BUFFER SIZING IN INTERNET 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

Yashar Ganjali Gavgani
March 2007

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
(Principal 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
Department of Management Science and Engineering

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.

Tim Roughgarden
Department of Computer Science

Approved for the University Committee on Graduate Studies.

Abstract

Internet routers require buffers to hold packets during times of congestion. The buffers need to be fast, and so ideally they should be small enough to use fast memory technologies such as SRAM or all-optical buffering. Unfortunately, a widely used rule-of-thumb says we need a bandwidth-delay product of buffering at each router so as not to lose link utilization. This can be prohibitively large.

In a recent paper, Appenzeller *et al.* challenged this rule-of-thumb and showed that for a backbone network the buffer size can be divided by $\sqrt{N}$ without sacrificing throughput, where N is the number of flows sharing the bottleneck. In this dissertation, we explore how buffers in the backbone can be significantly reduced even more, to as little as a few dozen packets, if we are willing to sacrifice a small amount of link capacity. We argue that if the TCP sources are not overly bursty, then 20-50 packet buffers are sufficient for high throughput. Specifically, we argue that $O(\log W)$ buffers are sufficient, where W is the congestion window size of each flow. We support our claim with analysis, a variety of simulations, and some experiments in real networks.

The change we need to make to TCP is minimal – each sender just needs to pace packet injections from its window. Moreover, there is some evidence that such small buffers are sufficient even if we do not modify the TCP sources so long as the access network is much slower than the backbone, which is true today and likely to remain true in the future.

We conclude that buffers can be made small enough for all-optical routers with small integrated optical buffers.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my adviser, Prof. Nick McKeown. He has been a great source of wisdom, support, and inspiration. He taught me how to "think big", "be persistent", and "be patient" at the same time. I cannot thank him adequately.

It is a great pleasure to acknowledge the work of the readers of this thesis, Prof. Nick McKeown, Prof. Ashish Goel, and Prof. Tim Roughgarden. This work has benefited from their thoughtful comments and suggestions. I would also like to thank Nandita Dukkipati, Salem Derisavi, Afshar Ganjali, and Hamideh Emrani for reading the manuscripts of this work, and for their valuable comments.

Professor David Dill, along with Nick, supported me during my first year at Stanford; I am very thankful.

Parts of this work were done in collaboration with others. I would like to acknowledge Neda Beheshti, Daniel Blumenthal, Mihaela Enachescu, Ashish Goel, Ramesh Rajaduray, Tim Roughgarden, Mei Wang. I enjoyed working with them, and learned a lot from each of them.

At the time of submission of this dissertation, I resided outside United States. Vahbod Pourahmad, Mohsen Bayati, and Neda Beheshti helped greatly with the submission process.

Parallel to my academic life, I have had a wonderful time at Stanford mainly because of all the good friends that I have had. They have been my big family during the time I was not able to visit home and see my family and friends.

A major part of this project has been conducted in collaboration with various research laboratories. Jean Bolot, Ed Kress, Kosol Jintaseranee, James Schneider,

and Tao Ye from Sprint ATL, Stanislav Shalunov, from Internet2, Shane Amante, Kevin Epperson, Niclas Comstedt, and Darren Loher from Level 3 Communications, Jeff Blanchard, Chris Chapman, Robert Gadbois, Max Kellogg, John Kenney, Deva Pandian, and Thuy Pham from Spirent Communications, T.V. Lakshman, Marina Thottan from Lucent Technologies, Pat Kush, and Tom Wilkes from Verizon communications, all helped during different stages of experiments. I am sincerely thankful to all of them.

I am specially thankful to the past and current members of High Performance Networking Group: Guido Appenzeller, Neda Beheshti, Martin Casado, Shang-Tse Chuang, Nandita Dukkipati, Sundar Iyer, Isaac Keslassy, Pablo Molinero, Jad Naus, Paul Tarjan, Rui Zhang-Shen. These are a group of the smartest people I have known in my life, and very good friends as well. I am specially thankful to Guido for his help during the initial phase of this project.

The members of Information Systems Networking Laboratory provided extremely helpful feedback throughout this work. I would like to acknowledge Prof. Balaji Prabhakar, Abtin Keshavarzian, Devavrat Shah and Mei Wang for their valued comments and discussions.

Last but absolutely not least, I wish to express how grateful I am to my wonderful family: my parents, and my brother, whose continuous love, caring, and support made me who I am. We were physically distant during the past few years, but I know they have kept me in their prayers. My lovely wife Hamideh has been the most wonderful companion I could have wished for during this journey. She has been the greatest source of love and support. This dissertation is dedicated to her.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Packet switching is an efficient way of sharing the cost of long-haul links in data networks. By statistically multiplexing different flows, we can significantly reduce the overall cost of the network, and make it more resilient to failures. In any packet switched network, such as the Internet, we need to have "some" amount of buffering in the routers to hold packets during times of congestion. The question is how much buffering Internet routers need in order to have an acceptable performance. We call this the *buffer sizing problem*, and will study it in this dissertation.

1.1.1 Why does router buffer sizing matter?

In a packet switched network, end-to-end latency of individual packets is an important performance metric, which quantifies the behavior of the system from a user's point of view. If we take a closer look at the end-to-end latency of a given packet (Figure 1.1), we can see that it consists of three main components.

1. **Transmission delay.** This is the time it takes for a packet to be transmitted by the source host, and by any intermediate router on its path.

2. **Propagation delay.** This is the time it takes for a packet to traverse the links connecting routers; and

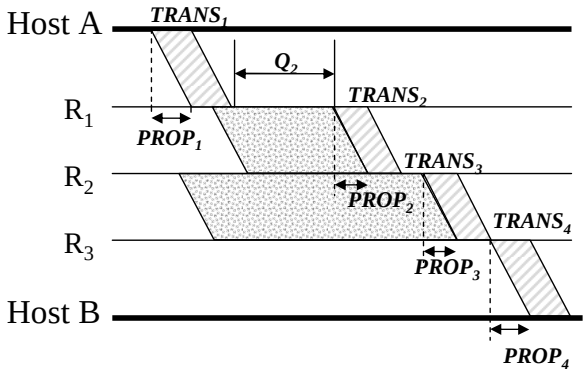

Figure 1.1: End-to-end latency of any packet consists of three components: transmission delay, propagation delay, and queueing delay. Here, the end-to-end latency is $\sum_i \text{TRANS}_i + \text{PROP}_i + Q_i$.

3. **Queueing delay.** This is the time that the packet sits in a buffer and waits for some system resource – usually the output port of the router which is blocked by other packets – to be released.

Of these three components, the first two (*i.e.* the transmission delay and the propagation delay) are fixed. Queueing delay is the only variable component of the end-to-end latency, and therefore it is what causes the variation in performance observed by the end users. In fact, one can argue that queueing delay is the single biggest cause of uncertainty in today's Internet. Clearly, queueing delay and jitter[1] are directly related to the buffer sizes. We need to understand the buffer sizing problem if we want to understand and control the queueing delay of packets.

Additionally, there are other reasons why buffer sizing is important; a considerable reduction in router buffer sizes, if it were possible, would have significant practical consequences.

First, if big electronic routers only needed very small buffers, it could reduce their complexity, making them easier to build and easier to scale. A typical 10 Gb/s router linecard today contains about one million packet buffers, using many external DRAM chips. The board space the DRAMs occupy, the pins they require, and the power

[1]Variations in queueing delay of different packets is called *jitter*.

they dissipate all limit the capacity of the router [7]. If very small buffers suffice, then packet buffers could be incorporated inside the network processor (or ASIC) in a small on-chip SRAM; in fact, the buffers would only occupy a tiny portion of the chip. Not only would external memories be removed, but the reduction in buffer sizes would allow the use of fast on-chip SRAM, which scales in speed much faster than DRAM. By eliminating some board space, and reducing the complexity of the router, we can increase the density of the system, and simply provide higher throughput.

Second, reduced buffer sizes could facilitate the building of all-optical routers. With recent advances in optical technology [33, 34, 40], it is now possible to perform all-optical switching, opening the door to routers with huge capacity and lower power than electronic routers. These advances make possible optical FCFS[2] packet buffers that can hold a few dozen packets in an integrated opto-electronic chip [40]. Larger all-optical buffers remain infeasible, except with unwieldy spools of optical fiber (that can only implement delay lines, not true FCFS packet buffers). Feasibility of having a network with a few dozen packets, which might result from studying the buffer sizing requirements, is an important step towards building an operational all-optical network.

1.1.2 Why is the router buffer sizing problem difficult?

Given the importance of the buffer sizing problem, one might reasonably think that it is already well understood. After all, we are equipped with tools like queueing theory, large deviations theory, and mean fields theory, which are focused on solving exactly this type of problem. One might think this is simply a matter of understanding the random process that describes the queue occupancy over time. Unfortunately, this is not the case.

The closed-loop nature of the flows in the Internet, *i.e.*, the fact that flows react to the current state of the network, makes it necessary to use control theoretic tools, rather than queueing theory techniques. However, we cannot simply apply control theoretic tools for this problem as they mainly emphasize the equilibrium state of the

[2]First-come, first-serve.

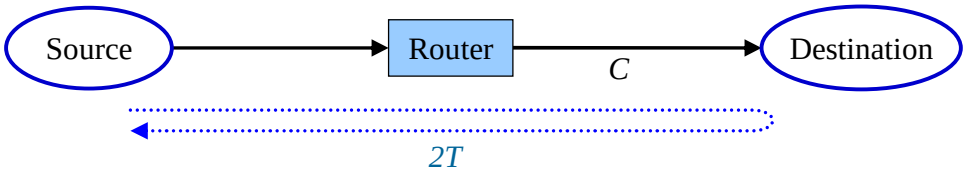

Figure 1.2: Rule-of-thumb for buffer sizing: $B = 2T \times C$.

system, rather than what we are interested here, *i.e.*, the transient state. Additionally, the discrete nature of the buffer sizing problem makes using classic control theoretic tools difficult, as they are typically designed for continuous systems.

It is also difficult to experimentally study the buffer sizing problem by reducing the buffers in a commercial backbone router. The problem is, how to decide the appropriate buffer size, without trying it in an operational network? But who would reduce buffers in an operational network, and risk losing customers' traffic, before knowing if the result is correct?

1.2 Rule-of-thumb for Buffer Sizing

If the buffer sizing problem is not easy, what do people do in practice? Buffer sizes in today's Internet routers are set based on a rule-of-thumb which says, if we want the core routers to have 100% utilization, the buffer size should be greater than or equal to $2T \times C$, also known as the *bandwidth-delay product*. Here, $2T$ is the effective round-trip propagation delay of a flow through the router (also denoted as RTT), and C is the capacity of the bottleneck link (Figure 1.2).

This bandwidth-delay rule is mandated in backbone and edge routers, and appears in several Internet RFCs [11], and architectural guidelines. Based on the rule-of-thumb, the buffer size increases linearly with capacity of the bottleneck link. Due to the limitations on the speed of light, we do not expect the propagation delay to change drastically over time, but capacity of the network is expected to grow very rapidly. Therefore, this rule can have major consequences in router design.

1.3 Related Work

There is evidence that shows the rule-of-thumb was known at the time TCP was invented [31, 30]. In the literature, however, this rule is commonly attributed to a paper by Villamizar and Song [48]. Based on experimental measurements of up to eight long-lived TCP flows on a 40 Mb/s link and with different buffer sizes, they concluded that any router needs a buffer size equal to the capacity of the router's network interface, multiplied by the round-trip time of a typical flow passing through the router, or the bandwidth-delay product.

Using *ns-2* [4] simulations, Morris [36] investigated the buffer sizing problem on a 10 Mb/s link with 25ms latency and carrying up to 1500 long-lived flows. He concluded that the required buffer size is a small multiple of the number of flows. The bandwidth-delay product of this network is approximately 217 packets, which means most of the flows have only a fraction of a packet in transit at any time. Thus, most flows are in timeout, which significantly reduces link utilization and fairness in the system. We note that it is not typical for a router used by a carrier or ISP to be under such a high load, and therefore, this result is related to access networks and has very limited implications for today's core Internet routers.

Appenzeller *et al.* proposed the rule $B = 2T \times C/\sqrt{N}$ instead, where N is the number of flows through the bottleneck link [7]. In a backbone network today, N is often in the thousands or the tens of thousands, and so the sizing rule $B = 2T \times C/\sqrt{N}$ results in significantly smaller buffers. The basic idea 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 bigger the buffers need to be so as to guarantee 100% throughput. As the number of flows increases, the aggregate window size process (the sum of all the congestion window size processes for each flow) decreases, following the Central Limit Theorem. The result relies on several assumptions: (1) That flows are sufficiently independent of each other to be desynchronized, (2) That the buffer size is dominated by long-lived flows, and perhaps most importantly (3) That there are no other significant, unmodeled reasons for buffering more packets. We call this the *small buffers* rule, and will scrutinize it

more carefully later.

Dhamdhere and Dovrolis [17] studied a particular network example to argue that when packet drop rates are considered, one might need much larger buffers – perhaps larger than the buffers in place today. Similar to Morris's work [36] they studied an example where a large number of flows share a heavily congested low capacity bottleneck link towards the edge of the network, and showed that one might get substantial packet drop rates, even if buffers are set based on the rule-of-thumb.

Recently, we (in collaboration with others) proposed reducing buffers much further to $O(1) \approx 20 - 50$ packets in backbone routers [19]. The main idea is considering the tradeoff between reducing buffers and losing some throughput – assumed to be 10-20%. In other words, when congested, links behave as if they run at 80-90% of their nominal rates. This could be an interesting assumption in networks with abundant link capacity, or in future optical networks where link capacity might be cheaper than buffers. The results depend on the network traffic being non-bursty, which can happen in two ways: (1) If the core links run much faster than the access links (which they do today), then the packets from a source are spread out and bursts are broken, or (2) TCP sources are changed so as to pace the delivery of packets. If the results are correct, and relevant, then a backbone link could reduce its buffers by five orders of magnitude. We call this the *tiny buffers* rule, and, as the main contribution of this dissertation, we will explain it in more detail in the following chapters.

The tiny buffers rule was also studied by Raina and Wischik independently [42]. They studied stability of a system under different buffer sizing rules, using control theory, and simulations, and concluded that a network with tiny buffers is stable.

To get some feel for different buffer sizing rules, consider the scenario where 1000 flows share a link of capacity 10Gb/s. Assume that each flow has a round-trip propagation delay of 100ms, a maximum window size of 64KB, and a packet size of 1KB. The peak rate is roughly 5Gb/s. The bandwidth-delay product rule-of-thumb suggests a buffer size of 125MB, or around 125,000 packets. The $2T \times C/\sqrt{N}$ rule suggests a buffer size of around 3950 packets, and based on the tiny buffers rule, we will only need a buffer size of twelve packets plus some small additive constant.

1.4 Organization of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we will describe the model used throughout the work, and give an overview of the theory behind the rule-of-thumb. In Chapter 3, we will present a theoretical analysis which shows the feasibility of having a network with a few dozen packets of buffering. We will also present simulations on the impact of buffer size reduction on network performance. In addition to verifying the limits of our theoretical work, we study the effect of other network parameters on buffer sizing requirements. Chapter 4 describes the set of buffer sizing experiments we have done in real networks or test-beds with real routers. In Chapter 5, we take a closer look at implicit/explicit assumptions in the tiny buffers rule that might cause problems when building a real system, and address these miscellaneous issues. Chapter 6 concludes this work.

Chapter 2

Overview: Rule-of-thumb and Small Buffers

In this chapter, we briefly overview the origins of the rule-of-thumb, and small buffers rule. We start by examining an extremely simple scenario (Section 2.1): an Internet core router carrying a single TCP flow. We assume the router is directly connected to the source and destination nodes of the TCP flow, and that the output link of the router is the bottleneck. The goal is to keep the bottleneck link at 100% utilization at all times. This simple scenario is the origin of the *rule-of-thumb* for buffer sizing.

In Section 2.2, we briefly review the case with more than one flow going through the router. This is the *small buffers* scenario studied by Appenzeller *et al.* [7]. We show that depending on whether flows are synchronized or not, we will have completely different requirements for buffer sizing. When flows are independent and desynchronized, the buffer sizes can be reduced by a factor of $\sqrt{N}$ without any degradation in link utilization, where N is the total number of long-lived TCP flows going through the router.

In Chapter 3, we will explore the *tiny buffers* rule – *i.e.*, how buffers in the Internet backbone can be significantly reduced to as little as a few dozen packets.

2.1 Single Flow Scenario – Rule-of-thumb

Today, Internet core routers carry thousands to tens of thousands of flows at any given time. However, the rule-of-thumb for buffer sizing comes from a setting with a single TCP flow. This discrepancy can be attributed to the shifting nature of the network traffic. In the early days of the Internet, it was quite typical for a long-haul link to carry a small number of flows at any time; this is the single flow origin of the rule-of-thumb. Even though the single flow assumption is not valid anymore, it is still useful to understand this assumption and its implications.

In this section, we first give a brief overview of how TCP works (Section 2.1.1), and why in a single flow setting we need a bandwidth-delay product of buffering based on the rule-of-thumb (Section 2.1.2).

2.1.1 Overview of TCP Behavior

Understanding how TCP works is essential for understanding the buffer sizing rules. Here, we will present a *very* simplified description of TCP Reno basic rules, enough for us to understand buffer sizing rules. We refer the interested reader to networking textbooks for a more comprehensive presentation [47, 10].

Let us consider a *source* node which is going to send some packets to a given *destination* node through a network (*e.g.*, the Internet). The source node needs to control its packet injection rate to the network (Figure 2.1); otherwise, if all nodes keep sending packets with their maximum capacity, the network might become extremely congested, and everyone's performance will suffer.

TCP is a congestion control scheme, designed to address exactly this problem. Each time the source node sends a packet to the destination, TCP stores the packet in a memory location, called the *congestion window*. The packet remains in the congestion window, until the source makes sure it has been received by the destination node. When the destination node receives the packet, it generates an *acknowledgement* packet and sends it back to the source. Upon receiving the acknowledgement packet, the source removes the corresponding packet from the congestion window.

The size of the congestion window, denoted by W, determines the packet injection

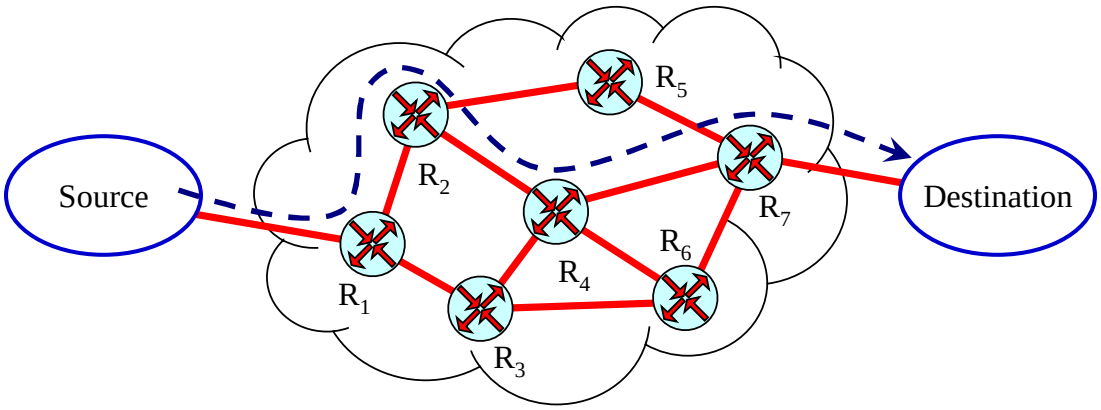

Figure 2.1: Congestion control in the Internet. To avoid overwhelming the network and causing congestion, the source needs to control packet injection rate. In this figure, packets originating at source pass through routers R_1, R_2, R_4, and R_7 before reaching the destination. Packet injection rate must be such that none of the links on the path become congested.

rate: it is effectively the number of packets which are sent out by the source during each *round-trip time* (RTT). We note that it takes one RTT from the time a packet is sent out by the source till the acknowledgement is received back.

How does TCP adjust W to make sure that the maximum capacity of the network is utilized, without the network being congested? The answer to this question depends on the state of the source node, which is either *slow-start,* or *congestion avoidance.* Initially, the source is in slow-start and W is set to two packets. The source increments W for each acknowledgement it receives – effectively doubling W every RTT.

If the source detects a packet loss, it halves its congestion window and enters congestion avoidance. In congestion avoidance, W is increased by $1/W$ each time an acknowledgement is received – effectively increasing W by one during each RTT. During congestion avoidance, any packet loss halves the congestion window, and the flow stays in the same state.

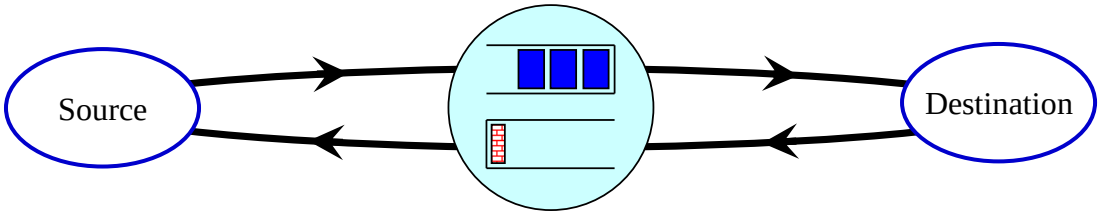

Figure 2.2: Single TCP flow going through a bottleneck link.

$$\texttt{Slow-Start:} \begin{cases} \texttt{No loss:} & W_{new} = 2W_{old} \\ \texttt{Loss:} & W_{new} = \frac{W_{old}}{2}; \text{enter congestion avoidance} \end{cases}$$

$$\texttt{Congestion Avoidance:} \begin{cases} \texttt{No loss:} & W_{new} = W_{old} + 1 \\ \texttt{Loss:} & W_{new} = \frac{W_{old}}{2} \end{cases}$$

If the source does not receive acknowledgements for several packets in time, it triggers a timeout, and goes back to the slow-start state and the initial congestion window size.

2.1.2 Origin of the Rule-of-thumb

Let us consider a TCP flow, between a pair of source-destination nodes, as shown in Figure 2.2. We use the following notation in this section.

W	The congestion window size of the source
$2T$	The Round-Trip-Time as measured by the source
C	The capacity of the bottleneck link (between router and destination)
C'	The capacity of the access link (between source and router)
R	The sending rate of the sender
Q	The length of the buffer queue (forward path)
B	The buffer size. $Q_{max} \leq B$

While in congestion avoidance, the source node will increment W once every

RTT. As long as the sending rate is less than the capacity of the bottleneck link, *i.e.*, $R < C$, all the buffers on the forward and reverse path remain empty. We note that the bottleneck link utilization is below 100% in this case and we have $W < 2T \times C$. Any increase in W will increment the number of packets which are traversing the forward or reverse path, or *on-the-fly* packets, without the need to buffer any packets.

Once W reaches $2T \times C$ (or equivalently when $R = C$) all the links are full, [1] and the bottleneck link utilization will reach 100%. From this point on, packets will start to accumulate at the forward path buffer, and we have

$$W = 2T \times C + Q.$$

This process will continue until the forward path buffer is full. At this time we have

$$W = 2T \times C + B.$$

The next increment in W will cause a packet drop since the forward path buffer is already full. When the source detects a loss, it halves W. Thus,

$$W = T \times C + B/2.$$

At this time, the number of packets which are either on-the-fly or in the forward path buffer is still $2T \times C + B$, which means the source has to stop sending packets out until it receives enough acknowledgements. If we want to keep the bottleneck link at 100% utilization at all times, we need to make the buffer size large enough to keep the bottleneck link busy from the time the source stops sending packets, until it starts again. Equivalently, the new congestion window size after the loss event, *i.e.*, $T \times C + B/2$, must be large enough so that the source will start sending packets before the number of on-the-fly packets goes below $2T \times C$ (or $R < C$). In other words, we should have

[1] The bottleneck link throttles the traffic, and therefore the capacity of bottleneck link C is what determines the total number of on-the-fly packets, even though other links have capacity $C' > C$.

$$T \times C + \frac{B}{2} \geq 2T \times C,$$

or

$$B \geq 2T \times C, \tag{2.1}$$

which is exactly the well-known rule-of-thumb for buffer sizing.

2.1.3 Verification of the Rule-of-thumb

Verifying the rule-of-thumb in its original setting is quite simple. We can easily setup a network with two nodes, and a router connecting them, or use simulations. Here, we present *ns-2* simulation results showing a single-flow network, with a bandwidth-delay product of 100 packets. Figure 2.3 depicts the congestion window size, queue occupancy, and bottleneck link utilization when the rule-of-thumb for buffer sizing is applied.

As we can see in this figure, after the initial slow-start phase, the congestion window size follows the classic saw-tooth shape. Once the congestion window size passes 100 packets (or the bandwidth-delay product), the forward and reverse paths are full. Therefore, packets start to accumulate in the buffer. When the source congestion window size reaches its maximum of 200 packets, there are about 100 packets on-the-fly, and 100 packets in the buffer. Halving the congestion window size reduces it to 100 packets, and therefore the source waits until the number of unacknowledged packets is 100 before starting to send more packets out. This is exactly when the buffer becomes empty, and thus the bottleneck link never goes idle.

Figure 2.4 represents a scenario where the buffer size (71 packets) is less than the bandwidth-delay product. Similar to Figure 2.3, the congestion window size follows the saw-tooth shape. The difference is that, here, the saw-tooth oscillates between 85 and 170 packets, and it has a higher frequency.[2] When the congestion window size equals 170 packets, the links and the buffers are all full, and the next incoming packet

[2]Since the buffer is smaller, it takes less time for it to fill up or drain.

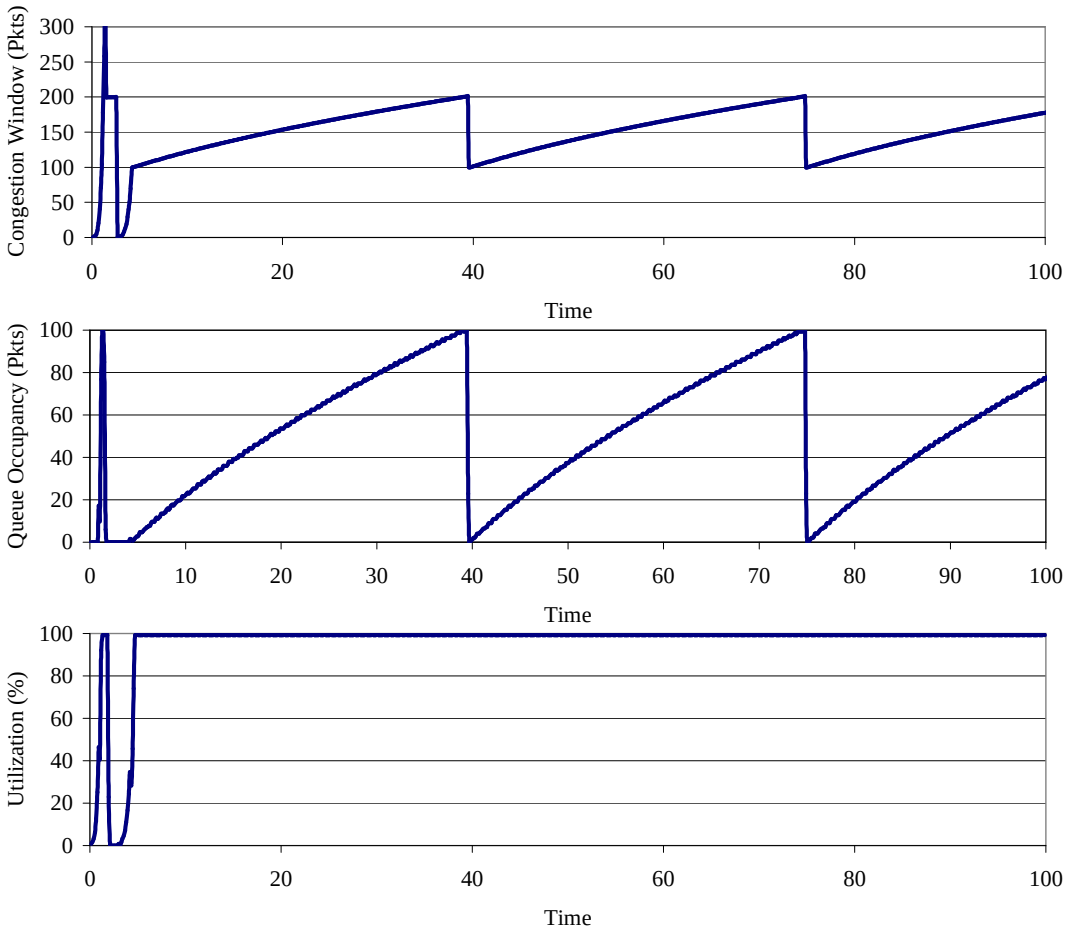

Figure 2.3: Congestion window size, queue occupancy, and link utilization for a single-flow network with buffer size equal to the bandwidth-delay product (100 packets).

will be dropped. The source will reduce its congestion window size to 85 packets, and will wait until the number of unacknowledged packets reaches this limit. Since the forward and reverse paths can hold up to 100 packets, in addition to the packets in the buffer being drained, 15 of the packets on-the-fly will leave the system as well; dropping the bottleneck link utilization below 100%. When the number of on-the-fly packets reaches 85, the source starts sending again, and fills up the pipes, pushing the bottleneck link utilization back to 100%.

Figure 2.5 depicts an over-buffered scenario. The saw-tooth shaped congestion window size oscillates between 114 and 228 packets in this case. Even when at its

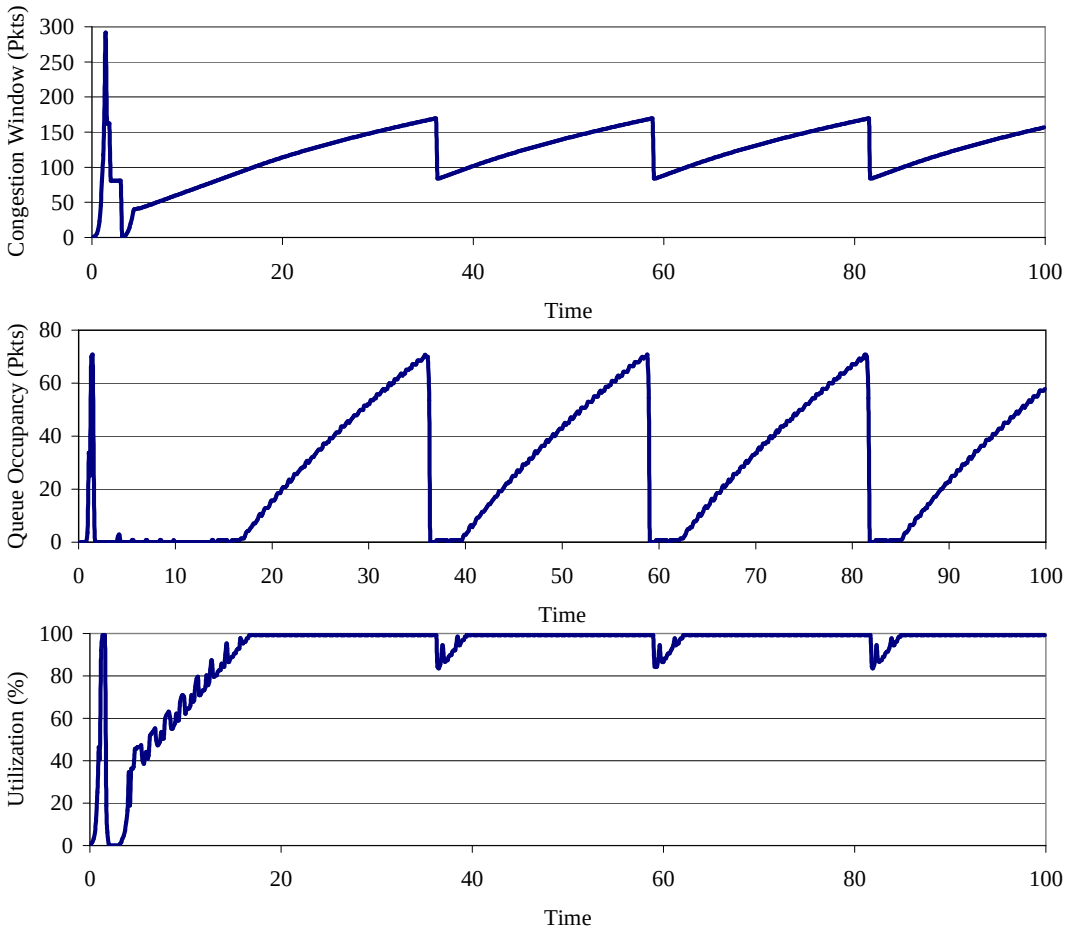

Figure 2.4: Performance of a network with buffers less than the bandwidth delay product (71 packets).

minimum, the congestion window size is larger than what is needed to keep the pipes full. In other words, when the congestion window size is halved, the buffer drains for a while, but before it becomes empty, the source starts sending out more packets. Consequently, the bottleneck link utilization remains at 100% at all times. We note that as a result of this over-buffering, every packet in the system encounters a larger delay than necessary. For instance, if the buffer size is set to $4T \times C$, or twice the rule-of-thumb, each packet will encounter a delay of $4T$, which is double the delay observed normally with a buffer of size $2T \times C$.

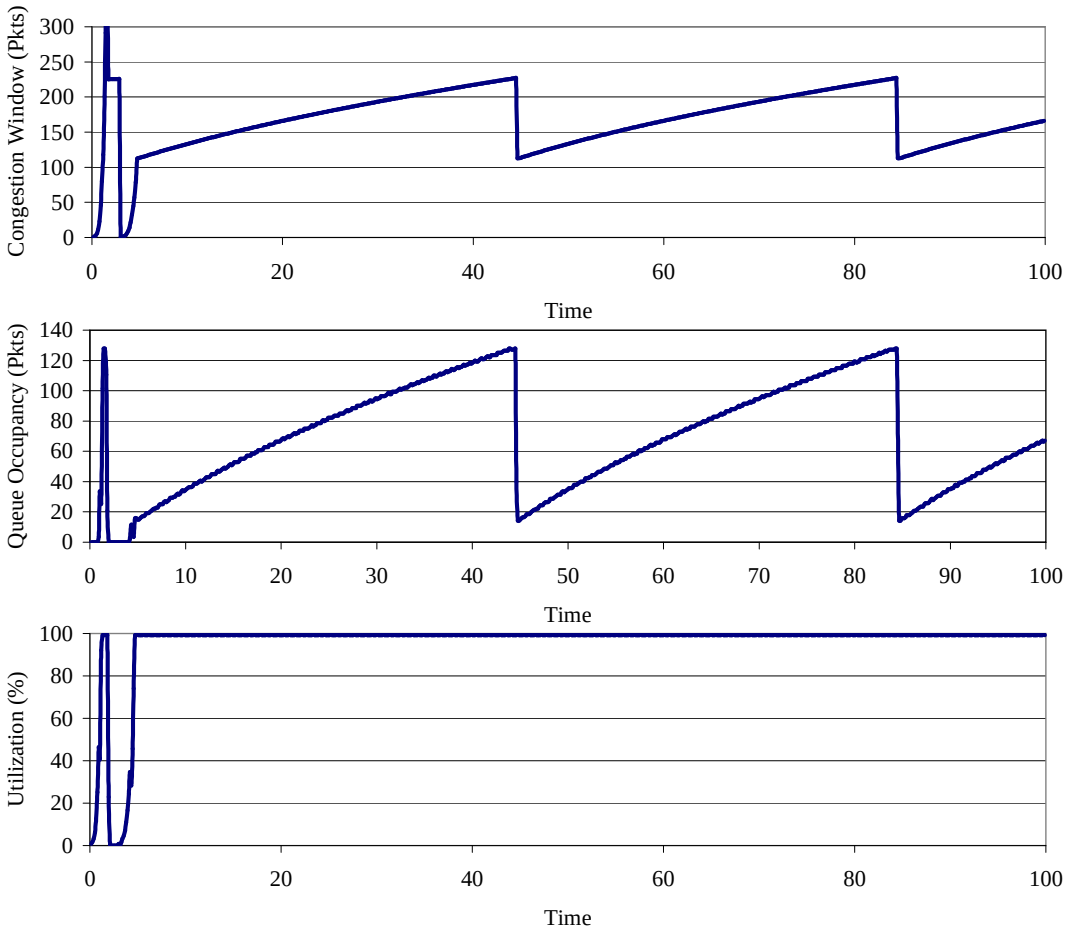

Figure 2.5: Performance of a network with buffers more than the bandwidth delay product (128 packets).

2.2 Small Buffers Rule

While the single-flow setting for the rule-of-thumb might have been appropriate in the early days of the Internet, it is not common for an Internet backbone router to carry a single-flow today. A 2.5 Gb/s (OC48c) link for example, typically carries over 10,000 flows at a time [24]. How does the change in number of flows assumption affect the rule-of-thumb?

Appenzeller *et al.* [7] studied this question in two different cases: (1) when all the flows going through the router are synchronized, *i.e.*, the saw-tooth shapes are

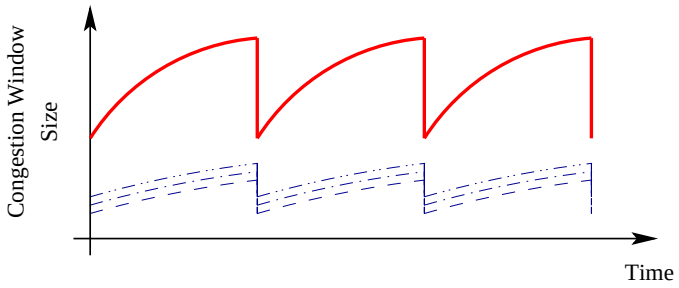

Figure 2.6: When the flows are perfectly synchronized (the dashed curve on the bottom), their aggregate (the solid curve on top) will have a similar shape to the saw-tooth shape of a single TCP flow.

perfectly in-phase, and (2) when the flows are desynchronized. They showed that the buffer size requirements of the system drops significantly in the second scenario, and provided evidence that this is what happens in practice. Here, we will give a brief overview of their results.

2.2.1 Synchronized Flows

Let us consider N long-lived TCP flows going through a shared bottleneck link. We denote the congestion window size of flow i at time t with $W_i(t)$, its two-way propagation delay with T_i, and the number of flow i packets which are in the queue with $Q_i(t)$. If all the flows are perfectly synchronized, they will all reach the peak of their saw-tooth at the same time (Figure 2.6). The total number of outstanding packets, or the *aggregate congestion window size*, $W(t)$, is then

$$\begin{aligned} W(t) &= \sum_{i=1}^{N} W_i(t) \\ &= 2T_i \times C + Q_i(t) \\ &= 2\overline{T} \times C + Q(t). \end{aligned}$$

Here, $Q(t)$ is the total number of packets in the queue, and $\overline{T}$ is the effective

one-way propagation delay of all flows.[3]

When the buffer is full, we have $Q(t) = B$. Thus,

$$W(t) = 2\overline{T} \times C + B.$$

Since all flows are synchronized, they halve their congestion window size at the same point of time, which means the aggregate congestion window size is reduced to

$$W(t) = \overline{T} \times C + B/2.$$

This is very similar to the single-flow case: if we want to have 100% link utilization the buffer size should be large enough to accommodate the variations in the aggregate congestion window size, or

$$B \geq 2\overline{T} \times C.$$

In other words, synchronized flows have the same buffer size requirements as a single-flow. In the following section, we will overview the case where flows are desynchronized.

2.2.2 Desynchronized Flows

When do flows become synchronized? The majority of results on flow synchronization are based on *ns-2* simulations [26, 50, 22], where the behavior of the system is deterministic, and there is very little randomness coming from the complex structure of the network, switching devices, link-level errors, host service times, and user behavior. On real networks, synchronization is much less frequent, and is largely limited to the cases with a small number of very heavy flows. There is no evidence of synchronization in large routers in the core of the Internet [24, 23, 28].

For a network with desynchronized flows, the aggregate congestion window size does not have a regular saw-tooth pattern as in the case of synchronized flows (Figure 2.7). Different flows, will cancel each other out, and the aggregate congestion

[3]One can easily show that $\overline{T}$ is the harmonic average of $\{T_1, T_2, \ldots, T_N\}$ defined as $\frac{1}{\overline{T}} = \sum_{i=1}^{N} \frac{1}{T_i}$.

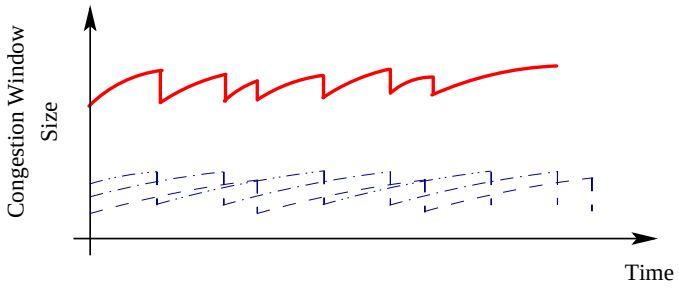

Figure 2.7: When the flows are not synchronized (the dashed curve on the bottom), their aggregate (the solid curve on top) will become smoother as the number of flows grows.

window size will become smoother. In fact, Appenzeller *et al.* have shown that as the number of flows is increased the distance from the peak to the trough of the aggregate window size will get smaller [7].

More formally, if we have N desynchronized TCP flows with random and independent start times, and propagation delays, by applying the Central Limit Theorem, we can show that the aggregate window size process will converge to a Gaussian process. Interestingly, the standard deviation of the aggregate congestion window size is decreased by a factor of $\sqrt{N}$.[4] Hence, given that the buffer size requirement is proportional to the standard deviation of the aggregate congestion window size, we can reduce the buffer sizes with a factor of $\sqrt{N}$ without degrading the performance of the system. This is what we call the *small buffer sizing* rule, or

$$B = \frac{2T \times C}{\sqrt{N}}. \tag{2.2}$$

We emphasize the underlying assumptions in this rule: we have a large number of flows that are independent, and desynchronized, and we want to have 100% link utilization throughout the network. This result has major practical implications for building routers, as it can result in a 2-3 orders of magnitude reduction in buffer sizes without any cost to the throughput of the system.

[4]We refer the reader to Appenzeller's Ph.D. thesis [6] for more detailed analysis, including cases with short-lived and long-lived flows, as well as simulations and experiments on this.

2.3 Summary

In this chapter, we gave a brief overview of the origins of the rule-of-thumb, and the small buffer sizing rule. We showed that the rule-of-thumb comes from a setting with a single TCP flow, which was typical at early days of the Internet. We also showed that increasing the number of flows changes the buffer size requirements of the network if the flows are desynchronized and independent. Specifically, we can reduce the buffer sizes in the core of the Internet with a factor of $\sqrt{N}$ if N independent flows share the core link.

Chapter 3

Tiny Buffer Sizing Rule

In this chapter, we explore how buffers in the Internet backbone can be significantly reduced to as little as a few dozen packets, if we are willing to sacrifice a small amount of link capacity. For a typical Internet router today, this is 2-3 orders of magnitude smaller than what is indicated by the small buffers rule, and the difference will grow as the capacity of the network grows.

We argue that if the TCP sources are not overly bursty, then 20-50 packet buffers are sufficient for high throughput. Specifically, we argue that $O(\log W)$ buffers are sufficient, where W is the window size of each flow. We support our claim with analysis, and a variety of simulations. The change we need to make to TCP is minimal – each sender just needs to pace packet injections from its window. Moreover, there is some evidence that such small buffers are sufficient even if we do not modify the TCP sources so long as the access network is much slower than the backbone, which is true today and likely to remain true in the future.[1]

3.1 Implications of Tiny Buffers

The problem of whether we can build networks with buffers as small as a few dozen packets is an interesting intellectual exercise in its own right. There would also be

[1]Theoretical analysis presented in this chapter was done in collaboration with Mihaela Enachescu, Ashish Goel, Tim Roughgarden, and Nick McKeown [18, 19].

practical consequences if it were possible.

First, if big electronic routers required only a few dozen packet buffers, it could reduce their complexity, making them easier to build and easier to scale. A typical 10 Gb/s router linecard today contains about one million packet buffers, using many external DRAM chips. The board space the DRAMs occupy, the pins they require, and the power they dissipate all limit the capacity of the router [7]. If a few dozen packet buffers suffice, then packet buffers could be incorporated inside the network processor (or ASIC) in a small on-chip SRAM; in fact, the buffers would only occupy a tiny portion of the chip. Not only would external memories be removed, but the reduction in buffer sizes would allow the use of fast on-chip SRAM, which scales in speed much faster than DRAM. By eliminating some board space, and reducing the complexity of the router, we can increase the density of the system, and simply provide higher throughput.

Second, reduced buffer sizes could facilitate the building of all-optical routers. With recent advances in optical technology [33, 34, 40], it is now possible to perform all-optical switching, opening the door to routers with huge capacity and lower power than electronic routers. These advances also make possible optical FCFS packet buffers that can hold a few dozen packets in an integrated opto-electronic chip [40].[2] Whether we can create an all-optical network then is a question of if having a network with tiny buffers is feasible.

In the following section, we will describe the intuition behind the tiny buffer sizing rule.

3.2 Tiny Buffers Intuition

Our main result in this chapter is that minor modifications to TCP would indeed allow us to reduce buffer-sizes to dozens of packets with the expense of slightly reduced link utilization. We obtain this result in a succession of steps. We will start by adopting two strong assumptions: (1) That we could modify the way packets are transmitted

[2]Larger all-optical buffers remain infeasible, except with unwieldy spools of optical fiber (that can only implement delay lines, not true FCFS packet buffers).

by TCP senders, and (2) That the network is over-provisioned. However, we will soon relax these assumptions.

We start by asking the following question: What if we kept the AIMD (Additive Increase Multiplicative Decrease) dynamics of TCP window control, but changed the TCP transmission scheme to "space out" packet transmissions from the TCP sender, thereby making packet arrivals less bursty? We assume that each TCP flow determines its window size using the standard TCP AIMD scheme. However, if the current window size at time t is W and the current round-trip estimate is RTT, then we assume the TCP sender sends according to a Poisson process of rate W/RTT at time t. This results in the same average rate as sending W packets per RTT. While this is a slightly unrealistic assumption (it can result in the window size being violated and so might alter TCP behavior in undesirable ways), this scenario yields important clues about the feasibility of very small buffers.

We are also going to assume that the network is over-provisioned – even if each flow is sending at its maximum window size, the network will not be congested.[3] Under these assumptions, we show that a buffer size of $O(\log W_{\max})$ packets is sufficient to obtain close to peak throughput, where $W_{\max}$ is the maximum congestion window size in packets. Some elements of the proof are interesting in their own right.[4] The exact scenario is explained in Section 3.3 and the proof itself is in Appendix A.

To get some feel for these numbers, let us consider the scenario where 1000 flows share a link of capacity 10 Gb/s. Assume that each flow has an RTT of 100ms, a maximum window size of 64 KB, and a packet size of 1 KB. The peak rate is roughly 5Gb/s. The bandwidth-delay product rule-of-thumb suggests a buffer size of 125MB, or around 125,000 packets. The $2T \times C/\sqrt{N}$ rule suggests a buffer size of around 3950 packets. Our analysis suggests a buffer size of twelve packets plus some small additive constant, which brings the buffer size down to the realm where optical buffers can be

[3] This assumption is less restrictive than it might appear. Current TCP implementations usually cap window sizes at 32 KB or 64 KB [35], and it is widely believed that there is no congestion in the core of the Internet. All optical networks, in particular, are likely to be significantly over-provisioned. Later we will relax this assumption, too.

[4] For example, we do not need to assume the TCP equation [39] or aggregate Poisson arrivals [42]—hence we do not rely on the simplifying assumptions about TCP dynamics and about a large number of flows that are required for these two results.

built in the near future.

We then systematically remove the two assumptions we made above, using a combination of simulations and analysis. We first tackle the assumption that TCP sends packets in a locally Poisson fashion. Intuitively, sending packets at fixed (rather than random) intervals should give us the same benefit (or better) as sending packets at a Poisson rate. Accordingly, we study the more reasonable case where the TCP sending agent "paces" its packets deterministically over an entire RTT. Paced TCP has been studied before [5, 49], and does not suffer from the problem of overshooting the window size. We perform an extensive simulation study of paced TCP with small buffers. When the network is over-provisioned, the performance of paced TCP closely mirrors our analytical bound of $O(\log W_{\max})$ for Poisson sources. This holds for a wide range of capacities and number of flows, and not just in the regime where one might expect the aggregate arrival process at the router to resemble a Poisson process [12]. These results are presented in Section 3.4. In Appendix B, we provide additional intuition for this result: if many paced flows are superimposed after a random jitter, then the packet drop probability is as small as with Poisson traffic.

The next assumption we attempt to remove is that of the network being over-provisioned. We consider a single bottleneck link, and assume that if each flow were to send at its maximum window size, then the link would be severely congested. In our simulations (presented in Section 3.5), Paced TCP results in high throughput (around 70-80%) with the relatively small buffers (10-20) predicted by the simple Poisson-transmissions analysis. While we have not been able to extend our formal analysis to the under-provisioned network case, some analytical intuition can also be obtained: if we assume that the TCP equation [39] holds and that the router queue follows the M/M/1/B dynamics, then buffers of size $O(\log W_{\max})$ packets suffice to utilize a constant fraction of the link capacity.

Our results are qualitatively different from the bandwidth-delay rule-of-thumb or from the results of Appenzeller *et al.* On the positive side, we have *completely removed* the dependence of the buffer size on the bandwidth-delay product. To understand the importance of this, consider the scaling where the RTT is held fixed at τ, but the maximum window size $W_{\max}$, the number of flows N, and the capacity C all go to ∞

such that $C = NW_{\max}/\tau$. This is a very reasonable scaling since τ is limited by the speed of light, whereas C, N, and $W_{\max}$ are all expected to keep growing as Internet traffic scales. Under this scaling, the sizing rule of Appenzeller *et al.* suggests that the buffer size should grow as $\sqrt{N}W_{\max}$, whereas our results suggest that the buffer size needs to grow only at the much more benign rate of $\log W_{\max}$. On the negative side, unlike the result of Appenzeller *et al.* , our result is a trade-off result—to obtain this large decrease in buffers, we have to sacrifice some fixed fraction (say around 20%) of link capacity. This is a good trade-off for an all-optical network where bandwidth is plentiful and buffers are scarce. But for electronic routers, this might possibly be a sub-optimal trade-off.

We give evidence that our result is tight in the following sense.

1. Under the scaling described above, buffers must at least grow in proportion to $\log W_{\max}$ to obtain a constant factor link utilization. In Section 3.6, we present simulation evidence that constant sized buffers are not adequate as the maximum window size grows to infinity. We also perform a simple calculation that shows the necessity of the log-scaling assuming the TCP equation and M/M/1/B queueing.

2. When we run simulations without using Paced TCP, we can not obtain reasonable link utilizations with log-sized buffers, even in the over-provisioned case (Section 3.4).

While TCP pacing is arguably a small price to pay for drastic reduction in buffer sizes, it does require a change to end-hosts. Fortunately, we suspect this may not be necessary, as two effects naturally provide some pacing in current networks. First, the access links are typically much slower than the core links, and so traffic entering the core from access links is automatically paced; we call this phenomenon "link-pacing". We present simulations showing that with link-pacing we only need very small buffers, because packets are spaced enough by the network. Second, the ACK-clocking scheme of TCP paces packets [5]. The full impact of these two phenomena deserves further study.

Of course, significant additional work—including experimental verification, more detailed analysis, and larger simulation studies—is required before we undertake a drastic reduction in buffer sizes in the current Internet.

3.3 Poisson Injections, Over-provisioned Network

Let us imagine for a moment that each flow is an independent Poisson process. This is clearly an unrealistic (and incorrect) assumption, but it serves to illustrate the intuition. Assume too that each router behaves like an M/M/1 queue. The drop-rate would be ρ^B, where ρ is the link utilization and B is the buffer size. At 75% load and with 20 packet buffers, the drop rate would be 0.3%, independent of the RTT, number of flows, and link-rate. This should be compared with a typical 10Gb/s router line-card today that maintains 1,000,000 packet buffers, and its buffer size is dictated by the RTT, number of flows and link-rate. In essence, the cost of not having Poisson arrivals is about five orders of magnitude more buffering! An interesting question is: How "Poisson-like" do the flows need to be in order to reap most of the benefit of very small buffers?

To answer our question, assume N long-lived TCP flows share a bottleneck 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 $1/W_i(t)$, 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 $W_i(t)/\text{RTT}$. Note that this is different from regular TCP, which typically sends packets as a burst at the start of the window.

We will assume that the window size is bounded by $W_{\max}$. Implementations today typically have a bound imposed by the operating system (Linux defaults to $W_{\max} = 64$ KB), or the window size is limited by the speed of the access link. We will make the simplifying assumption that the two-way propagation delay of each flow is RTT. Having a different propagation delay for each flow leads to the same results, but the analysis is more complicated. The capacity C of the shared link is assumed

to be at least $(1/\rho) \cdot NW_{\max}/\text{RTT}$ where ρ is some constant less than 1. Hence, the network is over-provisioned by a factor of $1/\rho$, *i.e.* the peak throughput is ρC. The effective utilization, θ, is defined as the achieved throughput divided by ρC.

In this scenario, the following theorem holds:

Theorem 1. To achieve an effective utilization of θ, a buffer of size

$$B <= \log_{1/\rho}\left(\frac{W_{\max}^2}{2(1-\theta)}\right) \tag{3.1}$$

suffices.

The proof comes in Appendix A.

As an example of the consequences of this simple model, if $W_{\max} = 64$ packets, $\rho = 0.5$, and we want an effective utilization of 90%, we need a buffer size of 15 packets *regardless of the link capacity*. In other words, the AIMD dynamics of TCP do not necessarily force us to use larger buffers, if the arrivals are well-behaved and non-bursty. So what happens if we make the model more realistic? In the next section we consider what happens if instead of injecting packets according to a Poisson process, each source uses Paced TCP in which packets are spread uniformly throughout the window.

3.4 Paced TCP, Over-provisioned Network

It should come as no surprise that we can use very small buffers when arrivals are Poisson: arrivals to the router are benign and non-bursty. Queues tend to build up—and hence we need larger buffers—when large bursts arrive, such as when a TCP source sends all of its outstanding packets at the start of the congestion window. But we can prevent this from happening if we make the source spread the packets over the whole window. Intuitively, this modification should prevent bursts and hence remove the need for large buffers. We now show that this is indeed the case. Throughout this section, we assume that the bottleneck link is over-provisioned in the same sense as in the previous section. In the next section we remove this assumption.

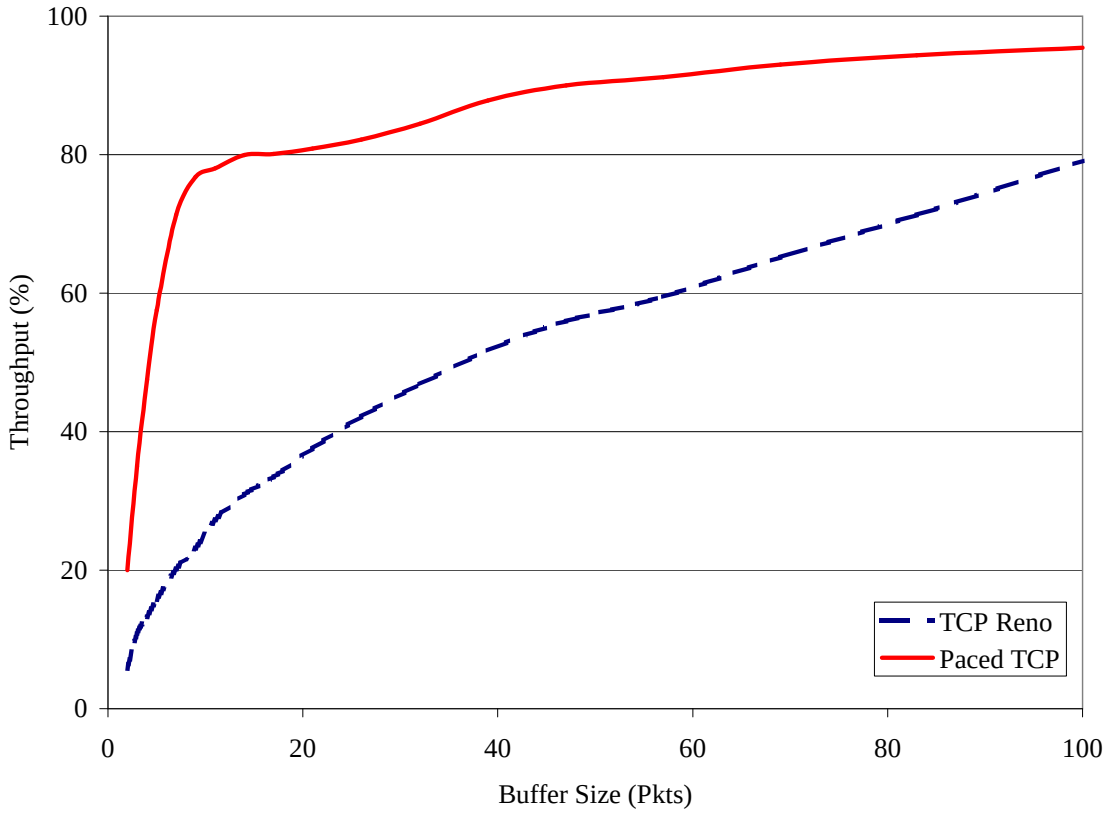

Figure 3.1: Bottleneck link utilization for different buffer sizes (TCP Reno vs. Paced TCP)

First, suppose N flows, each with maximum window size $W_{\max}$, share a bottleneck link. Then the following is true.

Theorem 2. Assume that: (1) The buffer size at the bottleneck link is at least $c_B \log W_{\max}$ packets, where $c_B > 0$ is a sufficiently large constant; (2) The rate of each flow is at least a $c_S \log W_{\max}$ factor slower than that of the bottleneck link, where c_S is a sufficiently large constant; and (3) Random jitter prevents a priori synchronization of the flows. Then the packet loss probability during a single RTT is $O(1/W_{\max}^2)$.

Please see Appendix B for proof of this theorem.

The packet loss probability in Theorem 2 is comparable to that for Poisson traffic with the same buffer size. The buffer size requirement for Theorem 2 (Assumption (1))

is comparable to that in Theorem 1—a few dozen packets for present-day window sizes, independent of the link capacity, number of flows, and RTT. This requirement appears to be necessary to achieve constant throughput, even with Paced TCP (see Section 3.6). The second assumption states that flows should be "sufficiently non-bursty". Note that some assumption of this form is necessary, since if flows can send at the same rate as the bottleneck link, then there is no pacing of traffic whatsoever and our simulations indicate that constant throughput is not achievable with log-sized buffers. Precisely, Theorem 2 requires that all flows send at a rate that is roughly a $\log W_{\max}$ factor slower than that of the bottleneck link, and obtains a Poisson-like throughput-buffer size trade-off under this requirement. This slowdown factor is only a few dozen for present-day window sizes, while access links are often orders of magnitude slower than backbone links. This huge difference in access link and backbone link speeds also seems likely to persist in the near future (especially with an all-optical backbone). The final assumption of Theorem 2 simply ensures that the flows are not initialized in a synchronized state.

To explore the validity of Theorem 2, we performed simulations using *ns-2* [4]. We implemented Paced TCP by adding a new timer to TCP Reno which regulates the injection time of packets. This is a high granularity timer, since using low granularity timers might generate bursts, which is undesirable.[5] Each flow is generated at a source node, goes through an individual access link, and then through a shared link. The capacity and the propagation delay of the shared and access links vary for different simulations. We add a random jitter to the propagation delay of the access links to model different RTTs in the network.

In Figure 3.1 we compare the buffer size needed by TCP Reno with Paced TCP. We plot the throughput of the system as a function of the buffer size used in the router. The capacity of the bottleneck link is 100Mb/s, there are 40 flows in the system, and the average RTT is 100ms. In this experiment, the maximum congestion window size is set to 32 packets, and the size of packets is 1,000 bytes, thus the maximum offered load is 100Mb/s with 40 flows. The simulation is run for 1,000

[5]The implementation of Paced TCP that comes with *ns-2* is not accurate and needs some modifications.

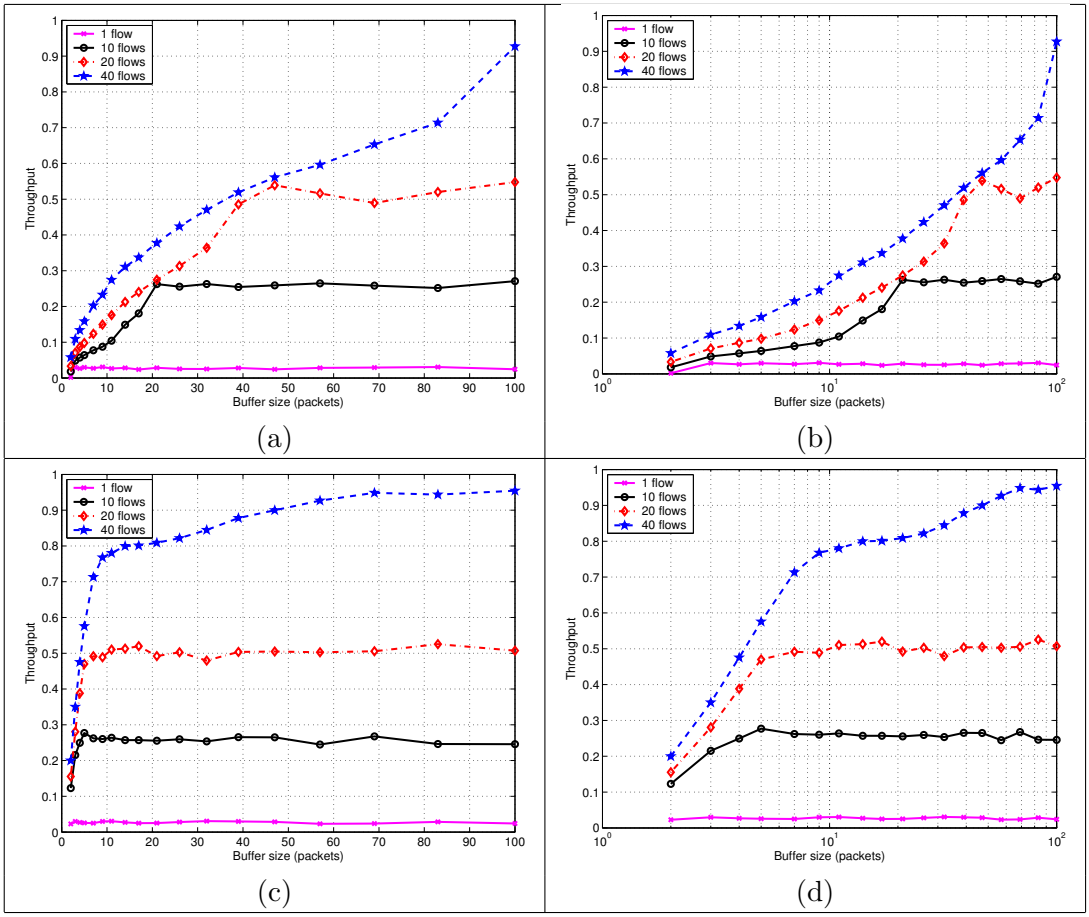

Figure 3.2: Bottleneck link utilization for different buffer sizes and number of flows. (a) TCP reno (b) TCP Reno with logarithmic x-axis (c) paced TCP (d) Paced TCP with logarithmic x-axis. The maximum possible offered load is 0.026 with one flow, 0.26 with 10 flows, 0.52 with 20 flows, and 1 with 40 flows.

seconds, and we start recording the data after 200 seconds.

As we can see, with 40 unmodified TCP (Reno) flows, we need to buffer about 100 packets to achieve a throughput above 80%. However, in the same setting, Paced TCP achieves 80% throughput with just 10 packet buffers.

Reducing the number of flows from 40 will reduce the overall system throughput, as individual flows have a limited congestion window size. For instance, with 20 flows we expect to have a throughput of around 50% if we have enough buffers. Figure 3.2 shows the bottleneck link utilization for various buffer sizes when we have 1, 10, 20,

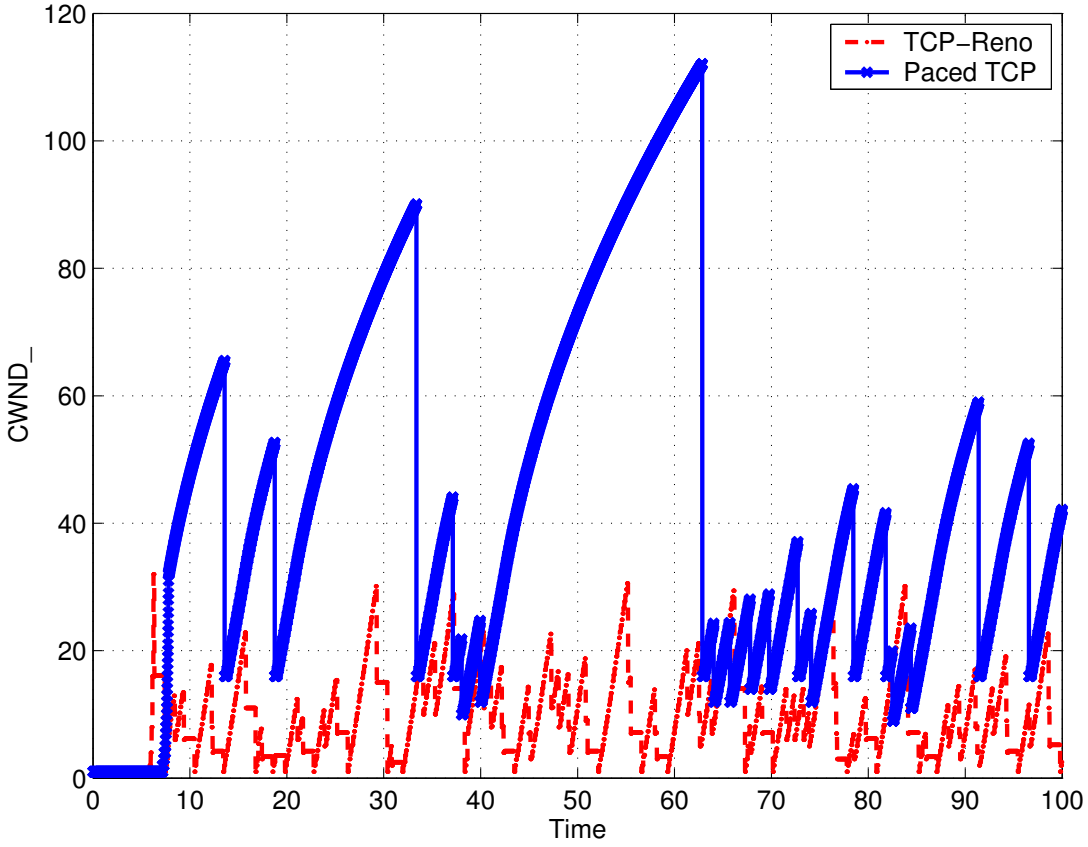

Figure 3.3: Congestion window size (TCP Reno vs. Paced TCP)

and 40 TCP Reno or Paced TCP flows in the system. We can see that TCP Reno still needs relatively large buffers even when we reduce the number of flows (and thus the load) in the system. For instance, for 20 flows, we will need 50 packets of buffering to reach the maximum utilization (which is roughly 50%). However, with Paced TCP we need less than 10 packets of buffering to gain the same utilization. In other words, with TCP Reno even though decreasing the number of flows, and the system load reduces the buffer size requirements, it still is in the order of the bandwidth-delay product. With Paced TCP on the other hand the tiny buffers rule can be applied regardless of the number of flows.

To understand the impact of pacing, we take a closer look at the congestion window (CWND_) of TCP Reno and Paced TCP in Figure 3.3. In this experiment, 500 flows

share a bottleneck link with a capacity of 1.5Gb/s; the buffer size is 10 packets; and each flow is limited to a maximum window size of 32 packets[6] and the average RTT is 100ms. We observe that TCP Reno rarely reaches the maximum window size of 32 packets, whereas Paced TCP has a larger congestion window at almost all times. The bursty nature of TCP Reno makes the flows experience packet drops more frequently when the buffer size is small, while Paced TCP flows experience fewer drops, and so CWND_ grows to larger values. Consequently, Paced TCP sends with its maximum possible capacity of 32 packets per RTT most of the time.

In Figure 3.2 we increased the system load as we increased the number of flows since the capacity of the shared link is kept fixed. It is interesting to see what happens if we keep the system load constant (at 80% in this case) while increasing the number of flows. This is illustrated in Figure 3.4, for flows with a maximum congestion window of 32 packets. As we increase the number of flows from one to more than a thousand, we also increase the bottleneck link capacity from 3.2Mb/s to 3.4Gb/s to keep the peak load at about 80%. The buffer size is still set to 10 packets. The graph shows that regardless of the number of flows, throughput is improved by Paced TCP. The throughput of Paced TCP is around 70% (i.e., the effective utilization is more than 85%) while the throughput of the TCP Reno is around 20% (with an effective utilization of around 25%) regardless of the number of flows in the system.

We have shown that Paced TCP can gain a very high throughput even with very small buffers. As we have already noted, if the capacity of the access links is much smaller than the core link, then packets entering the core will automatically have spacing between them even without modifying TCP. To verify this we repeat the previous experiment except instead of using Paced TCP, we limit the capacity of the access links to 5Mb/s. As shown in Figure 3.4, the spacing resulting from limited access link capacities has the same effect as using Paced TCP since the two curves follow each other very closely. In practice, the capacity of the access links is usually several orders of magnitude smaller than the capacity of the core links, which means even without any modification to TCP, we can gain high throughput with very small

[6]Note that CWND_ can go beyond 32 packets in practice, and the source can have up to min(32,CWND_) unacknowledged packets at any point of time.

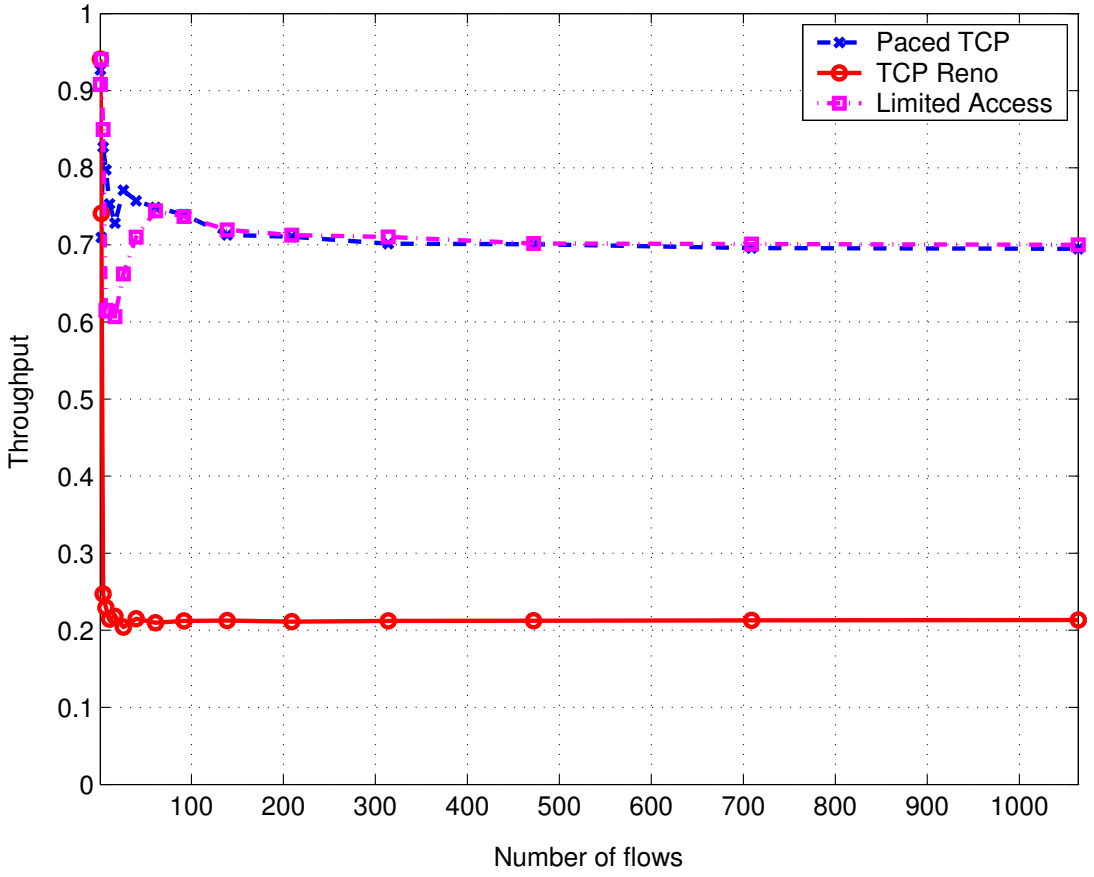

Figure 3.4: Throughput of Paced TCP vs. TCP Reno; the capacity of shared link is increased as we increase the number of flows.

buffers.

It is important to note that this significant discrepancy between the throughput of paced and regular TCP is observed only with small buffers. If we use the bandwidth-delay rule for sizing buffers, this discrepancy vanishes.

3.5 Paced TCP, Under-provisioned Network

So far we have assumed that the network is over-provisioned and we do not have congestion on the link under study. Even though this is true for most links in the

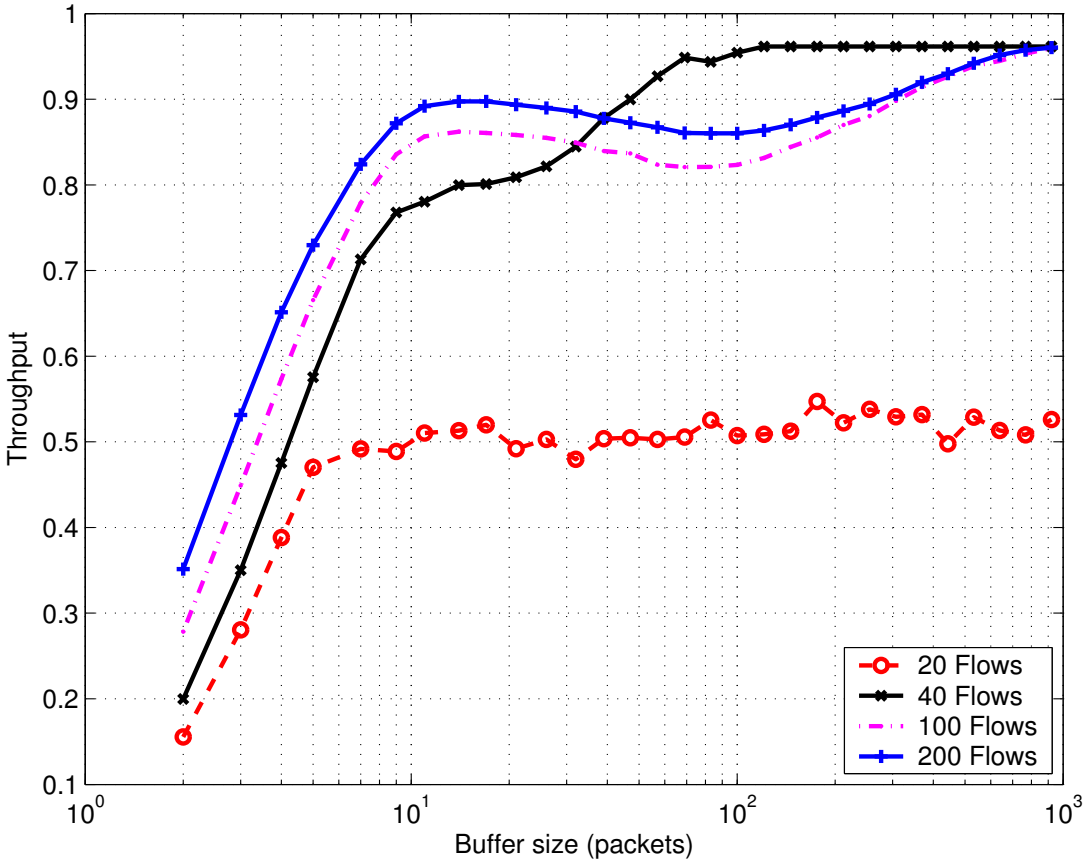

Figure 3.5: Bottleneck link utilization vs. the buffer size. With only 40 flows the core link becomes saturated, but even if we increase the number up to 200 flows, the throughput does not go below 80%.

core of the Internet, it is also interesting to relax this assumption. We next study, via simulations, how congestion affects link utilization.

We repeat an experiment similar to that depicted in Figure 3.1 (this time with Paced TCP only). We increase the number of flows to up to 200 so that the bottleneck link becomes congested. The average RTT is 100ms, and the maximum window size is 32 packets. Each packet is 1000 bytes, which means each flow can contribute a load of $32 \times 1000 \times 8/0.1 \simeq 2.5$Mb/s. The capacity of the core link is 100Mb/s, *i.e.* the core link will become congested if we have more than 40 flows. For 100 and 200 flows the bottleneck link is highly overloaded.

Figure 3.5 shows the throughput of the bottleneck link as a function of the buffer size for various number of flows. We can see that as we increase the number of flows from 20 to 40 (at which point the link starts to be saturated) the throughput goes from around 50% to about 80-90%. As we increase the number of flows to 100 and 200, for small buffers (1-12 packets) the system throughput slightly improves. For middle size buffers (15-100 packets), however, we see a degradation in throughput, although the throughput never goes below 80%. This minor degradation in performance is due to the partial synchronization between congestion windows of flows, which occurs for middle sized buffers. This phenomenon has been explained by Raina and Wischik [42] in more detail. We note that the throughput of the system remains above 80% even when the buffer size is 10 packets.

Figure 3.6 depicts the throughput of the bottleneck link as a function of the number of flows when the access link capacity is fixed to 5Mb/s and the buffer size is 10 packets. We change the offered load to the system by adjusting the capacity of the bottleneck link relative to the total capacity of the access links. To keep the maximum offered load fixed at a given value, we increase the capacity of the bottleneck link as the number of flows is increased. As we can see in this graph, as we increase the offered load beyond 100%, the overall throughput increases. In other words, even when the network is under-provisioned we expect reasonable throughput with very small buffers.

In a related experiment we set the core link bandwidth to 1Gb/s, and vary the capacity of the access links. The maximum window size is very large (set to 10,000), the buffer size is set to 10 packets, and the average RTT is set to 100ms. Figure 3.7 shows the throughput of the system as a function of the capacity of the access links. We can see that at the beginning the throughput increases (almost) linearly with access link capacity, until the shared link becomes congested. For example, with 100 flows, this happens when the access link capacity is below 8-9Mb/s. Note that the normalized throughput is close to 100% when the offered load is less than the capacity of the shared link since the core link is not the bottleneck. As we increase the access link capacity, the throughput gradually decreases. This is because we lose the natural spacing between packets as the capacity of access links is increased.

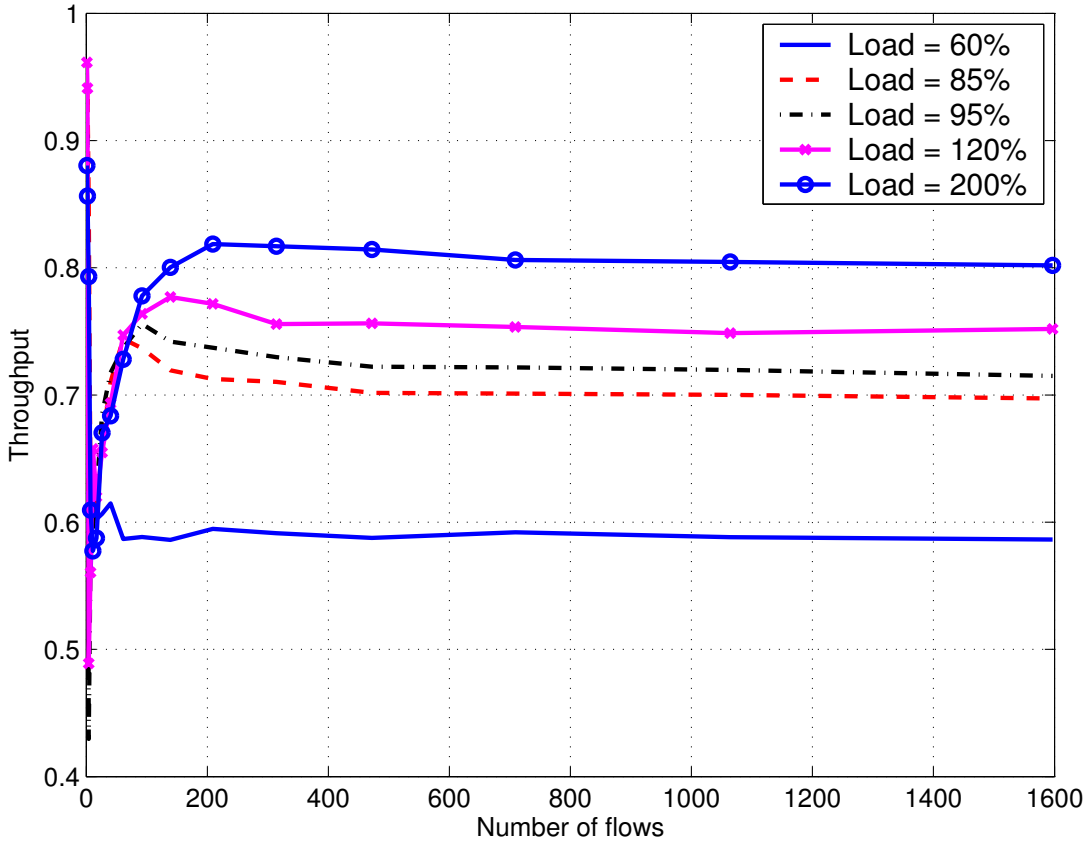

Figure 3.6: Throughput as a function of number of flows for various values of the offered load to the system.

3.6 The Necessity of Logarithmic Scaling of Buffer Sizes

We have not been able to extend our proof of theorem 1 to the case when the network is under-provisioned. However, the TCP equation [39] gives interesting insights if we assume that the router queue can be modeled as an M/M/1/B system [44]. Consider the scaling (described in the introduction) where the RTT is held fixed at τ, but the maximum window size $W_{\max}$, the number of flows N, and the capacity C all go to ∞. To capture the fact that the network is under-provisioned, we will assume that $C = \frac{NW_{\max}}{2\tau}$ *i.e.* the link can only support half the peak rate of each flow. Similarly,

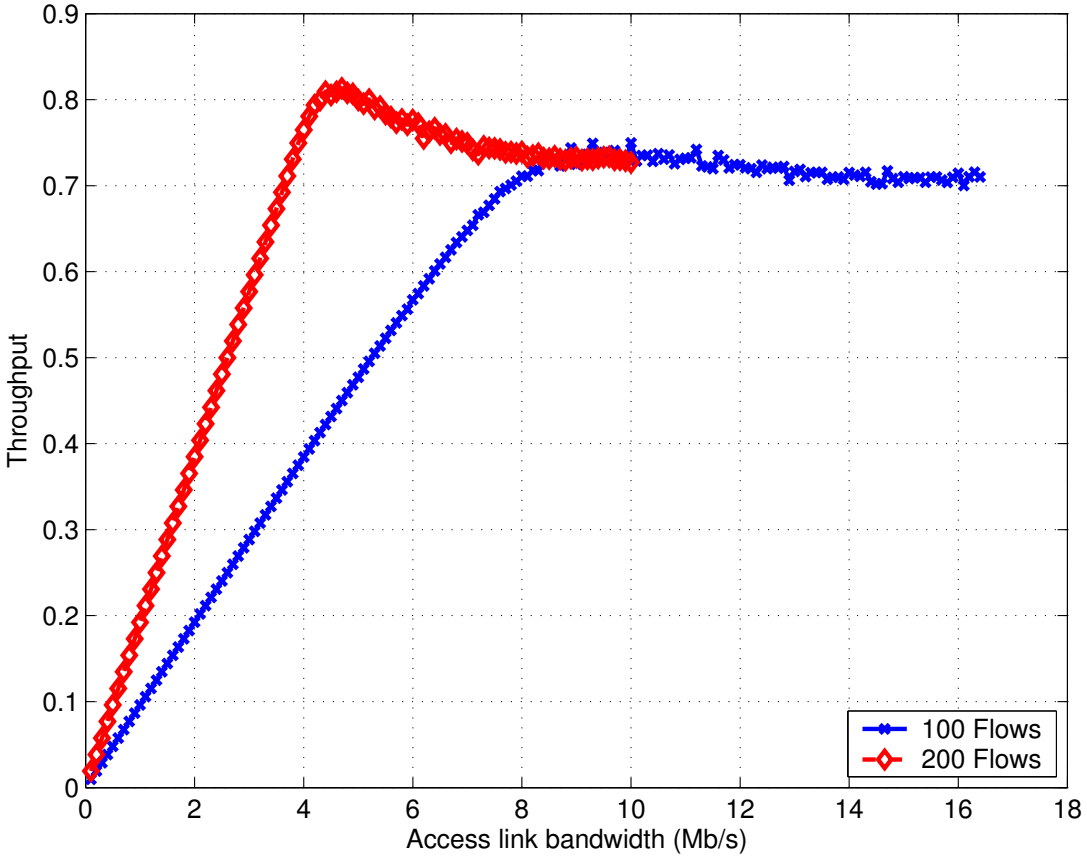

Figure 3.7: Throughput as a function of access link capacity.

$C = \frac{2NW_{\max}}{\tau}$ represents the under-provisioned case.

Let p be the drop probability, and ρ the link utilization. Clearly, $\rho = RN/C$, where R is the average throughput of each flow. Then, the TCP equation states:

$$R = \frac{1}{\tau}\sqrt{\frac{3}{2p}} + o(1/\sqrt{p}) \simeq \frac{1}{\tau}\sqrt{\frac{3}{2p}}. \tag{3.2}$$

The M/M/1/B assumption yields [32]:

$$p = \rho^B \frac{1-\rho}{1-\rho^B}\frac{\rho}{1+\rho} \simeq \rho^{B+1}. \tag{3.3}$$

Equations 3.2, 3.3 immediately yield the following:

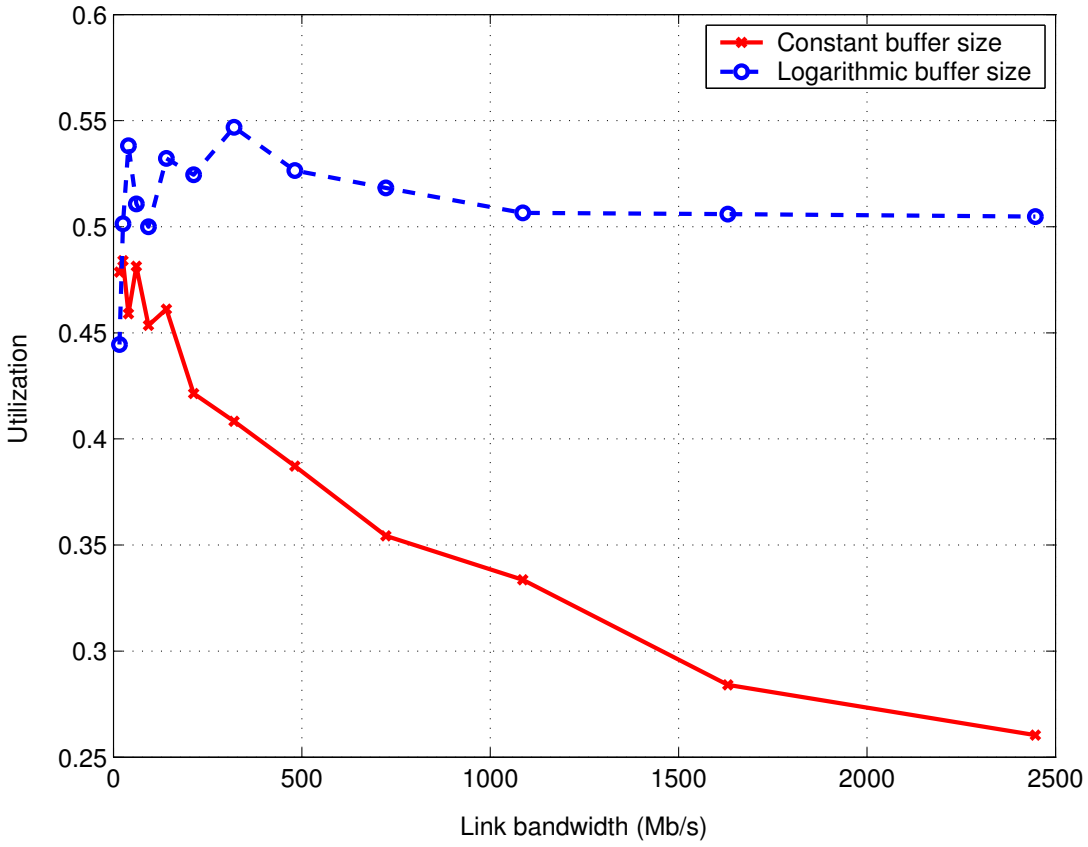

Figure 3.8: Constant vs. logarithmic buffers.

1. Assume $C = \frac{NW_{\max}}{2\tau}$. For every constant $\alpha < 1$, there exists another constant β such that setting $B = \beta \log W_{\max}$ yields $\rho > \alpha$. In other words, logarithmic buffer-sizes suffice for obtaining constant link utilization even when the network is under-provisioned.

2. Assume $C = \frac{2NW_{\max}}{\tau}$. If $B = o(\log W_{\max})$ then $\rho = o(1)$. In other words, if the buffer size grows slower than $\log W_{\max}$ then the link utilization drops to 0 even in the over-provisioned case.

Obtaining formal proofs of the above statements remains an interesting open problem. Simulation evidence supports these claims, as can be seen in Figure 3.8 which

describes the throughput for a constant vs. a logarithmic sized buffer. For this simulation we are using Paced TCP, N is held fixed at 10, $W_{\max}$ varies from 10 to 1529, and C varies as follows. Initially, C is chosen so that the peak load is constant and a little over 50%, and this choice determines the initial value for the ratio $\frac{C\overline{\text{RTT}}}{NW_{max}}$; then, since we fix N and $\overline{\text{RTT}}$, C varies proportionally to W_{max} to keep the above ratio constant as in our theoretical modeling. The buffer size is set to 5 packets when $W_{\max} = 10$. Thereafter, it increases in proportion with $\log W_{\max}$ for the log-sized-buffer case, and remains fixed at 5 for the constant buffer case. Here, initially the throughput is around 50% for both buffer sizing schemes. However, the throughput for the constant sized buffer drops significantly as C and $W_{\max}$ increase, while for the logarithmic sized buffer the throughput remains approximately the same, just as predicted by our theoretical model.

3.7 Summary

Our theoretical and simulation results in this chapter suggest packet buffers can be made much smaller; perhaps as small as 10-20 packets, if we are prepared to sacrifice some of the link capacity. It appears from simulation - though we have not been able to prove it - that the buffer size dictates directly how much link capacity is lost, however congested the network is. For example, a 40Gb/s link with 15 packet buffers could be considered to operate like a 30Gb/s link. Of course, this loss in link capacity could be eliminated by making the router run faster than the link-rate. In a future network with abundant link capacity, this could be a very good trade-off: Use tiny buffers so that we can process packets optically. In the past, it was reasonable to assume that packet buffers were cheap, while long-haul links were expensive and needed to be fully utilized. Today, fast, large packet buffers are relatively painful to design and deploy; whereas link capacity is plentiful and it is common for links to operate well below capacity. This is even more so in an all-optical network where packet buffers are extremely costly and capacity is abundant.

Our results lead to some other interesting observations. It seems that TCP dynamics have very little effect on buffer-sizing, and hence these results should apply to

a very broad class of traffic. This is surprising, and counters the prevailing wisdom (and our own prior assumption) that buffers should be made large because of TCP's saw-tooth behavior.

Chapter 4

Buffer Sizing Experiments

So far we have described several rules for buffer sizing in Internet backbone routers. The bandwidth-delay rule-of-thumb for buffer sizing is based on the saw-tooth dynamics of a single TCP flow. Based on this rule, Internet routers require extremely large buffers, and the buffer size needs to grow linearly with the capacity of the links. Clearly, this rule has significant implications in terms of complexity for Internet routers.

The small buffers rule on the other hand, is based on the observation that as the number of flows increases, the aggregate window size process (the sum of all the congestion window size processes for each flow) decreases, following a central limit theorem. The result relies on several assumptions: (1) that flows are sufficiently independent of each other to be desynchronized, (2) that the buffer size is dominated by the long-lived flows, and perhaps most importantly (3) that there are no other significant, unmodeled reasons for buffering more packets. If the result is correct, then a backbone link carrying 10,000 long-lived flows could have its buffer size reduced by a factor of 100 without loss in throughput. If, though, the results are wrong, then the consequences of reducing the buffers in a router, or in an operational commercial network, could be quite severe. The problem is, how to decide if the result is correct, without trying it in an operational network? But who would reduce buffers in an operational network, and risk losing customers' traffic, before knowing if the result is correct?

Based on the tiny buffers rule, we can reduce buffers much further to $O(1) \approx 20 - 50$ packets in today's backbone routers [18, 41, 19]. This conclusion is reached by considering the tradeoff between reducing buffers and losing some throughput – assumed to be 10-20%. In other words, when congested, links behave as if they run at 80-90% of their nominal rates. This could be an interesting assumption in networks with abundant link capacity, or in future optical networks where link capacity might be cheaper than buffers. The results depend on the network traffic being non-bursty, which can happen in two ways: (1) If the core links run much faster than the access links (which they do today), then the packets from a source are spread out and bursts are broken, or (2) TCP sources are changed so as to pace the delivery of packets. If the results are correct, and relevant, then a backbone link could reduce its buffers by five orders of magnitude.

Again, it is difficult to validate these results in an operational network, and we are not aware of any other laboratory or network experiments to test the $O(1)$ results. So it is the goal of our work to experiment with *small buffers* and *tiny buffers* in laboratory and operational networks.

Throughout this chapter, we describe a number of laboratory and network experiments that were performed (by us and by others) during 2003 to 2006. The laboratory experiments were performed in the WAIL laboratory at University of Wisconsin Madison, at Sprint Advanced Technology Laboratory, Verizon, and Lucent. Experiments were also performed on the following operational networks: Level 3 Communications' operational backbone network, Internet2 and Stanford University dormitory traffic. We should make clear that our results are necessarily limited: While a laboratory network can use commercial backbone routers and accurate TCP sources, it is not the same as a real operational backbone network with millions of real users. On the other hand, experiments on an operational network are inevitably limited by the ability to control and observe the experiments. Commercial routers do not offer accurate ways to set the buffer size, and do not collect real time data on the occupancy of their queues. And real network experiments are not repeatable for different buffer sizes, making apples-with-apples comparisons difficult.

In laboratory experiments, we generate live TCP traffic (ftp, telnet, or http) using

a cluster of PCs or commercial traffic generators. We measure the performance from either real-time or statistical traces collected from the system. On one hand, we have a lot of control on the experiments, and can observe almost anything. On the other hand, the traffic is synthetic and might not represent real users. We note that we cannot simply use traces gathered from operational backbone networks for buffer sizing experiments, because TCP uses a feedback loop to control congestion.

In our experiments on operational backbone networks we can test the results with real user traffic. However, we have no control over the traffic pattern or the system load. For example, Internet2 has very low load (about $20 - 30\%$), which means congestion does not happen naturally. Fortunately, at the time of our experiments, part of the Level 3 Communications network ran at very high link utilization (up to 96%). We report results from both networks.

Where possible, we ran experiments over a wide range of operating conditions for both the *small buffer* and *tiny buffer* models, including system load ranging from 25% up to 100%, different number of users, various traffic patterns and flow sizes, different propagation delays, access link capacities, and congestion window limits.

4.1 Small Buffers Experiments

We start with, perhaps, the most interesting experiments, which were performed on Level 3 Communications' operational commercial backbone network. We follow these experiments with summaries of other experiments in different networks.

4.1.1 Experiment Setup and Characteristics

Although we have limited control of an operational network, these experiments have several interesting properties. The links under study were highly utilized with real live traffic. Their utilization varied between 28.61% and 95.85% during a 24 hour period, and remained above $85 - 90\%$ for about four hours every day (an exceptionally high value [29]- new link capacity was added right after the experiments were done).

The link under study consisted of three physical, load-balanced links (Figure 4.1).

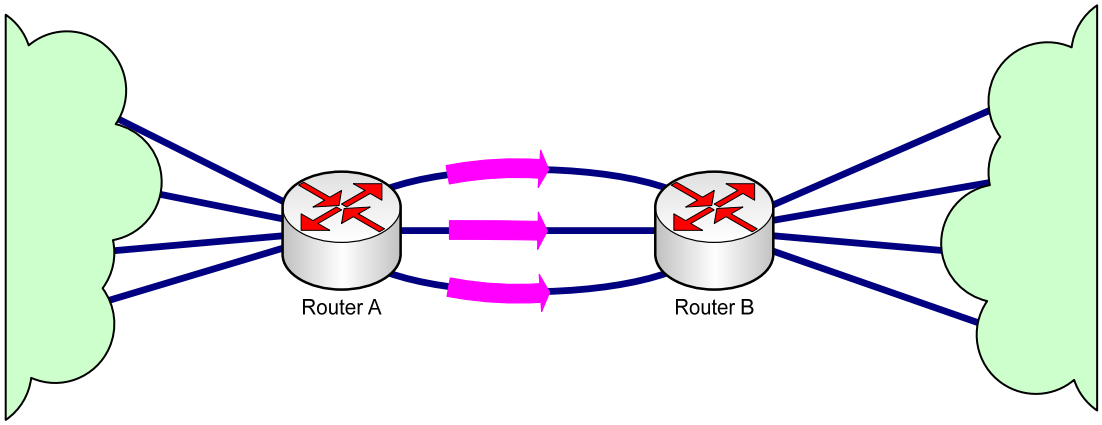

Figure 4.1: Setup used for buffer sizing experiments in Level 3 Communications' backbone network. The incoming traffic to Router A was divided amongst the three links connecting Router A to Router B using a static hash function balancing flows over the three links.

Traffic at the upstream router is divided equally among the three physical links. Each incoming flow is assigned to one of the three links using a static hash function based on the source-destination IP and port numbers of the flow. Ideally, there is equal traffic on each link (particularly as there are thousands of flows). If we give each physical link a different amount of buffering, we can perform an apples-with-apples comparison between different buffer sizes under almost identical conditions.

The three physical links are OC-48 (2.5Gb/s) backbone links, carrying mostly a mixture of traffic from cable modem and DSL users. Assuming an average rate of 250Kb/s per flow, each link carries about 10,000 flows when highly utilized. The default buffer size was 190ms per-link (60MBytes or 125,000 packets assuming an average packet size of 500B). We reduced the buffer sizes to 10ms (about 3MB or 6,000 packets), 5ms (1.5MB or 3,000 packets), 2.5ms (750KB or 1,500 packets) and 1ms (300KB or 600 packets). Based on the small buffer sizing model, we can expect to need a buffer size of about 2-6ms (depending on the actual value of N).

The buffer sizes were set for 5 days, so they capture the impact of daily and weekly changes in traffic. The whole experiment lasted two weeks in March, 2005. We gathered link throughput and packet drop statistics from each of the three links

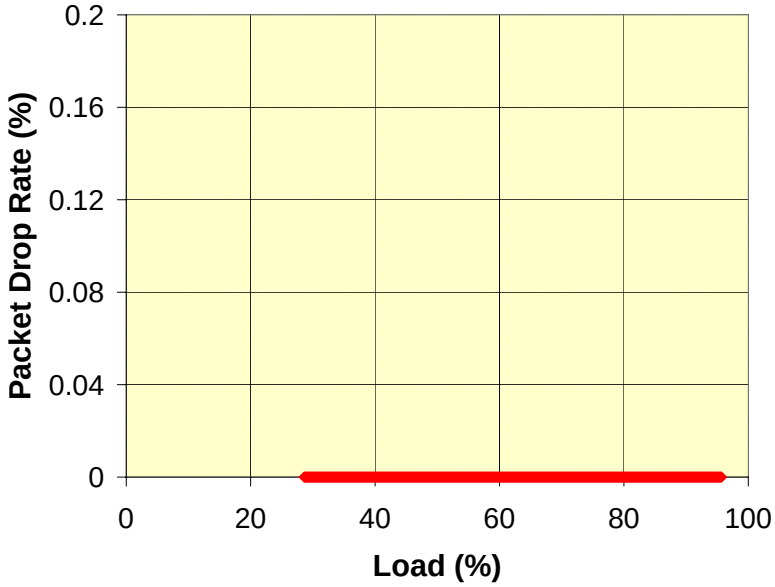

Figure 4.2: Packet drop rate as a function of load for buffer sizes equal to 190ms, 10ms, and 5ms. We did not observe any packet drops in these experiments.

which were collected by the router every 30 seconds. It would have been preferable to capture all packets and recreate the time series of the buffer occupancy in the router. But the network did not have the facility to do this. Still, we are able to infer some interesting results.

We also actively injected test flows, measuring the throughput and drops, and compared the performance of the flows going through different links to find out the impact of buffer size reduction. The amount of test flow traffic was kept small.

It is worth noting that the network does not use traffic shaping at the edges or in the core. The routers do not use RED, and packets are dropped from the tail of the queue.

4.1.2 Experiment Results

During the course of the experiments, we always kept the buffer size on one link at its original size of 190ms, and reduced the buffer size on the other two links.

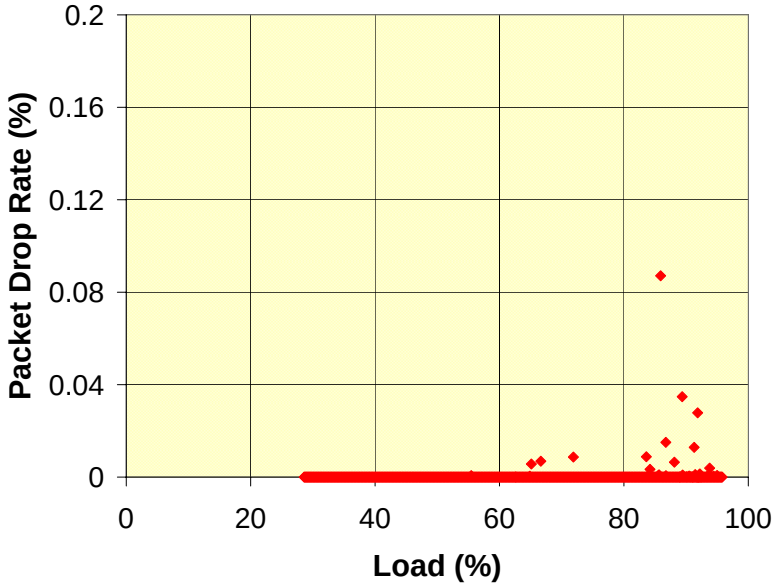

Figure 4.3: Packet drop rate as a function of load for buffer size of 2.5ms. We saw packet drops in only a handful of cases.

Figures 4.2, 4.3, and 4.4 show the packet drop rate as a function of system load for various buffer sizes. As explained before, both the load and drop rates are measured in time intervals of 30 seconds, and each dot in the graph represents one such interval. Figure 4.2 shows that for buffer sizes of 190ms, 10ms, and 5ms we did not see a single packet drop during the course of the experiments, which lasted more than 10 days for the 190ms buffer size, and about 5 days for each of the 10ms and 5ms buffer sizes!

Reducing the buffers by a factor of forty, without dropping packets, when the utilization frequently exceeds 95% is quite surprising. It suggests that the backbone traffic is very smooth (and presumably any self-similarity has no effect at this degree of multiplexing). Others have also reported smoothness in traffic in core networks [24, 23], despite somewhat older results which show self-similarity in core traffic [21, 20, 45]. Whether this is a result of a shift in traffic pattern and network characteristics over time, or simply the consequence of huge amounts of multiplexing, remains to be determined. We have found traffic to be extremely smooth in the laboratory and backbone networks we have studied, when there are a large number of flows.

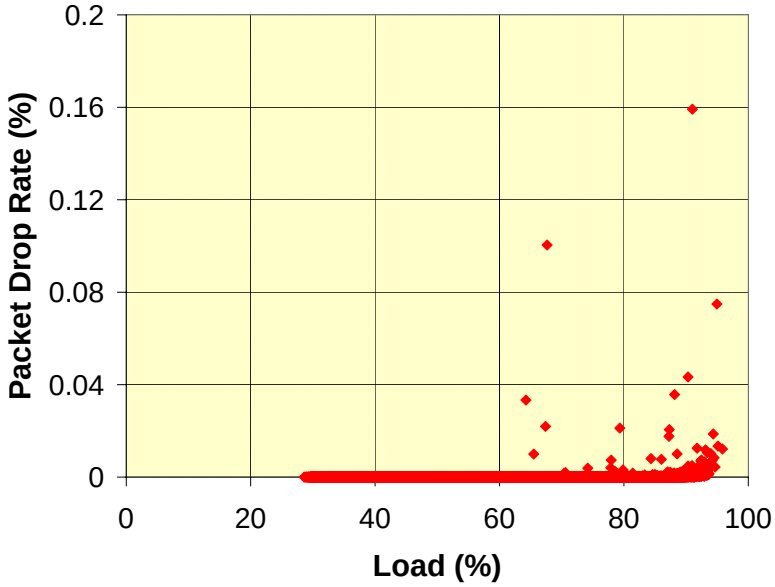

Figure 4.4: Packet drop rate as a function of load for a buffer size of 1ms. We observed packet drops during high utilization time periods.

Figure 4.3 shows the drop rate as a function of load, when the buffer sizes are reduced to 2.5ms. This is in the lower part of the range defined by the small buffer sizing model, and as expected we start to observe some packet drops. However, packet drops are still rare; less than 0.02% for the majority of samples. In two samples over five days, we see a packet drop rate between 0.02% and 0.04% and in one sample the drop rate is close to 0.09%.

Figure 4.4 shows what happens when we reduce the buffer size to 1ms. Here, there are a lot more packet drops, which we expect because the buffer size is now about half the value suggested by the small buffer sizing rule. It is interesting to note that almost all the drops occur when the load is above 90% - even though the load value is averaged over a period of 30 seconds. The instantaneous load is presumably higher, and the link must be the bottleneck for some of the flows. We conclude that having some packet drops does not lead to a reduction in available throughput; it appears that TCP's congestion control algorithms are functioning well. In none of these experiments, we observed a significant change in performance of the test flows

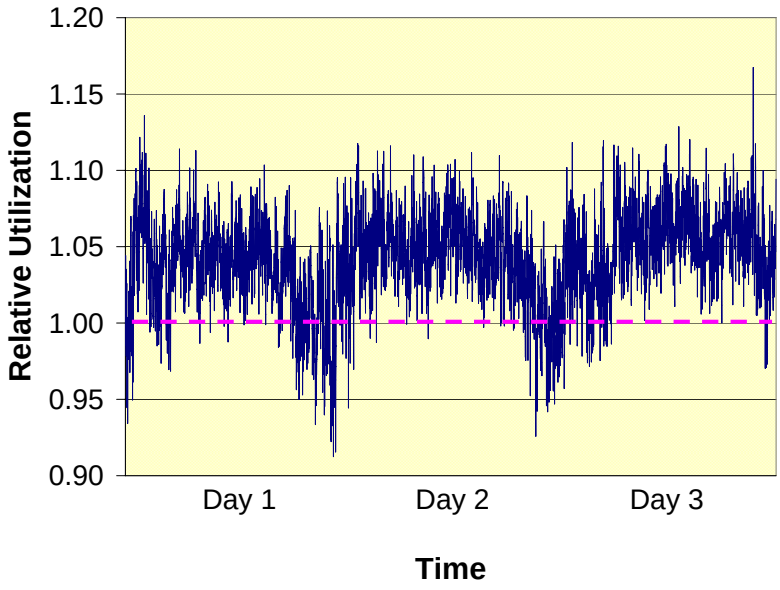

Figure 4.5: Relative utilization of two links with 1ms and 190ms of buffering over time.

injected to the network.

Figure 4.5 compares the link utilization for the links with 190ms and 1ms of buffering, over three days. Ideally, the utilization on both links would be equal at all times; which they are and their difference is almost always less than 10%. However, the differences are not symmetric. The link with 1ms buffering has a slightly higher utilization for the majority of the time.

We take a closer look at the cause of this asymmetry. Figure 4.5 depicts the link utilization as a function of time. By comparing this graph with Figure 4.6 we observe that during the periods when the overall load of the system is high, the link with 1ms has a slightly higher utilization than the link with 190ms. Figure 4.7 suggests the same, in a different yet more precise way. Each dot in this figure, represents the utilization of the two links in a period of 30 seconds. The majority of dots fall below the 45 degree line in the graph.

The higher utilization with smaller buffers can be associated with two possible reasons: (1) It might be due to higher loads on the link with 1ms of buffering. Since

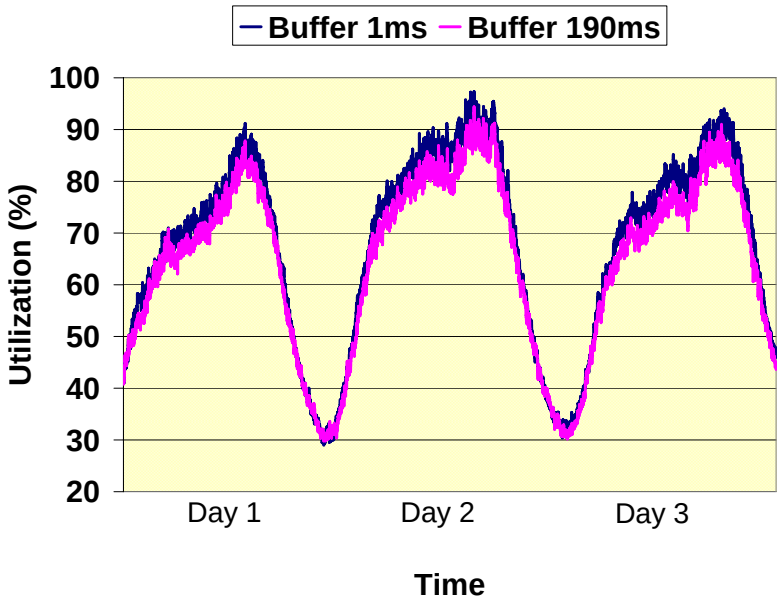

Figure 4.6: Utilization of links with 1ms and 190ms of buffering.

we have more drops on this link, sources need to send duplicates of the dropped packets, and that might be why we see a higher load. Or, (2) the load balancing scheme might be skewed and might divide the traffic somewhat unevenly among the three links, thus directing more traffic to one of the links.

We see the same phenomena (higher utilization in one link than the others) when the buffer sizes are set to 190ms, and 5ms on two links. Since we did not have any packet drops in these cases, there must be no difference in buffer occupancies, and therefore (1) cannot be the reason here, *i.e.*, any difference in utilization is most probably not a result of the reaction of TCP sources to packet drops.[1] In other words, we associate these slight differences between link utilizations with imperfections in the load balancing scheme deployed in the system, rather than the changes in buffer sizes. We conclude that reducing buffer sizes in this network does not have a significant impact on the performance.

[1]The load balancing scheme used in this system was changed after these experiments.

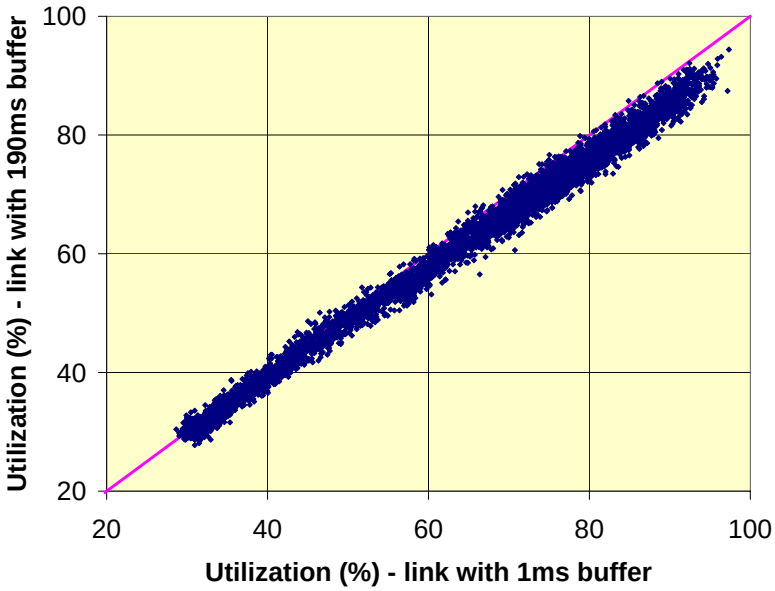

Figure 4.7: Utilization of 1ms buffer link vs. the utilization of the 190ms buffer link.

4.1.3 Other Small Buffer Experiments

Other than the experiments on Level 3 Communications' backbone network, other experiments on small buffer sizing model have been conducted by us and others. These experiments include University of Wisconsin Madison's Advanced Internet Laboratory (WAIL), and Stanford University's dormitory network.

The WAIL experiment, reported by Appenzeller et al. [7], a cluster of PCs is used to generate up to 400 live TCP flows, and the traffic is directed to a Cisco GSR router. The buffer sizes on the router are reduced by a factor of 10-20, which does not result in any degradation in the system throughput. In Stanford University experiment, a Cisco VXR 7200 router was used to connect the dormitories to the Internet via a 100Mb/s link. Traffic was from a mix of different applications including web, ftp, games, peer-to-peer, Streaming and others, and we had 400-1900 flows at any given time. The buffer sizes on the router were reduced by a factor of 20, with no impact on network throughput. We refer the interested reader to Appenzeller's Ph.D. thesis for more details on these experiments [6].

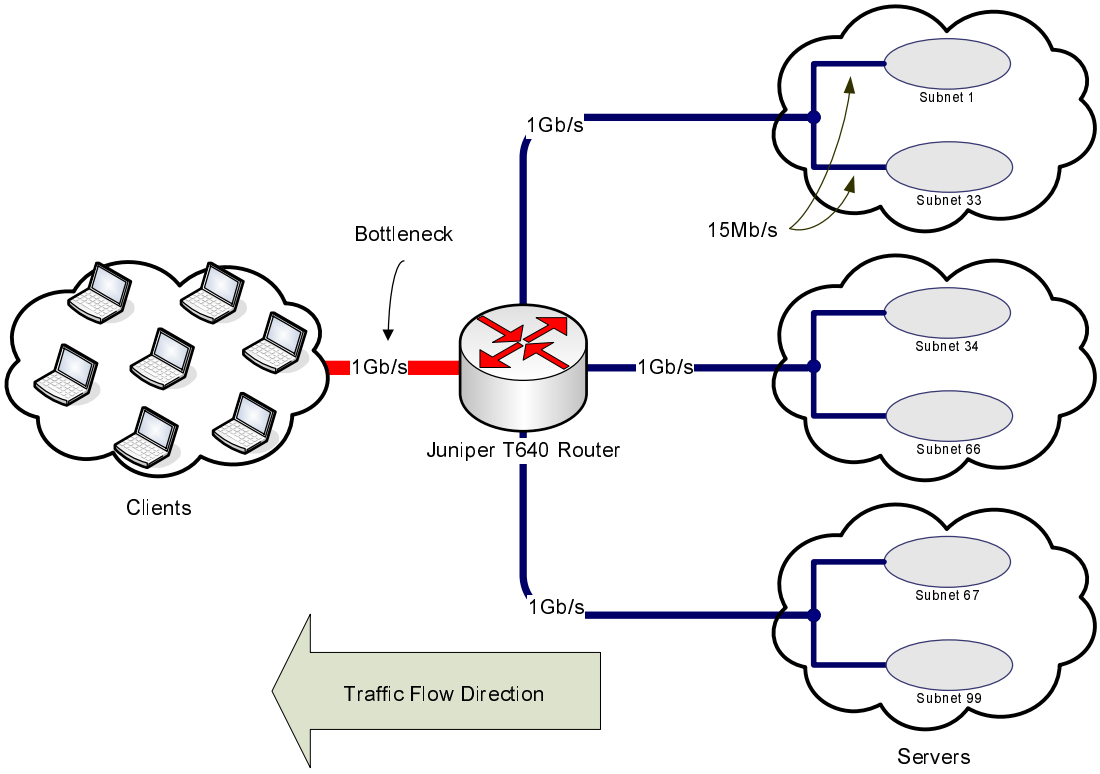

Figure 4.8: Topology of the network used in experiments. The capacity of core links is 1Gb/s, and the capacity of access links is 15Mb/s.

4.2 Tiny Buffers Experiments

In this section, we describe our experiments on tiny buffer sizing model. These experiments are carried out in the context of the tiny buffer sizing theory: *i.e.*, we consider a single point of congestion, assume core links run much faster than the access links, and expect a 10-20% reduction in network throughput. Without a guarantee that these conditions hold in an operational backbone network, it is not feasible to test tiny buffer model, and therefore, we have to content ourselves to laboratory experiments. We understand this is a limiting factor, and view our work as a first pass in a more comprehensive experimental study of the tiny buffer sizing model by us and others.

We built a test-bed in collaboration with Sprint ATL for tiny buffer experiments.

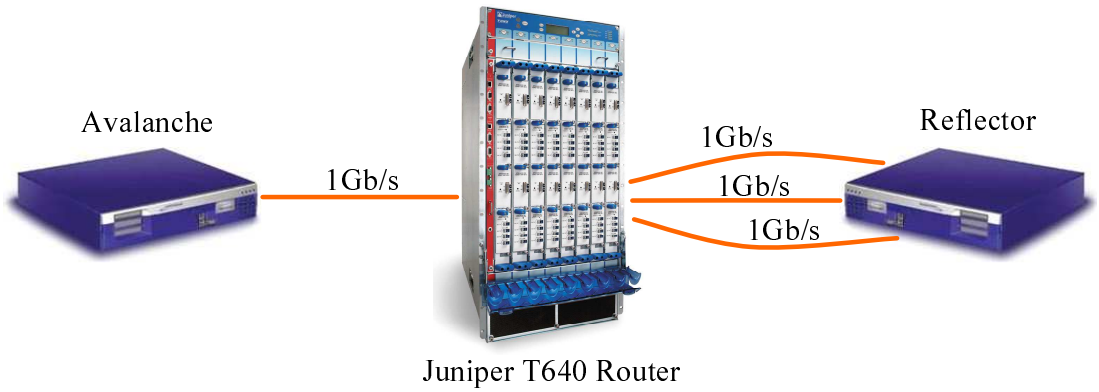

Figure 4.9: Tiny buffer experiment setup.

Figure 4.8 shows the topology of the emulated network, which is similar to the setting considered in tiny buffer sizing theory [19]. The core of the experiments is a Juniper T640 router, whose buffers are modified throughout the study.[2] The router is connected to four different networks through four Gigabit Ethernet interfaces. Each cloud in Figure 4.8 represents one of these networks. The cloud on the left contains all the users/clients, and the three clouds on the right hold the servers. Each server belongs to one of the 99 different subnets (33 for each of the three server networks). The capacity of the access link connecting each server to the rest of the network is set to 15Mb/s by default. The requests for file downloads flow from left to right (from clients to servers), and the actual files are sent back from right to left. In this direction, the router has three ingress and one egress line, which means by increasing the load we are able to create congestion on the link connection T640 router to the client network.

In practice, the clients and servers are emulated by two different boxes: Spirent Communications' Avalanche box plays the role of clients, and the Reflector box plays the role of servers (Figure 4.9). Each box has four Gigabit Ethernet Interfaces. Obviously, we use only one interface from the Avalanche box, and three interfaces from the Reflector box to connect the boxes to the T640 router. These links correspond

[2]We tried several other routers including Cisco GSR 12000, and Juniper M160. For the buffer sizes we were interested in this experiment, Juniper T640 seemed to be the most suitable choice.

to core links, and the link connecting the router to the Avalanche box is the target link. The access links are emulated by the Reflector box, which allows us to change the access link capacity to any desired value.[3] The delay associated with each access link is also emulated by the Reflector box. Since all other link delays are negligible, we can control the two-way propagation delay of packets by modifying these values.

Throughout the experiments, we use IPMon systems [2] to capture the headers of all the packets which go through the links connecting the router to the Avalanche and Reflector boxes. These headers are recorded along with high precision time-stamps. By matching the packet traces on ingress and egress lines of the router, we can measure the time each packet has spent inside the router, and thus, we can calculate the time-series representing the queue occupancy of the router. This also helps us identify any undocumented buffers inside the router. Such buffers could be fixed-delay buffers (*e.g.* part of a pipeline, or staging buffers), or could be additional FIFO queues.

4.2.1 Traffic Generator Evaluation

We evaluated the Avalanche/Reflector boxes to see if they generate accurate TCP Reno traffic. We started with a single TCP flow, then changed several parameters (link capacity, delay, congestion window size, packet drop rate, etc.) and manually studied the generated traffic. We concluded that packet injection times are accurate, except for a difference of up to 120μs (which can be attributed to processing times and queueing delays in the traffic generators). We also compared the traffic patterns generated by the Avalanche/Reflector boxes with patterns generated by the *ns-2* simulator in carefully designed scenarios. While *ns-2* is known to have problems of its own, we wanted to identify differences between experimental results and simulation. We found minor differences between *ns-2* and Avalanche output traffic. However, we believe these differences do not have any impact on buffer sizing experiments.[4]

[3] In the Internet, access links are slow on client side. We found out Avalanche does not enforce rate limitations for incoming traffic, and had to push slow accesses to the server side in this experiment so that we can emulate the impact of slow access links.

[4] These difference were reported to Spirent Communications, and some of them have been resolved in their current systems. We are working with them to resolve the remaining issues.

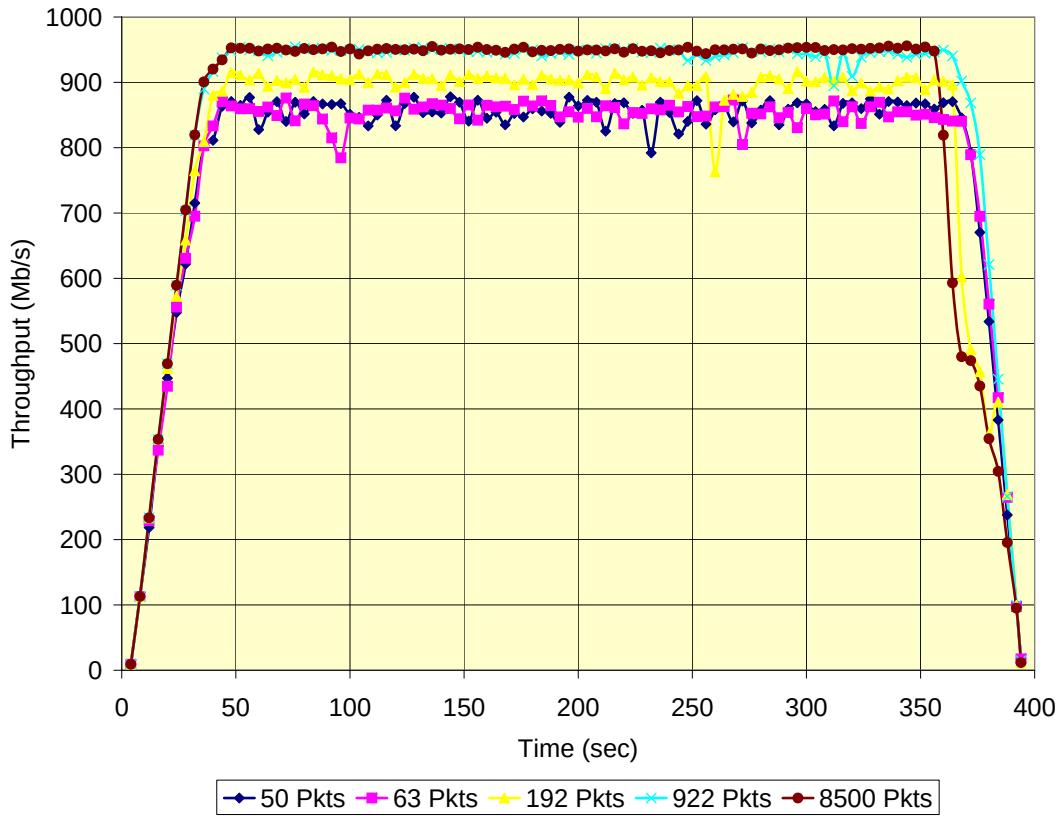

Figure 4.10: Throughput vs. time for various buffer sizes.

4.2.2 Experiment Results

In this section we study the impact of changing buffer sizes on network performance. When allowed by our testing equipment, we also study the effect of changing some other network properties (like traffic patterns, access link properties, number of flows, and others) on buffer sizing results.

Performance as a function of buffer size

We reduce the buffer sizes on the router from 8500 packets to just 50 packets, and measure the throughput, drop rate, and delay observed by individual packets as performance metrics. At 1Gb/s line speed, and an RTT of 50ms, 8500 packets is

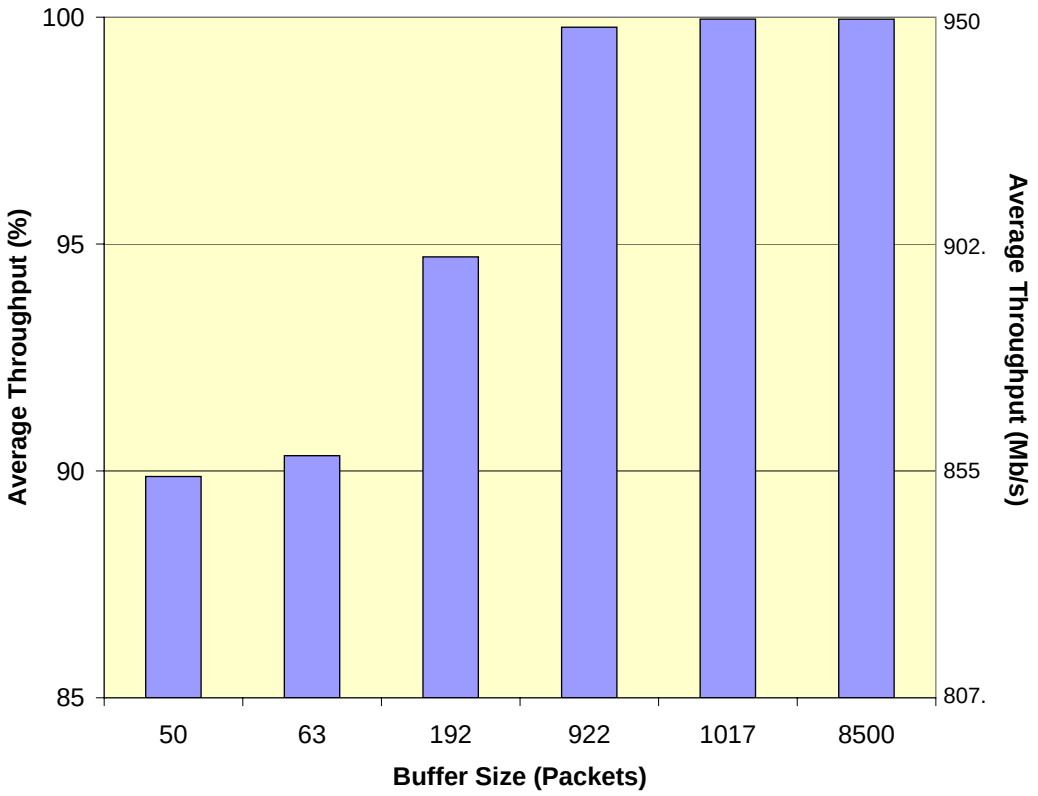

Figure 4.11: Average throughput vs. buffer size.

about twice the bandwidth-delay product, and 50 packets lies in the range of the tiny buffer sizing model. In this experiment we increase the number of users from 0 to 600 during a period of 50 seconds, and keep the number of users at 600 for 5 minutes, measuring throughput, delay, and drop rate during this time interval. The number 600 of users is chosen so that the effective load of the system is about 100%. Each user downloads a 1MB file from an ftp server. Once the file download is completed the user immediately starts downloading another file. The average RTT of the system is 50ms (more precisely $15 + U[0, 20]$ on each of the forward and reverse paths), and the capacity of access links connecting servers to the system is 15Mb/s. Both the server and clients have an advertised congestion window size of 16KB.

Figure 4.10 illustrates throughput as a function of time for various buffer sizes,

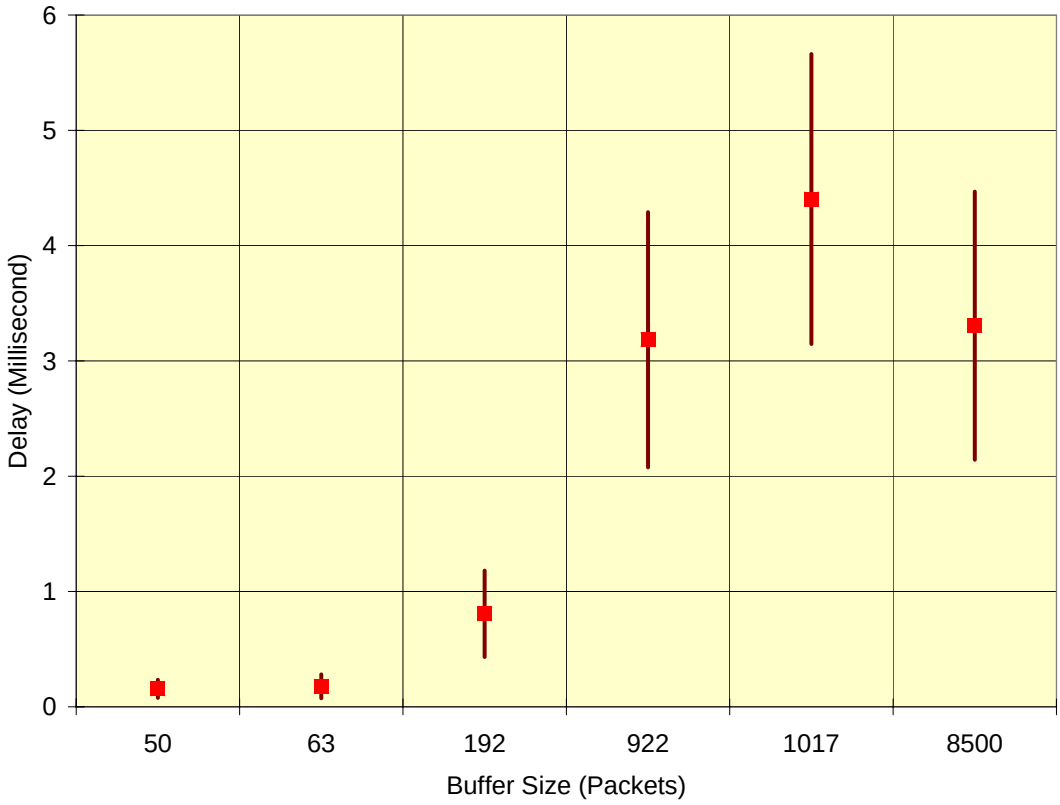

Figure 4.12: Delay statistics vs. the buffer size. The red square represents the average delay and the bar represents the standard deviation.

and Figure 4.11 represents the average throughput for different buffer sizes. If we consider the overhead of packet headers, the maximum throughput we can get is about 950Mb/s. We can see that a buffer size between 8500 and 922 packets, gives a throughput of about 100%. This is the range between the rule-of-thumb and the small buffer model. When we push the buffer size to 192, 63, and 50 packets, which is in the range of tiny buffers model, the throughput goes down by 10%, as predicted theoretically. The average level of throughput is maintained very smoothly throughout the experiments, as seen in Figure 4.10.

The other metrics which we consider here in order to measure the performance of the network are the delay statistics of packets going through the router, and the

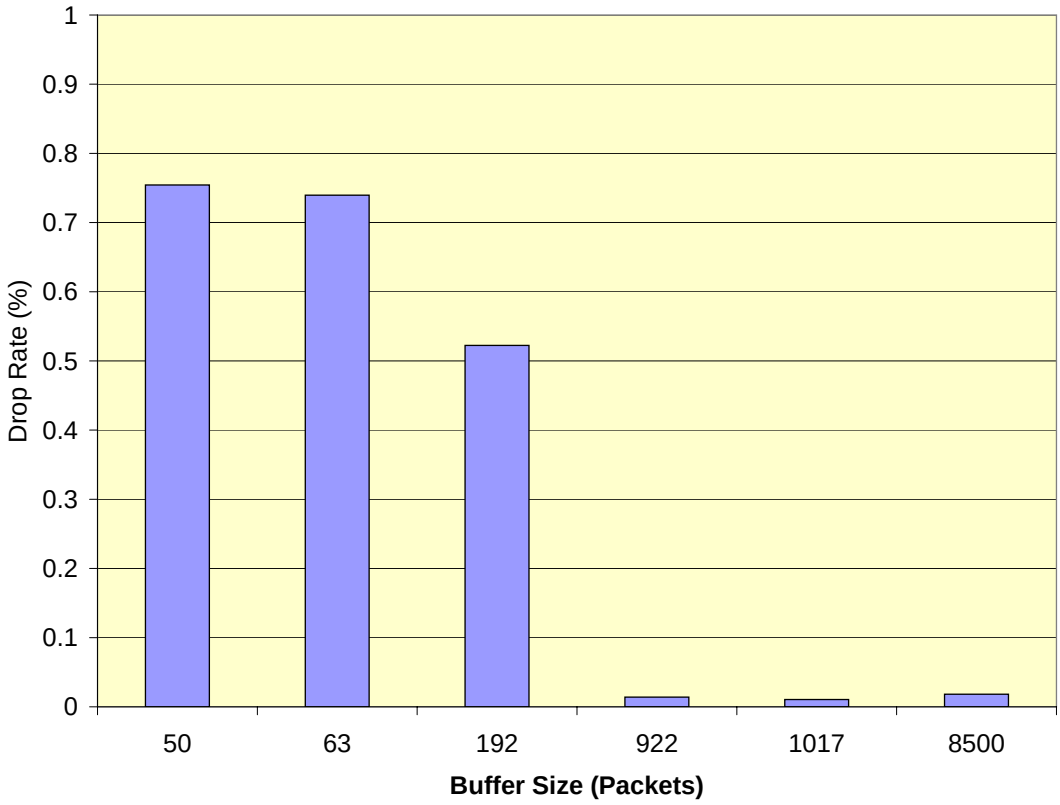

Figure 4.13: Drop rate vs. buffer size.

packet drop rate. Figure 4.12 shows that on average packets go through a delay of 155μs to 4.5ms (equivalent to 13 and 375 packets) for buffer sizes between 50 and 8500 packets. The average packet delay is considerably smaller than the maximum buffer size when it is set to 8500 packets. This is very similar to what we observed in Level 3 Communications' network.

The average delay increases as the buffer size is increased from 50 to 1017 packets, and is slightly reduced for 8500 packets. Since the packet drop rate is close to zero when buffer size is set to 1017 or 8500 packets, we expect these two to have similar average delays, and the observed reduction in average delay might be a result of activation/deactivation of some hidden buffer inside the router.

For buffer sizes between 922 and 8500 packets, the drop rate is very close to zero

(Figure 4.12). As expected, in these cases utilization is close to 100%. For smaller buffers we see a packet drop rate of up to 0.75%; only 0.25% more than a M/D/1 queue of similar size and arrival rate, confirming once more the smoothness of traffic going through the router.

The impact of increasing network load

In the previous experiment, parameters were chosen so that the effective system load is very close to 100%. What happens if we keep increasing the load? Does the throughput of the network collapse as a result of congestion? This is a valid concern, and to find out the answer, we performed another set of experiments. This time, we varied the *potential load* of the system between 25% and 150%. The potential load of the system is defined as the utilization achieved when the bottleneck link capacity is increased to infinity, and when the throughput is limited by other factors (like the maximum congestion window size, RTT, and the access link capacities). We control the system load by limiting the access link rates, and advertised congestion window, and by changing the number of end users from 150 to 1200.

Figure 4.14 plots the throughput of the system as a function of load and various buffer sizes in this scenario. For any given buffer size increasing the potential load monotonically increases the throughput. For large buffers, the throughput reaches 100% (950Mb/s) when the potential load is 100%, and remains at that level for increased potential load. For smaller buffers, the throughput reaches 90%-95% as we increase the potential load from 25% to 100%, and remains almost fixed beyond that point. This is good news in the sense that we do not see a collapse in throughput as a result of increased congestion. For a core network a potential load beyond 100% is very unlikely given that core networks are usually highly over-provisioned.

Performance as a function of the number of flows

We would like to see whether the number of flows affects the performance of the system. We cannot simply modify the number of flows, since the potential load to the system changes with the number of flows. To fix this problem we adjust the

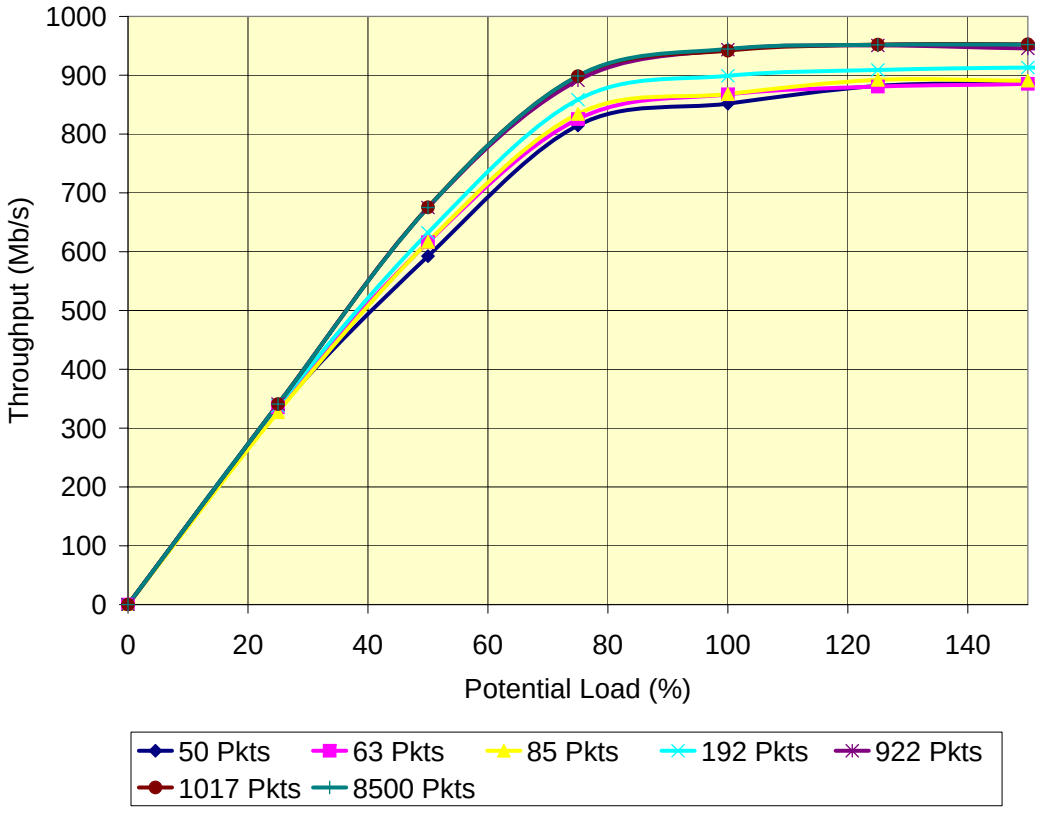

Figure 4.14: Throughput vs. potential load for different buffer sizes.

maximum congestion window size based on the number of flows in order to keep the potential load fixed. For 150 flows, the maximum congestion window size is set to 64KB. As we increase the number of flows to 300, 600, and 1200, we reduce the maximum congestion window size accordingly (to 32KB, 16KB, and 8KB). The buffer size is set to 85 packets in all these experiments.

Figure 4.15 illustrates the changes in network throughput as we increase the number of flows, and Table 4.1 summarizes the average throughput (in Mb/s) for various flow numbers and advertised congestion window sizes. When the number of flows is very low (*i.e.* 150-300) the system throughput is significantly less than 100%. Even when we increase the congestion window size (to increase the potential load), the system throughput is not significantly increased. This can be explained by tiny buffer

Congestion window	150 flows	300 flows	600 flows	1200 flows	4800 flows
1	-	-	-	-	0.1
2	-	-	-	-	764.9
4	-	-	-	548.0	818.7
8	-	-	651.0	876.9	-
16	-	596.3	868.8	893.7	-
32	492.7	752.8	879.5	-	-
64	558.3	742.0	-	-	-
128	522.8	-	-	-	-

Table 4.1: Throughput (Mb/s) as a function of advertised congestion window size and number of flows.

sizing model as follows: when the number of flows is low, we will not have a natural pacing as a result of multiplexing, and therefore, the throughput will not reach 100%.[5] When the number of flows is large (*i.e.* 600-1200), the system throughput easily reaches 90-95%, independent of the number of flows.

Increasing the number of flows beyond a few thousand can result in a significant reduction in throughput, as average the congestion window size becomes be very small (2-3 packets or even less), resulting in a very high drop rate and poor performance [37, 17]. This problem is not associated with tiny buffers, and unless we significantly increase the buffer sizes even more than the rule-of-thumb it would not be resolved. We believe this is a result of poor network design and increasing the buffer sizes is not the right way to address such issues.

Flow sizes

So far we have only considered users that download a 1MB fixed size file, which means all the flows are of the same length. To find out the impact of having flows of different sizes, we performed a set of experiments with several download patterns: fixed file sizes of 10KB, 100KB, and 1MB; uniformly random file size between 10KB and 500KB; and file sizes with a heavy-tailed distribution with parameters $\alpha = 2$ and

[5]This problem can be fixed by modifying traffic sources to use Paced TCP. Here, we do not have the tools to test this. The commercial traffic generator which we use, does not support Paced TCP.

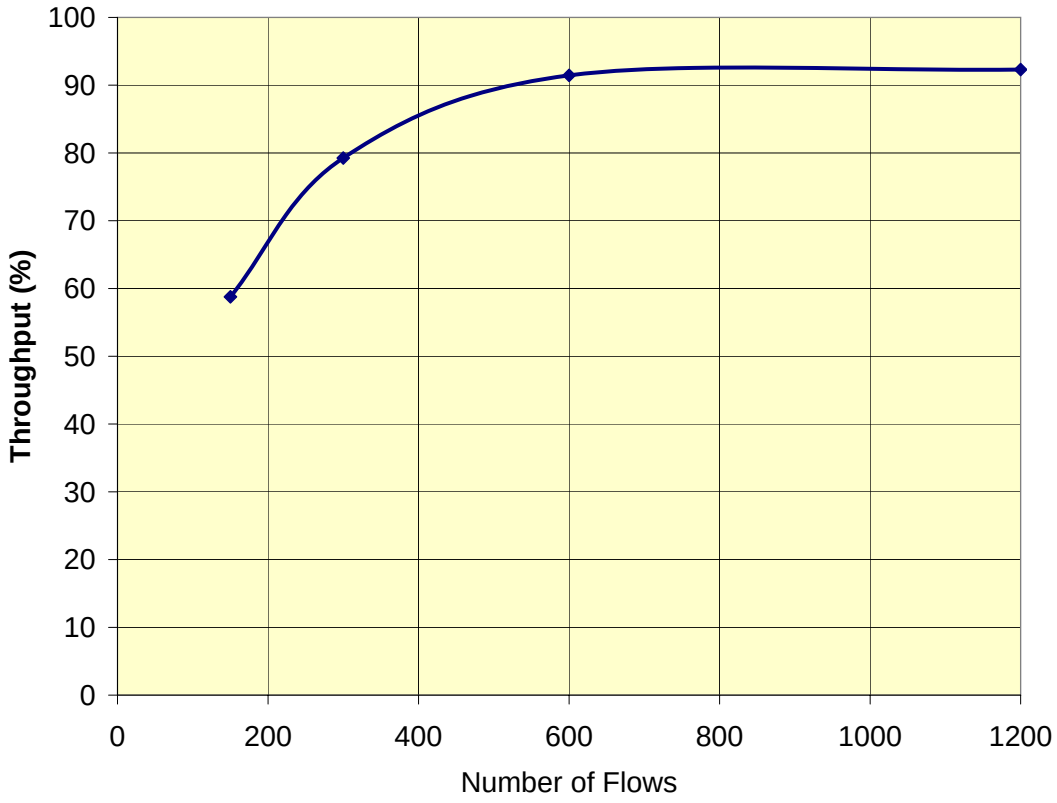

Figure 4.15: Throughput vs. the number of flows.

$K = 10$. In all these experiments the buffer size is set to 85 packets, and we emulate 600 users.

Table 4.2 summarizes the results and shows throughput and packet drop rates for different file size download patterns. We observe that regardless of file size distribution the drop rate is always below 1%, and the throughput remains above 74%. It is interesting to note that in patterns which have a lot of small flows, (10KB or heavy-tailed for example) we see more packet drops and lower link utilizations. Small flows spend the majority of their time in slow-start phase, during which packet injections are very bursty (due to sudden increases in the congestion window size). Large flows however, spend most of their time in congestion avoidance phase, during which the congestion window size changes much more slowly. A traffic pattern with a lot of

Flow Size	Throughput	Packet Drop Rate
10KB	74%	0.94%
100KB	82%	0.63%
1MB	91%	0.49%
Uniform	85%	0.21%
Heavy-tailed	80%	0.54%

Table 4.2: Drop rate vs. file size

small flows therefore, is more bursty; which means it will see more packet drops and less throughput compared to a traffic pattern with large flow sizes when the buffers are tiny.

Dependence on access link properties

The tiny buffer sizing model relies on natural pacing of flows which happens as a result of the difference between capacities of access links and core links in the network. Such difference between core and access link rates exists in today's network. A typical home user today, has an access link capacity of 56Kb to a few Mb. When the access links are very fast, the validity of tiny buffer sizing model depends on the heavy multiplexing of individual flows, or having sources which pace their injection, *e.g.* Paced TCP sources.

For the case where we rely on the disparity of the core and access link rates, a natural question is how slow the access links need to be in order to cause pacing needed for tiny buffer model? In order to answer this question, we performed a series of experiments in which the capacity of access links is set to 1Mb/s, 4Mb/s, 16Mb/s, 64Mb/s, and ∞. We increase the number of users from zero to 600, and measure the network throughput. The buffer size is 85 packets, and the average RTT is 50ms.

Figure 4.16 depicts the throughput of the system over time. When the access link capacity is very small (*i.e.* 1Mb/s and 4Mb/s), the system throughput remains low. This is because flows are throttled by access links, and the throughput cannot reach 100% even if we increase the buffer size. When the access link capacity reaches 16Mb/s we see the maximum throughput over time (93-94% on average). As we keep

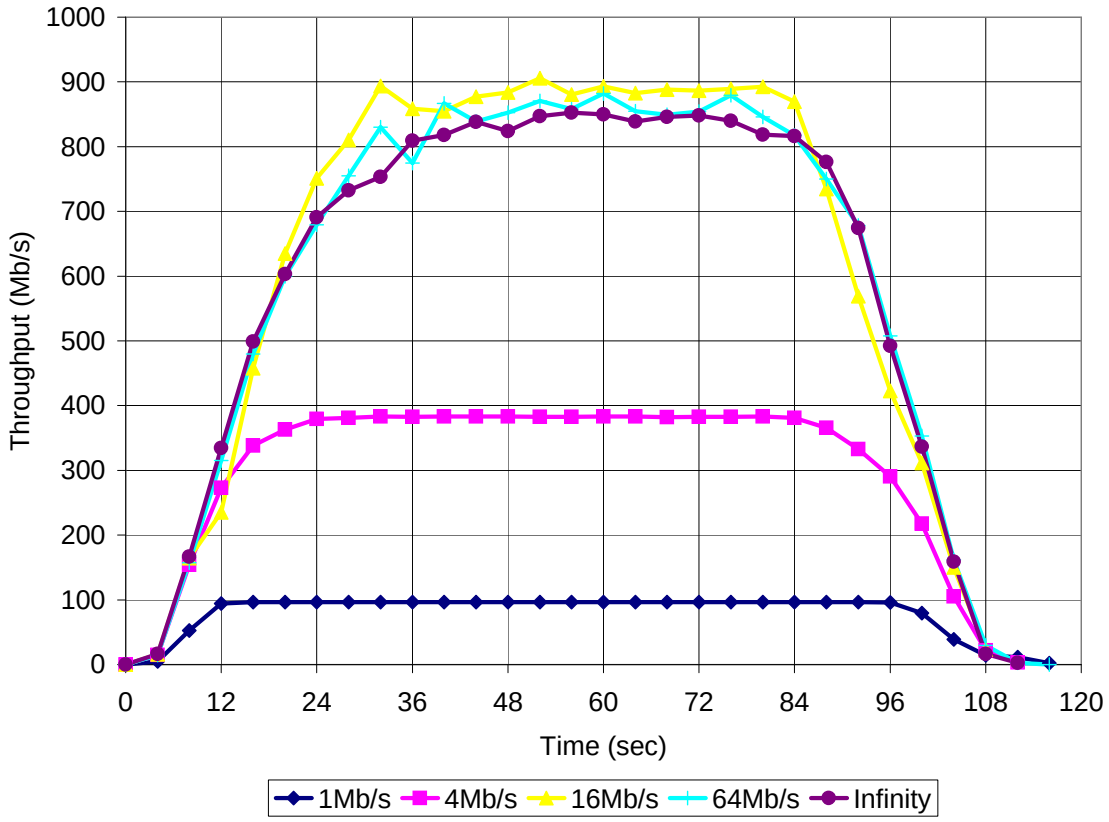

Figure 4.16: Throughput vs. access link capacity.

increasing the capacity of access links, the throughput slightly decreases to about 90%. This reduction happens because the natural pacing of flows vanishes as we increase the access link capacity. Because of the heavy multiplexing of individual flows, we still have a reasonably high throughput even when the access link capacities are very high. A test with fewer flows and with high access link capacities can result in significant reductions in throughput, as seen in Section 4.2.2.

Another property of access links that might impact the behaviors we have observed so far is the RTT associated with individual flows. To study this, we performed another experiment with four different link delay patterns:

1. 5+U[0,40] milliseconds forward, same reverse path

2. 20+U[0,10] milliseconds

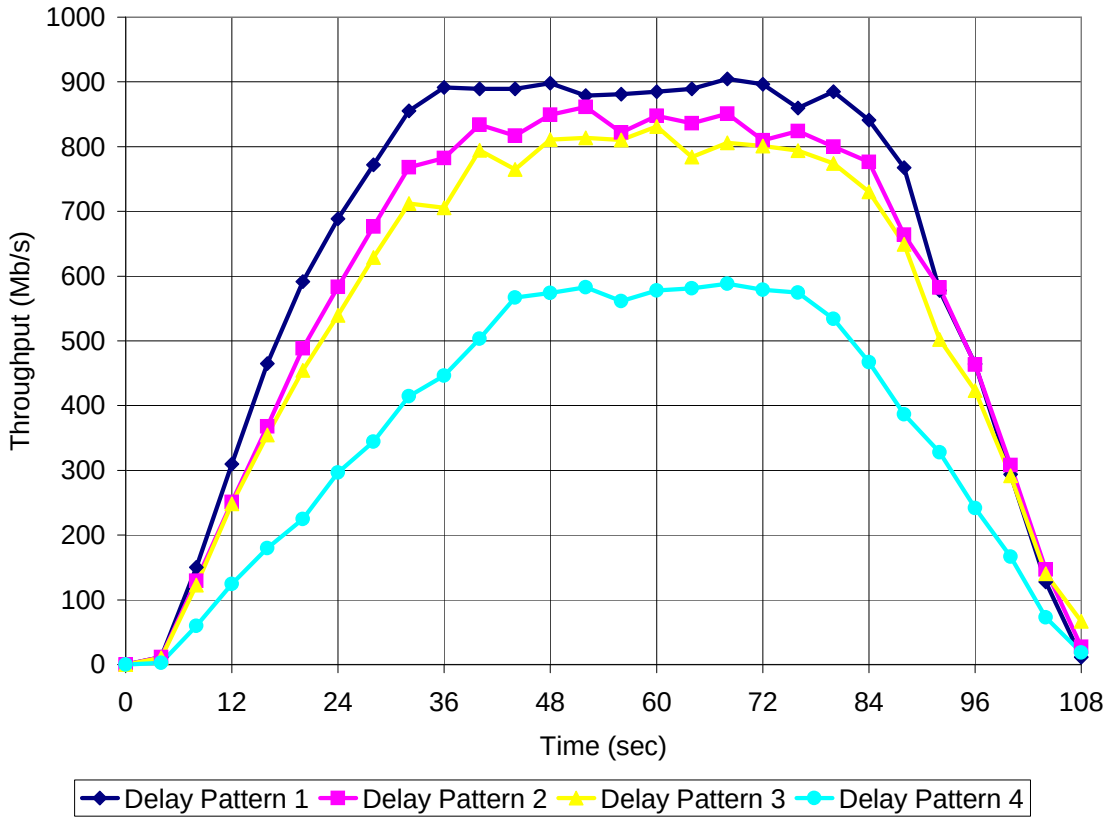

Figure 4.17: Throughput vs. delay pattern.

3. 25 milliseconds

4. 40+ U[0,20] milliseconds

Figure 4.17 depicts the throughput of the system over time for these link delay patterns. Other than the fourth pattern, modifying the RTTs does not have a significant impact on system throughput. In the case of the 4th pattern, we should note that we have effectively doubled the RTTs in this case, and that is why we see a significant reduction in throughput.

For patterns 1 to 3, even though the average RTT is fixed, the harmonic average is increased from pattern 1 to 3. Since the throughput is inversely proportional to the harmonic average of RTTs the overall throughput is slightly decreased in this case. However, these changes in throughput are not significant, and are similar to what

happens when buffers are large.

4.2.3 Hidden Buffers

Using commercial routers for buffer sizing experiments is difficult. Generally, we do not know the internal architecture of the router, or the location and exact size of buffers. And the router does not always allow buffer sizes to be set precisely. For example, in the Juniper T640 router which we used for Sprint ATL experiments, setting the buffer size to 1ms should result in a buffer which holds 83 packets at the given line rate of 1Gb/s. In reality, the buffer size is about 192 packets, which is almost twice the value set. Interestingly, decreasing the buffer size below 10ms significantly increases the actual delay observed by packets, presumably because of the activation of a hidden buffer. We determined the actual buffering by capturing traces of packets on ingress and egress lines of the router, along with high-precision time-stamps. These traces can be used to find the delay each packet has had inside the router, and hence the buffer size.

Any packet entering the router is either being processed in a pipeline, or sitting in a queue. By removing a constant minimum packet processing time, we can measure the delay observed by each packet. The maximum delay observed by any packet is considered to be the actual buffer size,[6] and by removing the minimum constant delay observed by all packets, we can find a good approximation of the buffer size.

All the buffer size values reported in Section 4.2 are based on measurements.

4.2.4 Other Tiny Buffer Experiments

The tiny buffer experiments have been independently performed by research groups at Lucent Technologies, and Verizon Communications, and they have found similar results. Other experiments have also considered some other scenarios like the impact of failures, cascading, and other traffic types like http and streaming protocols. The only operational network experiment we have done is performed in collaboration with

[6]The buffer size of 100ms is an exception here, since the buffer never becomes full. In this case, we estimate the actual buffer size based on the nominal value.

Internet2. We reduced the buffer size of a core router down to 50 packets, while measuring the performance through active flow injections, and passive monitoring tools. Our measurements of throughput and packet drops did not show any degradation. We note that Internet2 operates its network at very low utilization (20-30%) – not an ideal setup for buffer sizing experiments. For more details on these experiments, we refer the reader to the extended version of this paper [25].

4.3 Summary

The small buffers rule appears to hold in laboratory and operational backbone networks – subject to the limited number of scenarios we can create. We are sufficiently confident in the small buffers result to conclude that it is probably time, and safe, to reduce buffers in backbone routers, at least for the sake of experimenting more in a fully operational backbone network. The tiny buffer size experiments are also consistent with the theory. However, we suspect that there are many more scenarios and boundary cases to consider before deciding whether it is time to reduce buffers to just 20-50 packets. In the mean time, more experiments are needed to better understand these results.

Chapter 5

Tiny Buffers in Practice

In order to theoretically analyze or simulate any complex system, we need to discard some details present in the real system. Buffer sizing is not an exception. In this chapter, we will review some implicit simplifying assumptions that we have made previously, and show how one can apply buffer sizing results in practice despite these simplifications.[1]

Throughout this work, we have always assumed all routers and switches are *output queued* (OQ). In an OQ switch, packets arriving at input ports of the switch, are immediately transferred to the corresponding output port by the crossbar switching fabric. Packets are queued in the output port before they can depart the switch. In an N port switch, there might be as many as N packets destined to a given output port at any given time; thus, the switching fabric must be able to simultaneously transfer up to N packets from the input ports to a given output port. In other words, the switching fabric must operate at N times the ingress/egress line rates, which is a very restrictive requirement. In practice, most switches have buffers in both the input, and output ports, and are called combined input-output queued (CIOQ) switches, and the switching fabric operates at a rate which is usually within a 1-2 factor of the line rate. In Section 5.1 we will show how we can apply the OQ switch based results to the more realistic case of a CIOQ switch.

[1]Part of this work is done in collaboration with Neda Beheshti, Ramesh Rajaduray, Daniel Blumenthal, and Nick McKeown [9, 8].

In electronic switching devices, In electronic switching devices, we can randomly access any memory location. This is not trivial in all-optical routers, since we do not have random access memories. Using optical delay lines, a packet entering the delay line needs to remain in there for a fixed duration of time before one can retrieve it. In Section 5.2 we study the problem of how one can emulate FIFO queues using optical delay lines. We show the feasibility of constructing a FIFO queue of size N by using only $O(\log N)$ 2×2 switches. A simple scheduling algorithm that achieves this bound is developed, and the tradeoff between the number of required 2×2 switches and the maximum delay line length is discussed. The proposed structure provides an efficient way of storing optical packets using a minimal number of delay lines and switches.

5.1 Combined Input-Output Queued Switching

Throughout this work, we have always assumed all routers and switches are *Output Queued* (OQ). In an OQ switch, packets arriving at input port of the switch, are immediately transferred to the corresponding output port by the crossbar switching fabric. Packets are queued in the output port before they can depart the switch. In an N port switch, there might be as many as N packets destined to a given output port at any given time; therefore, the switching crossbar must be able to simultaneously transfer up to N packets from the input ports to a given output port. In other words, the switch must operate at N times the ingress/egress line rates, or have a *speedup* of N. This is a very restrictive requirement. In practice, the switching fabric in most routers operates at a speed which is 1-2 times faster than the line rates [1], and as a result routers have buffers in both the input, and output ports, and are called Combined Input-Output Queued (CIOQ) switches.

A CIOQ switch with speedup of s, has s scheduling cycles per time slot. Once per cycle, a scheduling policy determines which packets to transfer from the input ports to the output ports. In a crossbar switch, the scheduling algorithm must resolve two constraints per cycle: (1) at most one packet can be removed from each input port; and (2) at most one packet can be delivered to each output port. In a switch with N input/output ports, and with a speed up $1 < s < N$, buffers are required at both

input and output ports. Internet routers today commonly use virtual output queuing (VOQ), in which an input linecard buffers packets which are destined for different output linecards in separate FIFO queues.

In this section, we want to find out how big the buffer should be for a CIOQ router. We show that in theory, a CIOQ switch needs no more buffers than an OQ switch, *i.e.*, 20-50 packets should suffice under the same constraints described in Chapter 3. Using simulation, we explore how much buffer is needed when we use practical scheduling algorithms.

5.1.1 Theoretical Bounds

In this section we show how the tiny buffers result for OQ routers can be generalized to apply to a CIOQ router.

Definition 1. Consider two routers A and B, and assume the same input traffic is fed to both routers. B is said to *exactly emulate* A if packets depart at the same time from both routers.

At a given time t, let us denote the total number of packets buffered in a router A which are destined to output port j with $Q_A(j,t)$. Clearly, for an OQ router A, $Q_A(j,t)$ is the occupancy of the output queue j at time t.

Lemma 1. Let us consider a CIOQ router B which exactly emulates an OQ router A. At any point of time t, we have

$$Q_B(j,t) \leq Q_A(j,t).$$

Proof. Let us assume the contrary. There must be a time t, and an output port j such that $Q_B(j,t) > Q_A(j,t)$. Now, let us call the last packet which has entered the system P. Since A is an OQ switch, it will serve P in exactly $Q_A(j,t)$ time slots. There are more packets than $Q_A(j,t)$ buffered in router B, and they need to be served in $Q_A(j,t)$ time slots (since B exactly emulates A), a contradiction. □

Based on this lemma if we can find a way of emulating an OQ router with a CIOQ one, we can apply the tiny buffers results and prove that the CIOQ router needs tiny

buffers in order to gain very high throughput under the constraints mentioned before. Interestingly, Chuang *et al.* have shown that a CIOQ switch can emulate an OQ switch using a specific scheduling algorithm called the stable marriage scheduling and with a speedup of two [15]. Combining this result with Lemma 1, and the tiny buffers results we can prove the following theorem.

Theorem 3. There is a scheduling algorithm, which can guarantee a high throughput (for instance, above 80% of link capacity) in a CIOQ router with very small buffers (20-50 packets) under the constraints stated in Chapter 3 and with a speedup of two.

Theorem 3 shows the possibility of building a CIOQ router with very small buffers using the stable marriage scheduling algorithm. This algorithm is somewhat complex for implementation, and therefore it might be a good idea to take a look into simpler algorithms. We note that, even though in the above theorem we exactly emulate an OQ router with a CIOQ router, in practice exact emulation is not needed. If a CIOQ router can roughly emulate an OQ router we might still be able to use the same arguments as above to show the possibility of having realistic routers with tiny buffers. In what follows, we study the performance of a CIOQ packet switch with a speedup of 2, and a relatively simple scheduling algorithm.

5.1.2 Simulation Results

We enhanced *ns-2* [4] to include accurate router models with CIOQ switching. The simulated topology is depicted in Figure 5.1. The router has 32 input and output ports; where each input port carries 500 multiplexed TCP Reno flows. Flows are generated at separate source nodes on slow access links, then multiplexed together onto the backbone. The backbone link capacity is 40Gb/s. The multiplexer buffers are sufficiently large to avoid any losses other than possible ones at the input or output ports of the router. The data packet size is 1000 bytes for all flows. Packets arriving at the switch linecards are segmented into fixed size cells, then reassembled before they depart. The router has speedup of two; in each cycle a scheduling algorithm [43] removes either zero or one cell from every input port and sends it to the corresponding output. If a contention exists among input ports, *i.e.*, if there are more than one

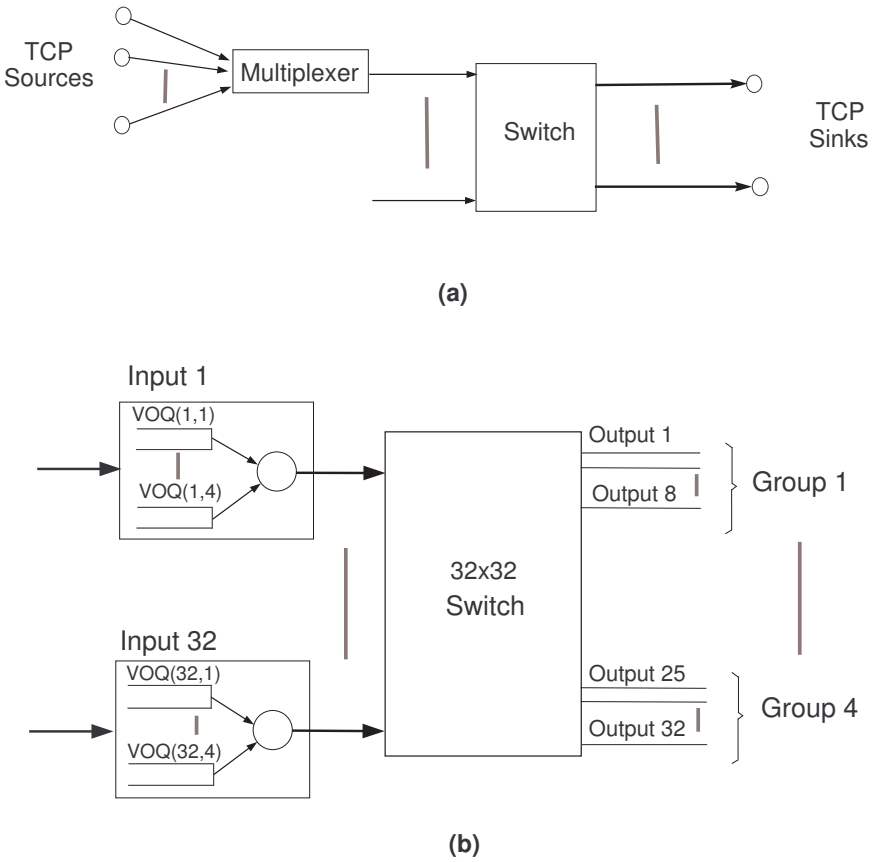

Figure 5.1: Topology of the switch used in *ns-2* simulations.

input ports requesting connection to a single output port, then the ports which are less recently served take priority over others. The same criterion holds if there is a contention among output ports; priority is given to less recently served output ports. The output traffic is distributed uniformly among different channels of each output port.

Output ports of the switch are divided into 4 groups of 8 optical channels. All the channels in a group carry traffic to one destination. The generated traffic consists of long-lived TCP flows with an average RTT of 40ms. Different RTTs are modeled by adding random jitter to the propagation delay of the access links.

To find out how much buffering is needed at input and output ports, we first assume that the size of buffers at the input side is unlimited. Simulation results show

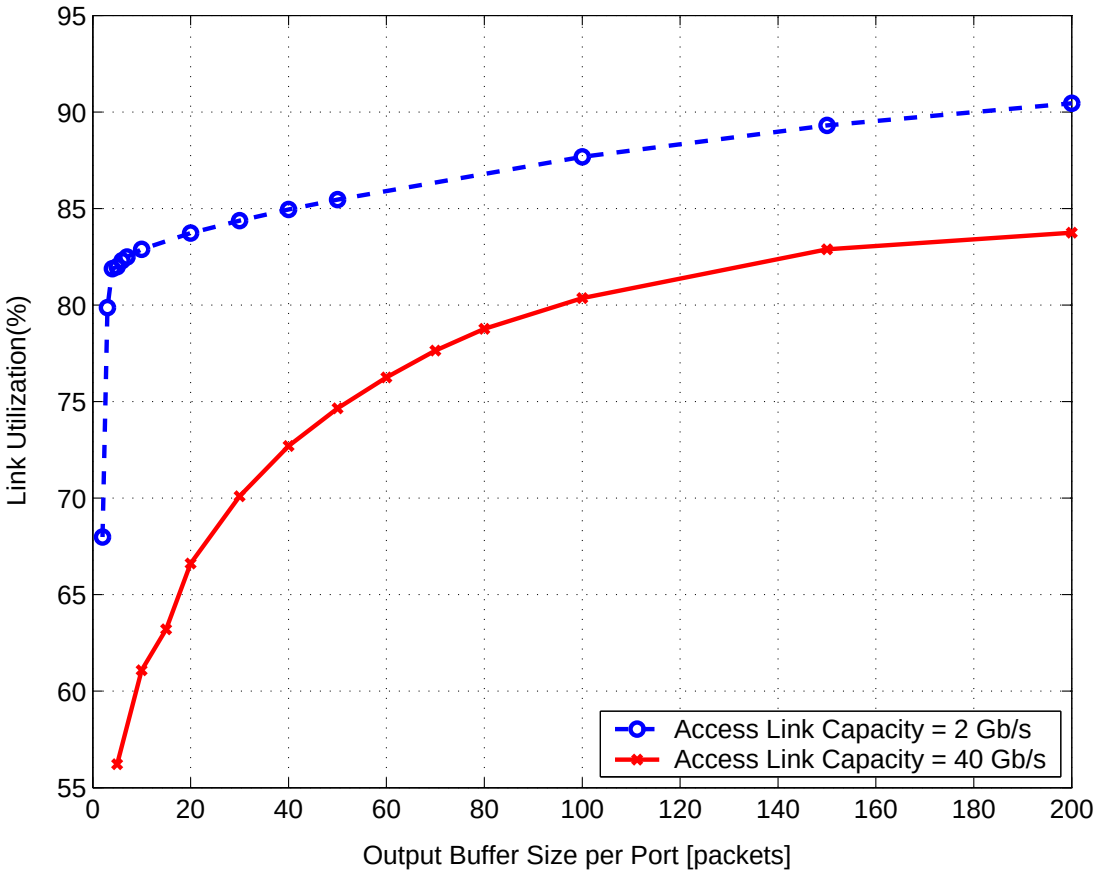

Figure 5.2: Throughput of a CIOQ switch as a function of buffer size.

that the tail probability of queue occupancy at input ports is a non-decreasing function of the output buffer size. But even with output buffers of size 1000 packets, the VOQ occupancy does not exceed 2 packets in the duration of simulation. Therefore in the next set of simulations we assume that the size of input queues is only 2 packets per VOQ, and investigate the appropriate size of the output queues.

The amount of buffering needed in a router depends on how bursty the TCP traffic is. As argued in Chapter 3, when the access links have limited capacity compared to the core links, the router finds the arrival traffic less bursty and this reduces the size of buffers needed for achieving high throughput. Figure 5.2 shows how link utilization varies as the size of output buffer increases considering two scenarios: 1) Access links

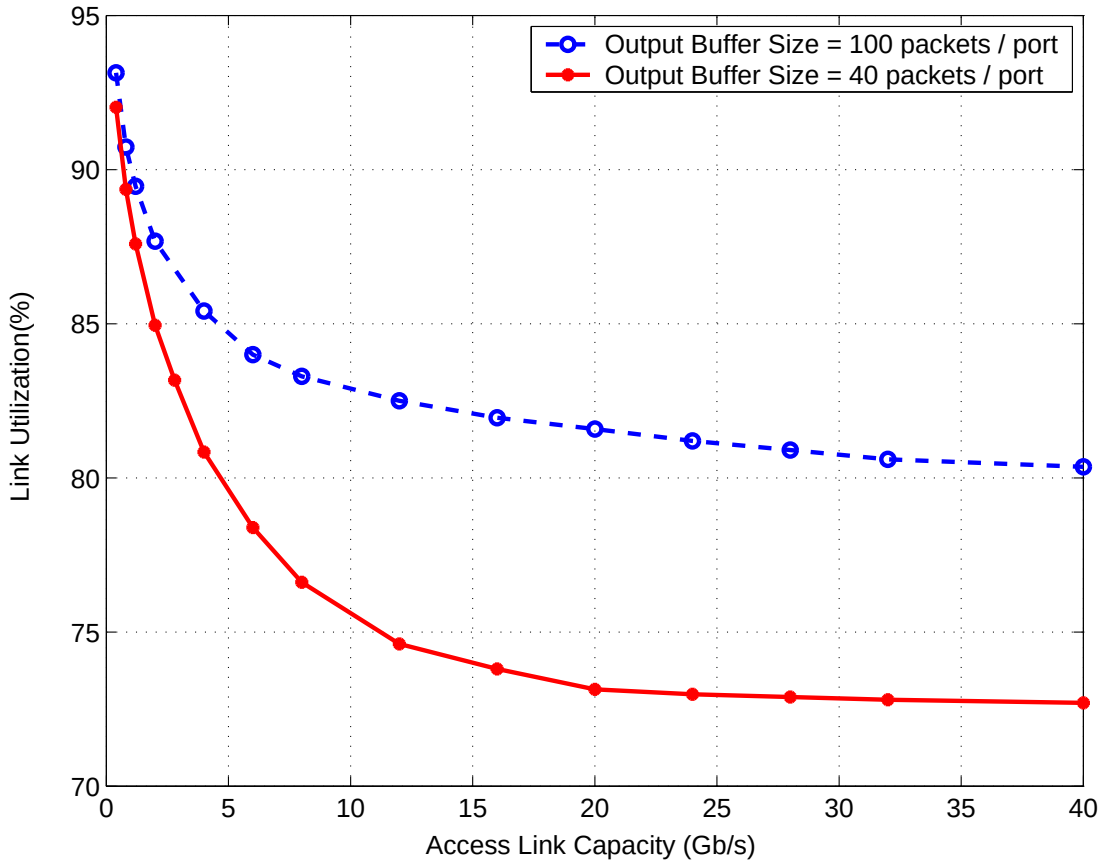

Figure 5.3: Throughput of a CIOQ switch as a function of access link capacity.

have the same capacity as the core links, and 2) Capacity of each access link is 2Gb/s, *i.e.*, 5% of the core link capacity. In both cases the maximum offered load to output ports is set to one. That is when each flow is sending at its maximum window size, the traffic destined for each output port is equal to the link capacity. The plot shows that with slow access links, we need only five packet buffers per output port to achieve 80% utilization. We would need about 100 packets if the access links run at the same speed as the backbone. In both cases, the input buffer size is two packets per VOQ.

Figure 5.3 depicts the throughput of the system as the access link capacity varies. Here, the capacity of the core links is kept fix at 40Gb/s, and the maximum offered load to the output links is set to one. It can be seen that as we increase the access link capacity, the throughput decreases. As mentioned before, this is because of losing the

natural spacing between packets by increasing the capacity of access links.

In all the above simulations the number of flows multiplexed on input ports is assumed to be 500. This number is bounded above due to the memory constraints of the *ns-2* simulator. In practice however, this number is typically much larger in the core of the Internet. An OC192 link for example, can carry up to 30,000 flows at a time. Larger number of flows make the traffic smoother and achieve a higher throughput. The simulation results therefore, provide a lower bound on the expected throughput of a network with tens of thousands of flows.

5.2 All-optical FIFO Queue Using Delay Lines

In electronic switching devices, we can randomly access any memory location. This is not trivial in all-optical switches. In fact, one of the main stumbling blocks to the rapid deployment of all-optical networks is the lack of optical FIFO buffers. The best solution which has been proposed for the optical buffering problem is to use optical delay lines in order to delay packets for a certain amount of time. Even though this is not as good as having a random access memory (like what we have in electronic networks), it can solve the buffering problem to some extent. Now, the question is whether we can emulate arbitrary queueing schemes, like FIFO, using optical delay lines. This is the problem we will address in this section.

Figures 5.4 (a) and (b) show how we can use a single optical delay line (or a fiber delay line) and a 2×2 optical switch to create a simple optical buffering element. Upon the arrival of a packet to the buffer, the switch must be in the crossed state (shown in Figure 5.4 (a)), directing the packet to the optical delay line. The length of the optical delay line must be large enough to hold the entire packet. Once the packet is completely on the optical delay line, the switch changes to the parallel state (shown in Figure 5.4 (b)). This will form a loop in which the packet will circulate.[2] When a packet is about to leave the buffer, the switch is changed back to the cross state and stays in that state until the packet leaves system.

[2]The packet can circulate in the buffer for a limited number of times due to the attenuation of the optical signal power.

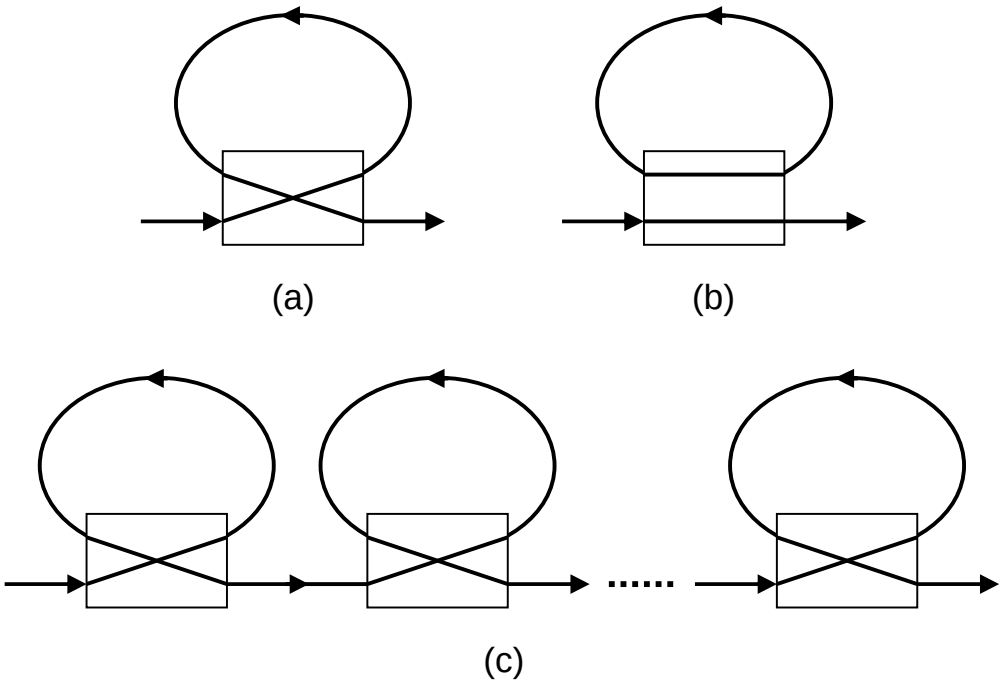

Figure 5.4: Building all-optical buffers from optical delay lines.

We can create an all-optical buffer of arbitrary size by using a series of optical buffering elements, as shown in Figure 5.4 (c). In order to store N packets, we need to have N optical delay lines, and N switches of size 2×2. Each packet is stored in the rightmost delay line which is not full. Upon departure of a given packet, all other packets are shifted to the right. Given the complexity of building switches in optics, this architecture – despite being relatively simple – might not be practical, as the number of 2×2 switches grows linearly with the number of packets that need to be stored.

There has been a considerable amount of work on emulating optical queues and multiplexers using optical delay lines and optical switches [27, 13, 46, 14, 16]. In a FIFO multiplexer the departure time of packets is known upon their arrival, whereas in a FIFO queue, the departure time can not be determined in advance. Sarwate and Anantharam proposed an architecture for emulating any priority queue of length N, in which the number of delay lines is $O(\sqrt{N})$ [46]. These delay lines are all connected

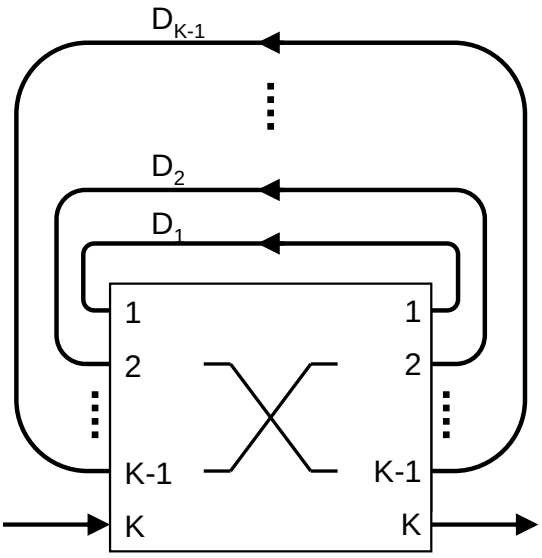

Figure 5.5: General architecture for building all-optical buffers from optical delay lines and switches.

to a $K \times K$ switch ($K = 2 \log N$), as shown in Figure 5.5. This switch transfers packets from a given delay line to the same delay line or any other one. A scheduling algorithm controls the state of the switch at each point of time.

Sarwate and Anantharam also proved that we need at least $O(\log N)$ delay lines for this problem. Recently, C.S. Chang *et al.* proposed a 3 stage design which achieves the $O(\log N)$ lower bound through recursive expansion of the same construction [13]. Here, we present a similar result which was devised independently, and is somewhat simpler in the sense that it does not need to keep track of the longest and the shortest delay lines in each step of the recursion. Our scheduling algorithm can be implemented in a distributed manner, as long as we notify individual units of the arrival and departure requests to the system.

5.2.1 Preliminaries and Assumptions

We consider the architecture depicted in Figure 5.5 as a starting point. Incoming packets arrive at input K of the switch, and departing packets leave the switch from the output line K.

We assume all packets are of fixed size, that time is slotted, and the length of a time slot is the time needed for a packet to enter (leave) an optical delay line. At each time slot, the system receives at most one arrival request and one departure request. The sequence of arrivals and departures is not known in advance.

The length of a delay line – assumed to be an integer – is the total number of back to back packets which the delay line can hold. If we have a delay line of length L it takes $L+1$ time slots for a packet from the time it starts entering the delay line to the time it has completely left the delay line.

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

We define the *order* of a given packet P as the total number of packets in the system (*i.e.* not including packets which have departed) which have an arrival time before P. In a FIFO queueing system, the order of each packet is decremented by one after each departure.

5.2.2 Emulating FIFO with $O(\log N)$ Switches

In this section we present a buffering architecture consisting of $O(\log N)$ optical delay lines that can emulate a FIFO queueing system, *i.e.*, if we apply the same sequence of arrival and departures requests to both systems, the corresponding output and drop sequences are the same.

Let us consider the structure depicted in Figure 5.6, where $D_1, ..., D_{\log N}$ are delay lines, and W is a *waiting line*; as defined later. We note that the $K \times K$ switch in Figure 5.5 has been replaced by a number of 2×2 switches here. The length of delay line D_i is $L_i = 2^{i-1}$; making the aggregate length of delay lines $\{D_i\}$ equal to $N-1$. Incoming packets are buffered by going through a number of these optical lines.

As time progresses, optical packets in each delay line move in the direction shown in Figure 5.6 towards the head of delay lines. At the end of each time slot, a scheduler determines the next position of the head of line packets. The system directs these

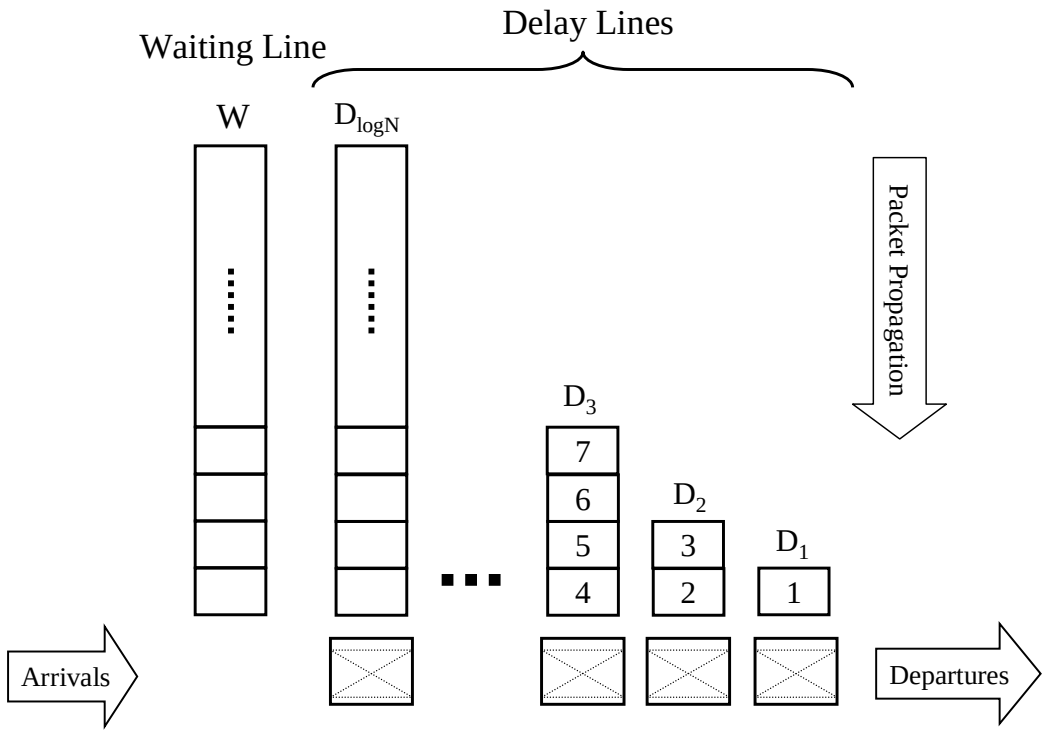

Figure 5.6: Emulating a FIFO queue using delay lines. The system regulates the position of the arrived packets by passing them through a waiting line.

head of line packets towards their scheduled positions by configuring the 2×2 switches corresponding to each delay line. The scheduler also checks whether there is any arrival or departure requests. Upon a departure request, the packet of order 1 will be scheduled to leave the system at next time slot. Upon an arrival request, 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.

The waiting line W operates as a regulator of the arriving packets. Since the time distance between successive arriving packets is not known in advance, the system uses this waiting line to adjust the location of packets in delay lines. W has a FIFO structure, with the property that the departure time of each packet that is placed in it, is known upon its arrival. In what follows we first show that this structure along with our proposed algorithm can emulate a FIFO queue of length $N-1$. We then show that the waiting line W can itself be constructed by $\log N-1$ delay lines. This

makes the total number of required delay lines equal to $2 \log N - 1$.

Algorithm A describes the scheduling procedure in details. The main idea of the scheduling algorithm is to place the packets in delay lines consecutively, *i.e.*, packets with successive departing order are placed back to back in delay lines. This is in contrast with what is proposed in [46], where arriving packets are allowed to fill out any available positions at the tail of delay lines. Clearly it might occur that a packet arrives at the system, while its preceding packet is in the middle of some delay line. In these cases, the system keeps the arrived packet in W for a proper number of time slots before directing it to one of the delay lines. This waiting time is equal to the time it takes for the preceding packet to traverse the delay line it is currently in, and gets switched to a tail position. During the waiting time, all the arrived packets are kept in W in a FIFO order. We say the waiting line W is *idle* if there are some packets in it, but there is no departures from it due to the scheduling algorithm's waiting policy.

At a departure request incident, if the system is non-empty, the packet with departing order 1 is delivered to the output link. Our scheduling algorithm guarantees that this packet is always at the head of some delay line. The departing order of every remaining packet in the system is then reduced by one.

At the end of each time slot, the next position of the head of line packets is determined by the scheduler. The scheduler decides whether to recirculate a packet in the same delay line it is currently in, or to send it to a shorter delay line. This decision is made independently for each head-of-line packet, based only on the packet's departing order as follows: in the next time slot, the packet will be transferred to the tail position of the longest delay line whose length is not greater than the order of the packet. More precisely, if the head-of-line packet has departing order m, then it will be placed in delay line $D_{\lceil \log m \rceil}$. As will be shown later, this ensures that packets are at the head of some delay line when their departing order goes down to 1.

The same scheduling policy applies when the waiting time of a packet in W goes to zero: the packet will be transferred to the tail position of the longest delay line whose length is not greater than the departing order of the packet.

Note that because there is no randomness in the scheduling algorithm, every

Algorithm A- Packet Scheduling

1. Arrival Event

   Let k be the total number of packets in the system.

   if $k = N$ **then**

   drop the arrived packet

   else

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

   if p_k is in W **then**

   place p_{k+1} in W

   else

   $waiting\_time \leftarrow (d+1) \mod l$, where l is the length of the delay line which contains p_k, and d 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 shortest queue with length greater than or equal to the order of the packet.

2. Departure Event

   Remove the packet with order 1 from the system, and decrease the order of all packets in the system 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 shortest delay line with length greater 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 shortest delay line with length greater than or equal to the order of the packet.

switch can be scheduled separately, as long as the local scheduler is informed of every arrival and departure event in the system. Based on this information, the scheduler can keep track of the order of all packets going through its corresponding delay line. Whenever the order of a packet at the head of the line is smaller than the length of the delay line, it will be sent out from the line. Also because all the arrival information is known at the local scheduler, at each time slot, the scheduler knows if there is a packet destined for its delay line, and will keep the switch close to make the arriving packet insert that link.

The following theorem states that the presented scheduling algorithm makes the overall system behave exactly as a FIFO queue.

Theorem 4. A FIFO queue of size N can be emulated by $O(\log N)$ delay lines.

To prove this theorem, we show that if $N = 2^n$ for some $n > 0$, by exactly $2 \log N - 1$ delay lines, a FIFO of size $N - 1$ can be emulated. In the sequel, N is assumed to be a power of 2, unless otherwise stated.

Lemma 2. The occupancy of W is less than or equal to $N/2$.

Proof. The maximum idle time of the non-empty W is equal to the length of the longest delay line, which is $N/2$. When W goes to the non-idle mode, it stays in that mode and sends out one packet in each time slot unless it gets empty. Therefore, the occupancy never exceeds $N/2$. □

The next step is to show that the scheduling of the head of line packets is contention free, *i.e.*, the scheduling algorithm guarantees that no more than one packet would be switched into a single delay line. Assume that all the $N - 1$ locations in delay lines are enumerated as shown in Figure 5.6. We say that at time slot t the location of a packet P is l, if the packet is located in the line slot numbered l.

Lemma 3. There is no contention among head of delay line packets for a given delay line.

The proof comes in Appendix C.

Lemma 4. Packet with departure order 1 is always at the head of a delay line.

Proof. When a packet P with departure order k gets to the head of some delay line, it will be scheduled to be placed in a delay line with length $l \leq k$ at the next time slot. In other words, there will be at most $k-1$ departures from the system before packet P reaches the head of line again, and hence its departure order can not be 1 unless it is at a head of some delay line. □

The above lemmas show that (1) an arriving packet will be buffered in the system as long as the total number of packets in the system remains less than N, (2) the scheduling algorithm relocates packets in such a way that there is no contention for a given location. Moreover, any packet can get to the head of line before its departing order drops to 1, and (3) the packets depart in the same order they arrive at the system. This completes the proof of theorem 4.

5.2.3 Construction of the Waiting Line W

To avoid future contention, the scheduling policy of algorithm A does not allow any void places between packets located in delay lines. To achieve this, an arriving packet is kept in the waiting line until its preceding packet – in arriving order, or equally in departing order – passes the tail of a delay line. In the following time slot, the waiting packet will be placed in a line according to algorithm A.

Theorem 5. Waiting line W can be constructed by $log(N)$ delay lines.

Consider a group of delay lines D'_i $i = 1, 2, ..., \log N - 1$, where the length of the delay lines grows as $1, 2, 4, ...2^{(logN)-2}$, generating an overall delay length of $N-1$. From algorithm A we know that: (1) the departure time of each packet from W is known upon its arrival, and (2) waiting time of each packet in W is always less than $N/2$.

Using the above facts, we develop algorithm B in the following way. When a packet P arrives to the system, and is scheduled by algorithm A to be placed in W then:

1. 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 i-th bit in the binary representation is non-zero, then the packet needs to traverse delay line D'_i.

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

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

Note that since the waiting time of each packet is known upon its arrival, the sequence of switch states in the next time slots can be determined at the time each packet arrives. Upon each arrival, this sequence is updated by the central scheduler. Therefore, the switching decisions are not required to be made exactly when packets arrive to the head of delay lines. More precisely, if a packet arrives at time t and the binary representation of its waiting time is $b_{(\log N)-1}...b_2b_1$, then for each nonzero bit b_i the switch of line D'_i will be closed twice; at time $t + \sum_{j=0}^{i-1} b_j 2^j$, to let the packet enter the delay line, and after the packet traverses the line, at time $t + \sum_{j=0}^{i} b_j 2^j$ to let it go to the next scheduled delay line.

Although the number of 2×2 switches in the proposed scheme is only $2 \log N$, the total length of delay lines used for exact emulation is $1.5N$. The suggested algorithm needs this extra $N/2$ length of delay line to ensure that packets are all placed in delay lines $\{D_i\}$ consecutively. As explained before, upon an arrival, if the prior packet is in the middle of a delay line, the scheduler waits until the prior packet is switched into the tail of some delay line. For this purpose, an extra length, equal to the length of the longest delay line, is needed to construct the waiting line W. Therefore, by reducing the length of the longest delay line, the extra length can be reduced too.

Figure 5.7 illustrates an example where the longest delay line is replaced by two half-sized delay lines. Consequently, the size of the waiting time W can be halved, resulting in only 0.25% extra length of fiber. This is in fact achieved at no expenses;

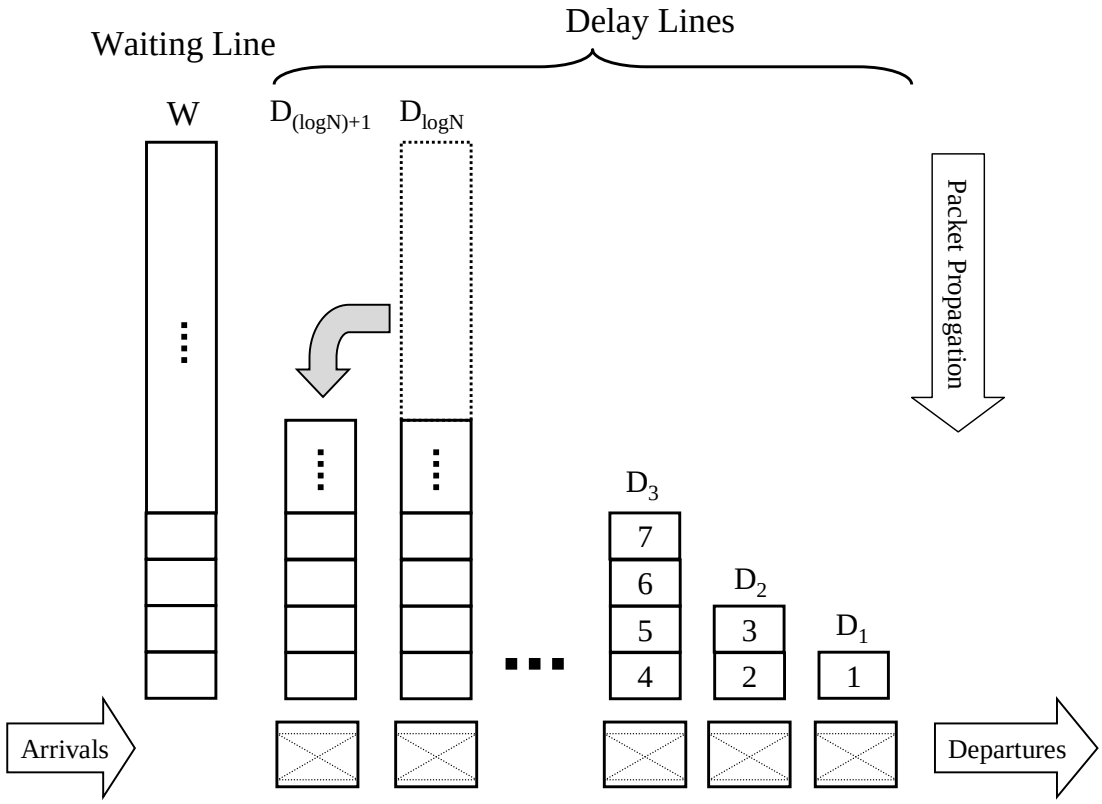

Figure 5.7: Trade-off between the number of delay lines and the maximum delay line length.

although one switch, and hence one delay line is added to the set of delay lines $\{D_i\}$, the size of W is halved, which means that one less switch is required for constructing W. It is easy to see that by repeating this procedure k times, the total number of switches added to the system would be $2^{k+1} - 2k - 2$, while the extra fiber length would be approximately $\frac{1}{2^{k+1}}$ that of the original scheme. By setting $K = 2^k$, the exact trade off can be stated as the following theorem.

Theorem 6. A FIFO queue of size $N - 1$, where N is a power of 2, can be emulated by $2(\log N + K - \log K) - 3$ delay lines with aggregate length of $N - 1 + \frac{N-2K}{2K}$, $K = 2^0, 2^1, ..., 2^{(\log N)-1}$.

5.3 Summary

In this chapter, we studied two implicit assumptions in our buffer sizing analysis, which are not necessarily realistic. The first assumption is that we have OQ routers. Even though analyzing such routers is easier than CIOQ router, they are not easy to build in practice. We showed how one can generalize buffer sizing results from OQ to CIOQ routers. We also studied the problem of building all-optical buffers. More specifically, we introduced a relatively simple architecture, which can be used to emulate FIFO queues by using all-optical switches and optical delay lines. The proposed structure, uses a simple mechanism to control the location of arriving optical packets by using only $O(\log N)$ 2x2 optical switches. In other words, the buffering capacity of the presented architecture grows exponentially with its size.

Chapter 6

Conclusion

In this dissertation, we have studied the problem of buffer sizing in Internet core routers. Through theoretical analysis, simulations, and real network experiments, we have examined the impact of small buffers and tiny buffers rules, which suggest two to six orders of magnitude reduction in buffer sizes in Internet core routers.

For the small buffers rule, the underlying assumptions, and thus the predictions appear to hold in laboratory and operational backbone networks – subject to the limited number of scenarios we can create. We are sufficiently confident in the small buffers result to conclude that it is probably time, and safe, to reduce buffers in backbone routers, at least for the sake of experimenting more in a fully operational backbone network.

In the tiny buffers context, our results suggest packet buffers in Internet core routers can be made much smaller; perhaps as small as 20-50 packets, if we are prepared to sacrifice some of the link capacity. It appears from simulation that the buffer size dictates directly how much link capacity is lost, however congested the network is. For example, a 40Gb/s link with 15 packet buffers could be considered to operate like a 30Gb/s link. Of course, this loss in link capacity could be eliminated by making the router run faster than the link-rate.

In a future network with abundant link capacity, this could be a very good trade-off: use tiny buffers so that we can process packets optically. In the past, it was

reasonable to assume that packet buffers were cheap, while long-haul links were expensive and needed to be fully utilized. Today, fast, large packet buffers are relatively painful to design and deploy; whereas link capacity is plentiful and it is common for links to operate well below capacity. This is even more so in an all-optical network where packet buffers are extremely costly and capacity is abundant.

The buffer size we propose depends on the maximum window size. Today, default settings in operating systems limit window size, but this limitation will probably go away over time. However, even if the maximum window size were to increase exponentially with time according to some form of "Moore's law", the buffer size would only need to increase linearly with time, which is a very benign scaling given recent technology trends.

Our results also assume that packets are sufficiently spaced out to avoid heavy bursts from one flow. Again, slow access links help make this happen. But if this is not true - for example, when two supercomputers communicate - the TCP senders can be modified to use Paced TCP instead.

The underlying assumptions in the tiny buffer size rule are more restrictive compared to the small buffers rule and might not hold in all Internet core networks. Our limited experiments in laboratories show as long as the theoretical constraints (*i.e.* non-bursty traffic as a result of pacing or slow access links) are valid tiny buffers result hold. However, we suspect that there are many more scenarios and boundary cases to consider before deciding whether it is time to reduce buffers to just 20-50 packets. In the mean time, more experiments are needed to better understand these results.

One of the major problems we encountered in buffer sizing experiments is that commercial routers do not allow buffer size adjustments with high accuracy, and are not able to provide precise buffer occupancy information. A simple way to address these issues in the future is to use NetFPGA-based 4x1GE routers [3]. The buffer sizes in these router can be adjusted with high accuracy at the packet or byte level. The router has the capability to collect accurate buffer occupancy data by recording every time a packet is written to and read from the buffer memory. None of the routers we know have these features, which are essential for buffer sizing, and we highly advise adding such features to future routers.

Appendix A

Proof of the Tiny Buffers Main Theorem

We will use the topology of Figure A.1. The capacity of the shared link (v, w), which is denoted by C, is assumed to be at least $(1/\rho) \cdot NW_{\max}/\text{RTT}$ where ρ is some constant less than 1. Hence, the network is over-provisioned by a factor of $1/\rho$, i.e. the peak throughput is ρC. The effective utilization, θ, is defined as the achieved throughput divided by ρC. We also assume that node v is an output queued switch, and has a buffer size of B.

The flow going through the link (v, w) is the superposition of the N long-lived TCP flows. Since the packet injection rate of the i-th flow is $W_i(t)/\text{RTT}$, its flow injection is dominated by a Poisson process of rate $W_{\max}/\text{RTT}$. More specifically, we can consider a virtual system in which flow i has an injection process which is Poisson and of rate $W_{\max}/\text{RTT}$. We can couple the packet injection processes in the real system and this virtual system such that the packets injected by the real system are a subset of the packets injected by the virtual system. Therefore, the aggregate of the N flows is dominated by a Poisson process of rate $NW_{\max}/\text{RTT}$.

Now, let us consider the output queue at node v. We assume this is a drop-tail queue, and prove the following lemma.

Lemma 5. The number of packet drops in the real system is less than or equal to the number of packet drops in the virtual system.

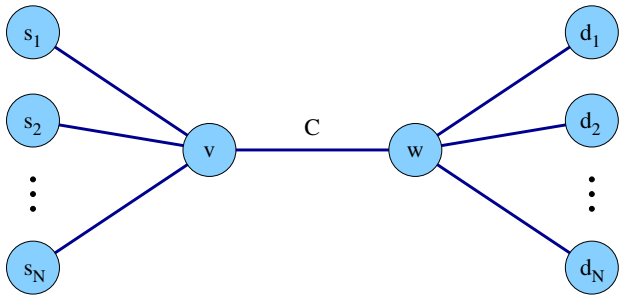

Figure A.1: Simplified topology of the network used in analysis.

At a given point of time t, let us denote the residual amount of data (queue occupancy plus part of the packet being served which has not left the system yet) in the real system with $Q_R(t)$ and the amount of data residing in the virtual system with $Q_V(t)$. We also denote the accumulative number of packet drops for the real system by $D_R(t)$ and the number of packet drops for the virtual system by $D_V(t)$. We claim that for any time t,

$$[Q_R(t) - Q_V(t)]^+ \leq (D_V(t) - D_R(t)). \tag{A.1}$$

Clearly this is true when both queues are empty at the beginning. Now we consider the following cases:

1. If we have no arrival and just time passes, the right hand side does not change, while the left hand side can only decrease or remain the same. This case also includes when packets depart either system.

2. If we have arrivals or drops at both queues at the same time, the inequality still holds.

3. If we have an arrival to the virtual system, and no arrivals to the real system no matter if we have a drop or not, the LHS doesn't increase, and the RHS might increase, which means the inequality still holds.

4. If we have an arrival to both of the queues and the real system drops the packet but the virtual system doesn't, we consider two cases. If $[Q_R(t) - Q_V(t)]^+ >= 1$,

then both sides go down by one unit. Otherwise, since the real system drops the packet but the virtual one doesn't, we can conclude that $Q_R(t) > Q_V(t)$ (t is the time right before this last arrival), which means $[Q_R(t) - Q_V(t)]^+$ is strictly greater than zero, and therefore $D_V(t)$ is greater than $D_R(t)$. Now, after the arrival and the drop, the LHS will become zero, while the RHS is greater than or equal to zero.

In all cases the inequality holds. Now, since $[Q_R(t) - Q_V(t)]^+$ is greater than or equal to zero (by definition), our inequality is translated to $D_V(t) >= D_R(t)$.

So far we have shown that the number of packet drops in the virtual system is more than the number of packet drops in the real system. The next step is to bound the number of drops in the virtual system. In this system the arrival process is Poisson. If we assume that the packets are all of the same size, the service time will be fixed. Therefore, we have an M/D/1 queue with a service rate of C, and arrival rate of ρC.

Lemma 6. The drop probability of the virtual system is bounded above by ρ^B.

The queue occupancy distribution of an M/D/1 FCFS queueing system is equal to that of an M/D/1 LCFS-PR (last-come first-served with preemptive resume) queue. This is because both queues have the same arrival process, are work conserving, and have the same service rate. Now, for any M/G/1 LCFS-PR system the steady state queue occupancy distribution is geometric with parameter ρ. Therefore, the drop probability of the M/G/1 LCFS-PR system equals ρ^B, which is an upper bound on the drop probability of the virtual system.

Note that this is not necessarily an upper bound on the packet drop probability of the real system.

Now that we have an upper bound on the packet loss probability of the virtual system, the next step is to find a lower bound on the throughput of the real system. Without loss of generality, we consider the dynamics of one of the flows, say flow number one. For simplicity, we assume that flow one is in congestion avoidance, *i.e.*, during each RTT the congestion window size is incremented by one if there is no packet loss, and the congestion window goes to zero if a packet loss is detected by

the source. Once the congestion window size reaches its maximum value (*i.e.* $W_{\max}$) it will remain fixed.

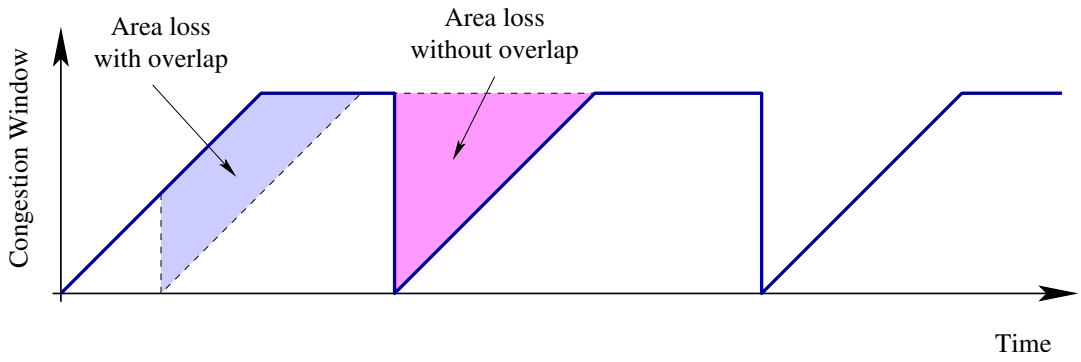

Figure A.2: Dynamics of the congestion window.

Figure A.2 depicts an example of the changes in congestion window size. The area under the curve indicates the total amount of data which has been sent by the flow. We can see that by each packet loss some portion of this area is lost, and the amount of loss is maximized when the overlap between the lost regions is minimum. We omit a formal proof. We are also ignoring slow-start in this system; it is not hard to see that considering slow-start can only lead to better bounds (i.e. smaller buffers)—again, we omit the formal proof.

Let us consider a long time interval of length Δ, and let us denote the number of packets injected by the sources in the virtual system during this interval with P_v, and the number of packet drops in the virtual system during this time interval with D_V. Choose an arbitrarily small $\epsilon > 0$. As Δ goes to ∞, we have:

$$Pr\left[P_v > \frac{\Delta N W_{\max}}{RTT}(1+\epsilon)\right] = o(1); \tag{A.2}$$

and,

$$Pr\left[P_v < \frac{\Delta N W_{\max}}{RTT}(1-\epsilon)\right] = o(1). \tag{A.3}$$

Since the probability of each packet being dropped is less than ρ^B, using Equation A.2, we can bound the total number of packet drops D as follows.

$$Pr\left[D_V > \frac{\rho^B \Delta N W_{\max}}{RTT}(1+\epsilon)\right] = o(1). \tag{A.4}$$

Based on Lemma 5 the number of packet drops in the virtual system is no less than the number of packet drops in the real system (henceforth denoted by D_R). Therefore, we get the following.

$$Pr\left[D_R > \frac{\rho^B \Delta N W_{\max}}{RTT}(1+\epsilon)\right] = o(1). \tag{A.5}$$

Now, if none of the flows in the real system encountered any losses during the time interval Δ, the amount of data that could have been sent during this time, U_T, can be bounded below as follows.

$$Pr\left[U_T < \frac{\Delta N W_{\max}}{RTT}(1-\epsilon)\right] = o(1). \tag{A.6}$$

We will lose some throughput as a result of packet drops in the system. As we can see in Figure A.2, the maximum amount of loss occurs when the triangles corresponding to packet losses have the minimum overlap. Therefore, we have

$$U_L \leq \frac{D_R W_{\max}^2}{2}. \tag{A.7}$$

In Equation A.5 we have bounded the number of packet losses in the real system with a high probability. Combining this bound, with Equation A.7 we get

$$Pr\left[U_L > \frac{\rho^B \Delta N W_{\max}^3}{2\text{RTT}}(1+\epsilon)\right] = o(1). \tag{A.8}$$

Now, if we want to guarantee an effective utilization throughput of θ, the following equation must hold.

$$\frac{U_T - U_L}{\rho C \Delta} >= \theta. \tag{A.9}$$

Since $\rho C = N W_{\max}/\text{RTT}$, we need to satisfy

$$U_L \leq N \Delta W_{\max}(1-\theta-\epsilon)/\text{RTT}. \tag{A.10}$$

Combining Equations A.6, A.8, and A.10 if we want to have a throughput of θ, we merely need to ensure

$$\frac{(1+\epsilon)\rho^B \Delta N W_{\max}^3}{2RTT} < \frac{\Delta N W_{\max}(1-\theta-\epsilon)}{RTT}, \tag{A.11}$$

which, in turn, is satisfied if the following holds:

$$\rho^B < \frac{2(1-\theta-O(\epsilon))}{W_{\max}^2}. \tag{A.12}$$

Since ϵ is arbitrarily small, it is sufficient for the buffer size B to satisfy

$$B \geq \log_{1/\rho}\left(\frac{W_{\max}^2}{2(1-\theta)}\right), \tag{A.13}$$

which is $O(\log W_{\max})$ since we assumed that ρ, θ are constants less than 1.

Appendix B

Pacing Analysis

In this Appendix we prove Theorem 2. We will consider the following discrete-time model of the packet arrivals at the bottleneck link during one RTT. There are a total $M = C \cdot RTT$ time slots, where C is the bandwidth of the bottleneck link. We assume that N flows will each send at most $W_{\max}$ packets, and that there are at least S time slots between consecutive packet arrivals of a single flow. The parameter S can be interpreted as a lower bound on the ratio of the bottleneck link speed to the access link speed. Note S is the crucial parameter in this section: when S is small traffic can be arbitrarily bursty and we cannot expect good throughput with small buffers (see also Section 3.6). We thus aim to prove that small buffers permit large throughput provided S is sufficiently large. Finally, we assume that the average traffic intensity $\rho = NW_{\max}/M$ is bounded below 1.

For the rest of this section, we adopt three assumptions.

(1) Buffers are sufficiently large: $B \geq c_B \log W_{\max}$, where $c_B > 0$ is a sufficiently large positive constant.

(2) The distance between consecutive packet arrivals of a single flow is sufficiently large: $S \geq c_S \log W_{\max}$, where $c_S > 0$ is a sufficiently large positive constant.

(3) Random jitter prevents a priori synchronization of the flows: flow start times are picked independently and uniformly at random from the M time slots.

As discussed in Section 3.4, the first two assumptions are often reasonable and are mathematically necessary for our results. The validity of the third assumption is less clear, especially in the presence of packet drops, which could increase the degree of synchronization among flows. Our simulations in Section 3.5 indicate, however, that our analytical bounds remain valid for long-lived flows that experience packet drops.

Our proof of Theorem 2 will focus on a particular (but arbitrary) packet. If the packet arrives during time slot t, then the probability that it is dropped is at most the probability that for some interval I of l contiguous time slots ending in time slot t, there were at least $l + B$ other packet arrivals. If this event occurs, we will say that the interval I is *overpopulated*. We will bound the probability of overpopulation, as a function of the interval length l, via the following sequence of lemmas. We first state the lemmas, then show how they imply Theorem 2, and finally prove each of the lemmas in turn.

The first lemma upper bounds the overpopulation probability for small intervals (of length at most $\log W_{max}$).

Lemma 7. In the notation above, if $l \leq \log W_{max}$, then the probability that the interval I is overpopulated is at most e^{-cB}, where $c > 0$ is a positive constant that depends only on c_B.

The second lemma considers intervals of intermediate size.

Lemma 8. In the notation above, if $\log W_{max} \leq l \leq SW_{\max}$, then the probability that the interval I is overpopulated is at most e^{-cS}, where $c > 0$ is a positive constant that depends only on c_S.

Finally, we upper bound the probability of overpopulation in large intervals.

Lemma 9. In the notation above, if $l \geq SW_{\max}$, then the probability that the interval I is overpopulated is at most $e^{-cl/W_{\max}}$, where $c > 0$ is a positive constant that depends only on c_S.

We now show how Lemmas 7–9 imply Theorem 2.

Proof of Theorem 2: We consider an arbitrary packet arriving in time slot t, and take the Union Bound over the overpopulation probabilities of all intervals that conclude with time slot t. First, by Lemma 7, the total overpopulation probability for small intervals (length l at most $\log W_{max}$) is $O(e^{-B} \log W_{max})$, which is $O(1/W_{\max}^2)$ provided c_B (in Assumption (1)) is sufficiently large. Next, Lemma 8 implies that the total overpopulation probability of intervals with length l in $[\log W_{max}, SW_{\max}]$ is at most $SW_{\max}e^{-\Omega(S)}$, which is $O(1/W_{\max}^2)$ provided c_S (in Assumption (2)) is sufficiently large. Finally, Lemma 9 implies that the total overpopulation probability of large intervals ($l \geq SW_{\max}$) is at most $\int_{SW_{\max}}^{\infty} e^{-\Omega(x/W_{\max})}dx$. Changing variables ($z = x/W_{\max}$), this quantity equals $W_{\max}\int_S^{\infty} e^{-z}dz$, which is $O(1/W_{\max}^2)$ provided c_S is sufficiently large. Taking the Union Bound over the three types of intervals, we obtain an upper bound of $O(1/W_{\max}^2)$ for the total overpopulation probability, and hence for the probability that the packet is dropped. This completes the proof.

Proof of Lemma 7: If the length l of interval I is at most $\log W_{max}$, then each flow contributes at most 1 packet to I (assuming that $c_S \geq 1$). This occurs with probability at most $W_{\max}l/M$. Let X_i denote the corresponding indicator random variable for flow i, and define $X = \sum X_i$. Note that $EX \leq \rho l \leq \rho \log W_{max}$. We use the following Chernoff bound (see e.g. [38]) for a sum of indicator random variables with expectation μ: $Pr[X \geq (1+\delta)\mu] < [e^{\delta}/(1+\delta)^{(1+\delta)}]^{\mu}$. Assuming that c_B is sufficiently large (in Assumption (1)), setting $(1+\delta)\mu = B$ gives $Pr[X \geq l + B] \leq Pr[X \geq B] \leq e^{-\Theta(B)}$.

Proof of Lemma 8: Suppose $(k-1)S \leq l \leq kS$ for some $k \in \{1, 2, ..., W_{\max}\}$. Then, each flow contributes at most k packets to I. Assume for simplicity that each flow contributes either 0 or k packets to I, with the latter event occurring with probability $W_{\max}S/M$ (so that $E[kX] \approx \rho l$). (Here X_i is the indicator for the latter event, so the total number of arrivals in I is at most kX, where $X = \sum_i X_i$.) A more accurate analysis that permits each flow to contribute any number of packets between 0 and k to I (with appropriate probabilities) can also be made, but the results are nearly identical to those given here.

We use the same Chernoff bound as in the proof of Lemma 7. We are interested in the probability $Pr[X \geq (l+B)/k]$, which we will upper bound by $Pr[X \geq l/k]$. Since $\mu = \Theta(\rho l/k)$, the Chernoff bound gives $Pr[X \geq l/k] \leq \exp\{-\Theta(l/k)\} = e^{-\Theta(S)}$

for fixed $\rho < 1$.

Proof of Lemma 9: The proof is similar to the previous one. Suppose that $l \geq W_{\max}S$ and assume that each flow i contributes either 0 or $W_{\max}$ packets to I, the latter event occurring with probability l/M. (So $E[W_{\max}X] = \rho l$.) The same Chernoff bound argument as in the proof of Lemma 8 gives $Pr[W_{\max}X >= l + B] \leq e^{-\Theta(l/W_{\max})}$ for fixed $\rho < 1$.

Appendix C

All-optical Buffering

Assume that there is no packet in the system at time 0, and that at each time slot $0 \leq \tau \leq t$ at most one packet is scheduled to be switched into delay line D_i, $i = 1, ..., \log N$. We prove that the same holds at time $t + 1$.

To prove this, we first show that if we assume that lemma 2 holds up to time t, then for any two consecutive packets p and p', with p' immediately following p in the departing order, the followings also hold for any $\tau \leq t$:

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

(‡)

$$l'(\tau) - l(\tau) = 1 \mod d(\tau) \tag{C.1}$$

where $l(\tau)$ and $l'(\tau)$ are the locations of packets p and p' respectively, and $d(\tau)$ is the length of the delay line containing packet p.

The assumption that Lemma 2 is valid until time t ensures that scheduling one packet has no effect on other packets, *i.e.*, when a packet is to be transferred to the tail of a delay line, that location is guaranteed to be empty. Hence, there is no packet drop caused by the scheduling of head of line packets. A packet changes line only when its priority gets smaller than the length of the current line. Therefore, a particular packet only moves upward, from longer delay lines to shorter ones, and

hence,

$$\begin{aligned} l(\tau) - l(\tau+1) &= 1 \mod d(\tau+1) \\ l'(\tau) - l'(\tau+1) &= 1 \mod d'(\tau+1). \end{aligned} \tag{C.2}$$

Now assume that † and ‡ are valid for some $\tau < t$. If at time τ packet p' is not at the head of some delay line, then clearly † holds at $\tau+1$ too. If at time τ packet p' is at the head of a delay line, then ‡ implies that at τ packet p is located at the tail of some delay line, or equivalently, at $\tau-1$ packet p has been at the head of a delay line. But there can be at most one departure from the system at time $\tau-1$. As a result, priority of packet p' at time τ will be greater than or equal to that of packet p at time $\tau-1$. Therefore, at time $\tau+1$ packet p will be located either in the same line where p is, or in a longer line, and hence, † holds at time $\tau+1$ too. To show that ‡ also holds at $\tau+1$, we need to keep in mind that the length of a delay line is divisible by the length of any shorter delay line. More specifically, $d'(\tau+1)$ is a multiple of $d(\tau+1)$ in Equation 2, and therefore, $l'(\tau+1) - l(\tau+1) = l'(\tau) - l(\tau)$ mod $d(\tau+1)$. Validity of ‡ at time $\tau+1$ immediately follows from this equation, and the fact that $d(\tau+1)$ is a multiple of $d(\tau)$.

To show that lemma 2 holds at time $t+1$, 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 $d_a < d_b$. Since † holds at time t, we know that packet a has a smaller priority than packet b. Also ‡ implies that the difference between the priorities is at least d_a. Note that if packet a is scheduled to be in a line with length d_0 at next time slot, then its priority is greater than or equal to $d_0 \leq d_a$. Therefore, the priority of packet b is greater than or equal to $d_0 + d_a \geq 2d_0$, and hence packet b will not compete for the same delay line.

Bibliography

[1] Cisco GSR routers. http://atantica.com/product/cisco/routers/12000.html.

[2] IP Monitoring Project. http://ipmon.sprint.com/.

[3] NetFPGA. http://yuba.stanford.edu/NetFPGA/.

[4] The network simulator – *ns2*. http://www.isi.edu/nsnam/ns/.

[5] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *Proceedings of the IEEE INFOCOM*, pages 1157–1165, Tel-Aviv, Israel, March 2000.

[6] G. Appenzeller. *Sizing Router Buffers*. PhD thesis, Stanford University, Department of Computer Science, March 2005.

[7] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of the ACM SIGCOMM*, pages 281–292, September 2004.

[8] N. Beheshti and Y. Ganjali. Packet scheduling in optical FIFO buffers. Technical Report TR06-HPNG-09-09-00, Stanford University, 2006.

[9] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. ck McKeown. Buffer sizing in all-optical packet switches. In *Proceedings of OFC/NFOEC*, Anaheim, CA, USA, March 2006.

[10] D. P. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, second edition, 1992.

[11] R. Bush and D. Meyer. RFC 3439: Some Internet architectural guidelines and philosophy, December 2002.

[12] J. Cao, W. Cleveland, D. Lin, and D. Sun. Internet traffic tends to Poisson and independent as the load increases. Technical report, Bell Labs, 2001.

[13] C. S. Chang, Y. T. Chen, and D. S. Lee. Construction of optical FIFO queues. *IEEE Transactions on Information Theory*, 52(6):2838–2843, June 2006.

[14] C. S. Chang, D. S. Lee, and C. K. Tu. Recursive construction of FIFO optical multiplexers with switches delay lines. *IEEE Transactions on Information Theory*, 50(12):3221–3233, December 2004.

[15] S. T. Chuang, A. G. N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. In *Proceedings of the IEEE INFOCOM*, pages 1169–1178, New York, NY, USA, March 1999.

[16] R. L. Cruz and J. T. Tasai. COD: alternative architectures for high speed packet switching. *IEEE/ACM Transactions on Networking*, 4(1):11–20, February 1996.

[17] A. Dhamdhere and C. Dovrolis. Open issues in router buffer sizing. *ACM/SIGCOMM Computer Communication Review*, 36(1):87–92, January 2006.

[18] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden. Part III: Routers with very small buffers. *ACM/SIGCOMM Computer Communication Review*, 35(3):83–90, July 2005.

[19] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden. Routers with very small buffers. In *Proceedings of the IEEE INFOCOM*, Barcelona, Spain, April 2006.

[20] A. Erramilli, O. Narayan, A. Neidhardt, and I. Saniee. Performance impacts of multi-scaling in wide area TCP/IP traffic,. In *Proceedings of the IEEE Infocom*, Tel-Aviv, Israel, March 2000.

[21] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA, USA, August 1999.

[22] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[23] C. Fraleigh, F. Tobagi, and C. Diot. Provisioning IP backbone networks to support latency sensitive traffic. In *Proceedings of the IEEE INFOCOM*, San Francisco, CA, USA, April 2003.

[24] C. J. Fraleigh. *Provisioning Internet Backbone Networks to Support Latency Sensitive Applications*. PhD thesis, Stanford University, Department of Electrical Engineering, June 2002.

[25] Y. Ganjali and N. McKeown. Experimental study of router buffer sizing. Technical Report TR06-HPNG-07-30-00, Stanford University, July 2006.

[26] A. Gilbert, Y. Joo, and N. McKeown. Congestion control and periodic behavior. In *Proceedings of LANMAN Workshop*, Boulder, CO, USA, March 2001.

[27] D. K. Hunter, M. C. Chia, and I. Andonovic. Buffering in optical packet switches. *Journal of Lightwave Technology*, 16(12):2081–2094, December 1998.

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

[29] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot. An approach to alleviate link overload as observed on an IP backbone. In *Proceedings of the IEEE INFOCOM*, San Francisco, CA, USA, March 2003.

[30] V. Jacobson. [e2e] Re: Latest TCP measurements thoughts. Posting to the end-to-end mailing list, March 7, 1988.

[31] V. Jacobson. Congestion avoidance and control. *ACM Computer Communications Review*, 18(4):314–329, August 1988.

[32] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, Chichester, 1979.

[33] V. Lal, J. A. Summers, M. L. Masanovic, L. A. Coldren, and D. J. Blumenthal. Novel compact InP-based monolithic widely-tunable differential Mach-Zehnder interferometer wavelength converter for 40Gbps operation. In *Indium Phosphide and Related Materials*, Scotland, 2005.

[34] M. L. Masanovic, V. Lal, J. S. Barton, E. J. Skogen, J. A. Summers, L. Rau, L. A. Coldren, and D. J. Blumenthal. Widely-tunable monolithically-integrated all-optical wavelength converters in InP. *Journal of Lightwave Technology*, 23(3), March 2005.

[35] Microsoft. TCP/IP and NBT configuration parameters for Windows XP. Microsoft Knowledge Base Article - 314053, November 4, 2003.

[36] R. Morris. TCP behavior with many flows. In *Proceedings of the IEEE International Conference on Network Protocols*, Atlanta, GA, USA, October 1997.

[37] R. Morris. Scalable TCP congestion control. In *Proceedings of the IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

[38] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[39] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM*, pages 303–314, Vancouver, BC, Canada, September 1998.

[40] H. Park, E. F. Burmeister, S. Bjorlin, and J. E. Bowers. 40-Gb/s optical buffer design and simulations. In *Numerical Simulation of Optoelectronic Devices (NUSOD)*, Santa Barbara, CA, USA, 2004.

[41] G. Raina, D. Towsley, and D. Wischik. Part II: Control theory for buffer sizing. *ACM/SIGCOMM Computer Communication Review*, 35(3):79–82, July 2005.

[42] G. Raina and D. Wischik. Buffer sizes for large multiplexers: TCP queueing theory and instability analysis. In *EuroNGI*, Rome, Italy, April 2005.

[43] R. Rajaduray. *Unbuffered and Limited-Buffer All-Optical Networks*. PhD thesis, University of California, Santa Barbara, Department of Electrical and Computer Engineering, August 2005.

[44] K. Ramanan and J. Cao. A poisson limit for buffer overflow probabilities. In *Proceedings of the IEEE INFOCOM*, New York, NY, USA.

[45] V. Ribeiro, R. Riedi, M. Crouse, and R. Baraniuk. Multiscale queuing analysis of long-range dependent network traffic. In *Proceedings of the IEEE Infocom*, Tel-Aviv, Israel, March 2000.

[46] A. D. Sarwate and V. Anantharam. Exact emulation of a priority queue with a switch and delay lines. *Queueing Systems: Theory and Applications*, 53(3):115–125, July 2006.

[47] W. R. Stevens. *TCP/IP Illustrated, Volume 1 - The Protocols*. Addison Wesley, 1994.

[48] C. Villamizar and C. Song. High performance TCP in ANSNET. *ACM Computer Communications Review*, 24(5):45–60, October 1994.

[49] V. Visweswaraiah and J. Heidemann. Rate based pacing for TCP. http://www.isi.edu/lsam/publications/rate_based_pacing/, 1997.

[50] L. Zhang and D. D. Clark. Oscillating behaviour of network traffic: A case study simulation. *Internetworking: Research and Experience*, 1(2):101–112, 1990.