SIZING ROUTER BUFFERS

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

Guido Appenzeller
March 2005

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.

Balaji Prabhakar

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.

Dawson Engler

Approved for the University Committee on Graduate Studies.

Abstract

All Internet routers contain buffers to hold packets during times of congestion. Today, the size of the buffers is determined by the dynamics of TCP's congestion control algorithms. To keep link utilization high, a widely used rule-of-thumb states that each link needs a buffer of size $B = \overline{RTT} \times C$, where $\overline{RTT}$ is the average round-trip time of a flow passing across the link, and C is the data rate of the link. For example, a 10Gb/s router linecard needs approximately 250ms $\times$ 10Gb/s = 2.5Gbits of buffers; and the amount of buffering grows linearly with the line-rate. Such large buffers are challenging for router manufacturers, who must use large, slow, off-chip DRAMs. And queueing delays can be long, have high variance, and may destabilize the congestion control algorithms.

In this thesis we argue that the rule-of-thumb ($B = \overline{RTT} \times C$) is now outdated and incorrect for backbone routers. This is because of the large number of flows (TCP connections) multiplexed together on a single backbone link. Using theory, simulation and experiments on a network of real routers, we show that a link with n long lived flows requires no more than $B = (\overline{RTT} \times C)/\sqrt{n}$, for long-lived or short-lived TCP flows.

The consequences on router design are enormous: A 2.5Gb/s link carrying 10,000 flows could reduce its buffers by 99% with negligible difference in throughput; and a 10Gb/s link carrying 50,000 flows requires only 10Mbits of buffering, which can easily be implemented using fast, on-chip SRAM.

Acknowledgements

Writing a thesis is not possible without the advice, support and feedback from researchers, collaborators, colleagues and friends.

First and foremost I would like to acknowledge Nick McKeown. He not only had an incredible instinct for suggesting a thesis topic that is both relevant to practitioners and academically deep, he also mentored me through the process with critical feedback and advice. It is hard to imagine a better advisor than Nick. I am very grateful to him.

Isaac Keslassy was an invaluable collaborator and great fun to work with. He kept asking the right questions at the right time, and provided major value with his superiour mathematical background.

Matthew Holliman helped me with the early experiments and deserves credit for being the first to suggest to look for a CLT based model. Many thanks go to Yashar Ganjali, Sundar Iyer, Shang-Tse Chuang, Martin Casado, Rui Zhang and Nandita Dukapi, Greg Watson, Ichiro Okajima and everyone else in the HPNG for discussions of my results while they were being developed.

The experiemnts on the GSR would not have been possible without the help of Joel Sommers and Paul Barford. Their expertise in setting up experiments with physical routers was a huge help.

Balaji Prabhakar and members of his research group gave valuable feedback on early results. I would also like to thank the other members of my orals comittee Dawson Engler, Nick Bambos and David Cheriton for their hard questions, they helped refine the written dissertation.

Early talks with Sally Floyd and Frank Kelly helped identify the issues and the existing knowledge in buffer research. Their feedback on the final results was a major step towards this work's conclusions. I would also like to thank the reviewers of the SIGCOMM article and specifically Craig Partridge for their valuable feedback.

Sunya Wang, Wayne Sung and Lea Roberts from the Stanford Networking team were extremely helpful in implementing the experiments on the Stanford Network and lending me equipment to get an understanding of routers. Many thanks go to them.

The team and investors at Voltage made it possible for me to complete my Ph.D. in parallel, I am very grateful for their patience.

Last but not least my thanks to Isabelle, who supported me in good and bad times, and was a calming influence when the Ph.D. stress was threatening to get the upper hand.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Internet routers are packet switches and employ buffers to hold packets during times of congestion. Arguably, router buffers are the single biggest contributor to uncertainty in the Internet. Buffers cause queueing delay and delay-variance. When they overflow, they cause packet loss, and when they underflow, they can degrade throughput. Given the significance of their role, we might reasonably expect the dynamics and sizing of router buffers to be well understood, based on a well-grounded theory, and supported by extensive simulation and experimentation. This is not so.

Router buffers are sized today based on a rule-of-thumb commonly attributed to a 1994 paper by Villamizar and Song [43].[1] Using experimental measurements of up to eight TCP flows on a 40 Mb/s link, they concluded that a router needs an amount of buffering equal to the round-trip time of a typical flow that passes through the router, multiplied by the capacity of the router's network interfaces. This is the well-known $B = \overline{RTT} \times C$ rule. We will show later that for a small number of long-lived TCP flows, the rule-of-thumb makes sense.

Network operators follow the rule-of-thumb and require router manufacturers to provide 250ms (or more) of buffering [38]. The rule is also found in architectural guidelines [9]. Requiring such large buffers complicates router design and is a big

[1] While attributed to this paper, it was already known to the inventors of TCP [25]

impediment to building routers with larger capacity. For example, a 10Gb/s router line card needs approximately $250ms \times 10Gb/s = 2.5Gbits$ of buffers; and the amount of buffering grows linearly with the line-rate.

Given the effect of buffer size on network performance and router design, it is worth asking if the rule-of-thumb still holds. Today, backbone links commonly operate at 2.5Gb/s or 10Gb/s and carry well over 10,000 flows [18]. Quantitatively, current backbone links are very different from 1994 and we might expect them to be qualitatively different too. It seems fair to say that it is currently not well understood how much buffering is actually needed or how buffer size affects network performance [15].

It is worth asking why we care to size router buffers accurately. With declining memory prices, why not just over-buffer routers and not run the risk of losing link utilization? We believe over-buffering is a bad idea for two reasons. First, it has architectural implications on high-speed routers, leading to designs that are more complicated, consume more power, and occupy more board space. Second, over-buffering increases end-to-end delay in the presence of congestion. This is especially the case with TCP, as a single TCP flow in the absence of other constraints will completely fill the buffer of a bottleneck link. In such a case, large buffers conflict with the low-latency needs of real time applications (e.g. video games, and device control). In some cases, large delays can make congestion control algorithms unstable [28] and applications unusable.

1.1.1 Buffer Size and Router Design

At the time of writing, a state of the art router line card runs at an aggregate rate of 40Gb/s (with one or more physical interfaces), has about 250ms of buffering, and so has 10Gbits (1.25Gbytes) of buffer memory.

Buffers in backbone routers are built from commercial memory devices such as dynamic RAM (DRAM) or static RAM (SRAM).[2] The largest commercial SRAM chip today is 32Mbits and consumes about 250mW/Mbit, which means a 40Gb/s line

[2]DRAM includes devices with specialized I/O, such as DDR-SDRAM, RDRAM, RLDRAM and FCRAM.

card would require more than 300 chips and consume 2.5kW, making the board too large, too expensive and too hot. If instead we try to build the line card using DRAM, we would need ten devices consuming only 40W. This is because DRAM devices are available up to 1Gbit and consume only 4mW/Mbit. However, DRAM has a random access time of about 50ns, which is hard to use when a minimum length (40byte) packet can arrive and depart every 8ns. Worse still, DRAM access times fall by only 7% per year [21]; thus, the problem is going to get worse as line-rates increase in the future.

In practice, router line cards use multiple DRAM chips in parallel to obtain the aggregate data-rate (or memory-bandwidth) they need. Packets either are scattered across memories in an ad-hoc statistical manner, or use an SRAM cache with a refresh algorithm [24]. Either way, a large packet buffer has a number of disadvantages: it uses a very wide DRAM bus (hundreds or thousands of signals) with a large number of fast data pins (network processors and packet processor ASICs frequently have more than 2,000 pins making the chips large and expensive). Such wide buses consume large amounts of board space and the fast data pins on modern DRAMs consume too much power.

In summary, it is extremely difficult to build packet buffers at 40Gb/s and beyond. Given how slowly memory speeds improve, this problem is going to get worse over time.

Substantial benefits could be gained from using significantly smaller router buffers, particularly if it was possible to use SRAM. For example, it would be feasible to build a packet buffer using about 512Mbits of off-chip SRAM (16 devices). If buffers of 5% of the delay-bandwidth product were sufficient, we could use SRAM to build buffers for a 40 Gb/s line card.

The real opportunity, however, lies in placing the memory directly on the chip that processes the packets (a network processor or an ASIC). In this case, very wide and fast access to a single memory is possible. Commercial packet processor ASICs have been built with 256Mbits of "embedded" DRAM. If memories of 2% the delay-bandwidth product were acceptable, then a single-chip packet processor would need no external memories. We will present evidence later that buffers this small would make little or no difference to the utilization of backbone links today.

1.1.2 Buffer Size and Latency

There is a second key argument for smaller buffers: buffers add queueing delay and thus increase latency of the Internet. Additional latency is undesirable for short flows and interactive applications. Excessive over-buffering can make some of these applications unusable in the case of congestion.

If a router's link is congested, the router's buffer will be filled most of the time. This introduces additional queueing delay in the order of $T_Q = \frac{B}{C}$. If the router buffer is sized using the rule-of-thumb, this additional queueing delay is

$$T_Q = \frac{\overline{RTT} \times C}{C} = \overline{RTT}.$$

An additional queueing delay of one RTT means that in the case of congestion, a router that is sized using the rule-of-thumb will double the latency of any flow going through it. If there are several points of congestion on a flow's path, each congested router will incur an additional $\overline{RTT}$ worth of queueing delay.

Additional latency affects real-time applications such as online gaming, IP telephony, video conferencing, remote desktop or terminal based applications. IP telephony typically requires a round-trip latency of less than 400ms. For competitive online gaming, latency differences of 50ms can be decisive. This means that a congested router that is buffered using the rule-of-thumb will be unusable for these applications. This is the case even if the loss rate of the router is still very small. A single, congestion-aware TCP flow that attempts to "just fill" the pipe will be sufficient to make online gaming or IP telephony on the router impossible.

ISP's routers are typically substantially over-buffered. For example, we measured up to five seconds of queueing delay at one router, making web surfing extremely cumbersome and any interactive applications all but impossible. The packet loss rate, however, was still well below 1%. With smaller buffers, this router could have remained usable in this case of congestion. An ISP using such an overbuffered router, has no choice but to overprovision its network, as congestion would immediately lead to customer complaints. Thus overbuffering indirectly is a key contributing factor to the low utilization of the internet today.

To summarize, over-buffering is harmful and can make routers unusable for interactive applications, even with minimal congestion. By reducing buffers substantially, we could almost halve the latency and latency jitter of a congested link.

1.2 Previous Work

There is surprisingly little previous work on sizing router buffers. Villamizar and Song report the $RTT \times BW$ rule in [43], in which the authors measure link utilization of a 40 Mb/s network with 1, 4 and 8 long-lived TCP flows for different buffer sizes. They find that for a drop-tail queue and very large maximum advertised TCP congestion windows, it is necessary to have buffers of $RTT \times C$ to guarantee full link utilization. We reproduced their results using $ns2$ [1] and can confirm them for the same setup. With such a small number of flows and large congestion windows, the flows are almost fully synchronized and have the same buffer requirement as a single flow.

Morris [30] investigates buffer requirements for up to 1500 long-lived flows over a link of 10 Mb/s with 25ms latency. He concludes that the minimum amount of buffering needed is a small multiple of the number of flows, and points out that for a bandwidth-delay product of 217 packets, each flow has only a fraction of a packet in transit at any time. Many flows are in timeout, which adversely effects utilization and fairness. We repeated the experiment in $ns2$ and obtained similar results. However, for a typical router used by a carrier or ISP, this result has limited implications. Users with fast access links will need several packets outstanding to achieve adequate performance. Users with slow access links (e.g. 32kb/s modem users or 9.6kb/s GSM mobile access) have a small bandwidth delay product and need additional buffers in the network to avoid excessive timeouts. These buffers should be at each end of the access link, e.g. the modem bank at the local ISP or GSM gateway of a cellular carrier, to buffer packets waiting to corss the slow link. We believe that placing these extra buffers in the core would be the wrong approach, as over-buffering increases latency for everyone — with fast and slow access links alike. It is also harder and more expensive to buffer at high line-rates. Instead, the access devices that serve slow, last-mile access links of less than 1Mb/s should continue to include about 7 packets per flow worth of buffering for each link. With line speeds increasing and the MTU size staying constant, we would also assume this issue to become less relevant in the future.

Avrachenkov et al [6] present a fixed-point model for utilization (for long flows) and flow completion times (for short flows). They model short flows using an M/M/1/K model that accounts for flows but not for bursts. In their long flow model, they use an analytical model of TCP that is affected by the buffer through the RTT. As the model requires fixed-point iteration to calculate values for specific settings and only one simulation result is given, we cannot compare their results directly with ours.

1.3 Organization of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, we will analyze a single flow through a router to understand the basics of TCP buffer interaction. It will show that the rule-of-thumb comes from the dynamics of TCP's congestion control algorithm. The buffer size is determined by the multiplicative decrease behavior of TCP. In Chapter 3, we present our model for flows in congestion avoidance mode and find that buffer requirements in this case are much smaller than predicted by the rule-of-thumb. Chapter 4 presents a model for short flows in slow-start mode and also concludes that small buffers usually suffice. The model is mainly of interest on uncongested routers and allows us to model TCP as well as other protocols. Chapters 5 and 6 contain extensive experimental verification of our findings using *ns*2, and experiments on physical routers, including experiments with traffic on live operational networks. We not only confirm and test the limits of our model, but also test the effect of a number of network parameters on buffer requirements. Chapter 7 offers our conclusion.

Chapter 2

A Single TCP Flow Through a Router

We start by modeling how a single TCP flow interacts with a router. Doing so will not only show where the rule-of-thumb for buffer sizing comes from, it will also give us the necessary tools to analyze the multi-flow case in Chapter 3. In Sections 2.1 and 2.2 we examine the case of a router that has the right amount of buffering vs. a router that has too much or too little buffering. This will confirm that the rule-of-thumb does hold for a single flow through a router. We then formally prove the rule-of-thumb using two different methods. In Section 2.3 we will do it based on rates and in Section 2.4 by accounting for outstanding packets.

Consider the topology in Figure 2.1 with a single sender and one bottleneck link. The sender is sending an unlimited amount of data using TCP. We define:

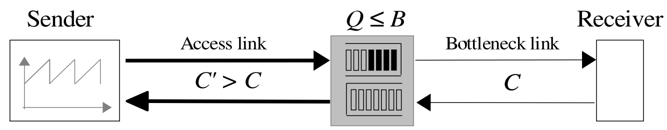


Figure 2.1: Topology for a Single TCP Flow

W	The TCP Window Size of the sender
T_P	The propagation delay from sender to receiver
RTT	The Round-Trip-Time as measured by the sender
C	The capacity of the bottleneck link
C'	The capacity of the access link
R	The sending rate of the sender
U	The link utilization measured on the link
Q	The length of the buffer queue
B	The buffer size, $Q \leq B$

The TCP sending rate is controlled by the congestion window W (for a brief summary of how TCP's congestion control algorithm works see Appendix A.1).

For this experiment, we assume that there is no congestion on the reverse path and that the capacity of the access link is higher than the capacity of the bottleneck link $C' > C$. We also assume that the window size and the sending rate of the TCP flow are not limited.

For simplicity, we will express data (Q, B, W) in packets and rates (U, R) in packets per second. This is a simplification as TCP effectively counts bytes and packets might have different lengths. Buffers of real routers may be organized as packets or smaller units (see Section 6.2), however, in practice, a flow sending at the maximum rate will behave close to this simplified model as it will primarily generate packets of the MTU size.

The RTT that a flow experiences is the two-way propagation delay, plus the queueing delay T_Q from the router queue:

$$RTT = 2T_p + T_Q = 2T_p + \frac{Q}{C}.$$

The sending rate of the TCP flow is well known to be the window size divided by the round trip time [42]:

$$R = \frac{W}{RTT} = \frac{W}{2T_p + T_Q} = \frac{W}{2T_p + \frac{Q}{C}}. \tag{2.1}$$

2.1 Router Buffers of Size $2T_p \times C$

2.1.1 Macroscopic Behavior

The first question we want to answer is the origin of the rule-of-thumb. The original paper [43] looks at how to adjust TCP and router buffers in order to get full link utilization. It finds experimentally that you need to do one of two things, either adjust the maximum window size W_{max} or have buffers of at least $2T_p \times C$ in the router. Our goal is to find out why this is the case.

To gain an understanding how buffers and TCP interact, we measured the window size W, queue length Q, RTT, rate R and utilization U of the TCP flow through a router with buffers of $2T_p \times C$. The result is shown in Figure 2.2. As expected, the window size of the TCP flow (after an initial slow-start phase) follows the familiar, slightly rounded TCP sawtooth pattern. In A.2, we present an analytical fluid model that provides a closed-form equation of the sawtooth, and closely matches the $ns2$ simulations.

The utilization is 100% (i.e. equals the link rate), which we would expect, as we used "sufficiently large" buffers according to the rule-of-thumb. Notice that the sending rate of the TCP sender is constant, except for a short drop every time the window scales down. Looking at Equation 2.1, we might have expected that if the window size fluctuates according to the sawtooth pattern, the sending rate would fluctuate as well.

The reason is as follows. In our experiment, the bottleneck link is always fully

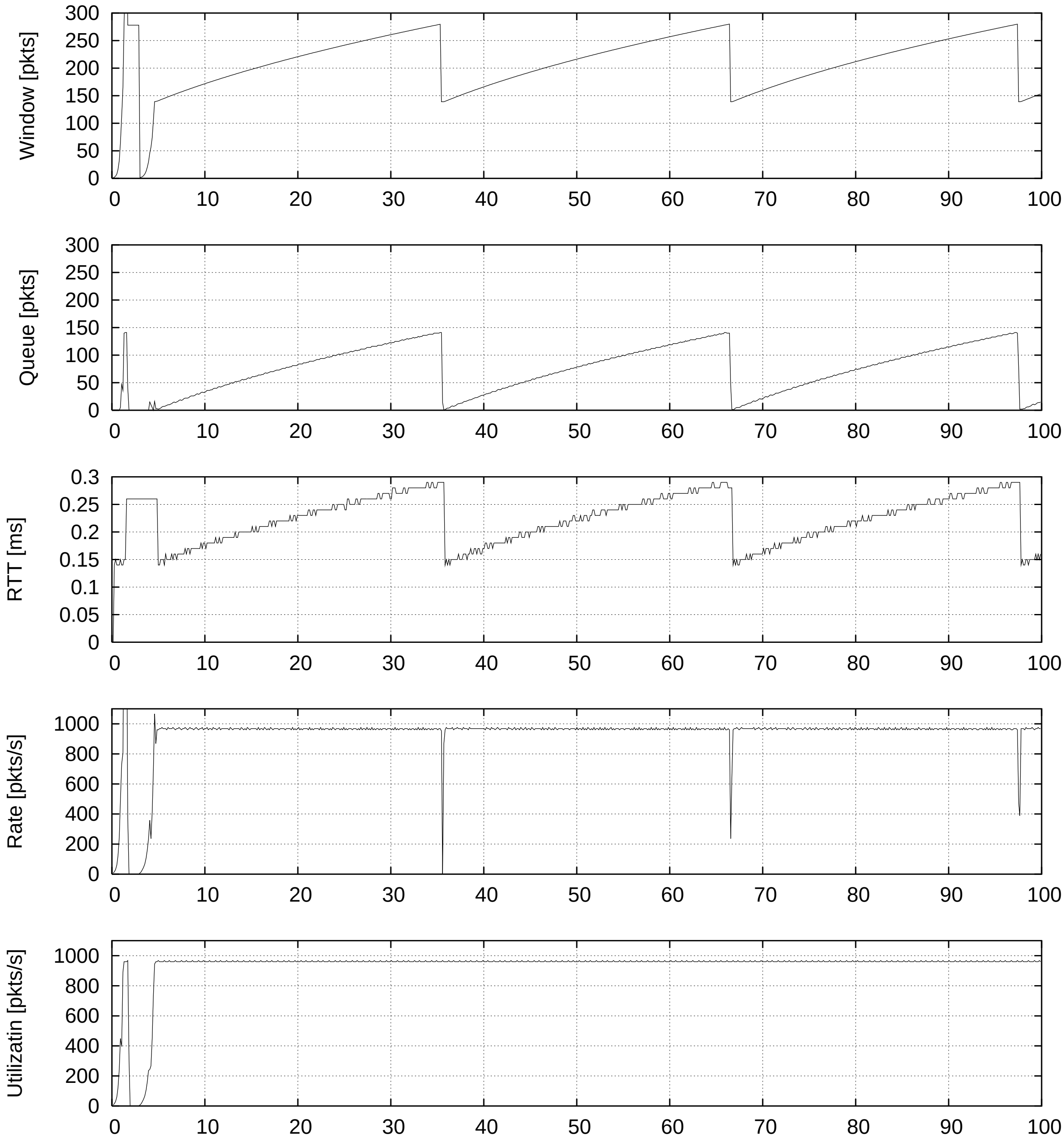


Figure 2.2: A single TCP flow through a single router with buffers equal to the delay-bandwidth product (142 packets).

utilized. This means that the router is sending packets at a rate $C \approx 1000$ packets per second. These packets arrive at the receiver which in turn sends ACK packets at the same rate of 1000 packets/s. The sender receives these ACK packets and will, for each packet received, send out a new data packet at a rate R that is equal to C, i.e. the sending rate is clocked by the bottleneck link. (There are two exceptions to this, when packets are dropped and when the window size is increased. If a packet was dropped and a sequence number is missing, the sender may not send new data. We'll treat this case below.) Once per RTT, the sender has received enough ACK packets to increase its window size by one. This will cause it to send exactly one extra packet per RTT. For this work we will treat this extra packet seperately, and not consider it as part of the sending rate. To summarize, for a sufficiently buffered router, a TCP sender sends (not counting the extra packet from a window increase) at a constant rate R that is equal to C:

$$R = C \tag{2.2}$$

That this is the case in practice can be easily confirmed by looking at Figure 2.2. This is compatible with our with our Rate Equation 2.1. The RTT follows a sawtooth pattern that matches that of the window size and at any given time we have

$$W \sim RTT$$

The reason for this can be seen by looking at the queue length Q in Figure 2.2. The router receives packets from the sender at a rate of $R = C$, and drains its queue at the same rate C. If W is increase by one, this causes an extra packet to be sent that increases Q by one. As $RTT = 2T_p + T_Q = 2T_p + \frac{Q}{C}$ this the increase of W by one will cause an increase of RTT and $\frac{W}{RTT}$ is constant.

2.1.2 Microscopic Behavior

Figure 2.3 shows what happens at a smaller time scale, specifically what happens when the TCP flow scales back its congestion window. Around $t = 35.15$, the router buffer overflows and a packet is dropped. This drop information (in the form of a

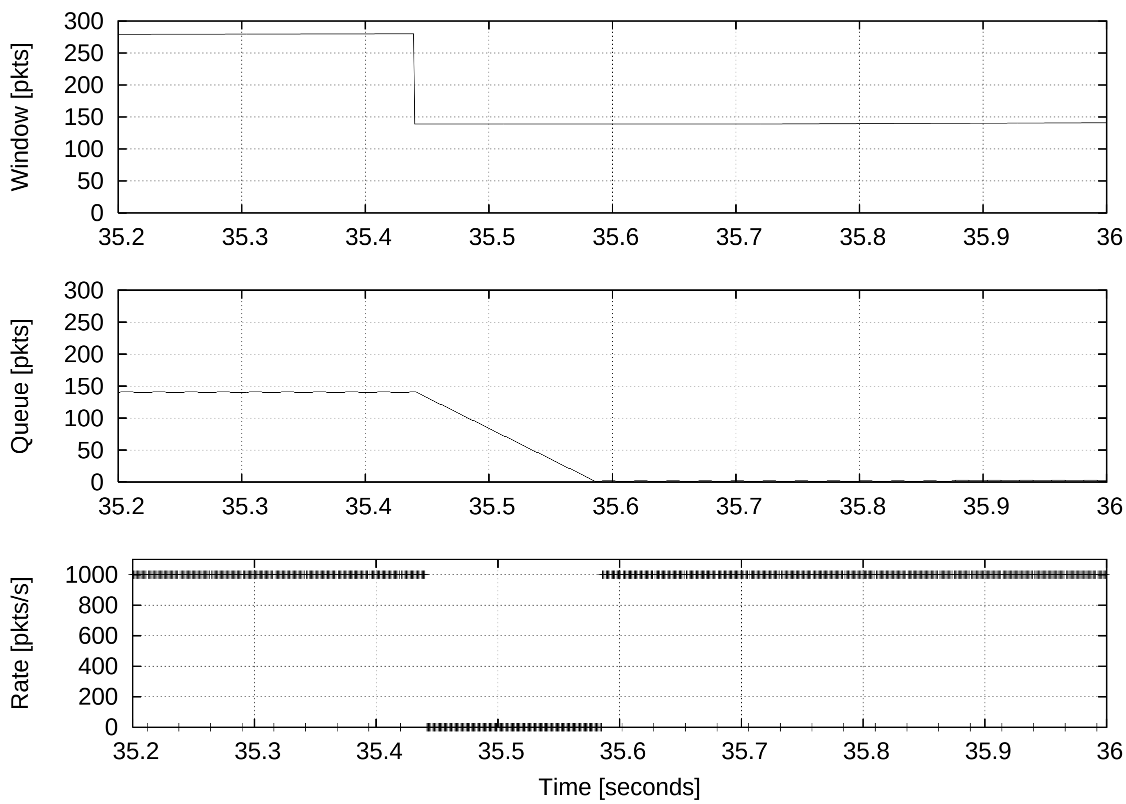


Figure 2.3: Microscopic behavior of a TCP flow at multiplicative decrease.

missing sequence number) takes some time to travel back to the sender. For a complete description of how TCP scales down, see [42], but the effect is that at $t = 35.44$, the sender halves its window size and stops sending for exactly one RTT. During this RTT, the bottleneck link is now serviced from packets stored in the router. For a router with buffers of $2T_p \times C$, the first new packet from the sender will arrive at the router just as it sends the last packet from the buffer, and the link will never go idle.

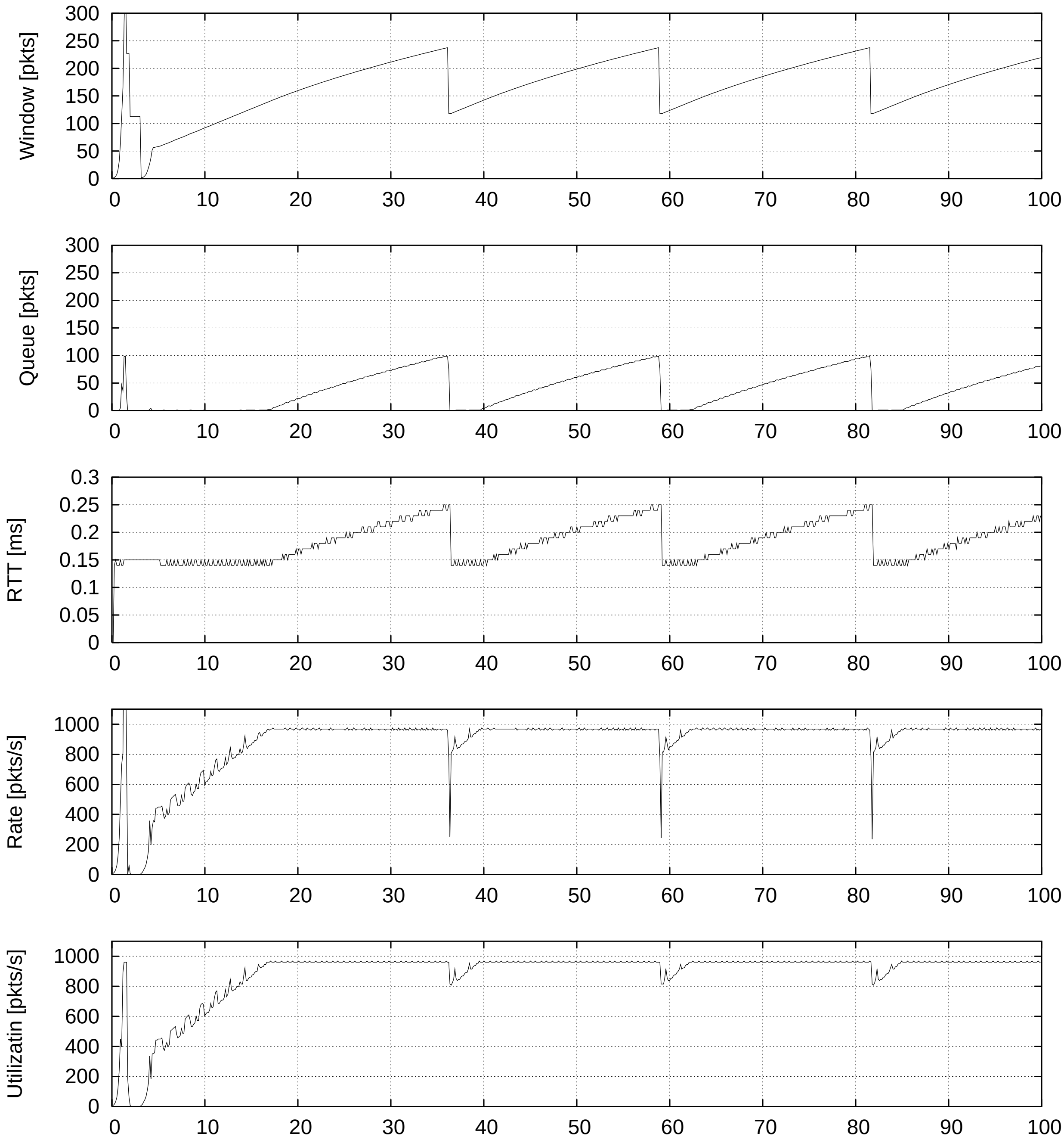


Figure 2.4: A TCP flow through an under-buffered router (100 packets).

2.2 Incorrectly Buffered Router

2.2.1 Under-buffered Router

Figure 2.4 shows what happens if we pick a buffer of less than $2T_p \times C$. The congestion window still follows the familiar sawtooth pattern, although with a lower maximum and a higher frequency. However, the shape of the RTT has now changed. The reason for this is apparent from the plot of the queue length. The queue stays empty (and hence the bottleneck link goes idle) over prolonged periods (e.g. 36 to 40). The reason for the queue staying empty is that the sending rate R is below the link capacity C. If the router receives less packets than it can send, the queue has to drain or stay empty. If $R < C$, the utilization will be less than 100%.

Why is the link not fully utilized? In order to achieve full utilization, we need $R \geq C$. From Equation 2.1, we can see that assuming we have an empty buffer and $T_Q = 0$, a minimum window size of $2T_p \times C$ is required. The minimum window size is half the maximum window size. The maximum window size in turn depends on the amount of buffering in the router. If we have more buffering, more packets can be stored in the router buffer and it will take longer for the TCP flow to scale down.

2.2.2 Over-buffered Router

Figure 2.5 shows what happens if we put more than $2T_p \times C$ of buffering in a router. The TCP window is again a sawtooth, but with a higher maximum and a lower frequency than the $2T_p \times C$ case. The RTT follows an identical sawtooth pattern and we have a constant sending rate of $R \approx C$ and maximum utilization $U = C$. The main difference is that after a window decrease, the buffer never runs empty. There are two important insights from this.

First, a buffer never running empty is not a good indicator of how much buffering is needed. A single TCP flow that is not constrained by window size will fill any buffer, no matter how high.

Second, over-buffering a congested router increases queueing delay (the buffer never empties) and increases the RTT. In the example in Figure 2.5, the added

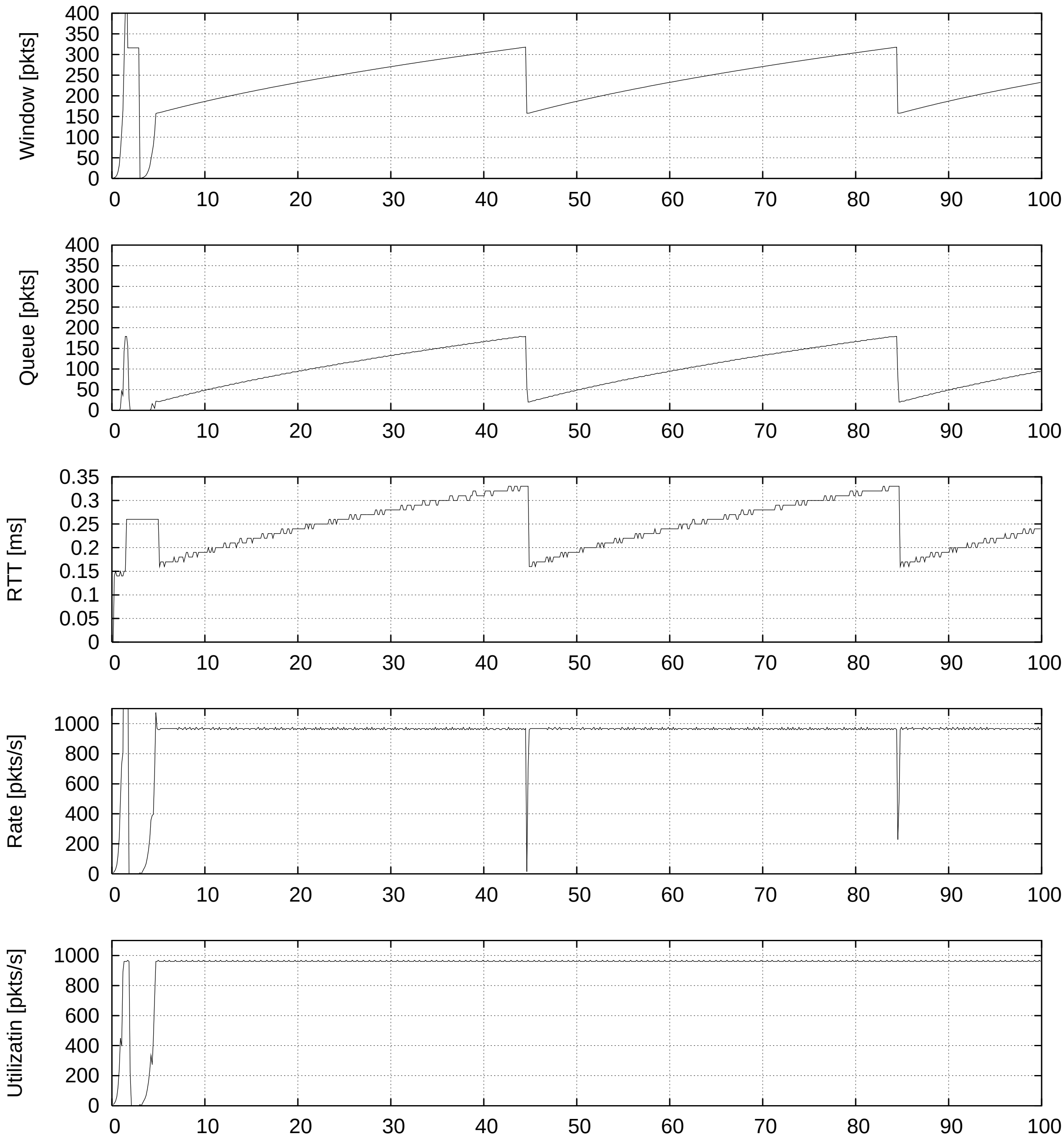


Figure 2.5: A TCP flow through an over-buffered router (180 packets).

queueing delay is about 10-15 ms.

2.3 The Rule-of-Thumb

In this and the next Section we will now derive the rule-of-thumb. We have seen that for a correctly buffered router, R is constant and equal to C directly before and after the window size is reduced. Using these results, we will now derive the rule-of-thumb in two different ways. As TCP behavior is most commonly described using rates, we will first use them to derive the rule. In Section 2.4 we derive the rule-of-thumb a second time by accounting for packets. The latter is the preferred method for the rest of this work.

To achieve full utilization, the sending rate before and after the window size is reduced from W_{max} to W_{min}, must be equal to the bottleneck link rate. From Equation 2.1, we obtain

$$\frac{W_{max}}{RTT_{max}} = \frac{W_{min}}{RTT_{min}} \tag{2.3}$$

With the minimum amount of buffering, we want the queue to be full (i.e. $Q = B$) before and empty (i.e. $Q = 0$) after the window has been halved. We also know that the minimum window size is half the maximum window size. Inserting into Equation 2.3

$$\begin{aligned} \frac{W_{max}}{2T_p \times C + B} &= \frac{\frac{1}{2}W_{max}}{2T_p \times C} \\ 4T_p \times C &= 2T_p \times C + B \\ B &= 2T_p \times C. \end{aligned}$$

This is the familiar rule-of-thumb. While not widely known, similar arguments were made elsewhere [12, 35], and our result can be easily verified using $ns2$ [1] simulation and a closed-form analytical model [5].

The rule-of-thumb essentially states that we need enough buffers to absorb the fluctuation in TCP's window size.

One interesting observation is that the amount of buffering directly depends on the factor TCP uses for multiplicative decrease. For TCP Reno, this is $\frac{1}{2}$. Generally, if TCP scales down its window as $W \rightarrow W(1 - \frac{1}{n})$, the amount of buffering that is required is

$$B = \left(\frac{1}{n-1}\right) 2T_p \times C.$$

Other types of TCP use smaller factors and require smaller buffers. For example High-Speed TCP [16] uses an adaptive mechanism that will require much smaller buffers for large window sizes.

TCP flavors that use latency (and not drops) to detect congestion [8, 26] have very different buffer requirements and are based on a fixed amount of buffering needed per flow, independent of the delay-bandwidth-product.

2.4 Accounting for Packets

We will now show how to derive the rule-of-thumb by accounting for individual packets. This different methodology will prove usful later.

Any packet that is outstanding from the sender's point of view can only be in one of three places:

- **In Transit** on one of the network links, either as a packet or as an ACK on its way back.
- **Queued** in in the router buffer.
- **Dropped** somewhere in the network.

A packet that is dropped will only be counted as dropped as long as the sender hasn't detected the drop yet. Once detected, the dropped packet is no longer counted as outstanding.

Assuming we have full utilization of the bottleneck link, we know that packets leave the sender as well as the router at a constant rate of C. The number of packets in transit with an empty buffer is $2T_p \times C$. The number of packets in the buffer is the length of the queue $Q(t)$.

The number of outstanding packets is commonly equated with the window size. This is not entirely correct as when the window-size is halved, the window size differs substantially from the actual number of outstanding packets [42]. However, both W and the number of outstanding packets share the increase and multiplicative decrease behavior. As the common intuition on W captures all of the essential aspects of the number of outstanding packets, we will use W for the number of outstanding packets for the remainder of this work. [1]

In summary, we can write the equation of outstanding packets

$$W(t) = 2T_p \times C + Q(t) + \Delta_{drop}(t) \tag{2.4}$$

This equation without losses holds well in practice over a wide range of TCP settings, for multiple flows (in this case we substitute $W(t)$ by the sum of the windows of all flows $\sum W^i(t)$), mixes of long and short flows and other network topologies. Figure 2.6 shows a link that receives bursty TCP traffic from many flows and is congested over short periods of time. For this graph, we plot utilization multiplied by the two-way propagation delay $2T_p \times U$ to allow a uniform scale on the y axis. When $t < 0.4$ utilization is below 100% and $2T_p \times U \approx \sum W^i(t)$. This makes sense as we have one RTT (with empty buffers is $RTT = 2T_p$) worth of traffic at rate U in transit at any given time and the buffer is still empty. For periods of full link utilization (e.g. from 3 to 4 seconds), we have $\sum W^i(t) = 2T_p \times C + Q$. The second observation is that $\sum W^i(t) < 2T_p \times C$ is a necessary and sufficient condition for the utilization dropping below 100%.

Using Equation 2.4, we can now also easily derive the rule-of-thumb. At the time before and after the window size is reduced, no packets are dropped and the network is "filled" with packets (i.e. no link is idle). The capacity of the network is $2T_p \times C$ packets. We therefore need a minimum window size (after decrease) of

$$W_{min} = 2T_p \times C$$

[1] For $ns2$ simulations we measured the actual number of outstanding packets as the difference between highest sequence number sent and highest sequence number acknowledged. This again will be referred to as W.

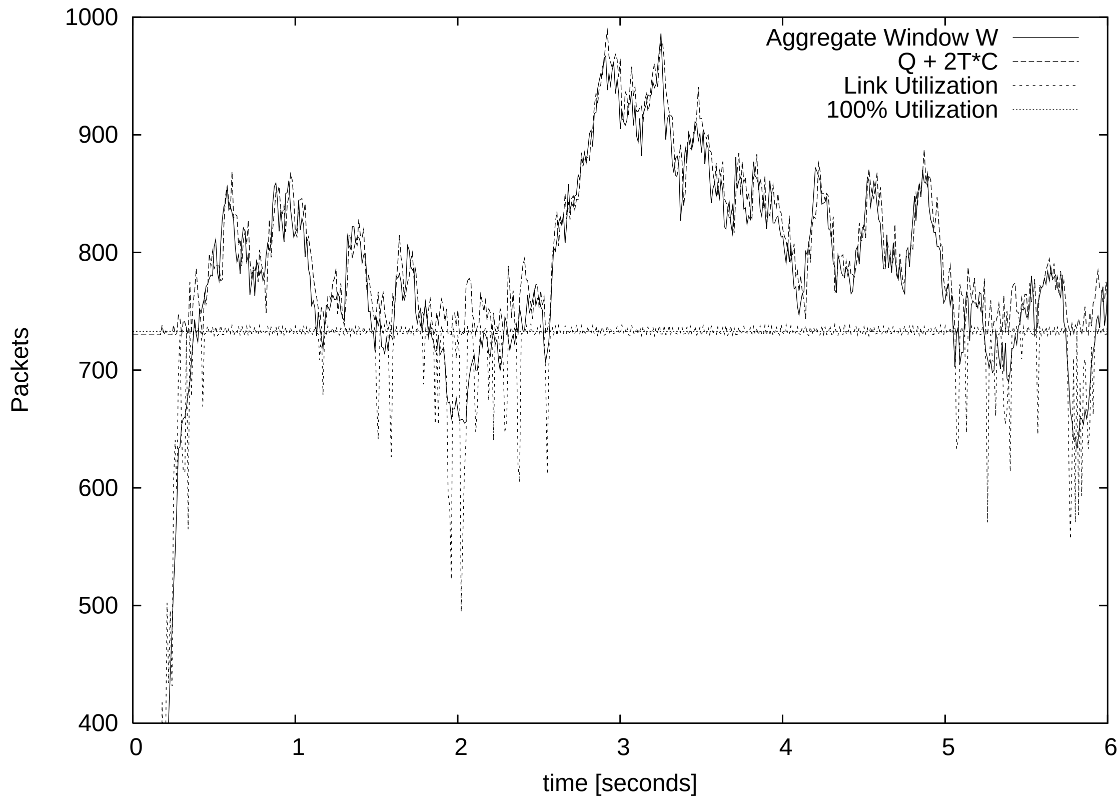


Figure 2.6: Comparison of the sum of window sizes $\sum W^i(t)$ and $2T_p \times C + Q(t)$

and for the maximum window size, the extra packets have to be in the router buffer

$$W_{max} = 2T_p \times C + B.$$

.

We know the maximum window size is half the minimum window size $W_{max} = 2W_{min}$. We substitute and find again the rule-of-thumb:

$$B = W_{max} - 2T_p \times C = 2W_{min} - 2T_p \times C = 2 \times 2T_p \times C - 2T_p \times C = 2T_p \times C$$

2.5 Summary

To summarize this Chapter, the role of a router buffer is to absorb the fluctuation in TCP window size and the fluctuation in the number of packets on the network. Because TCP in congestion avoidance mode varies its window size between W_{max} and $\frac{1}{2}W_{max}$, the number of packets on the network varies by a factor of two, and consequently, the buffer needs to hold an amount of packets that's equal to the capacity of the network. For one or a small number of flows, this capacity is one delay bandwidth product as correctly observed by Villamizar and Song [43].

Chapter 3

Long Flows

In a backbone router, many flows share the bottleneck link simultaneously. For example, a 2.5Gb/s (OC48c) link typically carries more than 10,000 flows at a time [18]. This should not be surprising: A typical user today is connected via a 56kb/s modem, and a fully utilized 2.5Gb/s can simultaneously carry more than 40,000 such flows. When it's not fully utilized, the buffers are barely used, and the link isn't a bottleneck. Therefore, we should size the buffers to accommodate a large number of flows. So, how should we change our model to reflect the buffers required for a bottleneck link with many flows? We will consider two situations. First, we will consider the case when all the flows are synchronized with each other, and their sawtooths march in lockstep perfectly in-phase. Then, we will consider flows that are not synchronized with each other, or are at least, not so synchronized as to be marching in lockstep. When they are sufficiently desynchronized — and we will argue that this is the case in practice — the amount of buffering drops sharply.

3.1 Synchronized Long Flows

Consider the evolution of *two* TCP Reno flows through a bottleneck router. The evolution of the window sizes, sending rates and queue length is shown in Figure 3.1. Although the two flows start at different times (time 0 and 10 seconds respectively), they synchronize quickly to be perfectly in phase. This is a well-documented and

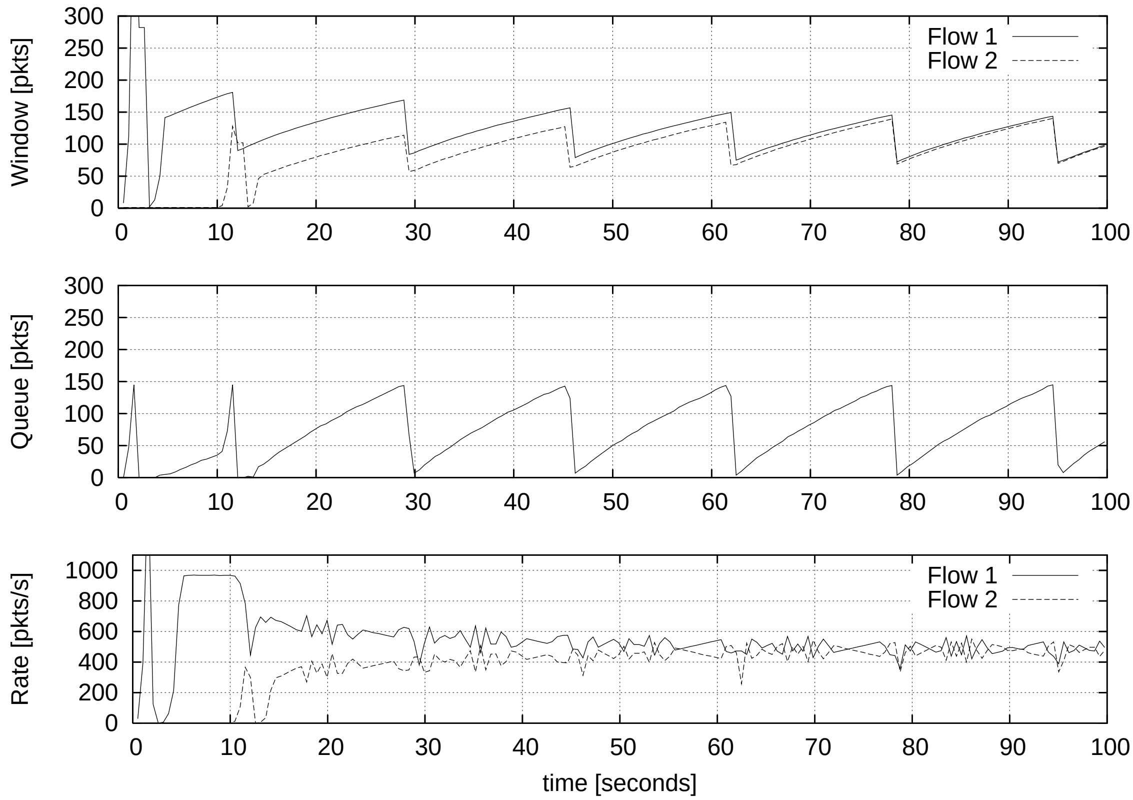


Figure 3.1: Time Evolution of two TCP flows sharing a Bottleneck Link in $ns2$

studied tendency of flows sharing a bottleneck to become synchronized over time [35, 4].

A set of precisely synchronized flows has the same buffer requirements as a single flow. Their aggregate behavior is still a sawtooth; as before, the height of the sawtooth is dictated by the maximum window size needed to fill the round-trip path, which is independent of the number of flows. Specifically, assume that there are n flows, each with a congestion window $W^i(t)$ at time t, and end-to-end propagation delay T_p^i, where $i = [1, ..., n]$. The window size is the maximum allowable number of outstanding packets, so from Equation 2.4, we have

$$\sum_{i=1}^{n} W^i(t) = 2\overline{T}_p \times C + Q(t) \tag{3.1}$$

where $Q(t)$ is the buffer occupancy at time t, and $\overline{T}_p$ is the average[1] propagation delay. As before, we can solve for the buffer size by considering two cases: just before and just after packets are dropped. First, because they move in lock-step, the flows all have their largest window size, W_{max} at the same time; this is when the buffer is full, so

$$\sum_{i=1}^{n} W_{max}^{i} = W_{max} = 2\overline{T}_p \times C + B. \tag{3.2}$$

Similarly, their window size is smallest just after they all drop simultaneously [35]. If the buffer is sized so that it just goes empty as the senders start transmitting after the pause, then

$$\sum_{i=1}^{n} W_{min}^{i} = \frac{W_{max}}{2} = 2\overline{T}_p \times C. \tag{3.3}$$

Solving for B, we find once again that $B = 2\overline{T}_p \times C = \overline{RTT} \times C$. Clearly, this result holds for any number of synchronized in-phase flows.

3.1.1 When are Flows Synchronized?

The basic mechanism for flow synchronization was first described in [35]. When a single flow without any sources of randomness passes through a router, it fills the router buffer packet by packet with the "extra" packets sent when it increases its window size. The packet that is eventually dropped is the first "extra" packet sent when the buffer is already full. It then takes the flow one RTT before this information travels back to the source and it can scale down and reduce its sending rate. If two flows share a router, during this RTT, the second flow will also increase its window size and send one "extra" packet. This packet will also be dropped and both flows scale down in the same RTT.

Such synchronization behavior has been widely observed in $ns2$ simulation (e.g. [4],[45]) and methods to reduce synchronization have been proposed [17]. In [33], the

[1] $\overline{T}_p$ is actually a weighted average that is skewed towards lower values of T_p. Flows with shorter T_p have a shorter RTT, therefore scale up more quickly and receive a larger share of the bandwidth, which means they have a larger share of the packets in transit.

authors find synchronization in $ns2$ for up to 1000 flows as long as the RTT variation is below 10%. Likewise, we found in our simulations and experiments that while in-phase synchronization is common for less than 100 concurrent flows, it is very rare above 500 concurrent flows.[2]

On real networks, synchronization is much less frequent. Anecdotal evidence of synchronization [3] is largely limited to a small number of very heavy flows. In a laboratory experiment on a throttled shared memory router, we found that two typical TCP flows were not synchronized, while an identical $ns2$ setup showed perfect synchronization. On large routers in the core of the network, there is no evidence of synchronization at all [18, 23]. Most analytical models of TCP traffic operate at a flow level and do not capture packet-level effects. They are therefore, no help in predicting the conditions under which synchronization occurs. The only model we know of [13] that predicts synchronization does not give clear bounds that we could use to guess when it occurs.

Today, we don't understand fully what causes flows to synchronize and to what extent synchronization exists in real networks. It seems clear that the $ns2$ simulator with long-lived flows does not correctly predict synchronization on real networks. This is not surprising as $ns2$ in itself introduces no randomness into the traffic, while real networks have a number of such sources (e.g. shared memory routers, shared medium collisions, link-level errors, end host service times, etc.).

It is safe to say though, that flows are not synchronized in a backbone router carrying thousands of flows with varying $RTTs$. Small variations in RTT or processing time are sufficient to prevent synchronization [33]; and the absence of synchronization has been demonstrated in real networks [18, 23]. Although we don't precisely understand when and why synchronization of TCP flows takes place, we observed that for aggregates of more than 500 flows with varying $RTTs$, the amount of in-phase synchronization decreases even in $ns2$. Under such circumstances, we can treat flows as being not synchronized at all.

[2] Some out-of-phase synchronization (where flows are synchronized but scale down their window at different times during a cycle) was visible in some $ns2$ simulations with up to 1000 flows. However, the buffer requirements are very similar for out-of-phase synchronization as they are for no synchronization at all.

3.2 Desynchronized Long Flows

To understand the difference between adding synchronized and desynchronized window size processes, recall that if we add together many synchronized sawtooths, we get a single large sawtooth, and the buffer size requirement doesn't change. If on the other hand, the sawtooths are not synchronized, the more flows we add, the less their sum will look like a sawtooth. They will smooth each other out, and the distance from the peak to the trough of the aggregate window size will get smaller. Hence, given that we need as much buffer as the distance from the peak to the trough of the aggregate window size, we can expect the buffer size requirements to get smaller as we increase the number of flows. This is indeed the case, and we will explain why and then demonstrate via simulation.

Consider a set of TCP flows with random (and independent) start times and propagation delays. We'll assume that they are desynchronized enough that the window size processes are independent of each other. We can model the total window size as a bounded random process made up of the sum of these independent sawtooths. We know from the central limit theorem that the aggregate window size process will converge to a Gaussian process.

More formally, we model the congestion windows $W_i(t)$ as independent random variables

$$E[W_i] = \mu_W \qquad var[W_i] = \sigma_W^2.$$

Now, the central limit theorem gives us the distribution of the sum of the window sizes as

$$\sum W_i(t) \rightarrow n\mu_W + \sqrt{n}\sigma_W N(0,1).$$

Figure 3.2 shows that indeed, the aggregate window size does converge to a Gaussian process. The graph shows the probability distribution of the sum of the congestion windows of all flows $W = \sum W_i$, with different propagation times and start times as explained in Chapter 5.

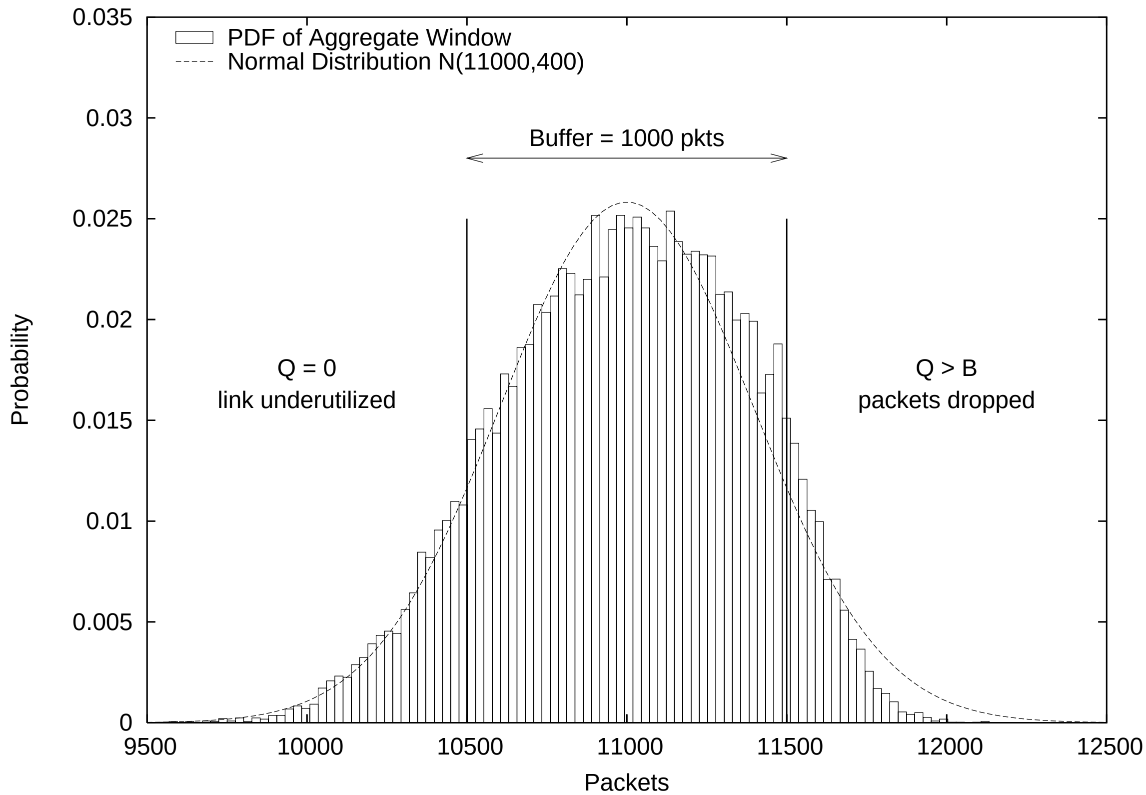


Figure 3.2: Probability Distribution of the Sum of the Congestion Windows for Desynchronized Flows sharing a Bottleneck Link from an $ns2$ simulation

With this model of the distribution of W, we can now find the necessary amount of buffering. From the window size process, we know from Equation 2.4 that the queue occupancy at time t is

$$Q(t) = \sum_{i=1}^{n} W_i(t) - (2T_p \times C) - \Delta_{drop} \tag{3.4}$$

This equation assumes full utilization and an unlimited buffer. What happens with a limited buffer can be seen in Figure 3.3. In the graph, we plotted the queue length against the right side of Equation 3.4. As we have many desynchronized flows, the sum of the TCP windows fluctuates rapidly at time scales below or around one RTT. The two dotted lines are for $Q = 0$ and $Q = B$, respectively. We can see that

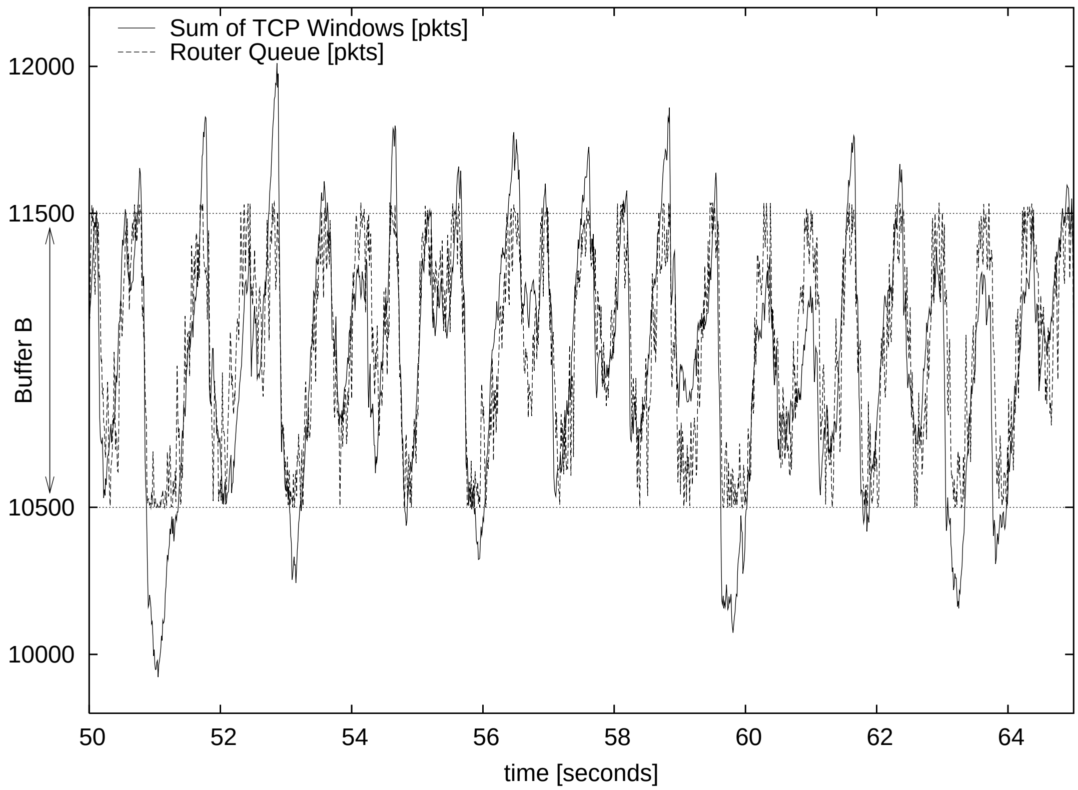


Figure 3.3: Plot of $\sum W_i(t)$ of all TCP flows, and of the queue Q offset by 10500 packets.

the above equation holds very well as long as the number of outstanding packets plus $2T_p \times C$ is within the buffer. If the queue drops below the lower line, our utilization is below 100%. If we are above the upper line, the buffer overflows and we drop packets. The goal of picking the buffer is to make sure both of these events are rare and most of the time $0 < Q < B$.

Equation 3.6 tells us that if W has a Gaussian distribution, Q has a Guassian distribution shifted by a constant (of course, the Guassian Distribution is restricted to the allowable range of Q). This is very useful because we can now pick a buffer size and know immediately the probability that the buffer will underflow and lose throughput.

Because it is Gaussian, we can determine the queue occupancy process if we know its mean and variance. The mean is simply the sum of the mean of its constituents. To find the variance, we'll assume for convenience that all sawtooths have the same average value (assuming different values would not fundamentally change the result). For a given network link, the overall bandwidth, and thus the sum of the congestion windows $\sum W$, will be the same for any number of flows n. If we denote the average and variance for a single flow as

$$E[W] = \mu_{n=1} \qquad var[W] = \sigma^2_{n=1},$$

then if n flows share this link, the mean and variance become

$$E[W] = \frac{\mu_{n=1}}{n} \qquad var[W] = \left(\frac{\sigma_{n=1}}{n}\right)^2$$

and the sum of the windows decreases to

$$\begin{aligned} \sum W_i(t) &\rightarrow n\frac{\mu_W}{n} + \sqrt{n}\frac{\sigma_W N(0,1)}{n} && (3.5) \\ &= \mu_{n=1} + \frac{1}{\sqrt{n}}\sigma_{n=1}N(0,1). && (3.6) \end{aligned}$$

We can model each window as a sawtooth oscillating between minimum $\frac{2}{3}\mu_W$ and maximum $\frac{4}{3}\mu_W$. Since the standard deviation of the uniform distribution is $\frac{1}{\sqrt{12}}$-th of its length, the standard deviation of a single window size σ_{W_i} is thus

$$\sigma_{W_i} = \frac{1}{\sqrt{12}}\left(\frac{4}{3}\overline{W}_i - \frac{2}{3}\overline{W}_i\right) = \frac{1}{3\sqrt{3}}\overline{W}_i$$

From Equation (3.4),

$$\overline{W}_i = \frac{\overline{W}}{n} = \frac{2\overline{T}_p \times C + \overline{Q}}{n} \leq \frac{2\overline{T}_p \times C + B}{n}.$$

For a large number of flows, the standard deviation of the sum of the windows, W, is given by

$$\sigma_W \leq \sqrt{n}\sigma_{W_i},$$

and so by Equation (3.4) the standard deviation of $Q(t)$ is

$$\sigma_Q = \sigma_W \leq \left(\frac{1}{3\sqrt{3}}\right) \frac{2\overline{T}_p \times C + B}{\sqrt{n}}.$$

Now that we know the distribution of the queue occupancy, we can approximate the link utilization for a given buffer size. Whenever the queue size is below a threshold, b, there is a risk (but not guaranteed) that the queue will go empty, and we will lose link utilization. If we know the probability that $Q < b$, then we have an upper bound on the lost utilization. Because Q has a normal distribution, we can use the error-function[3] to evaluate this probability. Therefore, we get the following lower bound for the utilization

$$Util \geq erf\left(\frac{3\sqrt{3}}{2\sqrt{2}} \frac{B}{\frac{2\overline{T}_p \times C + B}{\sqrt{n}}}\right). \tag{3.7}$$

Here are some numerical examples of utilization, using $n = 10000$.

Router Buffer Size	Utilization
$B = 1 \cdot \frac{2\overline{T}_p \times C}{\sqrt{n}}$	Util $\geq$ 98.99 %
$B = 1.5 \cdot \frac{2\overline{T}_p \times C}{\sqrt{n}}$	Util $\geq$ 99.99988 %
$B = 2 \cdot \frac{2\overline{T}_p \times C}{\sqrt{n}}$	Util $\geq$ 99.99997 %

This means that we can achieve full utilization with buffers that are the delay-bandwidth product divided by the square-root of the number of flows, or a small multiple thereof. As the number of flows through a router increases, the amount of required buffer decreases.

Whether this model holds can be easily verified using $ns2$ and the result is shown in Figure 3.4. Between one and 300 flows share a bottleneck link, and we iteratively found the minimum amount of buffering that allows us to achieve at least 95% link utilization. For fewer than 50 flows, our model clearly does not hold, we still have synchronization effects. However, as we reach 50 flows, the flows desynchronize and

[3] A more precise result could be obtained by using Chernoff Bounds instead. However to achieve our goal of determining a buffer size that gives us low packet loss and underutilization, this simple method is sufficient.

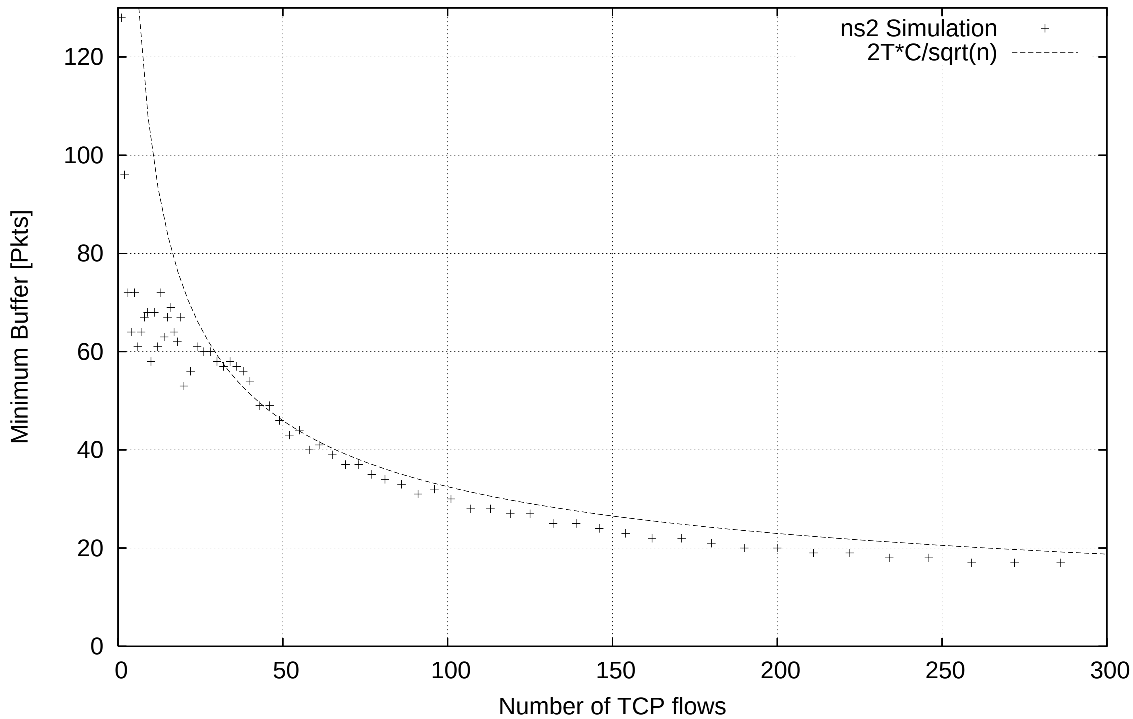


Figure 3.4: Buffer Requirements vs. number of Flows

we can see that our model predicts the actual required minimum buffer very well. We will verify our model more extensively with simulations in Chapter 5 and with experiments on real networks in Chapter 6.

This result has practical implications for building routers. A typical congested core router currently has 10,000 to 100,000 flows passing through it at any given time. While the vast majority of flows are short (e.g. flows with fewer than 100 packets), the flow length distribution is heavy-tailed and the majority of packets at any given time belong to long flows[4]. As a result, such a router would achieve close to full utilization with buffer sizes that are only $\frac{1}{\sqrt{10000}} = 1\%$ of the delay-bandwidth product.

[4]Live production networks have mixes of short flows and long flows. We will show that this model also holds for mixes of flows in Chapter 5 and present results on estimating n from live traffic in Chapter 6.

Chapter 4

Short Flow Model

4.1 Short Flows

Not all TCP flows are long-lived; in fact, many flows last only a few packets, never leave slow-start, and never reach their equilibrium sending rate [18]. Up until now, we've only considered long-lived TCP flows, so now, we'll consider how short TCP flows affect the size of the router buffer. We're going to find that short flows (TCP and non-TCP) have a much smaller effect than long-lived TCP flows, particularly in a backbone router with a large number of flows.

We will define a short-lived flow to be a TCP flow that never leaves slow-start (e.g. any flow with fewer than 90 packets, assuming a typical maximum window size of 65kB). In Section 4.2, we will see that our results hold for types of TCP flows and some non-TCP flows too (e.g. DNS queries, ICMP, etc.).

Consider again, the topology in Figure 2.1, with multiple senders on separate access links. As has been widely reported from measurement, we assume that new short flows arrive according to a Poisson process [32, 14]. In slow-start, TCP increases its congestion window by one for each successfully transmitted packet. This effectively doubles the congestion window for each full window worth of packets that is transmitted. Figure 4.1 shows an example of this behavior. After establishing the TCP session with the exchange of SYN packets, the flow sends out a first burst of two packets. Each subsequent burst has twice the length of the previous one (i.e. four,

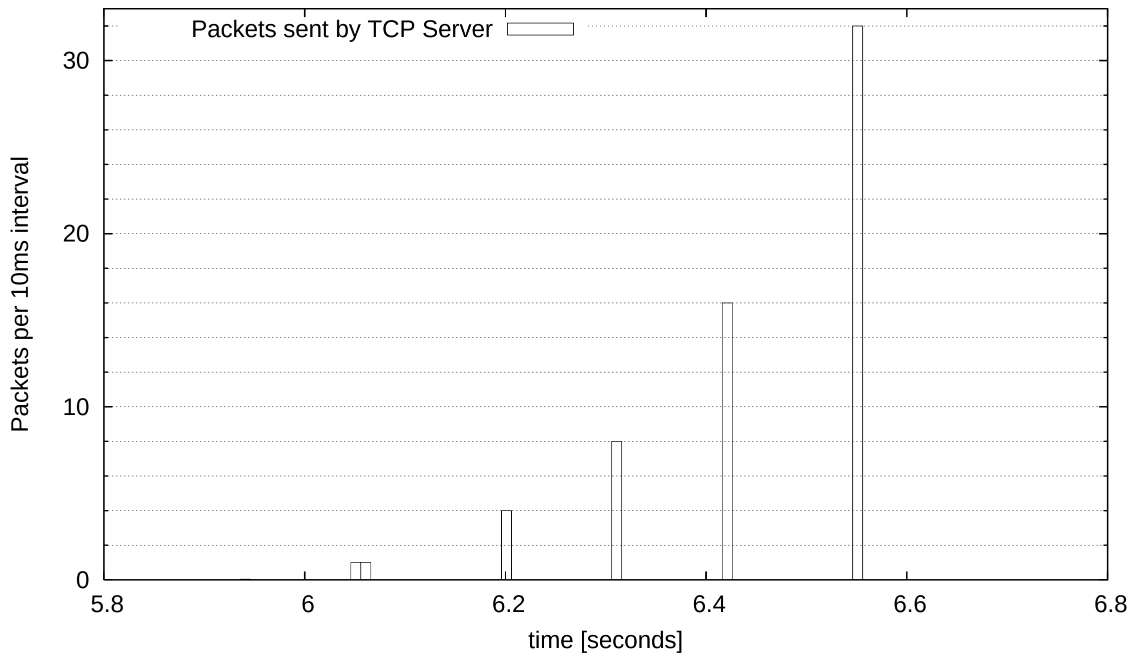


Figure 4.1: A Single Short-Lived TCP Flow

eight, sixteen, etc.). If the access links have lower bandwidth than the bottleneck link, the bursts are spread out and a single burst causes no queueing. We assume the worst case where access links have infinite speed; bursts arrive intact at the bottleneck router.

4.1.1 Short Flow Model

We will model bursts arriving from many different short flows at the bottleneck router. Some flows will send a burst of two packets, while others might send a burst of four, eight or sixteen packets and so on. There will be a distribution of burst-sizes; and if there is a very large number of flows, we can consider each burst to be independent of the other bursts, even of the other bursts in the same flow. In this simplified model, the arrival process of bursts themselves (as opposed to the arrival of flows) can be assumed to be Poisson. One might argue that the arrivals are not Poisson as a burst is followed by another burst one RTT later. However, under a low load and with many flows, the buffer will usually empty several times during one RTT and is effectively "memoryless" at this time scale.

For instance, let's assume we have arrivals of flows of a fixed length l. Because of the doubling of the burst lengths in each iteration of slow-start, each flow will arrive in n bursts of size

$$X_i = \{2, 4, ...2^{n-1}, R\},$$

where R is the remainder, $R = l \ mod \ (2^n - 1)$. Therefore, the bursts arrive as a Poisson process, and their lengths are i.i.d. random variables, equally distributed among $\{2, 4, ...2^{n-1}, R\}$.

The router buffer can now be modeled as a simple M/G/1 queue with a FIFO service discipline. In our case, a "job" is a burst of packets, and the job size is the number of packets in a burst. The average number of jobs in an M/G/1 queue is known to be (e.g. [44])

$$E[N] = \frac{\rho}{2(1-\rho)} E[X^2].$$

Here, ρ is the load on the link (the ratio of the amount of incoming traffic to the link capacity C), and $E[X]$ and $E[X^2]$ are the first two moments of the burst size. This model will overestimate the queue length because bursts are processed packet-by-packet while in an M/G/1 queue, the job is only de-queued when the whole job has been processed. If the queue is busy, it will overestimate the queue length by half the average job size, and so,

$$E[Q] = \frac{\rho}{2(1-\rho)} \frac{E[X^2]}{E[X]} - \rho \frac{E[X]}{2}.$$

It is interesting to note that the average queue length is independent of the number of flows and the bandwidth of the link. It only depends on the load of the link and the length of the flows.

A similar model is described in [20]. The key difference is that the authors model bursts as batch arrivals in an $M^{[k]}/M/1$ model (as opposed to our model that models bursts by varying the job length in a M/G/1 model). It accommodates both slow-start and congestion avoidance mode; however, it lacks a closed form solution. In the

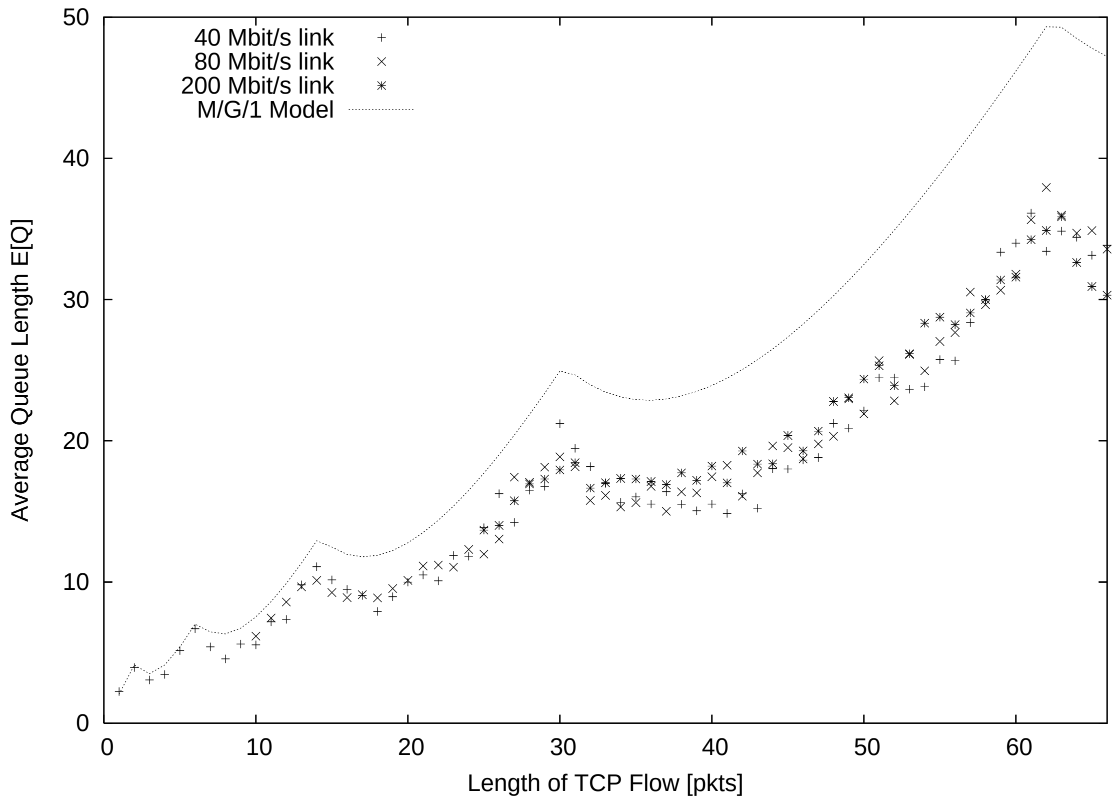


Figure 4.2: The average queue length as a function of the flow length for $\rho = 0.8$.

end, the authors obtain queue distributions that are very similar to ours.

4.1.2 Buffer Requirements

We can validate our model by comparing it with simulations. Figure 4.2 shows a plot of the average queue length for a fixed load and varying flow lengths, generated using $ns2$. Graphs for three different bottleneck link bandwidths (40, 80 and 200 Mb/s) are shown. The model predicts the relationship very closely. Perhaps surprisingly, the average queue length peaks when the probability of large bursts is highest, not necessarily when the average burst size is highest. For instance, flows of size 14 will generate a larger queue length than flows of size 16. This is because a flow of 14 packets generates bursts of $X_i = \{2, 4, 8\}$ and the largest burst of size 8 has a

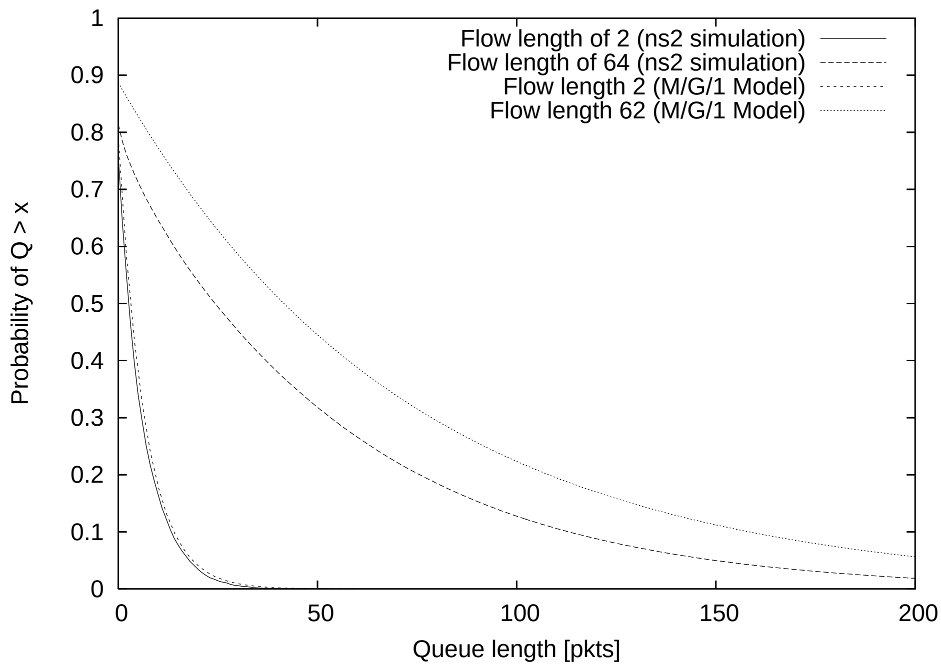


Figure 4.3: Queue length distribution for load 0.85 and flows of length 2 and 62

probability of $\frac{1}{3}$. A flow of 16 packets generates bursts of sizes $X_i = \{2, 4, 8, 4\}$, where the maximum burst length of 8 has a probability of $\frac{1}{4}$. As the model predicts, the bandwidth has no effect on queue length, and the measurements for 40, 80 and 200 Mb/s are almost identical. The gap between model and simulation is because the access links before the bottleneck link space out the packets of each burst. Slower access links would produce an even smaller average queue length.

To determine the buffer size, we need the probability distribution of the queue length, not just its average. This is more difficult as no closed form result exists for a general M/G/1 queue length distribution. Instead, we approximate its tail using the effective bandwidth model [27], which tells us that the queue length distribution is

$$P(Q \geq b) = e^{-b \frac{2(1-\rho)}{\rho} \frac{E[X_i]}{E[X_i^2]}}.$$

This equation is derived in Appendix A.3.

Figure 4.3 shows an experimental verification of this result. For traffic of only short flows of length two and length 62 respectively, we measured the queue occupancy using an $ns2$ simulation and compared it to the model predictions. The model gives us a good upper bound for the queue length distribution. It overestimates the queue length as the model assumes infinitely fast access links. The access links in $ns2$ spread out the packets slightly, which leads to shorter queues. The overestimation is small for the bursts of length two, but much larger for bursts of length 62. In Chapter 5, we will evaluate this effect in more detail.

Our goal is to drop very few packets (if a short flow drops a packet, the retransmission significantly increases the flow's duration). In other words, we want to choose a buffer size B such that $P(Q \geq B)$ is small.

A key observation is that, for short flows, the size of the buffer does not depend on the line-rate, the propagation delay of the flows, or the number of flows; it only depends on the load of the link, and length of the flows. Therefore, a backbone router serving highly aggregated traffic needs the same amount of buffering to absorb short-lived flows as a router serving only a few clients. Furthermore, because our analysis doesn't depend on the dynamics of slow-start (only on the burst-size distribution), it can be easily extended to short unresponsive UDP flows.

In practice, buffers can be made even smaller. For our model and simulation, we assumed access links that are faster than the bottleneck link. There is evidence [18, 10] that highly aggregated traffic from slow access links in some cases can lead to bursts being smoothed out completely. In this case, individual packet arrivals are close to Poisson, resulting in even smaller buffers. The buffer size can be easily computed with an M/D/1 model by setting $X_i = 1$.

In summary, short-lived flows require only small buffers. When there is a mix of short- and long-lived flows, we will see from simulations and experiments in Chapter 5 that the short-lived flows contribute very little to the buffering requirements, and the buffer size will usually be determined by the number of long-lived flows.[1]

[1] For a distribution of flows, we define short flows and long flows as flows that are in slow-start and congestion avoidance mode respectively. This means that flows may transition from short to long during their existence.

4.2 Other Types of Flows

The model described above is not only valid for short flows, but also for any other traffic that:

- Does not adapt its overall sending patterns to packet loss or changes in latency (other than simple retransmitting of packets)
- Sends packets in bursts where the time between bursts is large enough that the router buffer will usually empty at least once during this time. For a high-speed router at moderate loads, this is always the case if the inter-burst interval is in the order of the RTT.

If these two conditions hold, the arguments and methodology we used for short flows will equally hold and give us a good model of the queue distribution. Below, we discuss several examples of such flows.

4.2.1 Flows Constrained by Access Link and Window Size

Most flows from residential networks today are constrained by the access link. Internet access from home (56kB/s to 1Mb/s) is much slower than the links of ISPs (100's of Mb/s) or the backbone ($\geq$ 1 Gb/s). Links for residential access tend to have very large buffers such that the buffer of the access link is larger than the maximum window size of a sender $W_{max} < B$. We know that $W = 2T_p \times C + Q$, thus, $Q = W - 2T_p \times C < W \leq W_{max}$ and thus, $Q < B$. In other words, such an over-buffered access link can never drop a packet because the window is too small to ever fill the buffer.

The traffic of such an over-buffered access link appears to a core router as a constant rate sender. Aggregate traffic from such sources will behave very close to a Poisson arrival process (we believe this is the reason why Poisson arrivals at a packet level were seen in [18]) and the buffer requirements of the router can be calculated with the above model and a burst size of one packet. As we will see in Chapter 5 even for high loads a few ten packets are sufficient to achieve extremely low loss probabilities.

4.2.2 Real-Time Protocols and Games

Streaming media and online computer games are typically sent at a constant rate. However, in some encoding schemes, the rate varies slightly over small time scales. We can model constant rate senders (e.g. the Quake Engine protocol) with the model described in this Chapter and a burst size equal to the packet size. Varying the encoding rate (as it is done by some media streaming protocols as well as congestion-adaptive protocols) is more complicated and beyond the scope of this work.

4.2.3 UDP and ICMP Flows

UDP and ICMP flows with constant rate sending patterns (e.g. low-rate port scans, ping traces, DNS lookups, UDP based online gaming protocols) can be modeled using the short flow model described here, and should require very small buffers. Congestion aware UDP protocols would require a separate analysis but are very rare on the internet.

Chapter 5

Experimental Verification with $ns2$

This chapter is dedicated to verifying the models we built in the last two chapters using simulation with the $ns2$ [1] network simulator. We also give an overview of our experimental setup and methodology to facilitate further research in the area.

Some of the experimental scripts used for this work are available on the author's home page and use, modification and re-distribution for research purposes is encouraged.

We initially look at long flows and short flows separately. This allows us to verify quantitatively our models and get a better understanding on where they hold. We then combine long and short flows to see how they interact. Real traffic typically consists of flows with a continuous and typically heavy-tailed length distribution. We, therefore, study Pareto distributed flow lengths to see if the models we built for the separate classes still hold for this case.

Finally, we show that our model, at least in some cases, also holds for two-way congestion.

5.1 Experimental Setup and NS2

NS2 [1] is a freely available discrete event network simulator. It comes with basic models for links, routers, end hosts and a large number of protocols. It is the current defacto standard to analyze new protocols and models in the academic network research community. It is known to give a very accurate picture of how protocols will perform on real networks; however, it is also known to deviate in some areas, such as predicting synchronization in the network.

For the NS2 simulations in this thesis, we, in most cases, used a standard dumbbell topology. A number of senders send data over network links (herein referred to as "access links") to a router. Each sender has a separate access link with an individual latency and bandwidth. In most experiments, bandwidths were identical, however, the latency varies by about 20% of the total end-to-end latency unless otherwise noted. The router is connected to a receiving host via the bottleneck link. All receivers are on the receiving host. All links are bi-directional full-duplex links.

Focusing on this topology might seem restrictive at first. However, in practice, the model captures most characteristics of a complex network, or can at least be used to build a "worst case" topology that is similar to a complex topology, but will create more burstiness.

In a complex network, we can calculate the propagation delay for any sender-destination pair. By adjusting the variable latency of the access links, we can tune our simplified network to the same end-to-end propagation delay. In the absence of any (even short term) congestion, we can therefore create a simplified network that from the sender/receiver point of view is identical to the complex network.

In practice, a complex network would encounter congestion in two places. In the access networks leading to and from the router, and at the bottleneck link. The focus of this entire work is to investigate buffer requirements for networks with a single point of congestion. There still might be short-term congestion in the access network (e.g. two packets arriving at an access router at the same time), however this will mainly have the effect of reducing burstiness of the arriving traffic, which has little effect on long-lived flows and reduces queue size for short lived flows. The main effect of our simplified topology is to see an increased amount of burstiness and concurrent packet arrivals at our bottleneck router. In other words, we simplify the network in a way that, at least in most cases, should increase buffer requirements.

Multiple points of congestion would undoubtedly require different topologies. However, with the exception of two-way congestion, investigating this is beyond the scope of this work.

For the experiments, we use TCP Reno, as it is close to the most common flavors of TCP deployed in the internet today. Unless otherwise indicated, we had no limit on the maximum window size of a flow. In the real Internet, window sizes are typically limited to 12 packets (MS Windows Desktop OS [29]) or 43 packets (Linux, BSD, MS Windows Server OS). Again, we do this in order to find the maximum buffer requirements. As we will see below, fixed window sizes can considerably reduce buffer requirements.

We used the simple TCP implementation of ns2 with TCP senders and a TCP sink, as we did not need the additional features of the full, two-way TCP implementation. Simple TCP accounts for all traffic as well as for the window size in packets. While real TCP stacks use bytes, this makes little difference as long as we have flows sending at the maximum rate.

5.2 Long Flows

In this Section, we examine the behavior of many long flows in congestion avoidance mode that saturate a router. Flows are started at different times during an initialization phase and try to send an infinite amount of data. We then wait for the system to stabilize - typically between 10 and 60 seconds - and then measure for another 60 to 120 seconds. The propagation delay T_P of the flows was evenly distributed at an average with a standard deviation of at least 10%.

5.2.1 Utilization vs. Goodput

The first experiment tries to illustrate the relationship between the number of flows, the amount of buffering and the utilization or goodput. The top graph in Figure 5.1 shows the utilization vs. the number of concurrent long-lived flows passing through a 20 Mb/s link with 130 ms latency for a buffer of 50, 100 and 500. Overall, average

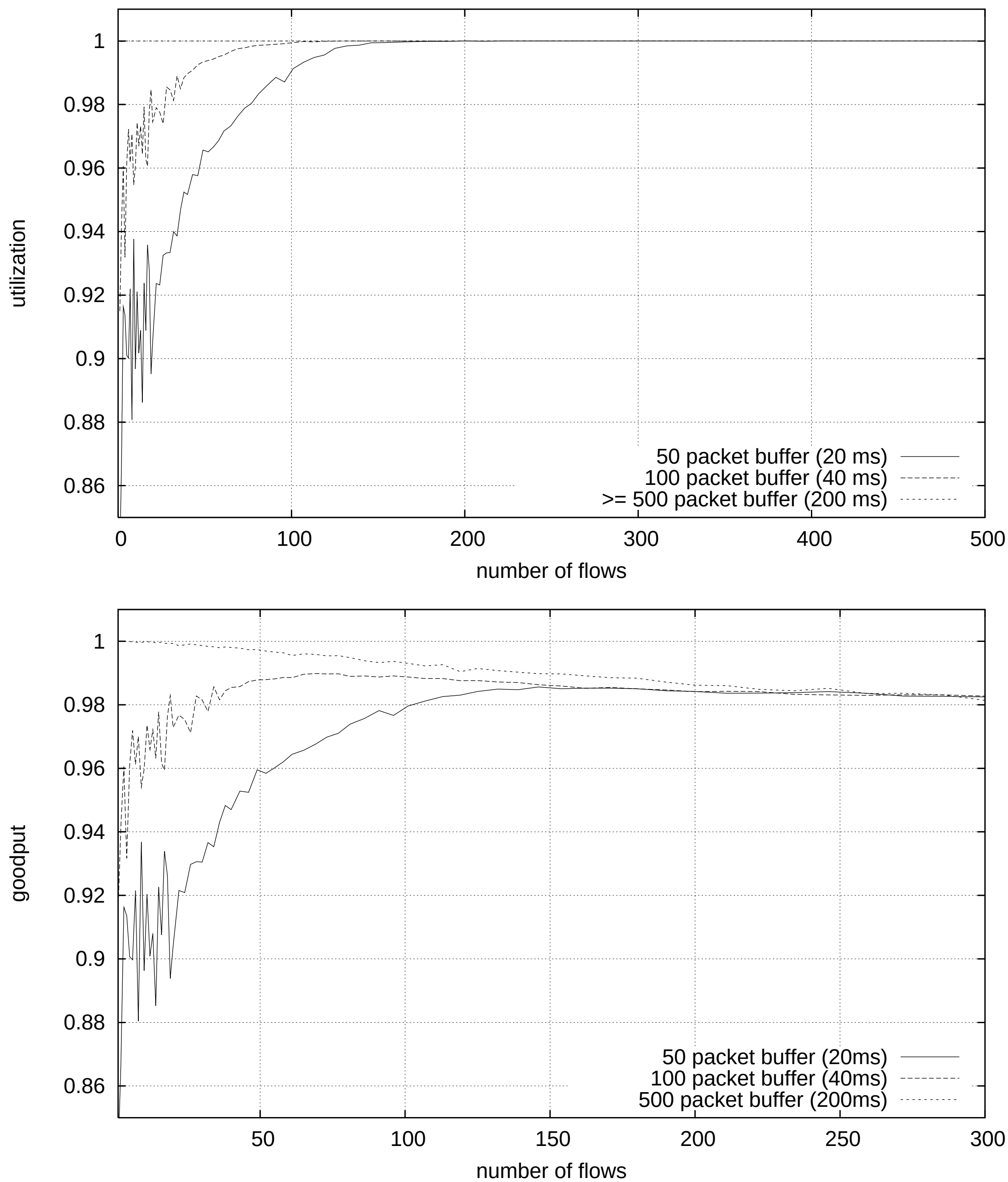


Figure 5.1: Utilization (top) and Goodput (bottom) vs. Number of Flows for different buffer sizes

two-way propagation delay was 130ms and the buffer sizes correspond to 20ms, 40ms and 200ms, respectively.

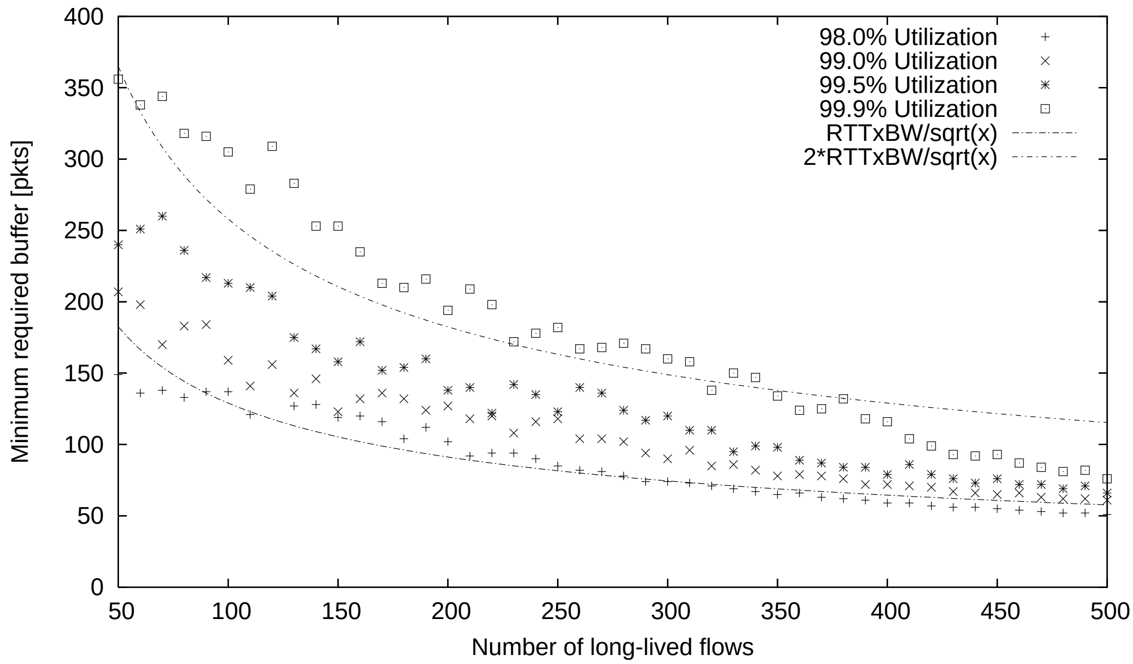
With buffers above $2T_p \times C$, we can achieve full utilization, even with a single flow. However, if we reduce the buffers, a small number of flows is no longer able to achieve full utilization. As we increase the number of flows, and with it the amount of statistical multiplexing, utilization increases until it eventually reaches 100%.

The bottom graph in Figure 5.1 shows goodput achieved in the same experiment. The results are essentially the same. With small buffers, we need a minimum number of flows to achieve full utilization. With large buffers, even a single flow can saturate the link. Goodput decreases as the number of flows increases and it is slightly higher for larger buffers. This is not surprising as more flows lead to smaller TCP windows and thus a higher drop rate [31].

5.2.2 Effect of Small Changes in Buffer Size

We know that utilization depends on buffer size and the number of flows. We have seen in Chapter 3 that we need buffers of approximately $\frac{2T_p \times C}{\sqrt{n}}$ to achieve close to full utilization. The upper graph in Figure 5.2 gives us an idea of the exact relationship between utilization and buffer size. For a specific utilization target and a specific number of flows, we iteratively found the minimum amount of buffering that was needed for an OC3 line. The number of flows was varied between 50 and 500 and we repeated the experiment for 98%, 99%, 99.5% and 99.9% utilization. For comparison, we also plotted lines for buffers of size $\frac{2T_p \times C}{\sqrt{n}}$ and $2 \times \frac{2T_p \times C}{\sqrt{n}}$.

The results largely confirm our intuition. In order to achieve higher Utilization, we need larger buffers. However, for a large number of flows, the additional amount of buffering required to increase utilization from 98% to 99.9% becomes very small. At 400 flows, it is much less than a factor of two. This graph also tells us that for a real router, we should probably use $2 \times \frac{2T_p \times C}{\sqrt{n}}$ as this will guarantee us very close to 100% utilization.



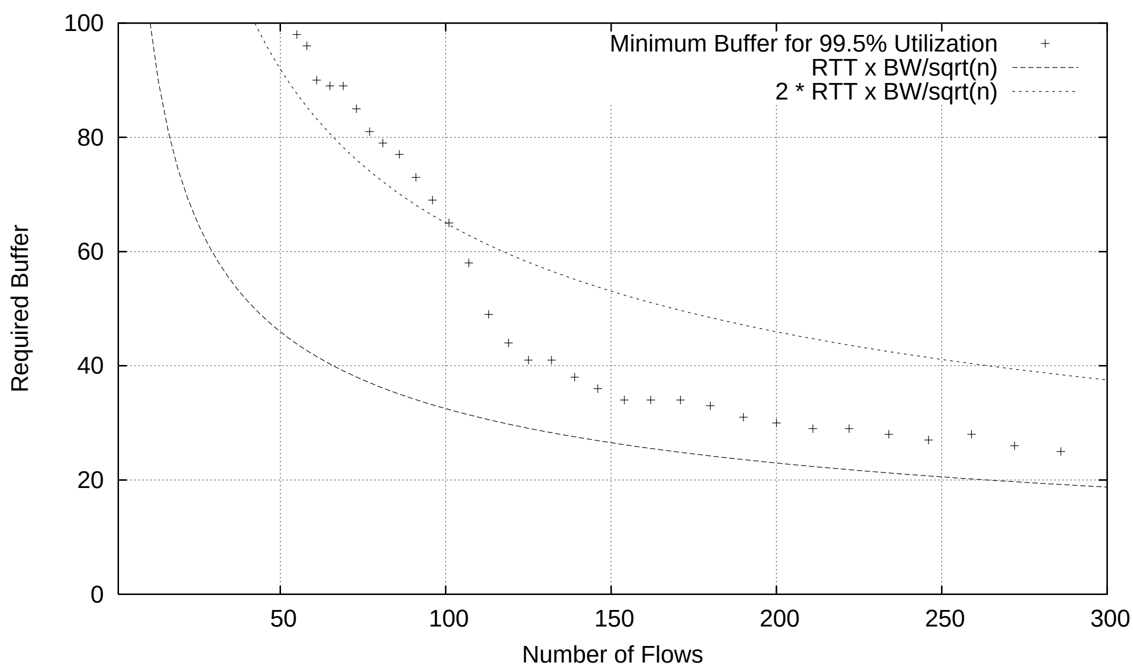


Figure 5.2: Amount of Buffering Required for different Levels of Utilization (Top) and Example of the $\frac{2T_p \times C}{\sqrt{n}}$ rule for a low bandwidth link (Bottom).

5.2.3 Small Delay Bandwidth Products

Our rule seems to hold well above 100 Mb/s, how does it behave for small delay bandwidth products? The lower graph in Figure 5.2 shows the minimum buffer requirement for 99.9% utilization on a 20 Mb/s link. At first glance, it seems that even at this low bandwidth, our model holds well, as long as we have at least 100 flows. However, for this experiment, our delay-bandwidth product is only 325 packets. With 100 flows, the average window size per flow would only be 3.25 packets and we would have frequent timeouts [31]. The insight here is that our new rule-of-thumb holds even in environments where timeouts are frequent, even if this is not a desirable point to operate a network.

We also ran experiments to see what happens in cases where the number of flows n is larger than the delay-bandwidth product in packets. In such a scenario, the average window size drops below one and a majority of flows will be in timeout at any given time. It seems that in this case, we need buffers that are slightly larger than $\frac{2T_p \times C}{\sqrt{n}}$. We did not investigate this further as networks operating under such extreme congestion should be very rare.

5.3 Short Flows

We have seen in Chapter 4 that buffering on uncongested routers can be modeled using M/G/1 model. A key result was that buffering depends only on flow lengths and load. In this Section, we want to verify these results and see how they are influenced by other network parameters.

5.3.1 Effect of Bottleneck Link Bandwidth on Queue Length

Our short flow model predicts that buffering is independent of the line speed. Even though our model made some simplifying assumptions, the results holds extremely well in ns2 simulations. Figure 5.3 shows a plot of the average queue length vs. the flow length. Our traffic consisted of TCP flows of a single length that arrived according to a Poisson process. The average rate of the flows generated a load of

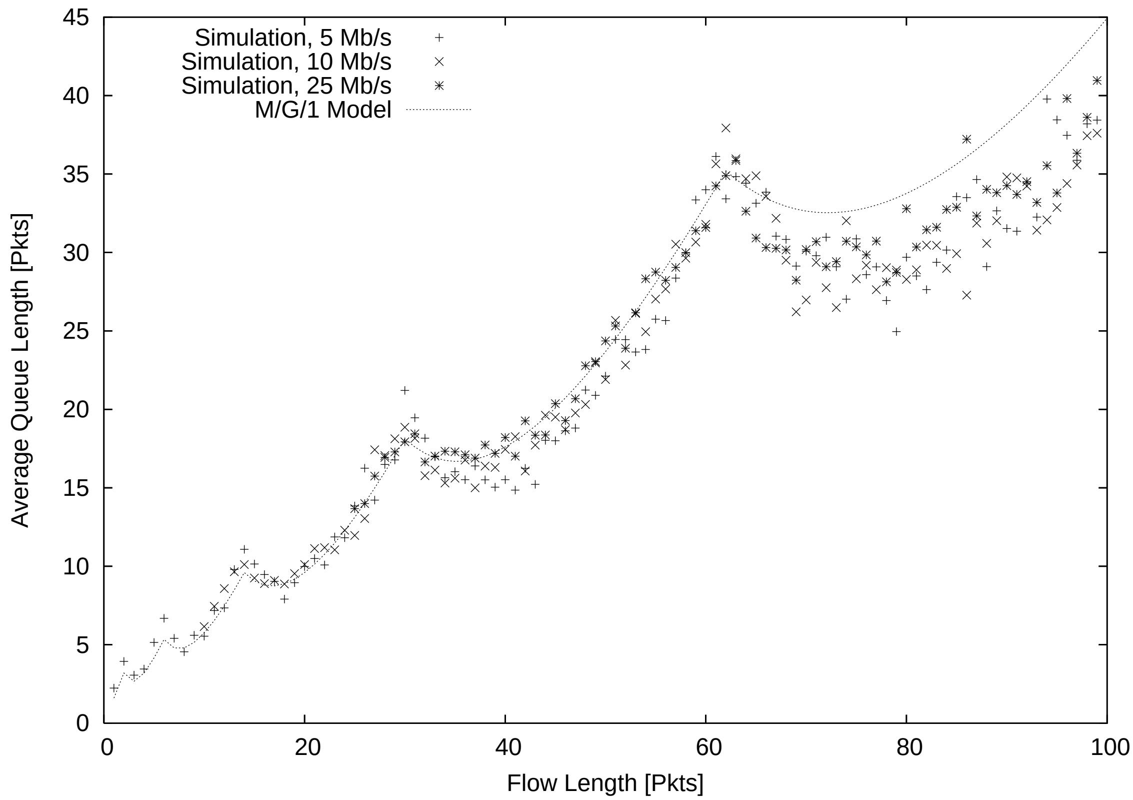


Figure 5.3: Average Queue Length for Short Flows at three Different Bandwidths

about 80% in the link. Access links had a speed of 10 Gb/s to create maximum burstiness. We varied the length of the flows from 1 to 100 and for each length measured over several minutes, the average length of the router's buffer queue. This experiment was repeated for bandwidths of 5 Mb/s, 10 Mb/s and 25Mb/s. In the graph, we also show the average queue length as predicted by the M/G/1 model from Chapter 4.

We can see from the graph that the model matches the actual measured average queue length very well. More importantly, however, the results from the measurements at three different bandwidths are almost identical. We measured queue occupancies at line rates of up to 100 Mb/s and found no substantial difference in queue length.

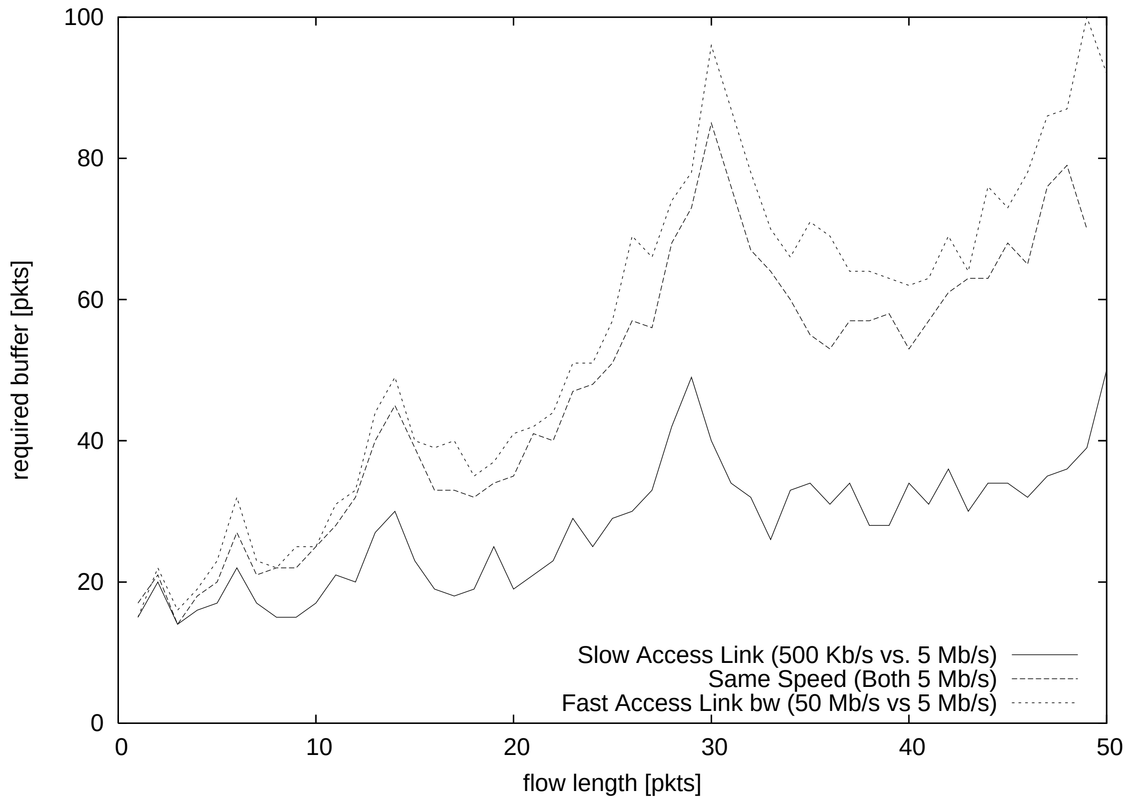


Figure 5.4: Effect of the Access Link Bandwidth on the Queue Length

5.3.2 Effect of Access Link Bandwidth on Queue Length

In most simulations, we assume that the access link has infinite speed. In practice, the access link to the router is of finite speed, and might be slower than the bottleneck link. We would expect that a slower access link would smooth out packet bursts and reduce the length of the queues. As long as the access link is at least as fast as the bottleneck link, this effect should be minimal. However, if the access link is slower than the bottleneck link, the packets from a burst are spaced out on the slow access link and arrive as individual packets at the bottleneck link, where they no longer generate queueing.

In practice, this intuition turns out to be correct. Figure 5.4 shows for a fixed maximum loss rate the minimum required buffer vs. the flow length. The bottleneck link has a bandwidth of 5 Mb/s. We repeated the experiment three times, once with access link speeds of 500 Kb/s, 5 Mb/s and 50 Mb/s. As we would expect, faster access links require more buffering as they preserve the burstiness of TCP the best. However, the difference between an access link that is faster than the bottleneck link and an access link that has the same speed as the bottleneck link is fairly small. Increasing the access link speed further does not change the results substantially. We can, therefore, treat access links that are at least as fast as the bottleneck link as having infinite speed. However, with a 500 Kb/s access link that is ten times slower than the bottleneck link, we require much less buffering. This is primarily of interest for the core of the network. Access links to a 10 Gb/s link are at 1 Gb/s or below, which would allow us to reduce the buffer for such links by a factor of two.

5.3.3 Effect of Picking Buffers that are too Small

The goal of our model is to predict the amount of buffering needed for traffic consisting of only short or otherwise constraint flows on an uncongested router. One important question is what happens if we pick a wrong buffer size. Figure 5.5 shows the effect of buffering on the flow completion time of short flows. We took a bottleneck link of 50 Mb/s and generated short flows of a fixed length that arrived according to a Poisson process. We then measured the flow completion time of these short flows over several minutes. We varied the arrival rate of the flows to create a load on the bottleneck link between 10% and 100%. This experiment was repeated for buffer sizes of 10, 20, 40, 80 and the bandwidth delay product of 644 packets.

At very light loads, the average flow completion time (AFCT) is a constant 350 ms. This is independent of buffer size as there is almost no queueing and the queue never runs over. As load increases, so does the queue size and very small buffers increase the probability of drops, which causes retransmits or time-outs and increases the AFCT. Retransmits also increase the total amount of traffic sent over the bottleneck link as now some packets have to be sent multiple times [1]. This effectively increases the utilization of the bottleneck link above the offered load. As the load increases, at

[1] We define the load as the amount of data the senders try to send divided by the capacity of the bottleneck link, not as the actual data sent over the link divided by the capacity

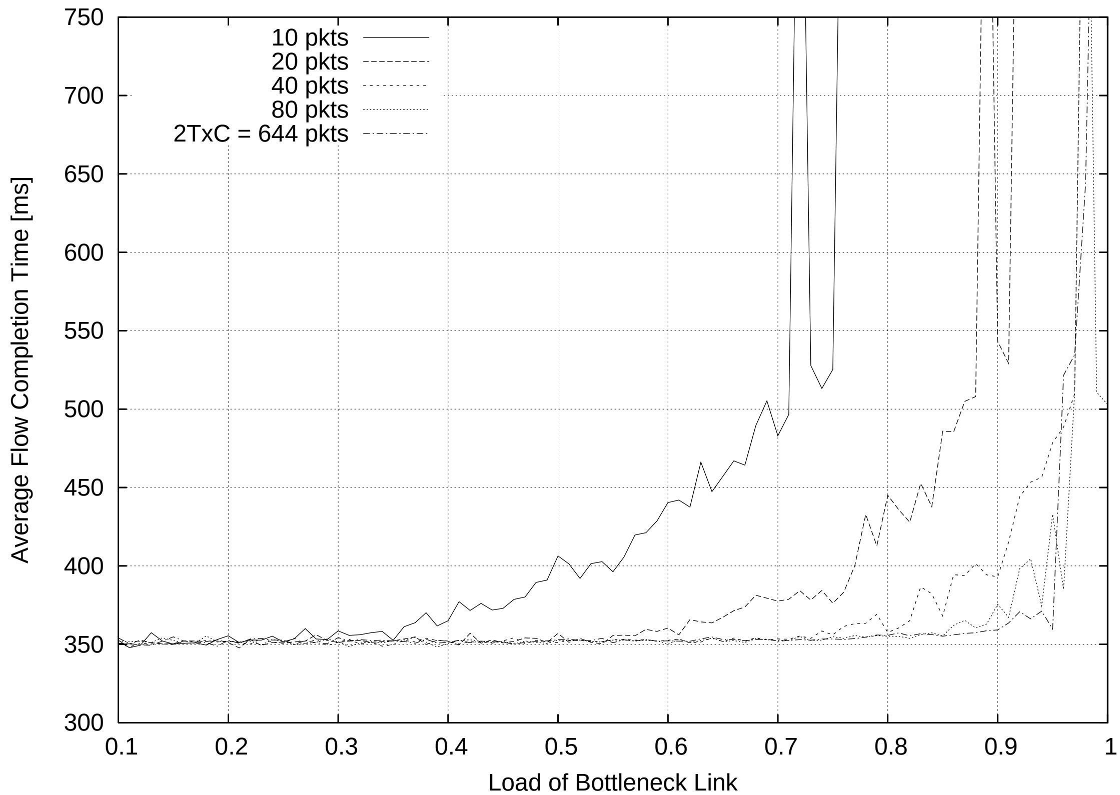


Figure 5.5: Average Flow Completion Time for short flows as a function of the Load of the Bottleneck Link and the amount of Buffering.

some point the rate of retransmits plus new flows will surpass the bottleneck capacity and the system will become unstable. This will happen earlier for small buffer sizes. For a buffer of only 10 packets, load cannot increase beyond 75% before a substantial number of flows no longer compete. Picking too small a buffer carries two penalties. First, it will increase flow completion times and second, it lowers the critical load at which the system becomes unstable.

In practice, this is of limited relevance. Even with buffers of only 40 packets, we can reach up to 90% load with little increase in AFCT. For a high-speed router, 40 packets of buffering is small and probably far below what is needed for long flows. And running a router that serves non-congestion-aware traffic at 90% would be a bad idea as a small error in estimating the load would make the system unstable.

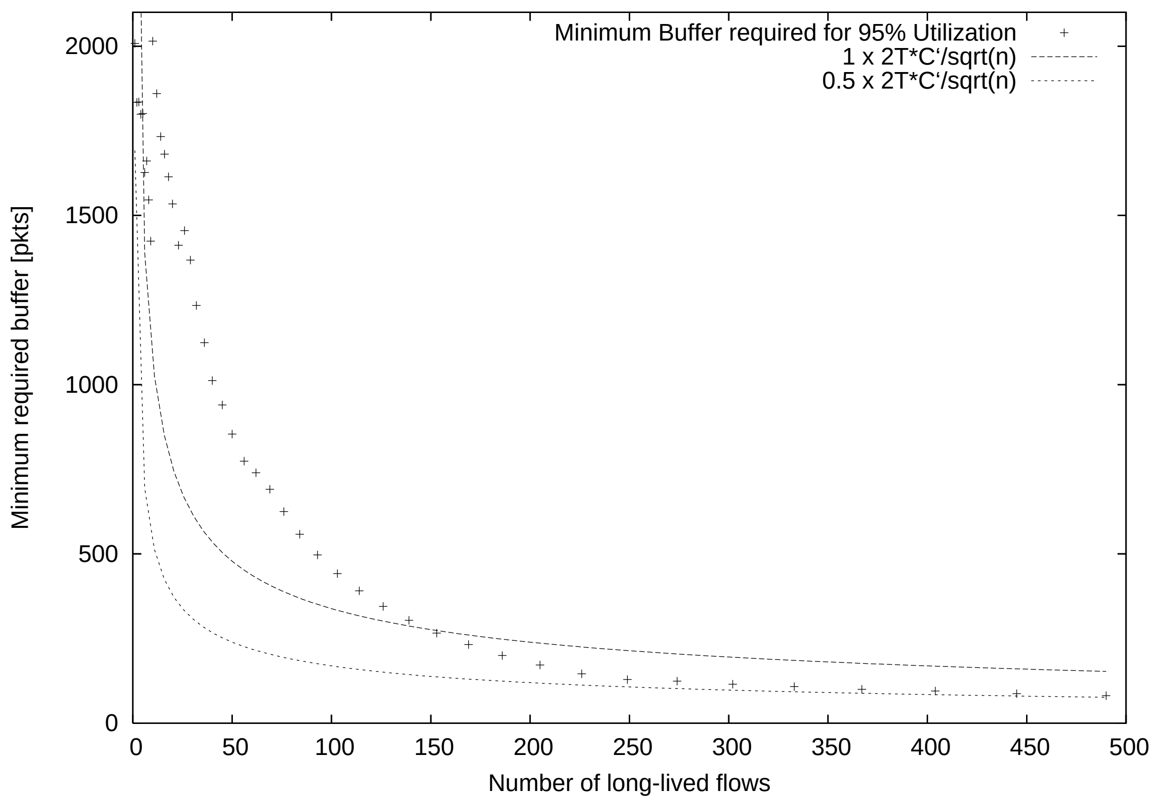


Figure 5.6: Minimum Required Buffer for a Mix of 20% Short Flows and 80% Long Flows.

5.4 Mixes of Flows

So far, we were looking at either only infinitely long flows, or only short flows on an uncongested router. The next logical step is to combine the two and see if our results still hold.

5.4.1 Buffer Requirements for Mixes of Flows

We postulated in Chapter 4 that in mixes of long and short flows, the effect of long flows dominates. We experimentally verified that this is the case and a sample result is shown in Figure 5.6. In this experiment, we generated 10 Mb/s of short flows that arrived at a bottleneck link according to a Poisson process. The bottleneck link had a capacity of 50 Mb/s, Therefore, 20% of the link's bandwidth was at any given time, taken up by short flows. We created a number of infinite length TCP flows that sent data over the bottleneck link that took up the remaining 80% of the bottleneck link.

The first thing we observed is that in such a mix of long and short flows, the short flows will always claim their share of the bandwidth with fairly minimal losses. The reason for this is the much more aggressive multiplicative increase of the short flows vs. the additive increase of the long flows in congestion avoidance mode.

More importantly, we see from Figure 5.6 that a buffer size of $\frac{2T_p \times C}{\sqrt{n}}$ is also sufficient for mixes of flows. In fact, in a mix of flow, we can calculate buffer size for both types of traffic separately and add them to obtain an upper bound of the total buffer. As the buffering needed for short flows is much smaller than the buffering needed for long flows, this effectively means we can calculate what fraction of the overall bandwidth comes from long flows and then use this reduced bandwidth to calculate the amount of buffering required. In this experiment, the delay-bandwidth product was 4230 packets. As 80% of the traffic is from long flows, we use a reduced delay-bandwidth product of $C' = C * 0.8$ to calculate our minimal required buffer. As we can see in the graph, the minimum required buffer we found experimentally fits very well with the amount of buffering predicted by our model.

It is also interesting to see that our new rule already holds at about 150 flows. We think that this is due to short flows causing long flows to desynchronize. In other words, mixes of long and short flows require *less* buffer than traffic consisting only of long flows.

5.4.2 Flow Completion Times for Short Flows

One argument raised against small buffers is that the higher loss rate hurts short flows that experience time outs and retransmits. In reality, it turns out that the opposite is true. Figure 5.7 shows the average flow completion time (AFCT) of short flows. We started with Poisson arrivals of short flows totaling 20% of the bandwidth of a bottleneck link. The link is shared with between one and 500 infinite-sized long-lived flows. For each number of long-lived flows, we measured the AFCT once with buffers

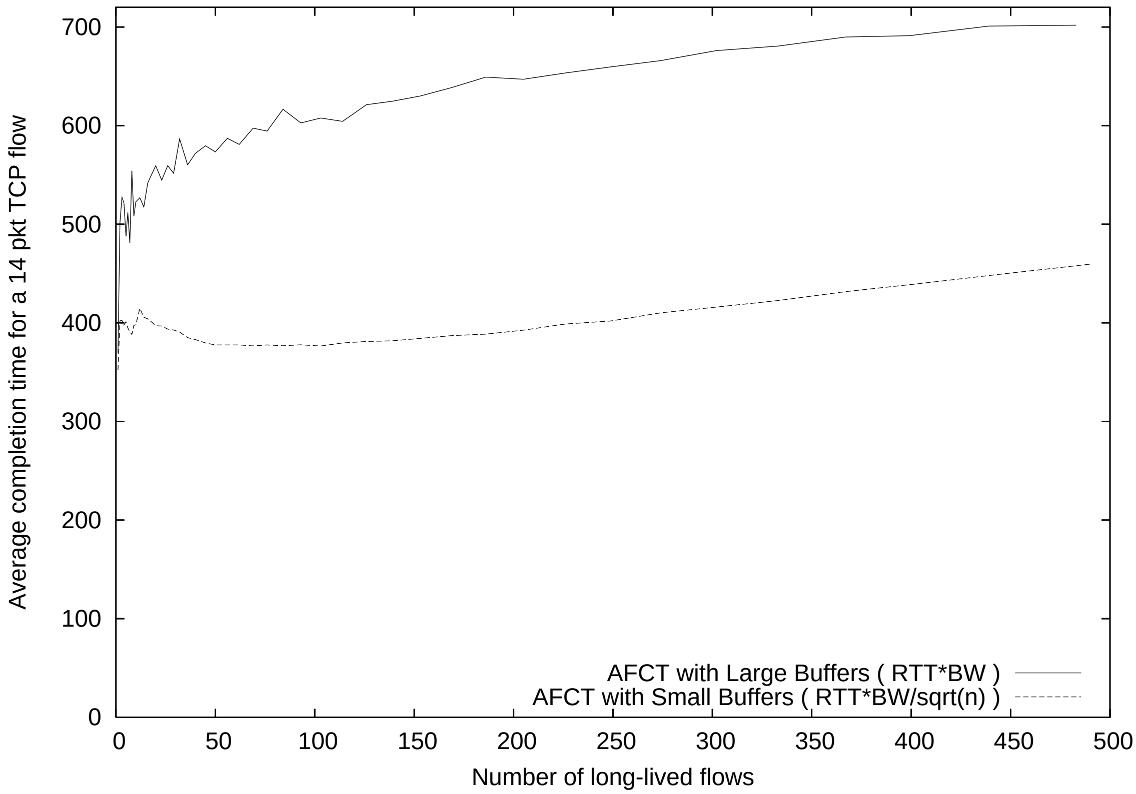


Figure 5.7: Average Flow Completion Time for Large ($2T_p \times C$) and Small ($\frac{2T_p \times C}{\sqrt{n}}$) Buffers.

of $2T_p \times C$ and once with buffers of $\frac{2T_p \times C}{\sqrt{n}}$. The results show that with smaller buffers, the flows complete much faster. This might be surprising at first, after all, smaller buffers will cause an increase in packet drops and thus, retransmits and time-outs. However, smaller buffers also decrease the RTT of the flows. A router with large buffers that is congested will have full or nearly full buffers all the time. This causes the RTT of flows to be:

$$RTT_{max} = 2T_P + \frac{\overline{Q}}{C} \approx 2T_P + \frac{2T_P \times C}{C} = 4T_P$$

With buffers of $\frac{2T_P \times C}{\sqrt{n}}$ the RTT will only be:

$$RTT_{min} = 2T_P + \frac{\overline{Q}}{C} \approx 2T_P + \frac{\frac{2T_P \times C}{\sqrt{n}}}{C} = 2T_P\left(1 + \frac{1}{\sqrt{n}}\right)$$

For large n, the term $\frac{1}{\sqrt{n}}$ becomes very small and the RTT for small buffers is close to only half the RTT for large buffers.

One retransmit will usually add one additional RTT to the flow completion time of a flow. The 14 packet flow in the experiment has a length of $4 \times RTT$ without losses. With small buffers and a halved RTT, the flow can incur up to four retransmits and still finish as quickly as a flow with large buffers, twice the RTT and no packet losses. For four retransmits in a 14 packet flow, we would need a loss rate of more than 20%! Smaller buffers can in the worst case scenario, quadruple the loss rate of TCP vs. full buffers of one delay-bandwidth product [31]. We can therefore argue that if the loss rate with full buffers is below 5%, smaller buffers will always lower the AFCT.

The above calculation is an oversimplification as it ignores time-outs and the details of the TCP retransmit algorithm. However, it seems fair to conclude that for typical loss rates in the internet, smaller buffers will always reduce the completion time of short TCP flows.

5.4.3 Effect of Changes in Long/Short Ratio

We have seen that our new rule-of-thumb overall holds for mixes of long and short flows. However our model only provides us with bounds on how much buffering is needed. Figure 5.8 shows the actual effect of changes in the ratio of short flows to long flows on buffer requirements and the average flow completion time.

The setup is similar to the last experiment. We generated a fixed amount of short flow traffic that utilized a certain percentage of a bottleneck link. We then added a number of infinite-sized long lived flows and iteratively found the minimum buffer size to meet a specific utilization goal. For this buffer size, we then measured the average flow completion time. We repeated this for traffic that consisted of 10% short flows, 20% short flows, 30% short flows. For comparison, we also ran the experiment with large buffers of $2T_p \times C$ and 20% short flows.

The top graph in Figure 5.8 shows the minimum buffer required to achieve 95% utilization for traffic consisting of 10%, 20% and 30% short flows. We can see that our model holds well in all cases. What is surprising though is that the difference in required buffering becomes virtually zero as soon as we are above 110 flows. The explanation for this can be found using our short flow model. If we consider the short flow traffic alone, without the long flows, we can determine the minimum amount of buffering needed for the short flows. It is plotted in the graph as the horizontal line at about $y = 100$. At around $n = 110$, the amount of buffering needed for 100% utilization drops below this threshold. What this means is that at this point, more and more short flows experience drops and are forced out of slow-start into congestion avoidance mode. As we increase n, soon all flows are in congestion avoidance mode and are effectively "long" flows. As we can see, the transition of the buffer dropping below the short flow threshold and the differences in flow lengths disappearing coincides very well.

With the amount of buffering being almost equal, the AFCT shown in the lower graph of Figure 5.8 only differs by a few percent. However, in any of the three settings, it remains much smaller than the AFCT with full buffers of $2T_p \times C$.

Overall, we can conclude that the amount of required buffer and the change in AFCT is not very sensitive by small changes in the ratio of long vs. short flows. Buffer requirements in mixes of flows are almost entirely driven by the number of long flows in congestion avoidance mode.

5.5 Continuous Flow Length Distributions

Real network traffic contains a broad range of flow lengths. The flow length distribution is known to be heavy-tailed [18] and in our experiments, we used Pareto distributions to model it. The flow arrival process is Poisson as in previous experiments. This simulation was done with a 155 Mb/s link and RTT of approximately 100ms.

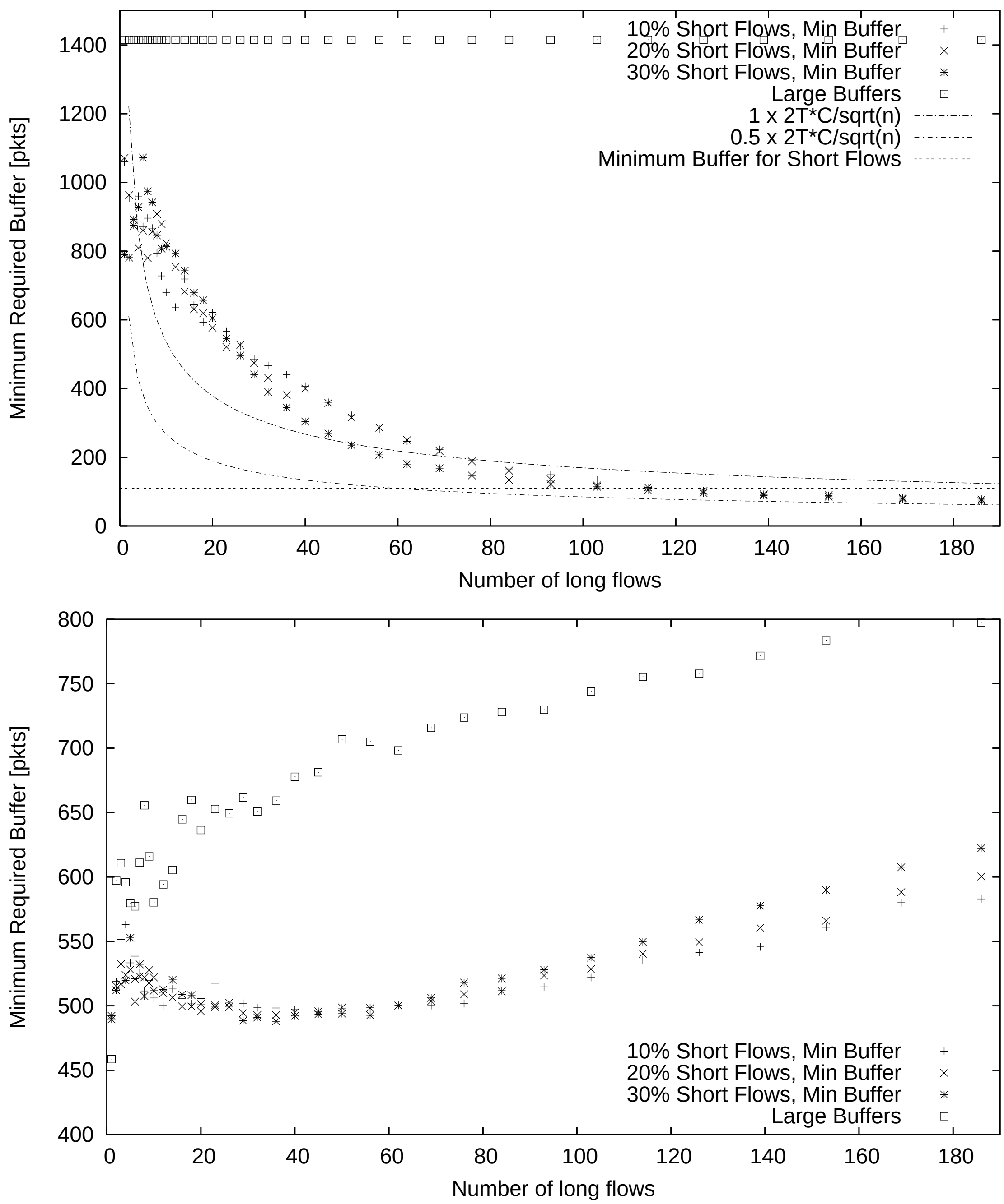


Figure 5.8: Utilization (top) and Average Flow Completion Times (bottom) for different Ratios of Long and Short Flows

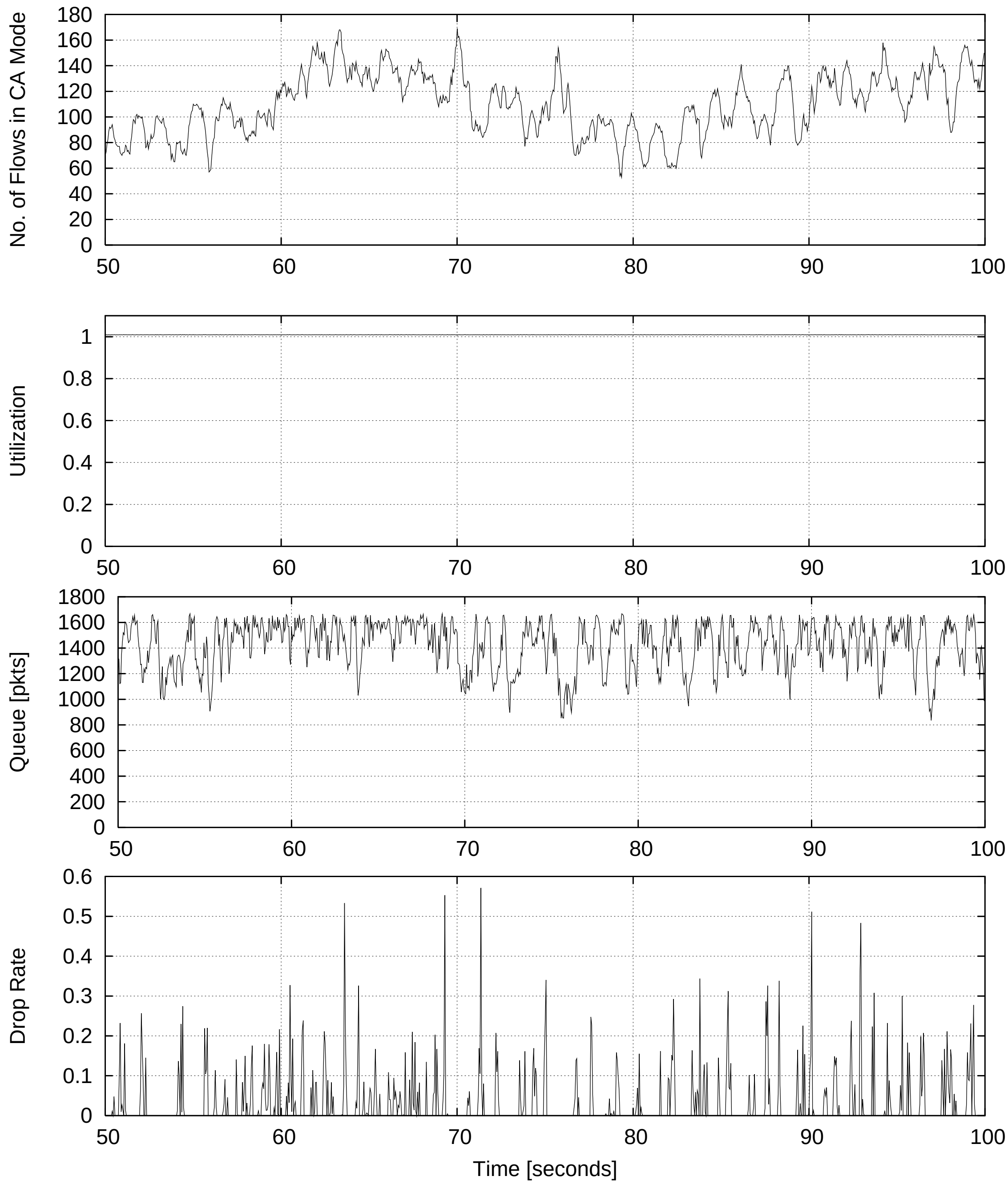


Figure 5.9: Number of Flows (Top), Utilization, Queue Length and Drops (Bottom) for Pareto Distributed Flow lengths with large Buffers of $2T_p \times C$

Figure 5.9 shows what happens for Pareto distributed flow lengths with a large buffer of size $\frac{2T_p \times C}{\sqrt{n}}$. We want to get an idea of the worst case congestion and we chose a load of just above 100%. A load above 100% might seem like a hopeless choice as it can not possibly be served by the router. However, in practice it simply means that a few heavy flows will not terminate for the duration of the experiment.

The top graph shows the number of flows in congestion avoidance mode. It fluctuates between 60 and 160 flows with an average of about 110 flows. The second graph shows the utilization, it was 100% for the entire duration of the experiment. The queue length shown in the third graph fluctuates between 800 and the maximum of 1600 packets. It never drops below 800 packets. Clearly, this buffer is too large and could be reduced by at least 800 packets. The bottom graph shows the loss rate for each 50ms interval. Drops occur only in some intervals, however, for some intervals, the drop rate goes as high as 50%. On average, the loss rate is around 1%. This is a very high loss rate for TCP however, it still works well at this rate.

To pick a reduced buffer size, we need to find the number of concurrent long flows n. Looking at the top graph of Figure 5.9, we pick $n = 100$ and size the buffer to $\frac{1}{10}$th of its original size or 165 packets. Note that this is not an intuitive choice. The buffer fluctuation that we can see in the queue length graph of Figure 5.9 is about 800 packets.

It does turn out to be the correct choice though. The top graph of Figure 5.10 shows the number of concurrent flows with the smaller buffers. We can see that the number of flows stays above 100 all the time, justifying our choice of $n = 100$. The second graph in Figure 5.10 shows the utilization. We can see that the utilization is close to 100% almost all the time. Averaging shows that the utilization is above 99%. The queue length shown in the third graph oscillates between zero and the buffer size of 165. This tells us that the buffer is fully used and gives us an indication that we are probably close to the minimum buffer. The drop rate shown in the lowest graph is quite different from the large buffer case. Overall, the drop rate is now higher (but still a low single digit percentage), however more intervals have drops and the maximum drop rate in a single interval is lower.

The most important result for this experiment is that we can achieve a utilization

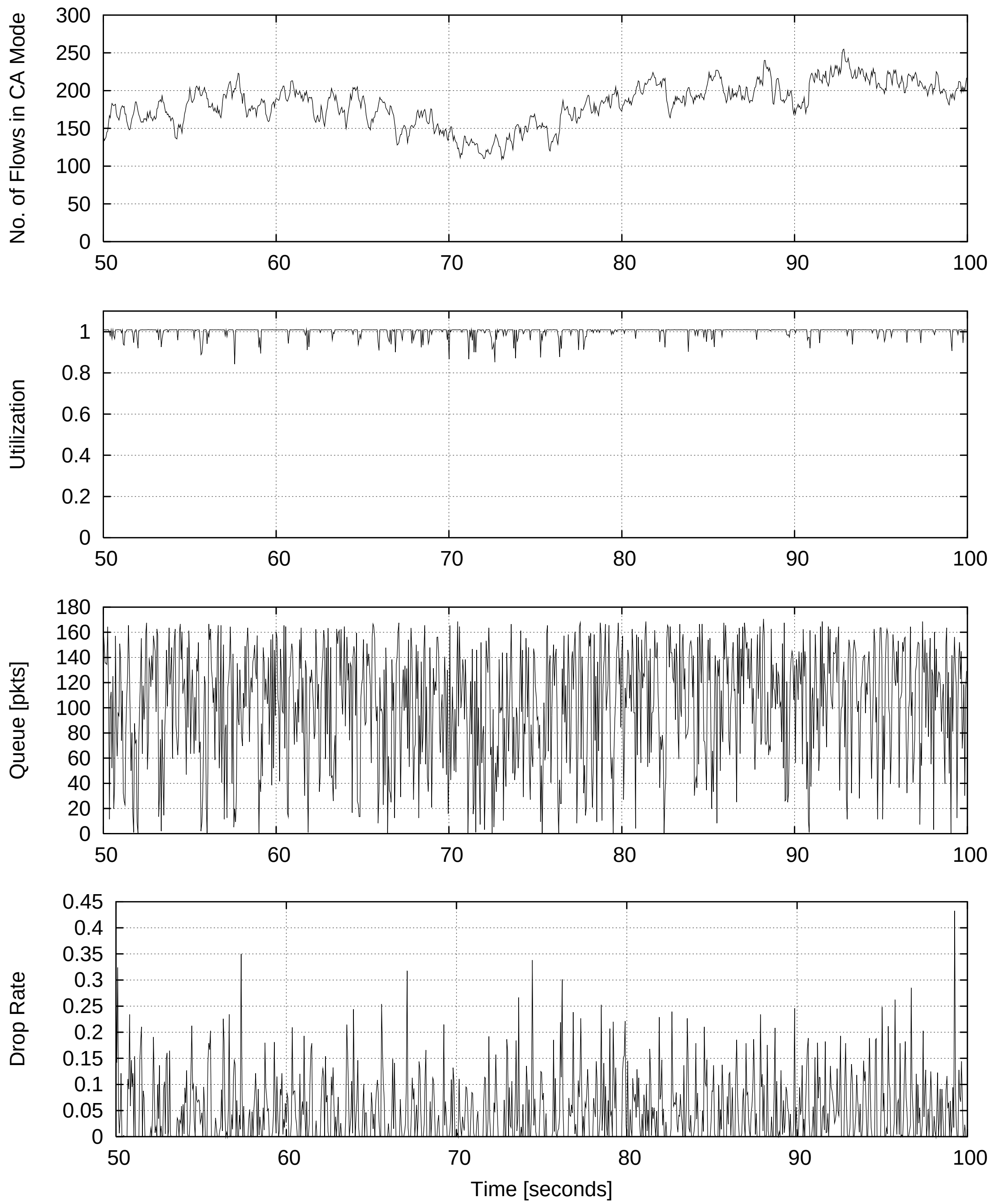


Figure 5.10: Number of Flows (Top), Utilization, Queue Length and Drops (Bottom) for Pareto Distributed Flow lengths with small Buffers of $\frac{2T_p \times C}{\sqrt{n}}$

of 99% with buffers of size $\frac{2T_p \times C}{\sqrt{n}}$. The main change from the long flow case is that now we have to measure or estimate the number of long-lived flows. We have also seen that smaller buffers increase the overall loss rate and decrease the probability of very large loss events.

One might be surprised that the number of long flows is this large for Pareto distributed traffic. It has been argued [19] that in a network with low latency, fast access links and no limit on the TCP window size, there would be very few concurrent flows. In such a network, a single very heavy flow could hog all the bandwidth for a short period of time and then terminate. But this is unlikely in practice, unless an operator allows a single user to saturate the network. And so long as backbone networks are orders of magnitude faster than access networks, few users will be able to saturate the backbone anyway. Even in our simulation where we have unlimited window sizes and very fast access links, TCP is not capable of utilizing a link quickly due to its additive increase behavior above a certain window size. Traffic transported by high-speed routers on commercial networks today [18, 2] has 10s of 1000s of concurrent flows and we believe this is unlikely to change in the future.

5.6 Summary of the ns2 Simulation Results

To summarize, we found in this chapter that our models hold well for a wide range of network parameters and traffic characteristics. We have seen that in traffic that consists of flows of different lengths, the effect of long flows dominates. In practice, we can size the buffer correctly by finding the number of flow in congestion avoidance mode and using it as the n from our long flow model.

Chapter 6

Experimental Results on Physical Routers

The last Chapter examined the experimental verification of our models using ns2 simulation. This Section verifies our model with experiments performed on actual, physical routers with TCP traffic generated by real TCP stacks.

While ns2 gives us valuable insights into how buffers and network traffic interact, it also fails to capture a number of characteristics of real networks:

- **Synchronization**. As previously discussed in Section 3.1, synchronization in ns2 is very different from synchronization on physical routers. In a laboratory experiment of a shared memory router, three TCP flows exhibited no signs of synchronization.

- **Packet Timings**. In ns2, all packet timings are deterministic. In a real network, there are many sources of randomization and burstiness such as context switching or busy intervals on end hosts or shared memory routers, routing messages, retransmits of packets by the link layer, etc.

- **Router Architecture**. ns2 assumes an idealized output queued router. Real routers can incur additional queueing delay due to congestion of the switching mechanism.

- **Other Protocols**. We assume only TCP flows with some length distribution in our simulation. Real networks will have a variety of traffic patterns and protocols.

The first set of experiments tries to evaluate the effect of the first three of the above issues. The experiments were done on a Cisco GSR [36] router in a laboratory setting at the University of Wisconsin-Madison. This setup allows us to create specific traffic patterns and directly compare the predictions of our model and ns2 with a physical router.

The other two experiments were done on live production networks. The key goal here is to verify the effect of different applications, protocols, packet types and usage patterns. The Stanford experiment measures performance of a congested router. The Internet2 experiment verifies our results for an uncongested router.

6.1 Laboratory Experiment on a CISCO GSR

6.1.1 Introduction and Setup

The goal of this experiment is to compare quantitatively our buffer model using a physical router and traffic generated by real TCP/IP stacks from current operating systems in a laboratory setting where we have full control over all parameters.

The topology of the extperimental network is as follow. Traffic is generated by a cluster of PCs running either Linux or BSD operating systems. Network traffic from the PCs is aggregated using a Cisco 6509 Ethernet switch with Gigabit Ethernet links. A Gigabit Ethernet link connects the switch to a Cisco Catalyst 12410 router [36] (this platform is also known as GSR). The GSR is connected to a second identical GSR with a 4 x OC3 POS “Engine 0” line card that switches IP packets using POS (PPP over Sonet) at 155Mb/s. Between the two GSRs is a delay generator that allows us to set arbitrary transmission delays between the routers. The second GSR distributes traffic via a second Ethernet switch to a number of Harpoon TCP traffic sinks. A separate network to the routers is used to poll data from the routers using SNMP commands.

The 1Gb/s links of the network aggregating traffic is, by a factor of 60, faster than the speed of the bottleneck link. Congestion will therefore, only happen on GSR interface serving the OC3 line. Reverse traffic consists only of ACK packets and will not cause congestion anywhere.

TCP traffic was generated using the Harpoon traffic generator [41]. Harpoon allows us to specify flow arrival patterns as well as flow lengths. The flows are real TCP flows that react to congestion and will terminate once all data has been transmitted.

For the delay generation, we originally used a PC running the Dummynet [34] network simulator. However, after running a first set of experiments, we found that Torment would sometimes queue traffic for several milliseconds, and then send it as a burst. The result was traffic with very different characteristics than the original network traffic. For example, the mean queue length was several times longer than without Dummynet. We replaced Dummynet with an Adrea delay generator and the burstiness in the traffic disappeared.

The goal of the experiment is to verify quantitatively both the long flow as well as the short flow model. For this, we need to measure:

- The capacity C. The line rate of an OC3 is 155 Mb/s, however after PPP and SONET encoding, we measured an effective throughput of 149.26 Mb/s. This is very close to the theoretical value.

- The number of flows, flow arrival pattern and overall load is directly controlled by the Harpoon traffic generators.

- The round-trip propagation delay is controlled by adjusting the AdTech Delay Generator and measured end-to-end.

- Utilization of the bottleneck link is measured on the GSR itself via two mechanisms: Netflow and byte counters on the interface that are polled via SNMP.

- The queue length Q can be polled on the GSR via the DOS command prompt.

6.1.2 The Cisco Catalyst 12000 Router

The Cisco 12000 [36] (or GSR) series of routers are designed to switch traffic at line rates of 10 Gb/s and primarily targeted for use in the core of the network and large edge nodes. The GSR is build as a central switching fabric that is connected to a number of line cards. An additional controller card is used for central functions such as routing protocols and configuration. For a complete description of the GSR architecture see [7].

A slightly simplified path of a packet passing through the GSR router takes the following path:

- Packet arrives on the physical interface. The line card inspects the packet and determines the packet's destination line card.
- The packet is queued in the input queue (called "ToFab" or ToFabric buffer on the GSR) until the switching fabric becomes available.
- The switching fabric switches the packet to the output line card.
- On the output line card, the packet is queued in an Output Queue (called FromFab buffer on the GSR).
- Once the output interface becomes available, the packet is de-queued and sent out on the network.

The switching fabric of the GSR is designed to switch traffic at rates of up to 10 Gb/s per line card. For our experiment, traffic through the router is limited to 155 Mb/s and we would expect that the switching fabric is never the bottleneck. We verified this experimentally and found the ToFab buffers to be empty all of the time.

The GSR keeps different buffer pools for different packet lengths. The relative size of these pools is centrally determined for the whole router and depends on the MTU sizes of the connected interfaces. This "carving" process takes place whenever a line card is added or removed or MTU sizes are changed. As we can not control the carving process, it is difficult to limit the buffer size directly.

On the “Engine 0” line card that we are using, it is possible to set directly the maximum queue lengths of the FrFab queue [1].

During our measurements, we initially were puzzled by the fact that we seemed to overestimate the queue length by a constant offset of about 43 packets or 64 kBytes. After contacting the router manufacturer, we found out that the GSR 4xOC3 Engine 0 line card has a FIFO buffer of 128 kByte on the interface management ASIC, of which in our setup the first 64 KByte are used. With an MTU of 1500 Bytes, this creates an additional 43 packets of buffering. Packets in this buffer are not visible to IOS and therefore explain the under reporting.

With the exception of the utilization, measuring the data we need on the GSR is fairly simple. The GSR allows us to poll directly the queue length of the ToFab and FrFab queue on the line card. Measuring utilization however, is difficult. The GSR has packet and byte counters, which together with a time stamp can convert into line rates. The problem is that the counters are on the line cards while the clock providing the time stamp is on the central controller. The counter on the line cards is sent to the central controller only once every 10 seconds. We tried polling the counters exactly once every 10 seconds, however, the timing jitter of the SNMP commands introduces a statistical error of several percent for each measurement.

We address this problem by a two-pronged approach. First, we measure utilization over several minutes, which reduces the statistical error to about 0.1%. Second, we use netflow records [11] to get a second estimate for the utilization. If Netflow differs too much from the SNMP byte counter measurements, we discard the experiment.

6.1.3 Results and Discussion

Long Flows

Figure 6.1 shows the results of the long flow measurements. The router memory was adjusted by limiting the length of the interface queue on the outgoing interface. The buffer size is given as a multiple of $\frac{\overline{RTT} \times C}{\sqrt{n}}$, the number of packets and the size of the

[1] Engine 0 line cards accept the IOS “TX-Queue limit” command, reportedly current engine 1 and engine 2 line cards do not support this feature. On these more modern cards, it might be possible to achieve the same effect with class-based shaping. See Section 6.2 for a detailed description.

TCP	**Router Buffer**			**Link Utilization (%)**		
Flows	$\frac{RTT \times BW}{\sqrt{n}}$	**Pkts**	**RAM**	**Model**	**Sim.**	**Exp.**
100	0.5 x	64	1 Mbit	96.9%	94.7%	94.9%
100	1 x	129	2 Mbit	99.9%	99.3%	98.1%
100	2 x	258	4 Mbit	100%	99.9%	99.8%
100	3 x	387	8 Mbit	100%	99.8%	99.7%
200	0.5 x	46	1 Mbit	98.8%	97.0%	98.6%
200	1 x	91	2 Mbit	99.9%	99.2%	99.7%
200	2 x	182	4 Mbit	100%	99.8%	99.8%
200	3 x	273	4 Mbit	100%	100%	99.8%
300	0.5 x	37	512 kb	99.5%	98.6%	99.6%
300	1 x	74	1 Mbit	100%	99.3%	99.8%
300	2 x	148	2 Mbit	100%	99.9%	99.8%
300	3 x	222	4 Mbit	100%	100%	100%
400	0.5 x	32	512 kb	99.7%	99.2%	99.5%
400	1 x	64	1 Mbit	100%	99.8%	100%
400	2 x	128	2 Mbit	100%	100%	100%
400	3 x	192	4 Mbit	100%	100%	99.9%

Figure 6.1: Comparison of our model, *ns*2 simulation and experimental results for buffer requirements of a Cisco GSR 12410 OC3 line card.

RAM device that would be needed. We subtracted the size of the internal FIFO on the line-card (see Section 6.1.3). *Model* is the lower-bound on the utilization predicted by the model. *Sim.* and *Exp.* are the utilization as measured by a simulation with *ns*2 and on the physical router respectively. For 100 and 200 flows, there is, as we expect, some synchronization. Above that the model predicts the utilization correctly within the measurement accuracy of about $\pm 0.1\%$. *ns*2 sometimes predicts a lower utilization than we found in practice. We attribute this to more synchronization between flows in the simulations than in the real network.

The key result here is that model, simulation and experiment all agree that a router buffer should have a size equal to approximately $\frac{\overline{RTT} \times C}{\sqrt{n}}$, as opposed to $\overline{RTT} \times C$ (which in this case would be 1291 packets). However, we can also see that our model predicts the transition point where utilization drops from 100% to values below 100% fairly accurately.

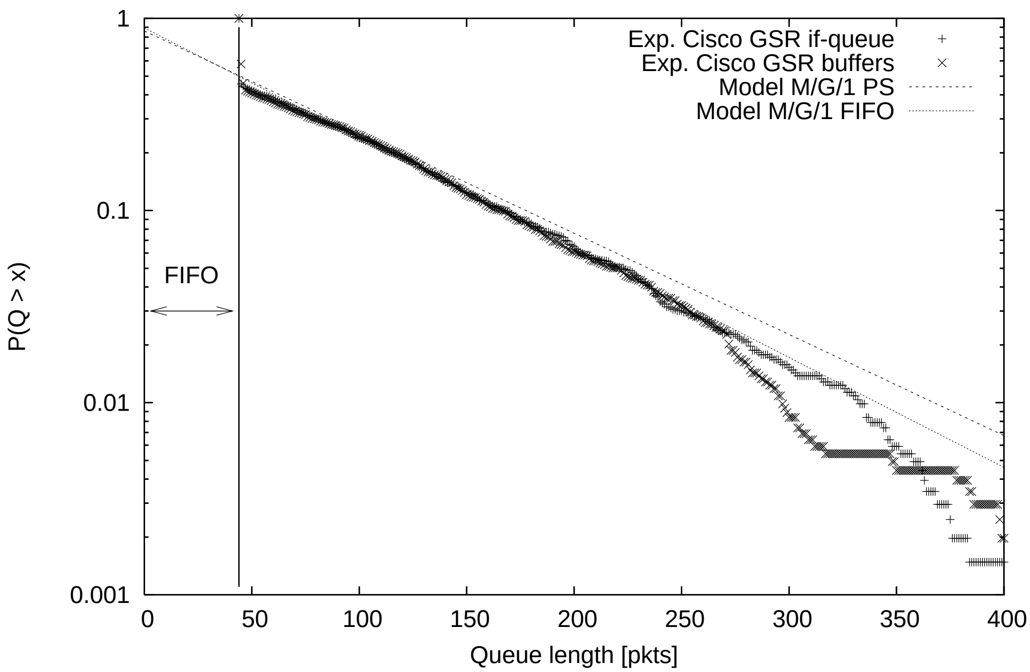


Figure 6.2: Short Flow Queue Distribution of 62 packet flows measured on a Cisco GSR compared to model prediction

Short Flows

In Section 4.1, we used a M/G/1 model to predict the buffer size we would need for short-lived, bursty TCP flows. To verify our model, we generated short-lived flows and measured the probability distribution of the queue length of the GSR. We repeated the experiment for flow lengths of 2, 14, 30 and 62 packets. The load for all three experiments was around 85%. For each case, we measured flow length using two different methods. First, we measured the number of packets in the output queue. Second, we measured the overall amount of buffering that was occupied. In practice, both methods agree very well. All measured queue lengths have to be adjusted by 43 packets to account for the FIFO buffer in the interface management chip.

Figure 6.2 shows the results for 62 packet flows compared to our M/G/1 model and the M/G/1/PS mode. The M/G/1 model matches the experimental data remarkably well. As expected, the M/G/1/PS model overestimates the queue length but can be

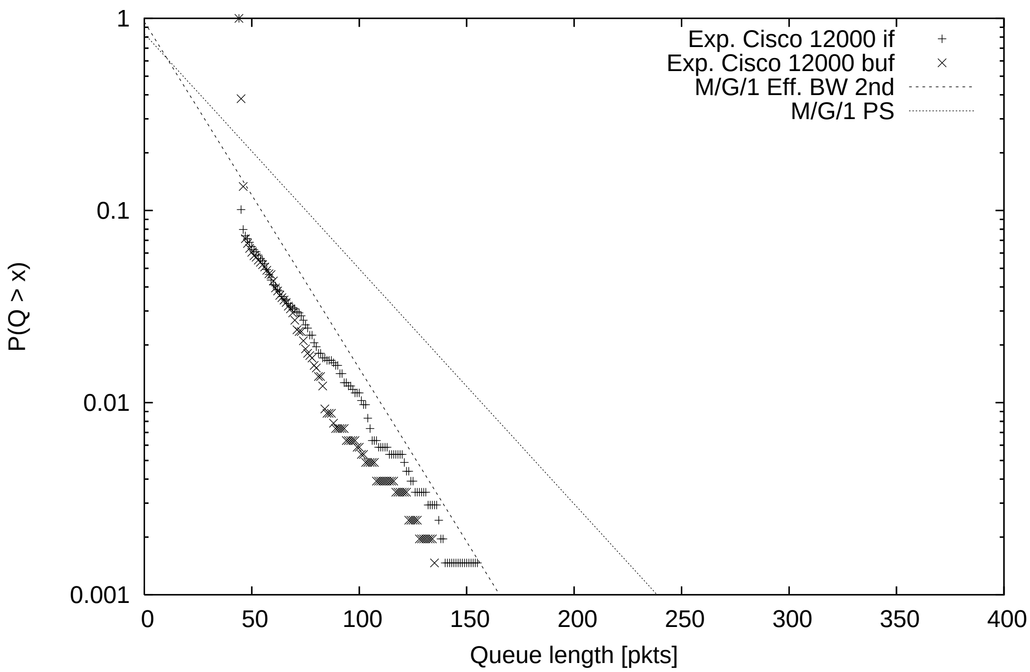


Figure 6.3: Short Flow Queue Distribution of 30 packet flows measured on a Cisco GSR compared to model prediction

usable to find an upper bound for the queue length.

Figure 6.3 and Figure 6.4 show the same graph for flows of 30 packets and 14 packets respectively. For 30 packets, the model still matches well, however it tends to overestimate the queue length. For 14 packet flows, we measured a non-empty queue for less than 1% of our samples. The built in FIFO was able to absorb most of the bursts. For six packet flows, the queue length was zero for almost all our samples.

The reason for the overestimation of the queue size for small flow lengths is most likely randomization and smoothing in the switch that aggregates the traffic as well as in the TCP senders and receivers. Even a slight spreading out of bursts can reduce the queue length sufficiently to push it below the 43-packet limit where it is invisible to our measurement.

Overall, we can see though that our M/G/1 model is a reliable tool to estimate queue distribution and the amount of buffering that is required to achieve certain loss guarantees. We see that it not only works in a simulation scenario but also for a real

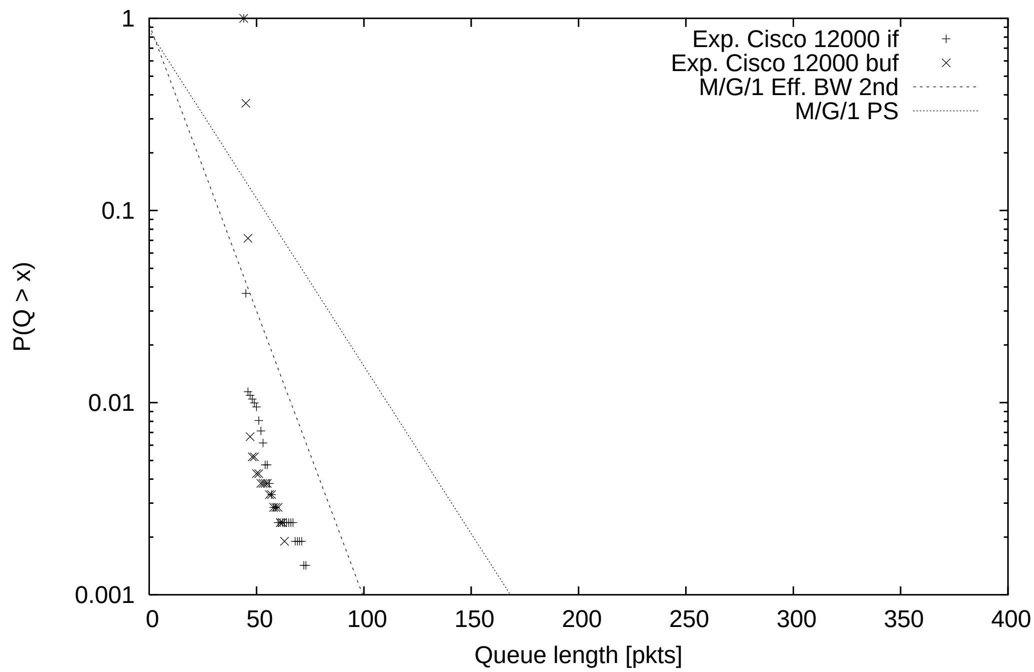


Figure 6.4: Short Flow Queue Distribution of 14 packet flows measured on a Cisco GSR compared to model prediction

physical router with traffic generated by physical network hosts.

6.2 Stanford Network Experiment

The Stanford Experiment verifies our model for a congested router serving traffic from live users in a production network. In the experiment, we throttled the bandwidth of the Cisco VXR 7200 router that forwards internet traffic to the student dormitories at Stanford. The VXR is a shared memory router that has very different properties than the GSR described in the last Section. Specifically, normal operation packets can be buffered in a number of different stages. A major part of this Section is devoted to understanding the VXR switching architecture and to explain how we can accurately control the amount of buffering that is used. The key result of this experiment is that even for a very diverse traffic mix, a router can achieve close to full link utilization with extremely small buffers.

6.2.1 Introduction and Setup

Stanford University routes traffic from and to on-campus student residences separately from traffic that is generated by administrative, educational and academic hosts. Residential traffic is aggregated via a 100 Mb/s network with Cisco 5000 Ethernet switches. The gateway to the internet is a Cisco 7200 VXR router with a 100 Mb/s link to one of the Cisco 5000 switches as well as a 100 Mb/s connection to a commercial ISP. Between the switch and the Cisco 5000, a Packeteer packet shaper is used to limit the outgoing bandwidth. A schematic outline of the topology is shown below.

```
                                                        ----100 Mb
          100 Mb                  100 Mb              /
Internet <------> Cisco 7200 VXR <---I--> Cisco 5000 ----100 Mb
                                     |                \
                                  Packeteer             ----100 Mb
```

The 100 Mb/s link from and to the internet is saturated a good part of the day. Traffic is from a mix of different applications including Web, FTP, Games, Peer-to-Peer, Streaming and others. The number of flows that are active varies by the time of day from 100s to at least 1000s of concurrent flows. Traffic is mainly TCP as well as some UDP. This traffic mix is far away from the idealized assumptions in the rest of the paper and fairly similar to network traffic on large commercial network backbones.

Between the VXR and the Cisco 5000 switch is a Paketeer packet shaper. This shaper can be used to throttle flows in both directions. For the experiment, the bandwidth of the router was below the trigger bandwidth of the Packeteer, which was basically inactive.

The goal of our experiment is to observe a congested router serving a large number of flows. The above setup is not directly usable for this purpose as the Cisco 7200 never experiences congestion - its input and output capacity is the same. In order to generate congestion at the router, we throttle the link from the Cisco 7200 router to the Cisco 5000 Ethernet switch to a rate substantially below 100 Mb/s. How this is done is explained in detail below.

The goal of the experiment is to verify if the $2T \times C/\sqrt(n)$ hypothesis holds. In order to do this, we measure before the start of the experiment:

- The capacity C. In practice, we will throttle the router's outgoing interface to a specific capacity C and leave it constant for the duration of the experiment.
- The number of flows n. We can estimate this via the Netflow mechanism of the Cisco router.
- The round-trip propagation delay 2T. We cannot measure or estimate the RTT distribution in a meaningful way and instead assume a typical average for backbone networks.

All of the above are assumed to be constant for the time of the experiment. During the experiment, we change the amount of buffering on the router, and measure the following parameters:

- Utilization of the bottleneck link.
- The queue length Q. This is mainly to verify that the router was congested and operating with full buffers.

Most of the measurements that we do are performed on the router. Additionally, we use the router for creating congestion. In order to set up a meaningful experiment and understand the results, it is necessary to understand how the VXR operates.

6.2.2 The Cisco 7200 VXR

The Cisco 720x VXR is a shared memory router. It has a central network processor (Cisco NPE-x00 series) and up to six interface cards that are all mapped into a shared memory space. Packets are transferred from the interface cards to memory via three PCI busses. Two PCI busses serve three interface cards each, and one serves the IO controller, which has an additional network interface. The VXR runs Cisco's IOS operating system that manages all router components directly. For a detailed architecture description see [7].

In our idealized router model, a packet is only queued once in a router. As we will see, the reality on the VXR is far more complex. As we have seen in the experiments for the GSR, it is important to understand where queueing can occur accurately for all buffering. The goal of this Section is to understand where in the VXR packets are queued and how these buffers are sized.

To understand fully the VXR, we ran a number of tests on a VXR in a laboratory setting before conducting the actual experiment on the VXR in the Stanford network.

Packet Switching Architecture

Before we explain the buffer architecture of the VXR, we need to discuss its packet switching architecture. Incoming packets on the VXR are copied into shared memory by the line card via DMA. Once the packet is copied, it triggers an interrupt that eventually causes the packet to be switched to the right outgoing line card. The VXR uses a number of different switching methods, depending on the packet and router state:

- Process switching. This is switching that is done completely in software in a special IOS process, no pre-cached information is used.
- Fast Switching. Again everything is done in software but a route cache is used and the packet is switched in the interrupt handler that was triggered by the received packet.
- Optimum switching. For IP packets only, there is a special optimized version of Fast switching called Optimal switching. It uses a faster caching method (M-Tree vs. Binary Tree)
- Cisco Express Forwarding (CEF). Basically, optimized Optimum Switching with an M-Trie instead of the M-Tree.
- Netflow switching. Netflow stores information for each flow it sees and uses this cached information to make very fast switching decisions. Netflow was switched off on the router used for the experiment.

The type of switching affects the path of the packet through the router and the OS as well as the maximum rate the router is able to achieve. In practice, the router in the experiment forwarded the vast majority of packets using CEF. In this case, the usual path of a packet is:

1. Packet is received by the network interface and transferred (via PCI bus DMA) into the Tx Ring. The Tx Ring is circular list of pointers to buffers. If the Tx Ring is full, the packet is dropped.

2. If the packet needs to be switched in software, it is moved to the input queue. As we use For CEF and Netflow switching, this step is usually skipped.

3. An interrupt is triggered and the buffers are removed from the Tx Ring. If the packet can be switched with anything but processor switching, the MAC header is rewritten and the packet is moved to an output queue. If no virtual output queueing is used, the output queue is one of the outgoing interface (and the next step is skipped). In our case, we use class-based VOQs and the packet is moved to the output queue of its class. If either type of output queue is full, the packet is dropped.

4. If virtual output queues (e.g. different queues for different classes of traffic) are used and a packet arrives at the head of the queue, it is switched to the output queue of the interface.

5. Once there is space available in the Tx ring, the packet is removed from the output queue to the Tx Ring.

6. The Tx Ring gets DMA'd to the interface card and the packet is transmitted.

Generally, packets never change their location in memory, all moving is done by putting pointers to the packet data into different queues. Process switching is the exception to the rule as packets have to be reassembled from fragments (see the buffer pool description below).

Buffer Pool Architecture

A further complication is that the 7200 uses particles instead of packets. For standard IP packets, a particle is approximately 745 bytes or half the most widely used Ethernet MTU size of 1500. Buffer pools contain either particles or full buffers (1500 bytes), but never both.

The 7200 keeps several different types of buffer pools. The most important ones are:

Private Particle Pools These are used to replenish the Rx rings after packets have arrived. After a packet has been sent from the Tx Ring, the buffer is returned to the private pool of the interface. That the buffers are associated with the **incoming** interface is a major difference compared to the idealized output queued model or central fabric routers such as the GSR. If we want to prevent a shortage in buffers, we have to increase the private pool of the incoming interface, not the outgoing interface.

Normal Particle Pools. These are global particle pools not associated with an interface. They are used as a back-up if private pools run out.

Public Pools. These pools are used to assemble packets temporarily that have to be switched in software. There are separate pools (small pool, middle pool, big pool) corresponding to typical packet lengths (40,...,1500,... bytes). Public Pools are global and are not associated with a particular interface.

All buffer pools are fairly small by default. If a buffer runs low, additional particles or buffers are added by IOS from the overall memory pool. If the CPU load of the router becomes too high, the software process that performs the buffer resizing can get starved and buffers might be too small, which eventually can cause additional packet drop. We tested the behavior of IOS both in a lab setting as well as on the real router, and at the line speeds that we are interesting in, IOS was always fast enough to re-size buffers as needed.

For our experiment, we want to limit the total amount of buffering available to packets through the router. Due to the VXR's architecture, it is impossible to do this by simply limiting the overall amount of memory. The dynamic allocation of memory between queues, the queues minimum sizes and the different paths through the queues make this impossible. Instead, our strategy will be to limit queue length directly, which for some queues we can do via IOS configuration commands.

Rate Limiting

In the experiment, we want to limit the rate of the VXR for two reasons. First, we need to throttle the bandwidth of the bottleneck link and create congestion in case there is not enough traffic. Second, by throttling the bandwidth substantially below the interface bandwidth, we can keep some of the internal queues in the router empty all of the time, which simplifies our analysis.

In a real router with limited buffers, the sending rate is limited by the outgoing hardware interface. Throttling the hardware interface directly is not possible for the VXR. However, we can rate limit the sending rate in software using Cisco IOS. Depending on how we do this, we will cause queueing in one or several of the queues described above. Understanding the details of how queueing works and which queues different methods for rate limiting use, is crucial to understand the results of the experiment.

In Cisco IOS, there are two approaches for reducing the sending rate of an interface: Policing and Shaping [37]. Policing (e.g. CAR, class based policing) drops packets that exceed a maximum rate selectively (usually based on a token bucket rule), but it never queues packets. This is very different from what a router with limited memory would do. Measuring router performance using policing will give us little insight in the behavior of a router.

Traffic Shaping (e.g. GTS, class-based shaping) first queues packets, and then drops packets based on the length of the queue. This is identical to what a router with limited memory would do.

The simplest shaping mechanism supported by IOS is General Traffic Shaping (GTS). It works on all IOS versions and can do simple rate limiting on most types of interfaces. The problem is that according to Cisco documentation, it is incompatible with Netflow [40]. In our laboratory setting, it did not limit sending rates when Netflow was turned on.

The second mechanism is Class-Based Shaping [39]. Class based shaping allows is to divide traffic into classes of packets. Each class can be queued separately and priorities between classes can be defined. Class based shaping uses its own output queue in addition to the standard output queue described above. Queue length can be adjusted with an IOS command. In our case, we configure a class that contains any type of packet and limit this class to our target rate. Configuration snippets can be found in Appendix A.5.

In addition to the queue, class-based shaping uses a token bucket to limit the sending rate. For our experiment, we are only interested in the buffering effect due to the router's queue and would like to set the token bucket size to zero (or one packet). In practice however, IOS does not allow us to set a token bucket below 10ms. This effectively creates additional buffering of 10ms multiplied with our throttled interface rate.

The queues that Class-Based Shaping uses, come before the interface's output queue and the TX Ring. If the output interface is the bottleneck, the effective queue length is the sum of all three queue lengths. In our case, however, the rate defined by the class-based shaping ($<<$ 100 Mb/s) is below the rate of the output interface (100 Mb/s). We can thus expect the output queue and the TX Ring to be empty at any given time. We verified this in an experiment with a VXR in a laboratory setting. We throttled a 1 Gb/s link down to 100 Mb/s and queried the TX Ring occupancy and output queue length. Except for packets due to the minimum token bucket size, both were empty or contained a single packet all the time.

Data Capture on the VXR

Ideally, we would have liked to measure network data using specialized measurement equipment in the network. In practice, doing so is difficult in a production environment. However, it turns out that we can capture all data that we need via the console of the router. Below we outline how this is done for each data type and what level of accuracy we can expect.

Link Capacity C. In the experiment, we set the link capacity using class-based shaping. In theory, this allows us to specify precisely a number of bits/s. We measured the accuracy of the shaped rate in a laboratory setting and the rate limiting in practice seems to be very accurate with an error far below one percent.

Number and type of flows. IOS has a feature called Netflow [11] that tracks all flows passing through a router and collects statistics about these flows. What we need for our experiment is the number of concurrent long-lived flows. Netflow is not able to provide this information directly, however we are able to get an estimate indirectly.

First, we measure the number of flows that are active over a 10 second interval. We do this by resetting the Netflow counter, and then dumping Netflow statistics. This count will be higher than the number of long-lived flows, as it includes short flows that were only active for a small fraction of the time, however it provides an upper bound. Netflow also gives us statistics about the type of flows (e.g. protocol, port numbers), the average number of packets per flow, and the average duration per flow. Using this additional information, we can estimate what percentage of flows were long lived. While this method is not very precise, we can use it to get a lower bound on the number of flows, which allows us to pick a sufficiently large buffer for the router.

Utilization. IOS automatically collects for each interface the rate in bits/s and packets/s. We tested the accuracy of this rate and found it to be unsuitable for our experiment. The reason for this is shown in Figure 6.5, which shows the time evolution of the measured rate in packets and bytes. At time zero, we saturate an interface of a previously idle router that is throttled to 1 Mbit/s and after about 23 minutes, we stop the traffic. First, we can see that the router uses a geometrically weighted average. This means we would have to run our experiment over long periods of time to achieve a stable state. Additionally, the rate calculated from the packet rate and the byte rate differ substantially and take different amounts of time to converge. We also found in a different experment that the convergence depends on the line rate.

A second measurement mechanism on the router are packet and byte counters on the interfaces. These counters have the advantage that they are exact, however we need to obtain the accurate time that corresponds to a counter value. On the VXR, the interface counter and the clock are both accessible directly by IOS running on the controller. In a lab setting, this method proved to be very accurate, with errors

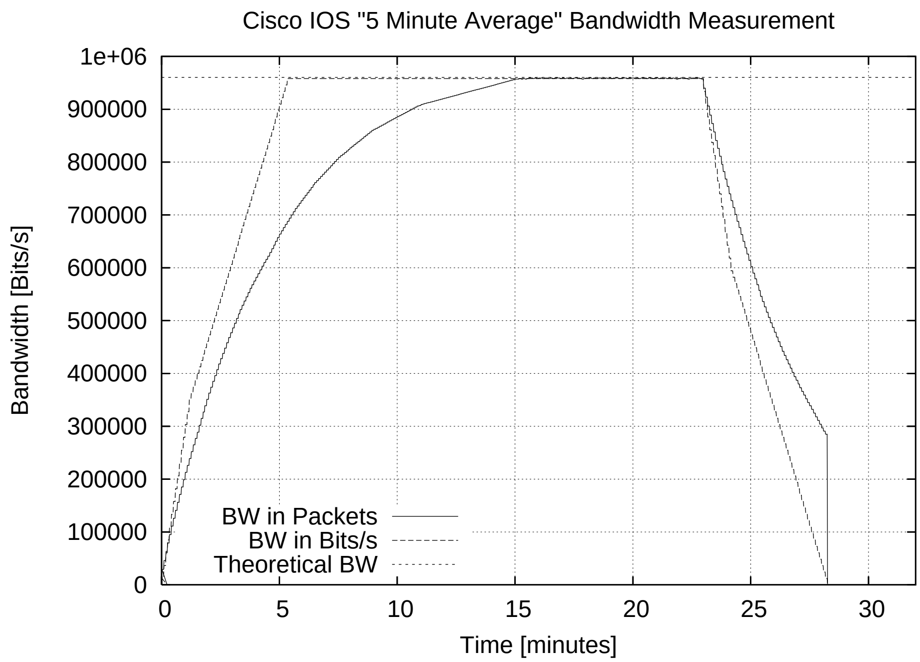


Figure 6.5: "average" sending rate of a router as measured by IOS. Actual sending pattern was an on/off source. The reported byte based rate and the packet based rate differ substantially.

below one percent over times of tens of seconds [2].

Queue length. The various queues of the router can all be monitored via IOS. We can only measure the current occupancy, but not averages or time evolution of queues. Queue length reporting for class-based-shaping queues proved to be unreliable in practice. Figure 6.6 shows the cumulative probability distribution of the class-based-shaping queue of a router. The maximum queue length was configured to be 40 packets, however the router reports that this length is frequently exceeded. We confirmed this behavior over a wide range of loads and bandwidths.

The above deviation can in a simple model have two causes. Either the queue length is misreported, or the queue limit is not enforced. To test this, we shaped a

[2] This way of measuring rates is only possible on a shared memory router. On the GSR, packet counters are on the line cards and are only sporadically transmitted to the controller. The time when a packet count was transmitted can only be determined within a few seconds.

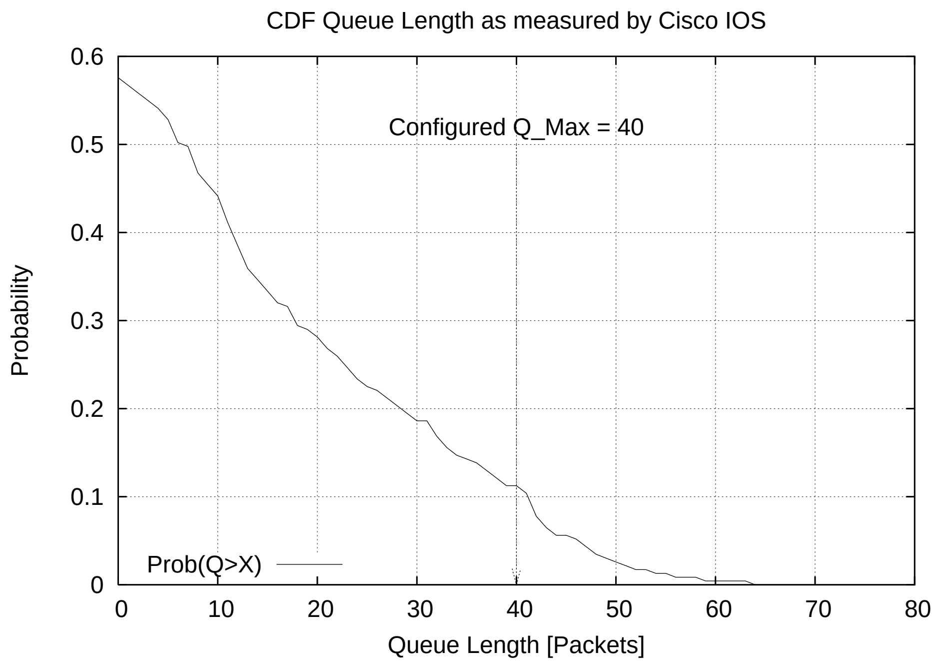


Figure 6.6: CDF of the class-based-shaping queue length reported by IOS on a VXR router. The maximum queue length was configured to be 40 packets, however the router reports queue lengths above this value.

network interface to 1 Mb/s and sent a TCP stream over a 1 Gb/s access link. The result was congestion on the 1Mb/s link with almost all outstanding packets in the router's buffer. In this experiment, the flow quickly reached its maximum window size of 42 packets (65,536 Bytes), which were all queued in the router. The reported queue length again frequently exceeded 40 packets and one data point exceeded 80 packets. We then reduced the queue length first to 43 packets and then to 40 packets. At 43 packets, the flow was unaffected, however at 40 packets, it would lose packets as the queue would overflow. This experiment suggests that the queueing mechanism works correctly, and it is only the reporting of the queue length that is incorrect.

In practice, we can use queue length measurements on the VXR only as qualitative data points. They give us a good idea of the typical queue lengths, however exact distribution as well as minimum and maximum values do not match other experimental evidence.

Packet Length	1-32	64	96	128	160	192	224	256	288	320
Probability	.001	.522	.034	.021	.012	.007	.009	.005	.004	.004
Packet length	352	384	416	448	480	512	544	576	1024	1536
Probability	.004	.006	.004	.003	.003	.003	.005	.011	.039	.293

Figure 6.7: Packet length statistics for the router in the experiment. Packet lengths are in bytes.

6.2.3 Experiment and Results

The actual experiment on the router was done during a maintenance window. At this time, both incoming and outgoing rate of the router were around 30 Mb/s. We throttled the incoming rate to 20 Mb/s using class-based shaping to create congestion. For the detailed IOS configuration, see AppendixA.5.

Packet Length

We need the average packet length to convert delay bandwidth product into a meaningful number of packets. Statistics on the packet length are collected by Netflow. The data generated by Netflow is shown in Figure 6.7. Half of the packets are between 32 and 64 bytes, these are TCP ACKs as well as games, real-time applications and port scans. 29% of the packets are packets of maximum length, these are typically from long-lived TCP flows from the data we calculate and average packet size of 557.1 bytes.

Bandwidth Delay Product

We cannot easily measure the RTT of flows through the router, instead we use the common 250ms assumption. With the 20 Mb/s throttled bandwidth and average packet size from above, this gives us a delay bandwidth product of:

$$2T \times C = 20Mb/s \times 250ms = 5Mbit = 1121Pkts \times 557.1Bytes \tag{6.1}$$

Protocol	Total	Flows	Packets	Bytes	Packets	Active(Sec)	Idle(Sec)
--------	Flows	/Sec	/Flow	/Pkt	/Sec	/Flow	/Flow
TCP-Telnet	4397974	1.0	36	490	37.7	10.3	23.9
TCP-FTP	15551970	3.6	3	172	14.4	4.2	23.5
TCP-FTPD	3168663	0.7	103	856	76.1	16.5	22.5
TCP-WWW	1389348486	323.4	16	752	5396.4	4.1	21.9
TCP-SMTP	2112872	0.4	3	131	1.7	10.5	24.9
TCP-X	919732	0.2	204	631	43.8	43.1	19.9
TCP-BGP	318326	0.0	10	190	0.7	25.4	19.2
TCP-NNTP	3192112	0.7	186	956	138.4	5.6	24.5
TCP-Frag	375573	0.0	23	262	2.0	13.0	23.4
TCP-other	6766607723	1575.4	11	495	17907.5	15.5	19.8
UDP-DNS	3206553	0.7	3	65	2.3	4.5	24.3
UDP-NTP	2101758	0.4	1	76	0.6	3.2	24.0
UDP-TFTP	22222	0.0	1	108	0.0	0.9	26.1
UDP-Frag	374985	0.0	426	1060	37.2	46.6	23.0
UDP-other	2879140323	670.3	4	204	2861.6	5.7	21.9
ICMP	729060088	169.7	1	48	219.0	0.9	28.6
IGMP	65	0.0	28	1447	0.0	0.1	29.5
IPINIP	4925	0.0	11	511	0.0	46.6	13.8
GRE	313657	0.0	492	377	35.9	63.4	20.2
IP-other	1305636	0.3	132	489	40.1	26.7	20.6
Total:	11801523643	2747.7	9	516	26816.2	10.9	21.1

Figure 6.8: Long term netflow statistics for the router used in the experiment

Number of long-lived Flows

Before the actual experiment, we measured that during a 10 second interval, about 3400 flows are active. This includes long-lived flows, as well as short-lived flows, real-time applications and port scans. To estimate what percentage of these flows are long-lived flows, we use Netflow statistics gathered by the router. The output of Netflow on the router used for the experiment is shown in Figure 6.8. The top three categories generate more than 90% of the traffic. These categories are:

Type	pkt/s	pktsize	Mb/s	% of BW
TCP-WWW	5396	752	32.46	30.0%
TCP-Other	17907	495	70.91	65.6%
UDP-Other	2861	204	4.67	4.3%

This adds up to a total bandwidth of 106 Mb/s. This makes sense as this is the bi-directional average for a 100 Mb/s link that is saturated during the day and underutilized at night.

If we care about what flows are competing for bandwidth, we can essentially ignore UDP. UDP flows are real-time applications such as games, streaming and port scanning tools. All of these traffic sources usually do not respond to congestion. Additionally, they compromise only 4.3% of the overall traffic.

The TCP-WWW flows are HTTP flows generated by web browsers or web services. They have an average length of only 16 packets. Therefore, the average http flow will never leave slowstart. We now try to estimate how many TCP flows we have in a 10-second interval. Web traffic is 30bytes. The number of packets per second generated by web flows is thus:

$$20Mb/s \times 0.3 = 750kByte = 1000pkts/s \tag{6.2}$$

To verify the rate, we compare this to the rate reported by Netflow. Netflow reports 5400 packets/s for a traffic rate of 106 Mb/s. As we have only a rate of 20 Mb/s, we would expect a rate of

$$5400pkts/s \times \frac{106Mb/s}{20Mb/s} = 1018pkts/s \tag{6.3}$$

Both methods of calculating the packet rate agree fairly well. Assuming average flow length of 10-20 packets (Netflow reports 16), this means 50 to 100 flows/s at 20 Mb/s. This is again is consistent with the 323 flows/s (at 106 Mb/s) that Netflow suggests. Netflow reports that UDP has twice as many flows/s, thus 100 to 200 per second.

It seems that short-lived web and UDP flows make up the majority of the flows we are seeing. The minimum and maximum number of long-lived flows during the 10 second interval should be:

$$n_{min} \approx n_{total} - n_{UDP} - n_{Web} = 3400 - 2000 - 1000 = 400 \tag{6.4}$$

$$n_{max} \approx n_{total} - n_{UDP} - n_{Web} = 3400 - 1000 - 500 = 1900 \tag{6.5}$$

Given the delay-bandwidth product of about 1100 packets, this gives us an average window size of only 2.5 packets, which is very low. Many of the flows will be throttled or constrained by the source. However, those that try to send at full speed will likely see frequent time-outs.

Buffer Size

Using the minimum and maximum number of flows, as well as the delay-bandwidth product, we can calculate the minimum and maximum buffer:

$$B_{min} \frac{2T \times C}{\sqrt{n_{max}}} = \frac{1121pkts}{\sqrt{1900}} = 26pkts \tag{6.6}$$

$$B_{max} \frac{2T \times C}{\sqrt{n_{min}}} = \frac{1121pkts}{\sqrt{400}} = 56pkts \tag{6.7}$$

For the experiment, we will assume a minimum buffer size of 56 packets. We can set the queue length of the class-based shaper in packets. However, the shaper uses a token bucket mechanism with a minimum size of the bucket of 10 ms. At our rate of 20 Mb/s, this translates to:

$$10ms \times 20Mb/s = 25kB = 45pkts \times 557Bytes \tag{6.8}$$

The token bucket effectively increases the queue length by 45. Thus, we should expect (and during the experiment observed) that we still have good link utilization with a maximum queue length set to zero.

Buffer	Bandwidth [Mb/s] (measured)	Utilization (measured)	Utilization (model)	$\frac{2T \times C}{\sqrt{n}}$
500	19.981	99.92 %	100%	>> 2x
40+45	19.709	98.55 %	99.9%	1.5 x
20+45	19.510	97.55 %	99.5%	1.2 x
1+45	19.481	97.41 %	95.9%	0.8 x

Figure 6.9: Utilization data from the router measured during the experiment. The buffer includes an extra 45 packets due to the minimum size of the toekn buffer that in this configuration acts like an additional buffer.

Results and Discussion

We measured the utilization using byte counters and timestamps for large buffers as well as three settings of small buffers. In each case, we measured the rate over several minutes. The results are shown in Figure 6.9.

The most important result of this experiment that we can achieve very close to full utilization with buffers that are in the order of $\frac{2T_p \times C}{\sqrt{n}}$. The delay-bandwidth product for our experiment is 557 packets, the buffer available on this router is in the 1000s of pockets, yet we can achieve 98.5% utilization with only the equivalent of 85 packets. This is the case not with idealized traffic, but a complex mix of long flows, short flows, UDP and TCP and a variety of non-congestion aware applications. It also holds in the worst case of a router experiencing heavy congestion.

Additionally, we can see that our model predicts the approximate amount of buffering needed. The utilization drop predicted by the model is much steeper than the drop-off we can observe in the experiment. However, the model predicts the point where utilization drops from full to below full utilization accurately within a small factor.

Chapter 7

Conclusion

We believe that the buffers in backbone routers are much larger than they need to be — possibly by two orders of magnitude. We have demonstrated that theory, $ns2$ simulation and experimental results agree that much smaller buffers are sufficient for full utilization and good quality of service.

The results we present in this paper assume only a single point of congestion on a flow's path. We don't believe our results would change much if a percentage of the flows experienced congestion on multiple links, however we have not investigated this. A single point of congestion means there is no reverse path congestion, which would likely have an effect on TCP-buffer interactions [46]. With these assumptions, our simplified network topology is fairly general. In an arbitrary network, flows may pass through other routers before and after the bottleneck link. However, as we assume only a single point of congestion, no packet loss and little traffic shaping will occur on previous links in the network.

We focus on TCP as it is the main traffic type on the internet today, however Chapter 4 shows that at least some other traffic types can be modeled with the same model we use for short flows and that in mixes of flows, long TCP flows will dominate. As in the internet today, the majority of traffic is TCP. Our results should cover a fairly broad range of scenarios and the experiment supports this. However, traffic with a large share of traffic that uses different congestion-aware protocols would likely require further study.

We did run some simulations using active queue management techniques (e.g. RED [17]) and this had an effect on flow synchronization for a small number of flows. Aggregates of a large number (> 500) of flows with varying RTTs are not synchronized and RED tends to have little or no effect on buffer requirements. However, early drop can slightly increase the required buffer since it uses buffers less efficiently.

Congestion can also be caused by denial of service (DOS) [22] attacks that attempt to flood hosts or routers with large amounts of network traffic. Understanding how to make routers robust against DOS attacks is beyond the scope of this thesis, however, we did not find any direct benefit of larger buffers for resistance to DOS attacks.

If our results are right, they have consequences for the design of backbone routers. While we have evidence that buffers can be made smaller, we haven't tested the hypothesis in a real operational network. It is a little difficult to persuade the operator of a functioning, profitable network to take the risk and remove 99% of their buffers. That has to be the next step, and we see the results presented in this thesis as a first step toward persuading an operator to try it.

If an operator verifies our results, or at least demonstrates that much smaller buffers work fine, it still remains to persuade the manufacturers of routers to build routers with fewer buffers. In the short-term, this is difficult too. In a competitive market-place, it is not obvious that a router vendor would feel comfortable building a router with 1% of the buffers of its competitors. For historical reasons, the network operator is likely to buy the router with larger buffers, even if they are unnecessary.

Eventually, if routers continue to be built using the current rule-of-thumb, it will become very difficult to build line cards from commercial memory chips. And so, in the end, necessity may force buffers to be smaller. At least, if our results are true, we know the routers will continue to work just fine, and the network utilization is unlikely to be affected.

Appendix A

A.1 Summary of TCP Behavior

Let's briefly review the basic TCP rules necessary to understand this paper. We will present a *very* simplified description of TCP Reno. For a more detailed and more comprehensive presentation, please refer to e.g. [42].

The sending behavior of the TCP sender is determined by its state (either slow-start or congestion avoidance) and the congestion window, W. Initially, the sender is in slow-start and W is set to two. Packets are sent until there are W packets outstanding. Upon receiving a packet, the receiver sends back an acknowledgement (ACK). The sender is not allowed to have more than W outstanding packets for which it has not yet received ACKs.

While in slow-start, the sender increases W for each acknowledgement it receives. As packets take one RTT to travel to the receiver and back this means W effectively doubles every RTT. If the sender detects a lost packet, it halves its window and enters congestion avoidance. In congestion avoidance, it only increases W by $1/W$ for each acknowledgment it receives. This effectively increases W by one every RTT. Future losses always halve the window but the flow stays in CA mode.

`Slow-Start:`	`No loss:`	$W_{new} = 2W_{old}$
	`Loss:`	$W_{new} = \frac{W_{old}}{2}$, `enter CA`
`CA:`	`No loss:`	$W_{new} = W_{old} + 1$
	`Loss:`	$W_{new} = \frac{W_{old}}{2}$

In both states, if several packets are not acknowledged in time, the sender can also trigger a timeout. It then goes back to the slow-start mode and the initial window size. Note that while in congestion avoidance, the window size typically exhibits a sawtooth pattern. The window size increases linearly until the first loss. It then sharply halves the window size, and pauses to receive more ACKs (because the window size has halved, the number of allowed outstanding packets is halved, and so the sender must wait for them to be acknowledged before continuing). The sender then starts increasing the window size again.

A.2 Behavior of a Single Long TCP Flow

Note: The following appendix is primarily the work of Isaac Keslassy, and not of the author of this thesis. It is reproduced here for completeness of argument.

Consider a single long TCP flow, going through a single router of buffer size equal to the delay-bandwidth product (Figure 2.1). It is often assumed that the round-trip time of such a flow is nearly constant. Using this assumption, since the window size is incremented at each round-trip time when there is no loss, the increase in window size should be linear with time. In other words, the sawtooth pattern of the window size should be triangular.

However, the round-trip time is *not* constant, and therefore the sawtooth pattern is not exactly linear, as seen in Figure 2.2. This becomes especially noticeable as link capacities, and therefore delay-bandwidth products, become larger and larger. Note that if the sawtooth was linear and the round-trip time was constant, the queue size increase would be parabolic by integration. On the contrary, we will see below that it is concave and therefore sub-linear.

Let's introduce a fluid model for this non-linear sawtooth behavior. Let $Q(t)$ denote the packet queue size at time t, with $Q(0) = 0$. As defined before, $2T_p$ is the round-trip propagation time, and C is the router output link capacity. For this fluid model, assume that the propagation time is negligible compared to the time it takes to fill up the buffer. Then, in the regularly increasing part of the sawtooth, the window size increases by 1 packet every RTT (Appendix A.1), and therefore

$$\dot{W}(t) = \frac{1}{RTT} = \frac{1}{2T_p + Q(t)/C}.$$

In addition the window size is linked to the queue size by

$$W(t) = 2T_pC + Q(t).$$

Joining both equations yields

$$\dot{W}(t) = \frac{C}{W(t)}. \tag{A.1}$$

Hence

$$\frac{d\left(\frac{W^2(t)}{2}\right)}{dt} = C$$
$$W^2(t) = 2Ct + W(0).$$

When the sawtooth pattern starts, $Q(0) = 0$, thus

$$W(0) = 2T_pC + Q(0) = 2T_pC.$$

We finally get a simple model for the increase of the window size in the sawtooth,

$$W(t) = \sqrt{2Ct + (2T_pC)^2}. \tag{A.2}$$

Therefore, as $t \gg T_p$, the queue size behaves in a square-root fashion, instead of a linear one, as commonly believed. The queue size behaves in a similar fashion,

$$Q(t) = W(t) - 2T_pC = \sqrt{2Ct + (2T_pC)^2} - 2T_pC. \tag{A.3}$$

As an application, it is now possible to determine the periodicity T of the sawtooth pattern. The maximum buffer size B is reached when $Q(T) = B$, i.e.

$$T = \frac{B^2}{2C} + 2T_pB.$$

When the buffer size is equal to the bandwidth-delay product ($B = 2T_pC$), we get

$$T = 1.5\frac{B^2}{C}.$$

For instance, in Figure 2.2, the modelled sawtooth period is $T = 1.5\frac{142^2}{1000} = 30.2$, which fits the observed value.

A.3 Queue Distribution using Effective Bandwidth

The goal of this appendix is to model the behavior of the router queue when there are several senders with short TCP flows. As explained in Section 4.1, each of these short TCP flows sends a given number of bursts. We assume that all these bursts are independent, and by considering all the independent bursts of all the flows, we model the arrival process to the router as the cumulative arrival of these independent bursts. Let $A(t)$ denote the cumulative amount of traffic arrived to the router at time t, $N(t)$ the number of bursts arrived before t, and X_i the size of each of these bursts. Then:

$$A(t) = \sum_{i=1}^{N(t)} X_i.$$

We assume that the bursts arrive as a Poisson process $N(t)$ of rate ν, and that their burst size follows a distribution function F. The queue is serviced with a capacity C. Therefore, we use an *M/G/1 model* for the job size in the router queue. Of course, the traffic arrival rate λ is the product of the burst arrival rate by the average burst size:

$$\lambda = \nu E[X].$$

The effective bandwidth theory describes the characteristics of a traffic source, and in many cases can be a powerful tool to derive properties that are otherwise hard to compute. For more information we refer the reader to [27], from which most of the results below are adapted. Consider a cumulative traffic arrival function $A(t)$, where

$A(t)$ has stationary increments. Its effective bandwidth is defined as:

$$\alpha(s,t) = \frac{1}{st} logE\left[e^{sA(t)}\right].$$

In our case, $A(t)$ has i.i.d. increments. It is a special case of Lévy process, and its effective bandwidth has a simpler form (section 2.4 of [27]):

$$\alpha(s) = \frac{1}{s}\int (e^{sx}-1)\nu dF(x) = \frac{\lambda}{sE[X]}E\left[e^{sX}-1\right]. \quad \text{(A.4)}$$

We can now use Cramér's estimate to model the tail of the distribution of the queue length Q (section 3.2 of [27]). This estimate assumes an infinite buffer size. For Cramér's estimate there has to exist a constant κ such that the effective bandwidth is the link capacity:

$$\alpha(\kappa) = C \quad \text{(A.5)}$$

Also the derivative $\alpha'(s)$ has to exist at κ. Both are satisfied in our case as $\lambda < C$ and $\alpha(s)$ is differentiable for any $s > 0$. Cramer's estimate is given by

$$P(Q \geq b) \approx \frac{C-\alpha(0)}{\kappa\alpha'(\kappa)} e^{-\kappa b} \quad \text{(A.6)}$$

To calculate κ we need to simplify Equation (A.5). We approximate the exponential function by its Taylor Series.

$$e^{sX} = 1 + sX + \frac{s^2}{2}X^2 + \frac{s^3}{6}X^3 + O(X^4) \quad \text{(A.7)}$$

We present solutions for the 2nd and 3rd order approximation. In our experience the 2nd order approximation is sufficient to estimate the required buffer.

Second Order Approximation We substitute Equation (A.7) into Equation (A.4) and obtain for the 2nd order approximation:

$$\alpha(s) \quad = \quad \frac{\lambda}{sE[X]}E\left[1 + sX + \frac{s^2}{2}X^2 - 1\right]$$

$$
\begin{aligned}
&= \lambda + \frac{\lambda s}{2}\frac{E[X^2]}{E[X]} \\
\alpha'(s) &= \frac{\lambda}{2}\frac{E[X^2]}{E[X]}
\end{aligned}
$$

The load ρ is defined as the ratio of the arrival rate λ by the capacity C:

$$\rho = \frac{\lambda}{C}.$$

Using Equation (A.5), we get:

$$\kappa = \frac{2(1-\rho)}{\rho}\frac{E[X]}{E[X^2]}$$

Finally, for the queue size distribution we obtain

$$P(Q \geq b) \approx \frac{\frac{\lambda}{\rho} - \lambda}{\frac{2(1-\rho)}{\rho}\frac{E[X]}{E[X^2]}\frac{\lambda}{2}\frac{E[X^2]}{E[X]}} e^{-b\kappa} = e^{-b\kappa}$$

Third Order Approximation. We substitute Equation (A.7) into Equation (A.4) and obtain for the 3rd order approximation:

$$
\begin{aligned}
\alpha(s) &= \frac{\lambda}{sE[X]} E\left[1 + sX + \frac{s^2}{2}X^2 + \frac{s^3}{6}X^3 - 1\right] \\
&= \lambda + \frac{\lambda s}{2}\frac{E[X^2]}{E[X]} + \frac{\lambda s^2}{6}\frac{E[X^3]}{E[X]} \\
\alpha'(s) &= \frac{\lambda}{2}\frac{E[X^2]}{E[X]} + \frac{\lambda s}{3}\frac{E[X^3]}{E[X]}
\end{aligned}
$$

Again we now need to solve $\alpha(\kappa) = C$ where $\rho = \frac{\lambda}{C}$. We obtain the quadratic equation.

$$\kappa^2 + \kappa\frac{3E[X^2]}{E[X^3]} - \frac{6(1-\rho)}{\rho}\frac{E[X]}{E[X^3]} = 0$$

and for κ

$$\kappa = -\frac{3}{2}\frac{E[X^2]}{E[X^3]} + \sqrt{\left(\frac{3}{2}\frac{E[X^2]}{E[X^3]}\right)^2 + \frac{6(1-\rho)}{\rho}\frac{E[X]}{E[X^3]}}$$

For the queue size distribution we find.

$$\begin{aligned} P(Q \geq b) &\approx \frac{\frac{\lambda}{\rho} - \lambda}{\frac{\lambda\kappa}{2}\frac{E[X^2]}{E[X]} + \frac{\lambda\kappa^2}{3}\frac{E[X^3]}{E[X]}} e^{-b\kappa} \\ &= \frac{1-\rho}{\rho}\frac{E[X]}{\frac{\kappa}{2}E[X^2] + \frac{\kappa^2}{3}E[X^3]} e^{-b\kappa} \end{aligned}$$

A.4 Configuration of ns2

Instead of trying to describe every experimental setting of ns2 in the main text, we include below the beginning of the experimental script used in most of the experiments. Complete scripts are available on the authors web page.

```
#
# TcpSim
# (c) Guido Appenzeller, 2003
#

# Defaults - May change for each experiment
set load 0.8          ;# 80% load
set bw   19.375       ;# OC3 speed, 155 Mb/s
set flows_data_kb 30  ;# 30 packets per flow
set bneck_buffer 1650 ;# Buffer of RTT*BW
set log 0
set long_n 0          ;# no of long (size infinity) flows
set time_length   50  ;# Duration
set bi 0              ;# One-way or two way traffic
set pareto 0          ;# Default is fixed size

# --- Flow Parameters

# Average amount of flow data in MBytes/s
set flows_bw_mb      [expr $load*$bw]  ;
# flows   - Average number of flows per second
set flows_n     [expr $flows_bw_mb*1000.0/$flows_data_kb*1.0];
# Average Inter Arrival Time
set flows_aiat  [expr 1/$flows_n];
```

```
# --- Bottleneck Link Parameters

set bneck_bw_mb  $bw      ;# MBytes / second
set bneck_lat    20       ;# miliseconds
set bneck_drop "DropTail"
set bneck_bw_n  [expr $bneck_bw_mb*8.0*1000]
set bneck_bw    ${bneck_bw_n}kb

# --- Access Link Parameters

set access_bw_n [expr $bneck_bw_n*100]
set access_bw ${access_bw_n}kb
set access_lat_min 20
set access_lat_max 30

# --- Packet parameters

set max_receiver_window 10000  ;# Maximum advertised receiver Window
set packet_len            960  ;# Length of packets

# --- Logging data

set time_slice      0.05            ;# Length of a time slice for logging
set time_startphase [expr $time_length/3] ;# Length of the startphase (util is not counted)

# ------ Distribution of Arrivals (Poission) ---------------------

set rng [new RNG]
$rng seed 0
set arrival_t [new RandomVariable/Exponential]
$arrival_t use-rng $rng

# ------ For pareto Flows (Poission) ----------------------------

set rng2 [new RNG]
$rng2 seed 0

# Important, what shape!
set pareto_shape 1.2

# This will be flow size
set npkts [new RandomVariable/Pareto]
$npkts use-rng $rng2
$npkts set avg_ $flows_data_kb
$npkts set shape_ $pareto_shape

remove-all-packet-headers    ;# removes all except common
```

```
add-packet-header Pushback NV
add-packet-header IP TCP Flags    ;# hdrs reqd for cbr traffic

# Nix routing rocks - this reduces memory by over a factor of two
$ns set-nix-routing

# --- Create Server, Router and bottleneck link ------------------

set n_server [$ns node]
set n_router [$ns node]

$ns duplex-link $n_server $n_router $bneck_bw ${bneck_lat}ms $bneck_drop
$ns queue-limit $n_router $n_server $bneck_buffer

for {set i 0} {$i < [expr $flows_nn]} {incr i} {
    set n_client($i) [$ns node]
    set lat_access [expr int(rand()*($access_lat_max-$access_lat_min))+$access_lat_min]
    set av_lat [expr $av_lat + $lat_access]
    $ns duplex-link $n_router $n_client($i) $access_bw ${lat_access}ms $bneck_drop
    $ns queue-limit $n_client($i) $n_router 32000
}
```

A.5 Cisco IOS Configuration Snippets for the VXR

This configuration snippet can be used to limit the sending rate of a shared memory router. Note that the token bucket size specified (here 10,000) for real line rates is far above the size of a single packet and that this incurs additional buffering that has to be accounted for.

```
! IOS snippet to shape traffic to 1 Mb/s with 64 packet queue
!
! Note: ** max-buffer does not show up in running-config
!          if you save/copy it to a drive or via tftp
!

! Define a packet class that matches any packet
class-map match-all guido-c1
```

```
  match any

! Define a policy map that shapes this class to 1 Mb/s
!    token bucket size is 10,000 Bits = 1.2 kByte
!    queue length is 64 packets
policy-map guido-map1
  class guido-c1
    shape average 1000000 10000 0
    shape max-buffers 64

! Activate policy for the interface
interface FastEthernet4/0
  service-policy output guido-map1
```

Bibliography

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

[2] Personal communication with stanford networking on characteristics of current stanford nework traffic.

[3] Guy Almes. [e2e] mailing list. Posting to the end-to-end mailing list, April, 2004.

[4] Youngmi Joo Anna Gilbert and Nick McKeown. Congestion control and periodic behavior. In *LANMAN Workshop*, March 2001.

[5] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. Technical Report TR04-HPNG-06-08-00, Stanford University, June 2004. Extended version of the paper published at SIGCOMM 2004.

[6] K.E. Avrachenkov, U. Ayesta, E. Altman, P. Nain, and C. Barakat. The effect of router buffer size on the TCP performance. In *Proceedings of the LONIIS Workshop on Telecommunication Networks and Teletraffic Theory*, pages 116–121, St.Petersburg, Russia, Januar 2002.

[7] Vijay Bollapragada, Curtis Murphy, and Russ White. *Inside Cisco IOS Software Architecture*. Cisco Press, 2000.

[8] L. Brakmo, S. O'Malley, and L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM*, pages 24–35, August 1994.

[9] R. Bush and D. Meyer. RFC 3439: Some internet architectural guidelines and philosophy, December 2003.

[10] 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.

[11] Inc. Cisco Systems. Netflow services solution guido, July 2001. `http://www.cisco.com/`.

[12] Constantine Dovrolis. [e2e] Queue size of routers. Posting to the end-to-end mailing list, January 17, 2003.

[13] Bohacek et al. A hybrid system framework for tcp congestion control. Technical report, University of California at Santa Cruz, 6 2002.

[14] Anja Feldmann, Anna C. Gilbert, and Walter Willinger. Data networks as cascades: Investigating the multifractal nature of internet WAN traffic. In *SIGCOMM*, pages 42–55, 1998.

[15] Dennis Ferguson. [e2e] Queue size of routers. Posting to the end-to-end mailing list, January 21, 2003.

[16] S. Floyd. RFC 3649: Highspeed TCP for large congestion windows, December 2003.

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

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

[19] S. Ben Fredj, T. Bonald, A. Proutière, G. Régnié, and J.W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *Proceedings of SIGCOMM 2001*, San Diego, USA, August 2001.

[20] Michele Garetto and Don Towsley. Modeling, simulation and measurements of queueing delay under long-tail internet traffic. In *Proceedings of SIGMETRICS 2003*, San Diego, USA, June 2003.

[21] John Hennessy and David Patterson. *Computer Architecture*. Morgan Kaufmann Publishers Inc., 1996.

[22] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of ACM SIGCOMM*, August 2003.

[23] Gianluca Iannaccone, Martin May, and Christophe Diot. Aggregate traffic performance with active queue management and drop from tail. *SIGCOMM Comput. Commun. Rev.*, 31(3):4–13, 2001.

[24] Sundar Iyer, R. R. Kompella, and Nick McKeown. Analysis of a memory architecture for fast packet buffers. In *Proceedings of IEEE High Performance Switching and Routing*, Dallas, Texas, May 2001.

[25] Van Jacobson. [e2e] re: Latest TCP measurements thoughts. Posting to the end-to-end mailing list, March 7, 1988.

[26] C. Jin, D. X. Wei, and S. H. Low. Fast tcp: motivation, architecture, algorithms, performance. In *Proceedings of IEEE Infocom*, March 2004.

[27] F. P. Kelly. *Notes on Effective Bandwidth*, pages 141–168. Oxford University Press, 1996.

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

[29] Microsoft. TCP/IP and NBT configuration parameters for windows xp. Microsoft Knowledge Base Article - 314053, November 4, 2003.

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

[31] Robert Morris. Scalable TCP congestion control. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, USA, March 2000.

[32] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[33] Lili Qiu, Yin Zhang, and Srinivasan Keshav. Understanding the performance of many TCP flows. *Comput. Networks*, 37(3-4):277–306, 2001.

[34] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[35] S. Shenker, L. Zhang, and D. Clark. Some observations on the dynamics of a congestion control algorithm. *ACM Computer Communications Review*, pages 30–39, Oct 1990.

[36] Cisco Support Web Site. Cisco 12000 series routers. `http://www.cisco.com/en/US/products/hw/routers/ps167/`.

[37] Cisco Support Web Site. Cisco ios quality of service solutions configuration guide. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fqos_c/fqcprt4/index.htm`.

[38] Cisco Support Web Site. Cisco line cards. `http://www.cisco.com/en/US/products/hw/modules/ps2710/products_data_sheets_list.html`.

[39] Cisco Support Web Site. Configuring class-based shaping - cisco ios quality of service solutions configuration guide. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fqos_c/fqcprt4/qcfcbshp.htm`.

[40] Cisco Support Web Site. Configuring generic traffic shaping - cisco ios software. `http://www.cisco.com/en/US/products/sw/iosswrel/ps1828/prod_configuration_guides_list.html`.

[41] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, Taormina, Italy*, October 2004.

[42] W. Richard Stevens. *TCP Illustrated, Volume 1 - The Protocols*. Addison Wesley, 1994.

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

[44] Ronald W. Wolff. *Stochastic Modelling and the Theory of Queues*, chapter 8. Prentice Hall, October 1989.

[45] Lixia Zhang and David D. Clark. Oscillating behaviour of network traffic: A case study simulation. *Internetworking: Research and Experience*, 1:101–112, 1990.

[46] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of ACM SIGCOMM*, pages 133–147, September 1991.