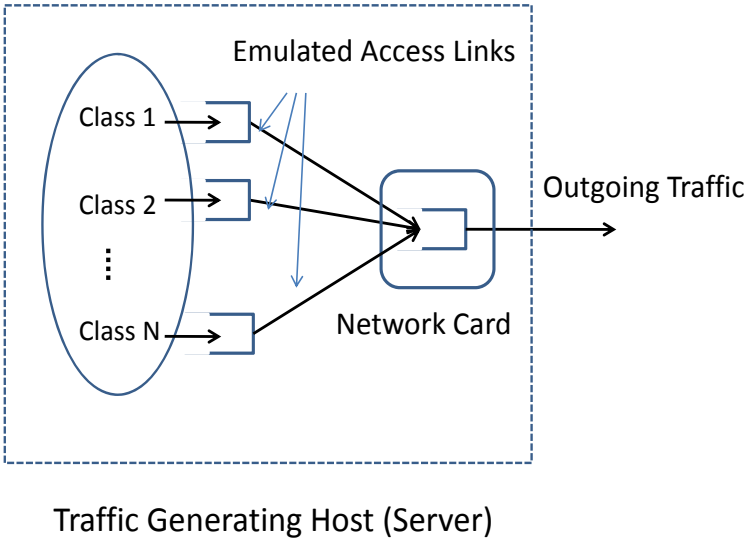


Figure 4.3: Emulating slow access links at end hosts. Flows generated on each physical machine are grouped into multiple classes, where flows in each class share an emulated access link before merging with all other flows into a single stream.

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

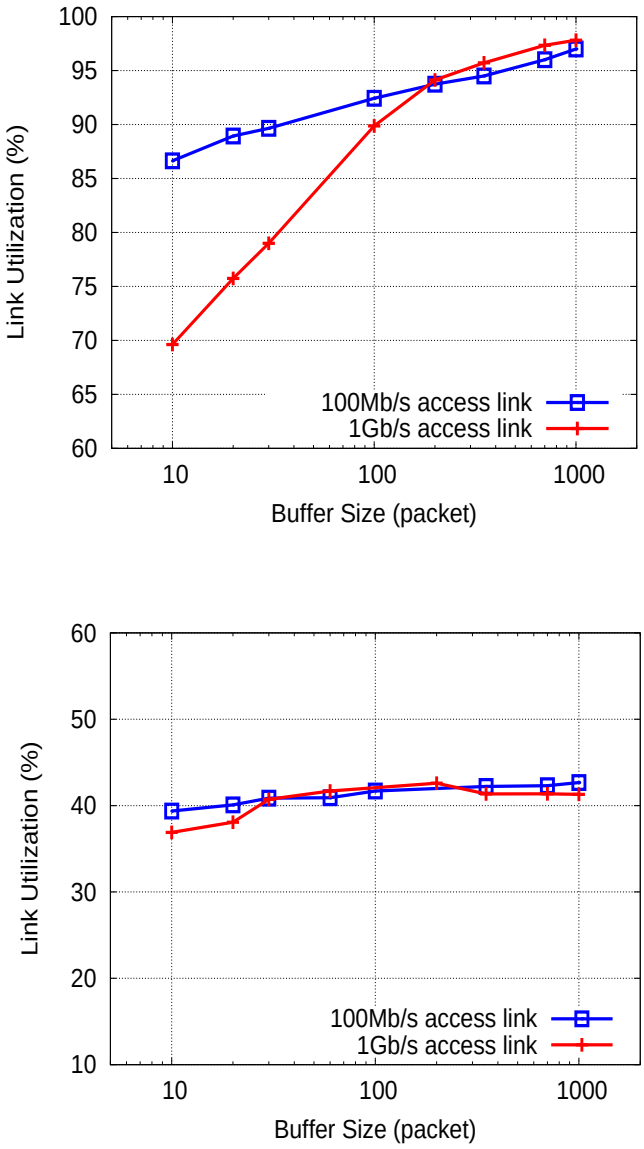


Figure 4.4: Link utilization vs. buffer size. *Top*: offered load = 125%; *bottom*: offered load = 42%.

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

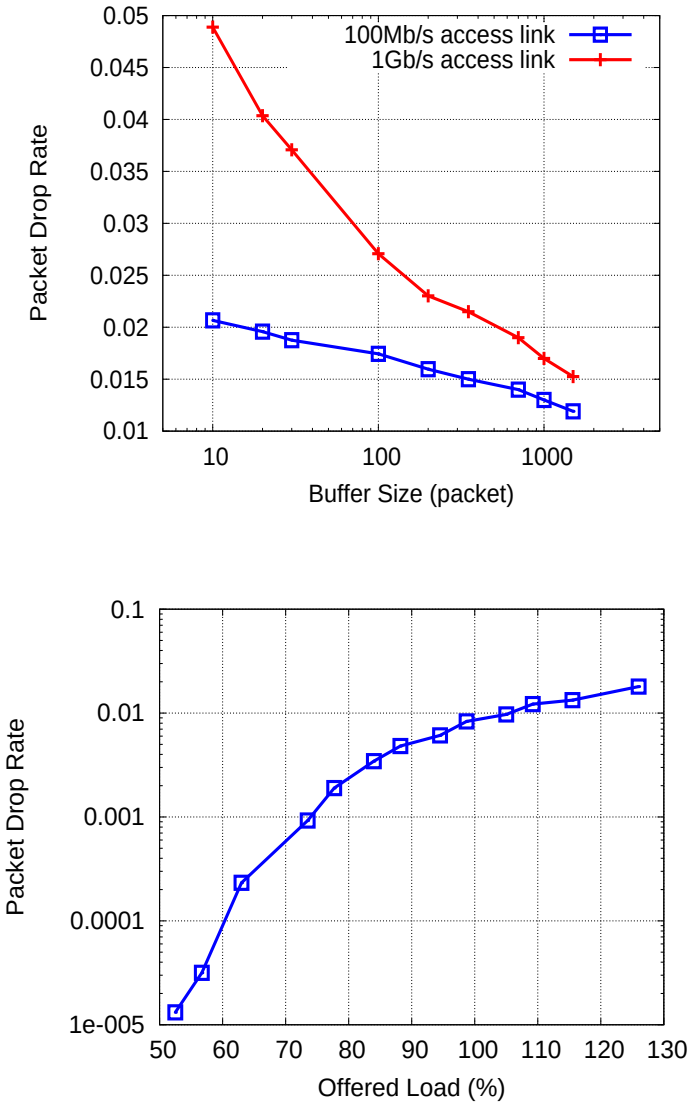


Figure 4.5: *Top*: packet drop rate vs. buffer size. The offered load is 125%; *bottom*: packet drop rate vs. offered load. Access links run at 100Mb/s and the buffer size is 50 packets.

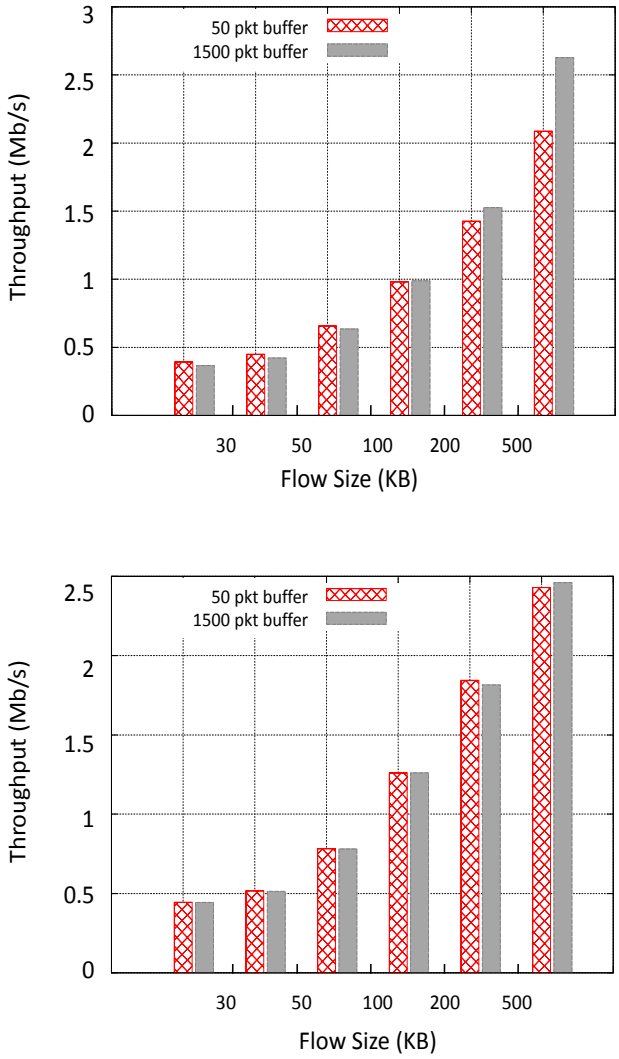


Figure 4.6: Average per-flow throughput vs. flow size. The left-most bars show the average throughput of flows smaller than 30KB. The right-most ones show the average throughput of flows larger than 500KB.
Top: offered load = 125% and $W_{\max}$ = 2MB; *bottom*: offered load = 42% and $W_{\max}$ = 64KB.

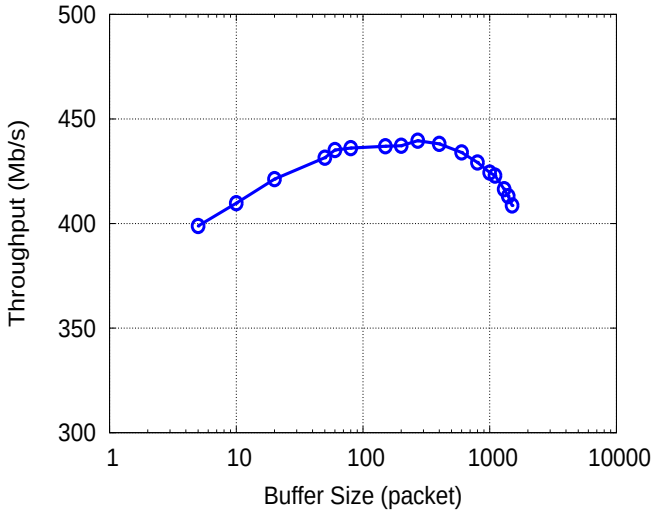


Figure 4.7: Average throughput of small flows ($\leq$ 50KB) vs. buffer size. Very large buffers add to the delay and reduce the throughput.

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 $0.2 - 0.4\%$

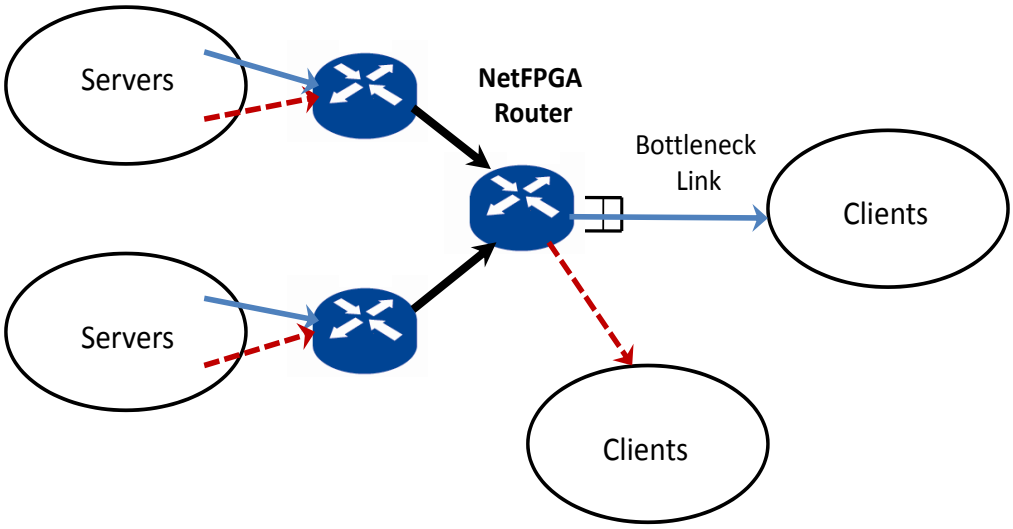


Figure 4.8: Network topology with cross traffic.

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.

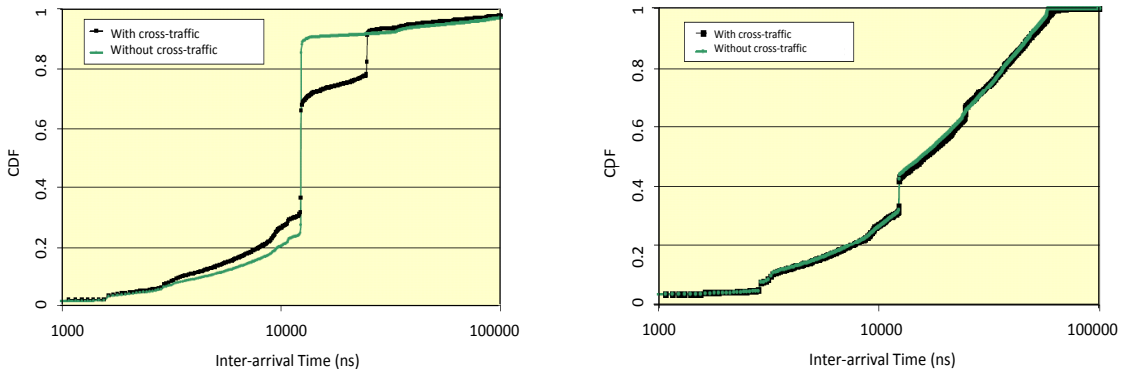
4.1.3 Cross traffic

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

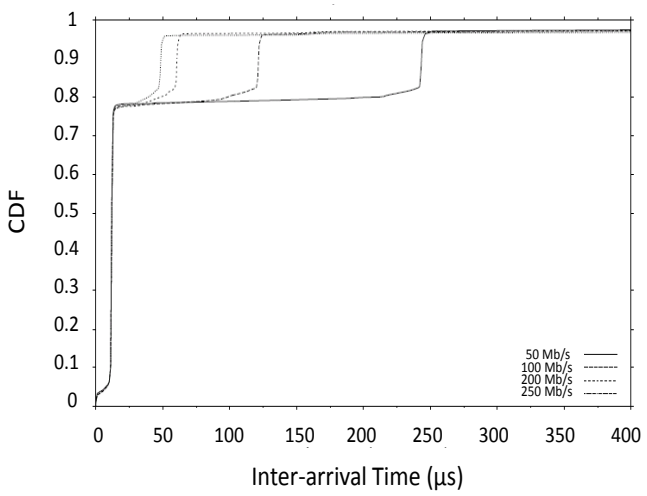


Figure 4.10: Cumulative distribution of packet inter-arrival time with TSO enabled. Even though PSPacer is implemented, the gap only appears between groups of back-to-back packets.

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

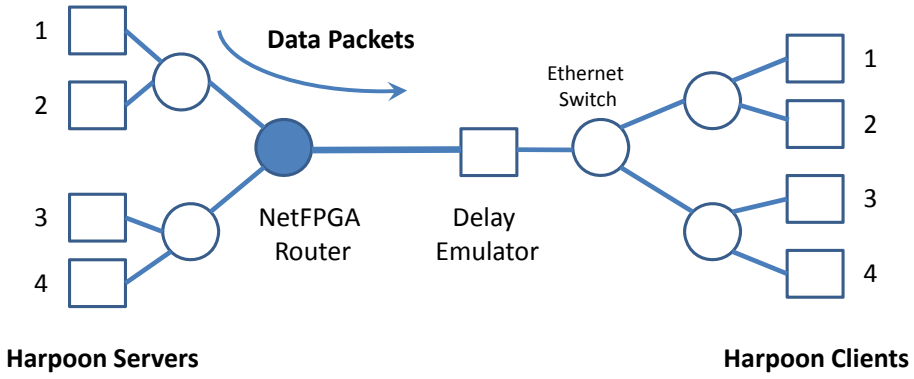


Figure 4.11: A network topology to compare Harpoon traffic generated by one pair versus four pairs of physical machines.

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

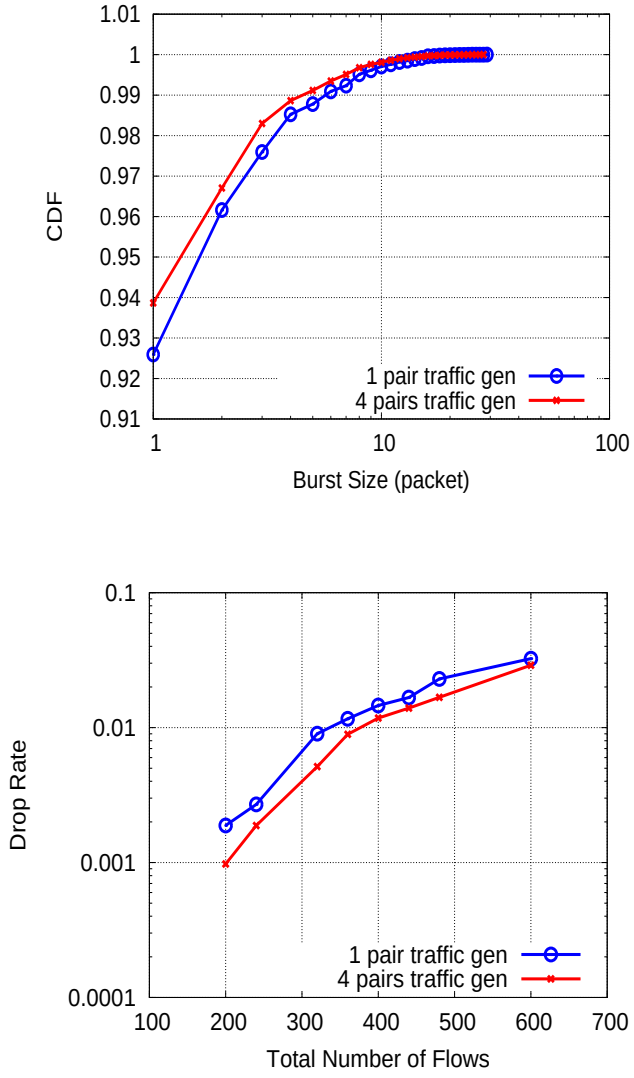


Figure 4.12: Comparison of traffic generated on two versus four hosts. *Top*: cumulative distribution of burst size in the aggregate traffic; *bottom*: packet drop rate at the bottleneck link.

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

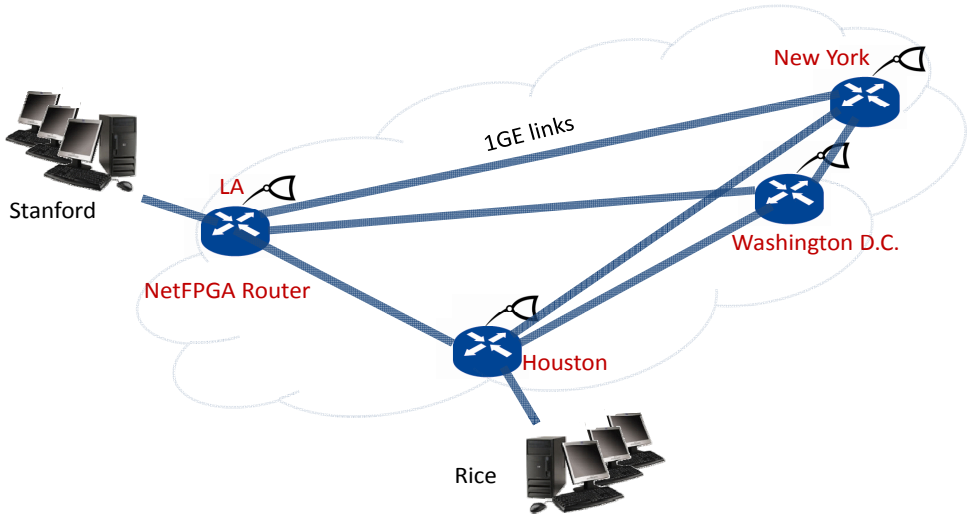


Figure 4.13: Experimental NetFPGA-based network over Internet2's backbone network.

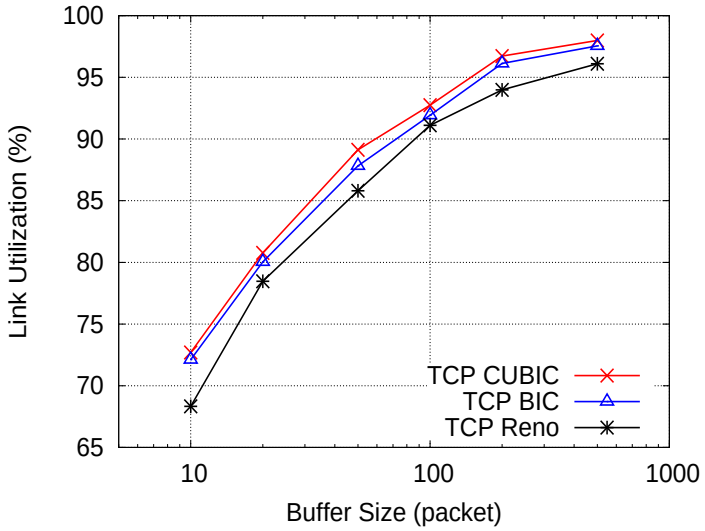


Figure 4.14: Utilization vs. buffer size with TCP Reno, BIC, and CUBIC.

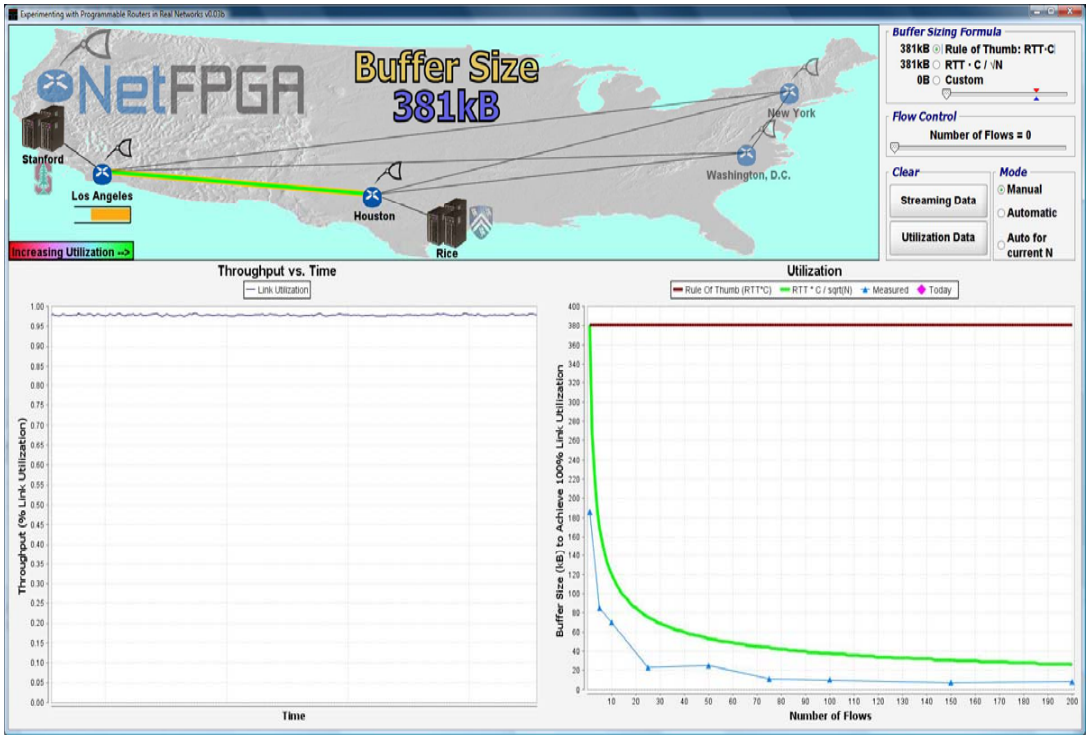


Figure 4.15: A screenshot of the real-time buffer sizing experiments.

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.

Chapter 5

Networks with Tiny Buffers

Enachescu et al. [25] show that under TCP traffic, a single router can achieve close-to-peak throughput with buffer size $O(\log W_{\max})$. The crucial assumption in this result is a non-bursty input traffic (see Section 2.1.3).

In a network with arbitrary traffic matrix and topology, however, the traffic pattern of flows may change across the network. Although there has been no thorough analytical work on how an arbitrary network of FIFO buffers changes the traffic pattern, there are examples that show the traffic may become bursty as it traverses the network [27].

In this chapter, we explore whether a network can maintain high throughput when all its routers have tiny congestion buffers. The running assumption is that traffic is smooth at ingress ports of the network. To answer this question, we will first study networks with tree structure (Section 5.2), where there is no cross traffic. We will show that the buffer occupancy of a router in a tree-structured network does not exceed the buffer occupancy of an isolated router into which the ingress traffic is directly fed. Therefore, the single-router results can be applied to all routers in this network. For general-topology networks (Section 5.3), we will propose an active queue management policy (BJP) which keeps the packet inter-arrival times almost unchanged as the flows traverse the network. Therefore, if traffic is smooth at ingress ports, it remains so at every router inside the network, and hence, the single-router results hold for all routers in the network.

5.1 Preliminaries and assumptions

Throughout this chapter, we assume that flows go through only one buffering stage inside each router. Hence, we set the number of buffering stages equal to the number of routers and refer to them interchangeably.

For a given network and traffic rate matrix, we define the offered load on each link to be the aggregate injection rate of flows sharing that link. The *load factor*, ρ, of the network is the maximum offered load over all the links in the network. We assume that the network is over-provisioned (i.e., $\rho < 1$), all packets have equal sizes, and all buffers in the network have equal service rates of one packet per unit of time.

5.2 Tree-structured networks

In a tree-structured network, routers form a tree, where the traffic enters the network through the leaves of the tree. After packets are processed at a hop (router), they are forwarded to the next hop towards the root of the tree, which is the exit gateway of the network.

Consider a router R_m at distance m from the leaves in a tree-structured network (Figure 5.1, top). This router is the root of a sub-tree that receives traffic on a subset of the ingress ports. The queue size of router R_m (i.e., number of packets queued in the router) can be compared to that of an isolated router R, which directly receives the same ingress traffic (Figure 5.1, bottom). At a given time t, there is an arrival at router R if and only if there is one at the input ports of the sub-tree.

Assume that the propagation delay on the links is negligible. By comparing the queue sizes of these two routers, it can be shown that the drop rate at R_m with buffer size B is upper bounded as stated in the following lemma.

Lemma 5.1. With Poisson ingress traffic and a load factor $\rho < 1$, the drop rate at router R_m is smaller than or equal to ρ^B.

The proof is in Appendix B.

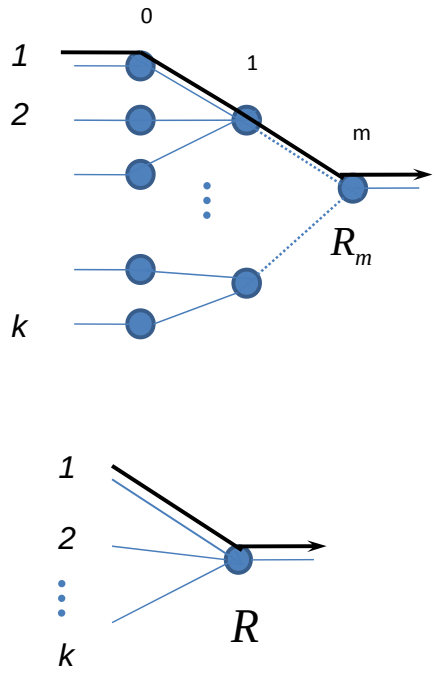


Figure 5.1: Tree-structured network (top). The buffer occupancy of router R_m in this network does not exceed the buffer occupancy of an isolated router R into which the ingress traffic is directly fed (bottom).

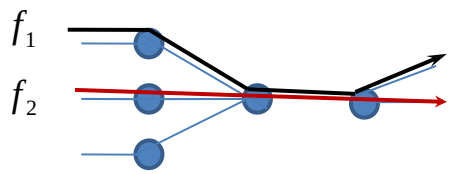


Figure 5.2: An example of a general-topology network.

This lemma implies that the overall packet drop rate will be less than $n\rho^B$ if each packet goes through at most n routers. The following theorem immediately follows.

Theorem 5.1. In a tree-structured network with Poisson ingress traffic and a load factor $\rho < 1$, packet drop rate of ϵ can be achieved if each router buffers

$$B \geq \log_{1/\rho}(\frac{n}{\epsilon})$$

packets, where n is the maximum number of routers on any route.

5.3 General-topology networks

Figure 5.2 shows an example of a network with general topology where flows can share a part of their routes and then diverge. The results obtained in Section 5.2 depend crucially on the tree-structured nature of the network (see Appendix B for details) and cannot be applied to a general network.

But what happens if routers in a general-topology network delay packets by exactly D units of time? Clearly, in this network the inter-arrival times of packets in each flow will remain intact as the flow is routed across the network. In particular, if the ingress traffic of the network is Poisson, the input traffic at each intermediate router will continue to be Poisson. However, delaying every packet by a fixed amount of time may not be feasible. Packets arriving in a burst cannot be sent out in a burst if

they arrive faster than the transmission capacity of the output interface. Therefore, some small variations should be allowed in the delay added by each router. This is the basic idea behind the Bounded Jitter Policy, which is explained below.

5.3.1 Bounded Jitter Policy (BJP)

Figure 5.3 illustrates the scheduling of BJP. This policy is based on delaying each packet by D units of time but also allowing a cumulative slack of Δ, $\Delta \leq D$. We call Δ the *jitter bound* of the scheme.

In order to implement BJP, packets need to be time stamped at each router. When a packet first enters the network, its time stamp is initialized to its actual arrival time. At each router on the route, the time stamp is updated and incremented by D units of time, regardless of the actual departure time of the packet.

Consider a router that receives a packet with time stamp t. The router tries to delay the packet by D units of time and send it at time $t + D$. If this exact delay is not possible, i.e., if another packet has already been scheduled for departure at time $t + D$, then the packet will be scheduled to depart at the latest available time in the interval $[t + D - \Delta, t + D]$. If there is not any available time in this interval, the packet will be dropped.

The packet scheduling of BJP is based on time stamps rather than actual times. It follows that if a packet leaves a router earlier than its departure time stamp $(t+D)$, it will likely be delayed more than D units by the next hop, and this can compensate the early departure.

The following theorem shows that if the ingress traffic is Poisson, BJP can achieve a logarithmic relation between the drop rate and the buffer size.

Theorem 5.2. In a network of arbitrary topology with Poisson ingress traffic, packet drop rate ϵ can be achieved if each router buffers

$$B \geq 4 \log_{1/\alpha}(\frac{n}{(1-\alpha)\epsilon})$$

packets, where n is the maximum number of routers on any route, $\alpha = \rho e^{1-\rho}$, and

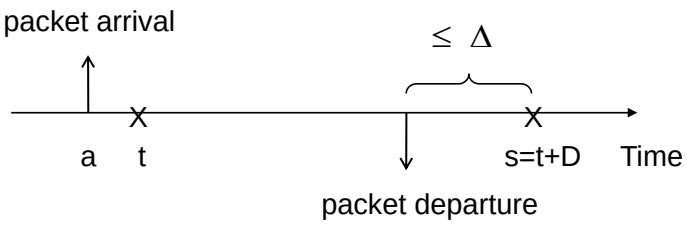


Figure 5.3: Packet scheduling under BJP. A packet that arrives at time a is scheduled based on its arrival time stamp $t \geq a$. The packet is scheduled to depart the buffer as close to $t + D$ as possible, where D is a constant delay.

$\rho < 1$ is the load factor of the network.

The proof is in Appendix C.

5.3.2 Local synchronization

Although BJP requires a time stamp to be carried with each packet inside the network, it only needs synchronization between the input and output linecards of individual routers, not a global synchronization among all routers. In fact, all that is needed is the slack time of the packet, not its total travel time. Consider a router that receives a packet at time a with a slack $\delta \leq \Delta$. Ideally, the router sends out the packet at time $a + D + \delta$. When the packet leaves the router, the difference between this ideal departure time and the actual departure time will be the updated time stamp of the packet.

5.3.3 With TCP traffic

As we explained in Section 2.1.3, the burstiness in TCP is not caused by the AIMD dynamics of its congestion control mechanism. Spreading out packets over a round-trip time can make the traffic smooth without needing to modify the AIMD mechanism.

To analyze the throughput of the network under smooth TCP traffic, we take an approach similar to [25] and assume that packet arrivals of each flow (at the rate dictated by the AIMD mechanism of TCP) follow a Poisson model. This lets us use the upper bound on the drop rate derived in the previous section.

Consider an arbitrary link l with bandwidth C packets per unit of time, and assume that N long-lived TCP flows share this link. Flow i has time-varying window size $W_i(t)$, and follows TCP's AIMD dynamics. In other words, if the source receives an ACK at time t, it will increase the window size by $1/W_i(t)$. If the flow detects a packet loss, it will decrease the congestion window by a factor of two. In any time interval $[t, t')$ when the congestion window size is fixed, the source will send packets as a Poisson process at rate $W_i(t)/RTT$. We also assume that the network load factor ρ is less than one. This implies that $\rho_l = \frac{N \times W}{RTT} < C$, where ρ_l is the offered load on link l. The effective utilization, θ_l, on link l is defined as the achieved throughput divided by ρ_l.

Under the above assumptions, the following theorem holds.

Theorem 5.3. To achieve an effective utilization θ on every link in the network, a buffer size of

$$B \geq 4 \log_{1/\alpha}(\frac{nW}{(1-\alpha)(1-\theta)})$$

packets suffices under the BJP scheme.

The proof is in Appendix D.

Chapter 6

Optical FIFO Buffers

All-optical packet switches require a means of delaying optical packets and buffering them when the output link is busy. This chapter addresses the problem of emulating the FIFO queueing scheme using optical delay lines.

6.1 Elements of optical buffering

Figure 6.1(a) shows how a single optical delay line and a 2×2 optical switch can create an optical buffering element. When a packet arrives at the buffer, the switch goes to the crossed state in order to direct the packet to the delay line. The length of the delay line is large enough to hold the entire packet. Once the entire packet enters the delay line, the switch changes to the parallel state. This forms a loop where the packet circulates until its departure time, when the switch goes to the crossed state again and lets the packet leave the buffer.

A FIFO buffer of size N can be built by concatenating N delay loops, each capable of holding one packet (Figure 6.1(b)). Each packet is stored in the rightmost delay line which is not full. When a packet leaves the buffer, all other packets are shifted to the right. However, given the complexity of building optical switches, this architecture may not be practical for large N, as the number of 2×2 switches grows linearly with the number of packets that need to be stored. Moreover, the power attenuation caused by passing through the optical switches imposes a constraint on the number of packet

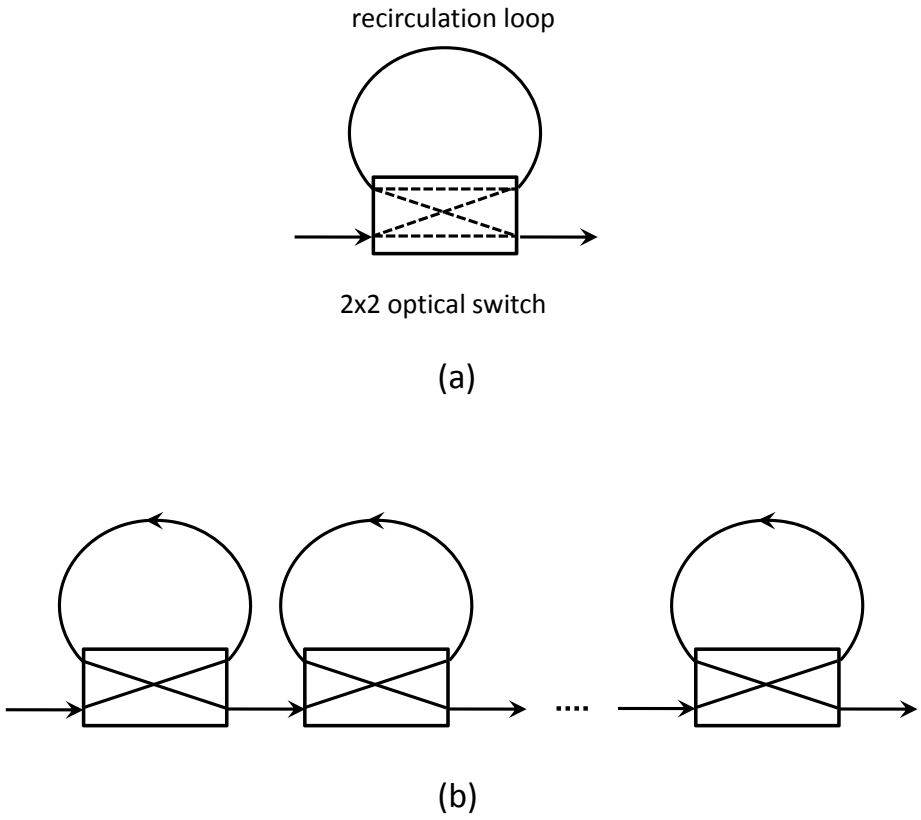


Figure 6.1: (a) Building an optical buffer with a delay line and a 2×2 optical switch. (b) Buffering multiple packets.

recirculations. The challenge is to design a buffer that works with a smaller number of switches, yet has the capacity of the linear architecture shown in Figure 6.1(b).

The problem of constructing optical queues and multiplexers with a minimum number of optical delay lines and optical switches has been studied in a number of recent works [28, 18, 19, 22, 43]. Sarwate and Anantharam have shown that for emulating any priority queue of length N, the minimum number of required delay lines is $O(\log N)$ [43]. They also propose a construction that achieves this emulation by $O(\sqrt{N})$ delay lines.

Recently, C.S. Chang et al. proposed a recursive approach for constructing optical

FIFO buffers with $O(\log N)$ delay lines [18]. Their proposed construction needs to keep track of the longest and the shortest queues in each step of the construction. Here, we present an architecture with a same number of delay lines, which is simpler in the sense that packet scheduling is independent for each delay line, as we will explain in the following sections.

6.2 Preliminaries and assumptions

We assume that all packets are of equal sizes, time is slotted, and the length of a time slot is the time needed for a packet to completely enter (or leave) an optical delay line. At each time slot, the buffer receives at most one arrival request and one departure request. The arrival and departure sequences are not known in advance.

The length of a delay line is the total number of back-to-back packets the delay line can hold. If we have a delay line of length L, it takes $L+1$ time slots for a packet to enter the delay line, traverse it once, and leave.

The states of the optical switches are controlled by a scheduling algorithm, and are changed at the beginning of each time slot. Depending on the state of a switch, a packet at the head of a delay line is either transferred to the tail of the same delay line, to a different delay line, or to the output port.

We define the (departure) order of a packet p as the total number of packets in the buffer that have arrived before p. In a FIFO queueing system, the order of each packet is decremented by one after each departure.

6.3 Buffering architecture

Exact emulation of a FIFO queue requires that with any arbitrary sequence of arrival and departure requests, the buffer has exactly the same departure and drop sequences as its equivalent FIFO buffer. What makes the delay-line based design of optical FIFO challenging is that the arrival and departure requests are not known in advance.

Figure 6.2 shows our optical FIFO construction for buffering $N-1$ packets. The delay loops have exponentially growing lengths of $1, 2, 4, ..., N/2$, and each comes with

a 2×2 switch connecting the delay loop to the path between the input and the output ports. Without loss of generality, we assume that N is a power of 2.

Incoming packets are buffered by going through a subset of the optical delay lines. When a packet arrives at the buffer, the scheduler decides if the arriving packet needs to go through the waiting line first, or if it can be directly delivered to one of the delay lines. As time progresses, optical packets in each delay line move towards the head of the delay line. At the end of each time slot, the scheduler determines the next locations of the head-of-line packets, and directs these packets towards their scheduled locations by configuring the 2×2 switches.

The waiting line W operates as a time regulator of the arriving packets. Since the time intervals between successive arriving packets are not known in advance, the scheduler uses this waiting line to adjust the locations of the packets in the delay lines. When the scheduler decides to direct an arriving packet to the waiting line, it knows exactly how long the packet needs to be kept there before moving to one of the delay lines. Packets leave W in a FIFO order. Section 6.3.1 shows that our proposed scheduling algorithm can make this buffering system emulate a FIFO buffer of size $N - 1$. Section 6.3.2 describes how the waiting line W, too, can be constructed by $\log N - 1$ delay lines. This makes the total number of delay lines equal to $2 \log N - 1$.

6.3.1 Packet scheduling

The main idea of our scheduling algorithm is to place the packets in the delay lines consecutively, i.e., packets with successive departure orders are placed back to back in the delay lines [1]. To achieve this, the scheduler places an arriving packet in a delay line only if the preceding packet has been in a tail location in the previous time slot. Otherwise, the scheduler holds the packet in W for a proper number of time slots. The waiting time is equal to the time it takes for the preceding packet to traverse its current delay line and go to a tail location. During this waiting time, the scheduler keeps all the arriving packets in W in a FIFO order.

When there is a departure request, if the buffer is nonempty, the packet with order

[1] This is in contrast with what is proposed by Sarwate and Anantharam [43], where arriving packets are allowed to fill out any available locations at the tail of the delay lines.

one is delivered to the output port. The departure orders of all the remaining packets are then reduced by one.

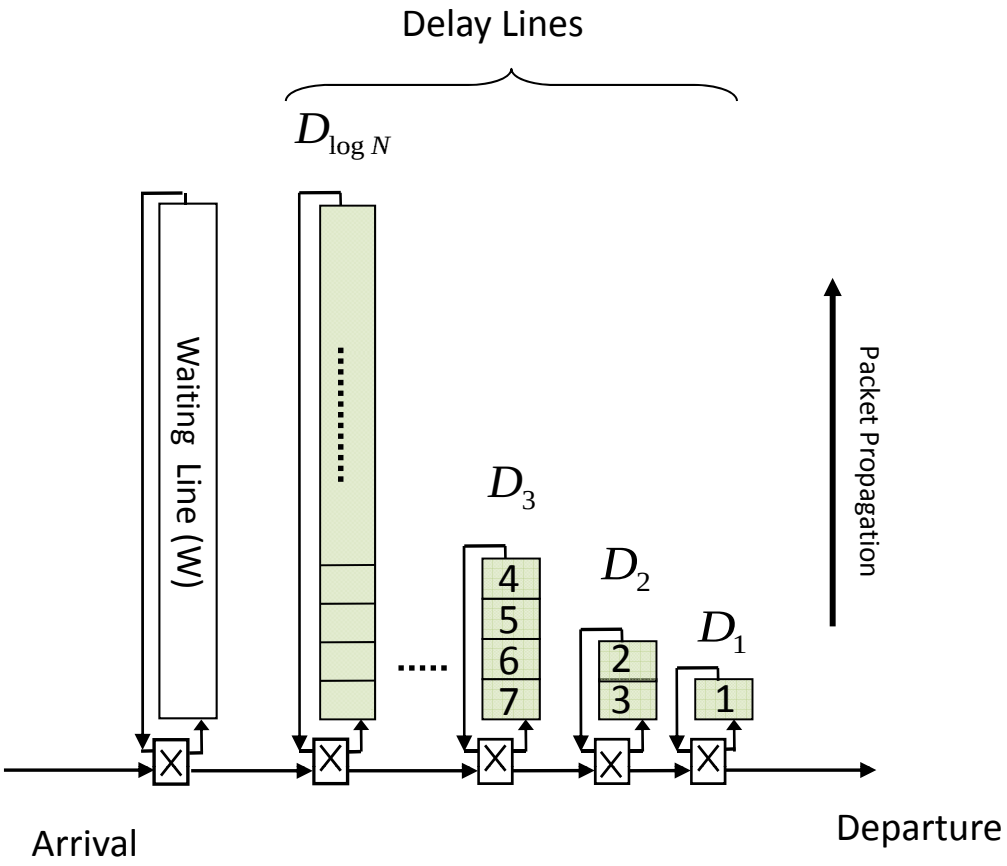


Figure 6.2: Constructing a FIFO buffer with optical delay loops. The scheduler controls the locations of the arriving packets by passing them through a waiting line first. Here, the delay loops are shown as straight lines for the sake of presentation.

At the end of each time slot, the scheduler determines the next locations of the head-of-line packets. The scheduler decides whether to recirculate a packet in its current delay line or to send it to a shorter delay line. This decision is made independently for each head-of-line packet, based only on the order of the packet: in the following time slot, the packet will be transferred to the tail location of the longest

delay line whose length is not greater than the departure order of the packet. More precisely, if the head-of-line packet has departure order k, then it will be placed in delay line $D_{\lfloor \log k \rfloor + 1}$. This ensures that the packet with departure order one is always at the head of one of the delay lines.

The same scheduling policy applies when the waiting time of the head-of-line packet in W is decremented to zero; the packet will be transferred to the tail of the longest delay line whose length is not greater than the departure order of the packet.

Theorem 6.1. Under scheduling Algorithm A, the set of delay lines $\{D_i\}$ and waiting line W exactly emulates a FIFO buffer of size $N - 1$.

The proof is in Appendix E, where we show that Algorithm A has the following three properties.

(i) Occupancy of W is always smaller than $N/2$.

(ii) There is no contention among head-of-line packets.

(iii) The packet with departure order one is always at the head of a delay line.

The first property is required for constructing the waiting line W with $\log N - 1$ delay lines. This construction is explained in the next section.

The second property is required to avoid dropping packets after they enter the buffer. The scheduling algorithm needs to relocate the head-of-line packets in such a way that there is no contention for a given location, i.e., at any time slot, at most one packet should be switched into each delay line.

The last property is required so that the departure sequence will exactly follow that of a FIFO, without any delay.

The above properties guarantee that no arriving packet will be dropped as long as the total number of packets in the buffer remains less than N. Packets depart in the same order they arrive, and each packet is delivered to the output link at the same time as in the emulated FIFO buffer.

Algorithm A- Packet Scheduling

1. arrival event

   let k be the total number of packets in the buffer.

   if $k = N - 1$ **then**

   drop the arrived packet

   else

   denote by p_{k+1} the arrived packet and by p_k its preceding packet in the buffer.

   if p_k is in W **then**

   place p_{k+1} in W

   else

   $waiting\_time \leftarrow (h+1) \mod d$, where d is the length of the delay line which contains p_k, and h is the distance of p_k from the head of the line.

   if $waiting\_time > 0$, place the arrived packet in W. Otherwise, place it in the longest queue with length smaller than or equal to the order of the packet.

2. departure event

   remove the packet with order 1 from the buffer, and decrease the order of all the remaining packets by 1.

3. scheduling the head-of-line packets

   for $i = 1, 2, ..., \log N$, move the packet at the head of D_i to the longest delay line with length smaller than or equal to the order of the packet.

   if $(waiting\_time > 0)$ **then**

   $waiting\_time \leftarrow waiting\_time - 1$

   if $(waiting\_time = 0)$ & (W is nonempty) **then**

   remove the head-of-line packet from W. Place the packet in the longest delay line with length smaller than or equal to the order of the packet.

6.3.2 Constructing the waiting line

To avoid future contention among head-of-line packets, the scheduling algorithm does not allow any void places between packets with successive departure orders. To achieve this, an arriving packet is kept in the waiting line until its preceding packet goes to a tail location. The waiting line needs to be able to buffer at most $N/2$ packets because our algorithm always keeps the number of packets in the waiting line smaller than $N/2$ (property (i) in Theorem 6.1).

Consider a set of delay lines $D'_i, i = 1, 2, ..., \log N - 1$, where the lengths of the delay lines grow as $1, 2, 4, ..., 2^{(logN)-1}$, generating an overall delay length of $N - 1$. Upon the arrival of each packet, the scheduler knows how long the packet needs to be kept in W before being transferred to one of the delay lines. Based on the availability of this information, we develop Algorithm B in the following way.

When a packet p arrives at the buffer and is scheduled by Algorithm A to be placed in W, then

- Calculate the binary expansion of the waiting time of p, i.e., the duration that p needs to wait in W before moving to one of the delay lines $\{D_i\}$. This determines which delay lines from the set $\{D'_i\}$ the packet should traverse before leaving W: if the jth bit in the binary representation is non-zero, then the packet needs to traverse delay line D'_j.

- Starting from the shortest line, when the packet reaches the end of a delay line, place it in the next delay line corresponding to the next non-zero bit.

Packet p leaves W after it traverses all the delay lines corresponding to the non-zero bits in its binary expansion. A *waiting* packet traverses any delay line in the set $\{D'_i\}$ at most once, and never recirculates in the same delay line.

Because the waiting time of each packet is known upon its arrival, the sequence of switch states in the following time slots can be determined at the time the packet arrives. Therefore, the switching decisions do not need to be delayed until packets reach tail locations. Consider a packet that arrives at time t and the binary representation of its waiting time is $b_{(\log N)-1}...b_2b_1$. For each nonzero bit b_i, the switch of

line D'_i needs to be closed twice; at time $t + \sum_{j=0}^{i-1} b_j 2^j$, to let the packet enter the delay line, and at time $t + \sum_{j=0}^{i} b_j 2^j$ to let the packet move to the next scheduled delay line (b_0 is assumed to be zero).

Tradeoff between number of switches and total fiber length. Although the number of 2×2 switches in our proposed scheme is only $2 \log N$, the total length of delay lines used for exact emulation is $1.5N$. The scheduling algorithm needs this extra $N/2$ length in order to place the packets in delay lines $\{D_i\}$ consecutively by sending the arriving packets to the waiting line first. The length of the waiting line must be equal to the length of the longest delay line (which corresponds to the longest possible time it takes for a packet in any of the delay lines goes to a tail location). Consequently, the extra fiber length will be reduced if the length of the longest delay line is reduced.

Figure 6.3 illustrates an example where the longest delay line is replaced by two half-sized delay lines. With this configuration, the size of the waiting time W can be halved too, which results in only 0.25% extra fiber length. By repeating this procedure k times, the total number of switches added to the system will be $2^{k+1} - 2k - 2$, while the extra fiber length will approximately be $\frac{1}{2^{k+1}}$ that of the original scheme. By setting $K = 2^k$, the exact tradeoff is stated in the following theorem.

Theorem 6.2. A FIFO queue of size $N - 1$ can be emulated by $2(\log N + K - \log K) - 3$ delay lines with an aggregate length of $N - 1 + \frac{N-2K}{2K}$, where $K = 2^0, 2^1, ..., 2^{(\log N)-1}$.

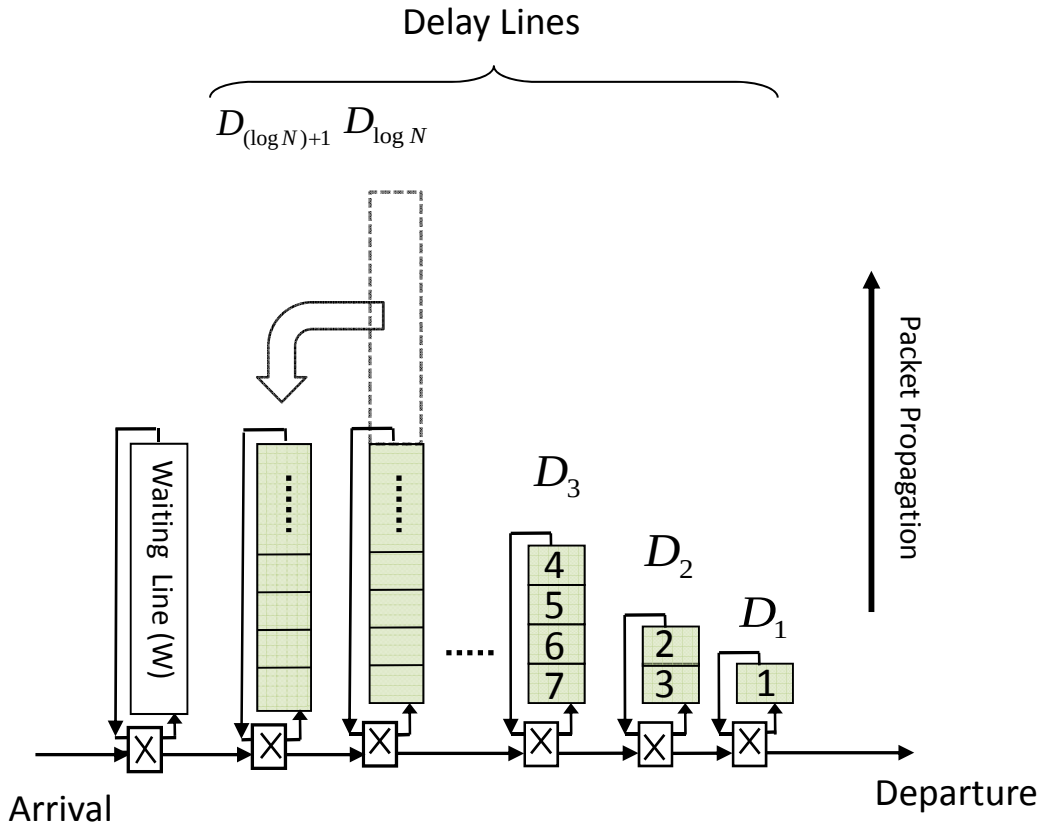


Figure 6.3: Trade-off between the number of delay lines and the maximum delay line length.

Chapter 7

Conclusion

The analysis, simulations, and experiments carried out in this work substantiates and elaborates on the suggested *tiny buffers* rule for Internet backbone routers [25]: backbone routers perform well with buffers of size 10-50 packets if their arrival traffic is non-bursty. We saw (in a wide variety of simulation and experimental settings) that if a bottleneck link runs a few tens of times faster than the access links, then the traffic will be sufficiently smooth to achieve over 80% link utilization with tiny buffers. In our simulations and experiments we assumed a core-to-access bandwidth ratio of 10-100. Our measurements on commercial backbone links, however, show that the vast majority of flows come from much slower access links; this could result in even higher utilization.

From the users' perspective, short flows could benefit from tiny buffers and experience faster completion times. This is because of a shorter round-trip time in a network with tiny buffers: reducing the buffer size to a few packets almost eliminates queueing delays. Larger flows could experience a longer completion time with tiny buffers if the link is heavily congested. A sufficiently over-provisioned bottleneck link eliminates this increased download time (Chapter 4).

The tiny buffers result is not limited to a single router. We can implement such buffers in all routers inside a backbone network as long as the ingress traffic is smooth. In general, we don't expect to see a difference between traffic burstiness at ingress ports and inside the backbone network. For cases where adversarial traffic matrix or

network topology make the ingress traffic bursty, we proposed Bounded Jitter Policy (Chapter 5), which makes the traffic at each router behave as if it comes directly from the ingress ports of the network.

While our simulations and experiments strongly suggest that routers perform well with tiny buffers, ultimately, network operators and service providers need to verify if it is so in operational backbone networks. When verified, this can have a significant implication in building all-optical routers, which have very limited buffering capacities.

Most of the simulations and experiments run in this work are with TCP Reno implemented at the end hosts. As we saw in Chapters 3 and 4, more recent variants of TCP, which are designed for large delay-bandwidth networks, achieve even higher throughput with tiny buffers compared to TCP Reno. Buffer sizing for traffic that uses other (non-TCP) congestion-aware algorithms requires further study.

The crucial assumption in the results of this work is that the arrival traffic is non-bursty. Buffer sizing in networks with different traffic patterns (e.g., data centers) needs to be studied separately. The problem has attracted interests in data center design because low-cost commodity switches, suggested as the connecting fabric in these networks [8], have small buffering capacities.

Appendix A

NetFPGA

The NetFPGA platform [5] consists of a PCI form-factor board that has an FPGA, four 1-Gigabit Ethernet ports, and memory in SRAM and DRAM. Figure A.1 shows the components in more details. Several reference designs have been implemented on NetFPGA: An IPv4 router, a learning Ethernet switch, and a NIC.

All the designs run at line-rate and follow a simple five-stage pipeline. The first stage, the Rx Queues, receives packets from the Ethernet and from the host CPU via DMA and handles any clock domain crossings.

The second stage, the Input Arbiter, selects which input queue in the first stage to read a packet from. This arbiter is currently implemented as a packetized round-robin arbiter.

The third stage, the Output Port Lookup, implements the design specific functionality and selects the output destination. In the case of the IPv4 router, the third stage will check the IP checksum, decrement the TTL, perform the longest prefix match on the IP destination address in the forwarding table to find the next hop, and consult the hardware ARP cache to find the next hop MAC address. It will then perform the necessary packet header modifications and send the packet to the fourth stage.

The fourth stage is the Output Queues stage. Packets entering this stage are stored in separate SRAM output queues until the output port is free. At that time, a packet is pulled from the SRAM and sent out either to the Ethernet or to the host

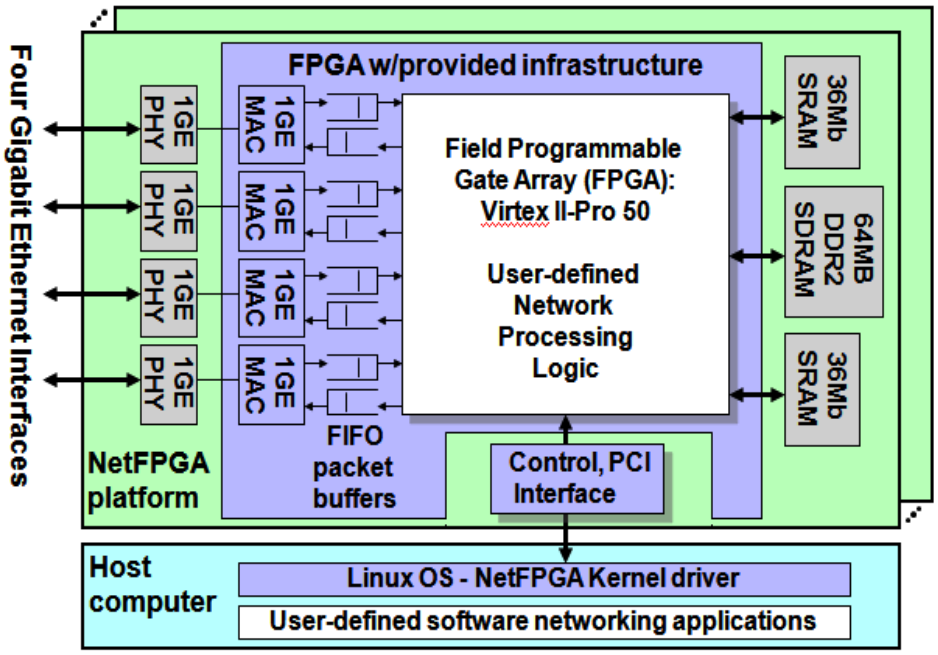


Figure A.1: A block diagram of the NetFPGA hardware platform. The platform consists of a PCI card which hosts a user-programmable FPGA, SRAM, DRAM, and four 1 Gb/s Ethernet ports.

CPU via DMA.

The fifth stage, the Tx Queues, is the inverse of the first stage and handles transferring packets from the FPGA fabric to the I/O ports.

The Buffer Monitoring design augments the IPv4 router by inserting an Event Capture stage between the Output Port Lookup and the Output Queues (Figure A.2). It allows monitoring the output queue evolution in real time with single cycle accuracy. This stage consists of two main components: an Event Recorder module and a Packet Writer module.

The Event Recorder captures the time when signals are asserted and serializes the events to be sent to the Packet Writer, which aggregates the events into a packet by placing them in a buffer. When an event packet is ready to be sent out, the Packet Writer adds a header to the packet and injects it into the Output Queues. From there the event packet is handled just like any other packet.

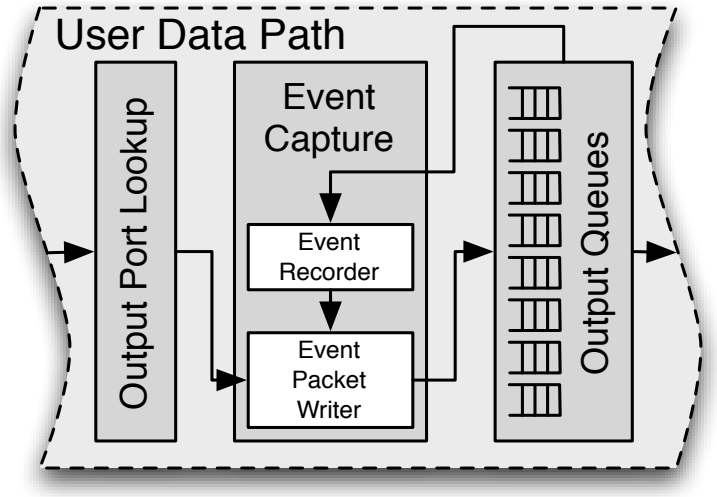


Figure A.2: The Buffer Monitoring Module in NetFPGA.

Appendix B

Proof of Lemma 5.1

To find an upper bound on the drop rate at router R_m, we first assume that both routers R_m and R have unlimited buffer sizes.

As shown in Figure 5.1, router R_m is at distance m from the ingress ports in a tree-structured network, and router R is in a single-router network. Both networks have the same input sequences. In the tree-structured network, packets go through exactly m FIFO queues—each with a departure rate of one packet per unit of time—before reaching R_m. In the single-router network, packets are directly forwarded to R.

With any arbitrary ingress traffic (not limited to Poisson traffic), we can compare the queue size of R_m (number of packets in this router) at time t with the queue size of R at time $t-m$, denoted by $q_m(t)$ and $q(t-m)$, respectively.

Consider router R_m with $A_m(t-\tau,t)$ denoting its number of arrivals during the interval $[t-\tau,t)$. We define τ_0 as follows:

$$\tau_0 = \max_{\tau}\{0 \le \tau \le t; A_m(t-\tau,t) - \tau \ = q_m(t)\}. \tag{B.1}$$

Here, τ_0 corresponds to the longest possible interval during which the backlog $q_m(t)$ is built up. By definition of τ_0, there are no arrivals at R_m in the interval $[t-\tau_0-1, t-\tau_0)$; otherwise, τ_0 should have been replaced by τ_0+1 in Equation B.1.

All packets arriving at R_m in the interval $[t-\tau_0,t)$ must have entered the network

in the interval $[t - \tau_0 - m, t - m)$, based on the following two arguments.

1. It takes at least m time units for each packet to reach the m-th buffer because the departure rate (in all buffers) is one packet per unit of time. Therefore, all these packets have entered the network earlier than time $t - m$.

2. The system is work-conservative: a packet is queued in a buffer for d time units only if there are d departures during this time. Therefore, all packets that arrive at R_m in the interval $[t - \tau, t)$ must have entered the network later than $t - \tau - m$. Otherwise, there should have been at least one arrival at R_m in the interval $[t - \tau_0 - 1, t - \tau_0)$, which contradicts our assumption.

The conclusion is that router R_m has received at least $A_m(t - \tau_0, t)$ packets in the interval $[t - \tau_0 - m, t - m)$.

Now consider the single router R. We have assumed that both networks have the same arrivals; therefore, router R also receives at least $A_m(t - \tau_0, t)$ packets in the interval $[t - \tau_0 - m, t - m)$. It follows that

$$q(t - m) \geq A_m(t - \tau_0, t) - \tau_0 = q_m(t). \tag{B.2}$$

In other words, the queue size of router R at time $t - m$ is not smaller than the queue size of router R at time t. Note that this inequality holds even if the intermediate routers in the tree-structured network have limited buffer sizes and drop packets.

Now we use the above inequality to calculate an upper bound on the drop rate when the buffer size in limited. If router R_m has buffer size B, its drop probability, $P_{\text{drop}}(B)$, will be smaller than the probability that the queue size of R_m exceeds B in the previous case (with unlimited buffer size). This is in turn smaller than the probability that the queue size of R exceeds B (as shown in Equation B.2).

With Poisson ingress traffic of rate $\rho < 1$, the queue occupancy distribution of an M/D/1 FCFS queueing system is geometric with parameter ρ. Therefore, the following inequality holds.

$$P_{\text{drop}}(B) \leq \rho^B.$$

Appendix C

Proof of Theorem 5.2

.

We assume that Δ is an integer, i.e., the jitter bound is a proper multiple of the time unit. Time slot i in this proof refers to the time interval $[i-1, i)$.

Consider a router that receives a packet with time stamp t. The router updates the time stamp to $t + D$ (which we call the departure time stamp), and tries to send the packet in the latest available time slot among $s - \Delta + 1$, $s - \Delta + 2$, ... , s. If none of these time slots are available, then the router drops the packet.

Note that under BJP, when a packet arrives at a buffer, it never finds more than $D + \Delta$ packets in the buffer; otherwise, at least one packet stays in the buffer longer than $D + \Delta$ time units, and our jitter policy is violated. Therefore, if we set the buffer size B of all routers greater than or equal to $D + \Delta$, no packet will be dropped due to buffers being full. With this buffer size, packets will be dropped only if they cannot be scheduled for in-time departure, i.e., when there are exactly Δ packets in the buffer whose departure time slots are $s - \Delta + 1$, $s - \Delta + 2$, ... , s.

Let P_m be the probability of having m consecutive busy time slots i, $i + 1$, ... , $i + m - 1$ followed by an available time slot. BJP tries to send out each packet in a time slot that is as close as possible to (and not later than) the departure time stamp of the packet. Therefore, if the $(m + 1)$-th time slot is available, the departure time stamps of all these m packets must fall within the interval $[i, i + m - 1)$. This means that there are at least m packets whose departure time stamps are in this interval

of length m. The drop probability at each buffer can be upper bounded as shown in Equation C.1.

$$P_{\text{drop}} \leq \sum_{m=\Delta}^{\infty} P_m \leq \sum_{m=\Delta}^{\infty} P\{A_{[m]} \geq m\}, \tag{C.1}$$

where $A_{[m]}$ is the number of packets whose departure time stamps are in an interval of length m. But we have assumed that at the ingress ports of the network, packet arrivals of each flow follow a Poisson model. This means that the departure time stamps of packets in each flow are also distributed according to a Poisson process (or a subsequence of a Poisson process if some packets are dropped by upstream routers); these time stamps are the arrival times (at ingress ports of the network) added by a fixed delay. Therefore, the following inequalities hold [31]:

$$\begin{aligned} P\{A_{[m]} \geq m\} &< (1 - \frac{\rho m}{m+1})^{-1} \frac{e^{-m\rho}(m\rho)^m}{m!} \\ &< \frac{1}{1-\rho} e^{m(1-\rho)} \rho^m. \end{aligned}$$

By substituting each term of the geometric sum in Equation C.1 by its upper bound, we get

$$P_{\text{drop}} < \frac{(\rho e^{1-\rho})^{\Delta}}{(1-\rho)(1-\rho e^{1-\rho})} \leq \frac{(\rho e^{1-\rho})^{\Delta}}{(1-\rho e^{1-\rho})^2}.$$

Let n be the maximum number of routers on each route in the network. To make the overall drop rate smaller than ϵ, it suffices to make the drop rate at each router smaller than ϵ/n, and hence, choose Δ such that

$$\frac{\alpha^{\Delta}}{(1-\alpha)^2} \leq \epsilon/n, \tag{C.2}$$

where $\alpha = \rho e^{1-\rho}$. This puts the following logarithmic constraint on the size of the jitter:

$$\Delta \geq 2 \log_{1/\alpha}(\frac{n}{(1-\alpha)\epsilon}). \tag{C.3}$$

Equation C.2 shows that the drop rate is only a function of the jitter bound Δ, and not of the fixed delay D. But we have assumed that D is greater than or equal to Δ. Therefore, a buffer size of 2Δ is enough to achieve a drop rate of ϵ across the network. This completes the proof of theorem 5.2.

Appendix D

Proof of Theorem 5.3

An effective utilization of θ will be achieved if the overall drop rate of packets in the network is upper bounded as follows [25].

$$P_{\text{drop}} < \frac{2(1-\theta)}{W^2}.$$

Using the result of Theorem 5.2, it suffices that

$$\frac{n\alpha^{\Delta}}{(1-\alpha)^2} \leq \frac{2(1-\theta)}{W^2}.$$

Therefore, buffer size B,

$$B = 2\Delta \geq 4\log_{1/\alpha}(\frac{nW}{(1-\alpha)(1-\theta)})$$

results in effective utilization θ.

Appendix E

Proof of Theorem 6.1

We show that our proposed FIFO construction has the following three properties.

(i) Occupancy of W is always smaller than $N/2$.

The scheduler keeps an arriving packet p in the waiting line until its preceding packet is in a tail location. In the time slot following that, the scheduler moves packet p from the waiting line, and transfers it to one of the delay lines.

If W is empty when packet p arrives, i.e., if the preceding packet is in one of the delay lines, then the maximum waiting time will be $N/2 - 1$, which corresponds to the travel time in the longest delay line. During this waiting time, all the arriving packets that are admitted to the buffer (according to step 1 of Algorithm A) will be kept in W. When the waiting time goes to zero, packet p and all other packets in W will be sent to the delay lines consecutively, one at each time slot, until the waiting line is empty. Therefore, the occupancy of W never exceeds $N/2 - 1$.

(ii) There is no contention among head-of-line packets.

Assume that there is no packet in the buffer at time 0, and that at any time before time $t + 1 > 0$, at most one packet is transferred to delay line D_i, $i = 1, ..., \log N$. We prove that the same holds at time $t + 1$.

Note that we have assumed that time is slotted (see Section 6.2). Therefore, t is

an integer, which we refer to as either *time* or *time slot*.

To prove that at time $t+1$ at most one packet is transferred to each of the delay lines, we first show that for any two packets p and p', where p' immediately follows p in departure order, the following also holds at any time $0 < \tau \leq t$:

(†) p' is either in the same line that contains p or in a longer line.

(‡)
$$l'(\tau) - l(\tau) \equiv 1 \mod d(\tau),$$

where $l(\tau)$ and $l'(\tau)$ are the locations of packets p and p' (enumerated as shown in Figure 6.2), respectively, and $d(\tau)$ and $d'(\tau)$ are the lengths of their delay lines, respectively.

When packet p' is transferred from the waiting line to one of the delay lines (at time t_0), step 1 of Algorithm A ensures that both † and ‡ hold. In the following, we show that they also hold at any time up to time t. Our arguments are mainly based on our scheduling policy of placing head-of-line packets in delay lines shorter than the departure orders of packets.

At any time slot τ, $t_0 < \tau \leq t$, packet p' changes line only if it is at a head location and its departure order is smaller than the length of the current line. But from one time slot to the next one, the departure order of packet p' either remains constant or decreases by one. In other words, the departure order of packet p at time $\tau - 1$ must have been smaller than or equal to that of packet p' at time τ. This means that at time $\tau - 1$ (and hence at time τ) packet p must be in a shorter line than the line packet p' is transferred to at time τ. Therefore, † holds at time τ.

Head-of-line packets either stay in their current lines (recirculate) or move to shorter delay lines. Therefore,

$$\begin{aligned} l(\tau-1) - l(\tau) &\equiv 1 \mod d(\tau), \\ l'(\tau-1) - l'(\tau) &\equiv 1 \mod d'(\tau). \end{aligned} \tag{E.1}$$

The length of each delay line is divisible by the length of all shorter delay lines, i.e., $d'(\tau-1)$ is a multiple of $d(\tau)$ in Equation E.1. Therefore,

$$l'(\tau) - l(\tau) \equiv l'(\tau-1) - l(\tau-1) \mod d(\tau).$$

By applying this equation to all successive time slots down to t_0, the following holds.

$$l'(\tau) - l(\tau) \equiv l'(t_0) - l(t_0) \mod d(\tau).$$

According to step 1 of Algorithm A, $l'(t_0) - l(t_0) = 1$. It immediately follows that ‡ holds at time τ.

To show that (ii) holds at time $t+1$, we consider two head-of-line packets a and b at time t, located at the head of two delay lines with length d_a and d_b, where $d_a < d_b$. We denote the departure orders of these packets by π_a and π_b, and their locations by l_a and l_b, respectively.

Since † holds at time t, π_a must be smaller than π_b. By successive application of ‡ we have

$$l_b - l_a \equiv \pi_b - \pi_a \mod d_a.$$

But $l_b - l_a \equiv 0 \mod d_a$; therefore, $\pi_b - \pi_a = kd_a$ for some integer $k \geq 1$. Our scheduling policy is based on placing each head-of-line packet in the longest line whose length is not greater than the departure order of the packet. If packet a is scheduled to be at the tail of a delay line with length d_0 in the following time slot, then $\pi_a \geq d_0$ and

$$\pi_b = \pi_a + kd_a > (k+1)d_0 \geq 2d_0.$$

Therefore, there is at least one longer delay line that can accommodate packet b without violating our scheduling policy. This shows that none of the head-of-line packets will compete for the same location at time $t+1$.

(*iii*) The packet with departure order one is always at the head of a delay line.

When packet p with departure order k reaches the head of a delay line, it will be placed in a delay line with length $l \leq k$ in the following time slot. There is at most one departure from the buffer in any time slot; therefore, the departure order of packet p cannot be one unless it is at a head location.

Bibliography

[1] http://www.qdrsram.com. 5, 6

[2] http://www.micron.com/products/dram. 5

[3] http://www.rldram.com. 6

[4] The CAIDA Anonymized 2008 Internet Traces, http://www.caida.org/data/passive/passive-2008-dataset.xml. 19

[5] The NetFPGA Project, http://www.netfpga.org/. 37, 75

[6] The network simulator - ns-2, http://www.isi.edu/nsnam/ns/. 21

[7] A. Aggarwal, S. Savage, T. Anderson, "Understanding the performance of TCP pacing," *Proceedings of the IEEE INFOCOM '00*, pp. 1157-1165, Tel-Aviv, Israel, March 2000. 9, 15

[8] M. Al-Fares, A. Loukissas, A. Vahdat, "A scalable, commodity data center network architecture," *Proceedings of the ACM SIGCOMM '08*, Seattle, WA, August 2008. 74

[9] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing router buffers," *Proceedings of the ACM SIGCOMM '04*, pp. 281-292, New York, NY, USA, 2004. 9, 14

[10] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, J. Naous, G. Salmon, "Performing time-sensitive network experiments," *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, San Jose, California, November 2008. 48

[11] N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, "Experimental study of router buffer sizing," *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, Vouliagmeni, Greece, October 2008. 14

[12] D. J. Blumenthal, "Optical packet switching," *Proceedings of European Conference on Optical Communication (ECOC2004)*, Stockholm, Sweden, pp. 830-833, 2004. 7

[13] D. J. Blumenthal, B. E. Olsson, G. Rossi, T. E. Dimmick, L. Rau,M. Masanovic, O. Lavrova, R. Doshi, O. Jerphagnon, J. E. Bowers, V. Kaman, L. A. Coldren, J. Barton, "All-optical label swapping networks and technologies," *Journal of Lightwave Technology*, vol. 18, no. 12, pp. 2058-2075, December 2000. 7

[14] E. F. Burmeister, D. J. Blumenthal, J. E. Bowers, "A comparison of optical buffering technologies," *Optical Switching and Networking*, vol. 5, no. 1, pp. 10-18, March 2008. 8

[15] E. F. Burmeister, J. E. Bowers, "Integrated gate matrix switch for optical packet buffering," *IEEE Photonics Technology Letters*, 18 (1)(2006) 103105. 8

[16] A. Carena, M. D. Vaughn, R. Gaudino, M. Shell, D. J. Blumenthal, "OPERA: An optical packet experimental routing architecture with label swapping capability," *Journal of Lightwave Technology*, vol. 16, no. 12, pp. 2135-2145, December 1998. 7

[17] M. Carson, D. Santay, "NIST Net: a linux-based network emulation tool," *ACM SIGCOMM Computer Communication Review*, 33(3):111-126, July 2003. 39

[18] C. S. Chang, Y. T. Chen, D. S. Lee, "Constructions of optical FIFO queues," *IEEE Transactions on Information Theory*, vol. 52, pp. 2838-2843, June 2006. 64, 65

[19] C. S. Chang, D. S. Lee, C. K. Tu, "Recursive construction of FIFO optical multiplexers with switched delay lines," *IEEE Transactions on Information Theory*, vol. 50, pp. 3221-3233, 2004. 64

[20] N. Chi, Z. Wang, S. Yu, "A large variable delay, fast reconfigurable optical buffer based on multi-loop configuration and an optical crosspoint switch matrix," *Optical Fiber Communication Conference*, Anaheim, CA, 2006. 8

[21] S. T. Chuang, A. Goel, N. McKeown, B. Prabhakar, "Matching output queueing with a combined input output queued switch," *Proceedings of the IEEE INFOCOM '99*, pp. 1169-1178, 1999. 2, 18, 19

[22] R. L. Cruz, J. T. Tsai, "COD: alternative architectures for high speed packet switching," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 11-20, February 1996. 64

[23] J. Dai, B. Prabhakar, "The throughput of data switches with and without speedup," *Proceedings of the IEEE INFOCOM '00*, vol. 2, pp. 556-564, Tel-Aviv, Israel, March 2000. 2, 23

[24] A. Dhamdhere, H. Jiang, C. Dovrolis, "Buffer sizing for congested Internet Links," *Proceedings of the IEEE INFOCOM '05*, 2005. 10

[25] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, "Routers with very small buffers," *Proceedings of the IEEE INFOCOM '06*, Barcelona, Spain, April 2006. 9, 11, 16, 56, 62, 73, 84

[26] C. J. Fraleigh, "Provisioning Internet backbone networks to support latency sensitive applications," PHD thesis, Stanford University, Department of Electrical Engineering, June 2002. 14

[27] M. Grossglauser, S. Keshav, "On CBR service," *Proceedings of the INFOCOM '96*, pp 129-137, March 1996. 56

[28] D. K. Hunter, M. C. Chia, I. Andonovic, "Buffering in optical packet switches," *Journal of Lightwave Technology*, vol. 16, pp. 2081-2094, December 1998. 64

[29] G. Iannaccone, M. May, C. Diot, "Aggregate traffic performance with active queue management and drop from tail," *ACM SIGCOMM Computer Communication Review*, 31(3):4-13, 2001. 14

[30] I. Keslassy, S-T Chang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, N. McKeown, "Scaling Internet routers using optics," *Proceedings of the ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003. 7

[31] B. Klar, "Bounds on tail probabilities of discrete distributions," *Probability in the Engineering and Informational Sciences*, 14(2):161-171, April 2000. 82

[32] S. H. Low, F. Paganini, J. Wang, S. Adlakha, J. C. Doyle, "Dynamics of TCP/RED and a scalable control," *Proceedings of the IEEE INFOCOM '02*, New York, USA, June 2002. 5

[33] N. McKeown, "iSLIP: A scheduling algorithm for input-queued switches," *IEEE Transactions on Networking*, vol. 7, no. 2, pp. 188-201, April 1999. 26

[34] N. McKeown, V. Anantharan, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM*, vol. 1, pp. 296-302, San Francisco, CA, March 1996. 23

[35] P. Ohlen, B. E. Olsson, D. J. Blumenthal, "All-optical header erasure and penalty-free rewriting in a fiber-based high-speed wavelength converter," *IEEE Photonnics Technology Letters*, vol. 12, no. 6, pp. 663-665, June 2000. 7

[36] E. Olsson, P. Ohlen, L. Rau, G. Rossi, O. Jerphagnon, R. Doshi, D. S. Humphries, D. J. Blumenthal, V. Kaman, J. E. Bowers, "Wavelength routing of 40 Gbit/s packets with 2.5 Gbit/s header erasure/rewriting using an all-fiber wavelength converter," *Electronics Letters*, vol. 36, pp. 345-347, 2000. 7

[37] R. S. Prasad, C. Dovrolis, M. Thottan, "Router buffer sizing revisited: the role of the output/input capacity ratio," *Proceedings of the ACM CoNEXT Conference*, pp. 1-12, New York, NY, USA, 2007. 37

[38] R. Prasad, M. Jain, C. Dovrolis, "Effects of interrupt coalescence on network measurements," *Passive and Active Measurements (PAM) Conference*, April 2004. 50

[39] R. Prasad, M. Thottan, C. Dovrolis, "Router buffer sizing revisited: the role of the input/output capacity ratio," *Proceedings of the ACM CoNext Conference*, New York, December 2007. 10

[40] L. Qiu, Y. Zhang, S. Keshav, "Understanding the performance of many TCP flows," *Computer Networks*, 37(3-4):277-306, 2001. 14

[41] G. Raina, D. Wischik, "Buffer sizes for larger multiplexers: TCP queueing theory and instability analysis," *EuroNGI*, Rome, Italy, April 2005. 5, 10

[42] I. Rhee, L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *Proceedings of the 3rd PFLDnet Workshop*, February 2005. 31

[43] A. D. Sarwate, V. Anantharam, "Exact emulation of a priority queue with a switch and delay lines," *Queueing Systems: Theory and Applications*, vol. 53, no. 3, pp. 115 -125, July 2006. 64, 66

[44] R. R. Schaller, "Moores law: Past, present and future," *IEEE Spectrum*, 34(6):52-59, June 1997. 6

[45] D. Shah, P. Giaccone, B. Prabhakar, "Efficient randomized algorithms for input-queued switch scheduling," *IEEE Micro*, vol. 22, pp. 10-18, 2002. 26

[46] J. Sommers, P. Barford, "Self-configuring network traffic generation," *ACM/USENIX IMC*, 2004. 37

[47] R. Takano, T. Kudoh, Y. Kodama, M. Matsuda, H. Tezuka, Y. Ishikawa, "Design and evaluation of precise software pacing mechanisms for fast long-distance networks," *3rd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet)*, 2005. 37, 39

[48] C. Villamizar, C. song, "High performance TCP in ANSNET," *ACM Computer Communications Review*, 24(5):45-60, 1994. 9

[49] A. Viswanathan, N. Feldman, Z. Wang, R. Callon, "Evolution of multiprotocol label switching," *IEEE Communucations Magazine*, vol. 36, no. 5, pp. 165-173, May 1998. 7

[50] L. Xu, K. Harfoush, I. Rhee, "Binary increase congestion control (BIC) for fast, long distance networks," *Proceedings of the IEEE INFOCOM '04*, pp. 2514-2524, March 2004. 31