RATE CONTROL PROTOCOL (RCP): CONGESTION CONTROL TO MAKE FLOWS COMPLETE QUICKLY

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

Nandita Dukkipati
October 2007

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

(Nick McKeown) Principal Adviser

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

(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.

(Scott Shenker)

Approved for the University Committee on Graduate Studies.

Abstract

Users typically want their flows to complete as quickly as possible. This makes Flow Completion Time (FCT) an important—arguably the most important—performance metric for the user. Yet research on congestion control focuses entirely on maximizing link throughput, utilization and fairness, which matter more to the operator than the user. This thesis is about a new congestion control algorithm—Rate Control Protocol (RCP)—designed for fast download times (i.e., aka user response times, or flow completion times). Whereas other modifications/replacements to TCP (e.g. STCP, Fast TCP, XCP) are designed to work for specialized applications that use long-lived flows (scientific applications and supercomputer centers), RCP is designed for the typical flows of typical users in the Internet today.

We will show that with typical Internet flow sizes, existing (TCP Sack) and newly proposed (XCP) congestion control algorithms make flows last much longer than necessary—often by one or two orders of magnitude. In contrast, RCP makes flows finish close to the minimum possible, leading to a perceptible improvement for web users, distributed computing, and distributed file-systems. We also address several theoretical as well as practical issues under RCP—how RCP's flow completion times compare with TCP, XCP and ideal Processor Sharing, the impact of RCP's short FCTs for the general Internet, stability of an RCP network and how RCP copes with sudden network changes such as flash-crowds, RCP's buffering requirements at routers, implementation of RCP in routers and end-hosts, and how RCP can be incrementally deployed in real networks.

Acknowledgments

It has been my privilege to work with my adviser, Nick McKeown. He is one of the few individuals who has had a powerful impact on my thinking. Nick has shaped my approach to problem solving, doing research, presenting my work and teaching. Nick has provided one of the best research milieus to work in, plenty of opportunities to present my work outside of the group, and perhaps most importantly he kept me motivated during the many difficult times when working on RCP. In fact with utmost confidence, I can say that the past few years at Stanford have been the best years of my life so far—many thanks to Nick.

I would like to thank my dissertation committee members Prof. Balaji Prabhakar, Prof. Scott Shenker and Prof. Mendel Rosenblum. I have immensely enjoyed and learnt a lot from Prof. Prabhakar's classes related to Networking theory. Prof. Shenker sat through several of my talks on RCP, and has given invaluable comments that has improved my work.

It was my pleasure to have interacted and collaborated with several researchers along the way while working on RCP. My special thanks to Rui Zhang-Shen and Masayoshi Kobayashi who I worked with in the earlier stages of RCP, Hamsa Balakrishnan and Claire Tomlin who formalized the non-linear stable region of RCP, Ashvin Lakshmikantha and Prof. R. Srikant for collaborating on the buffering requirements of RCP routers, Glen Gibb for implementing the RCP algorithm in hardware on NetFPGA platform, Chia-Hui Tai and Jiang Zhu for simulations and experiments on incremental deployment of RCP. I have also learnt a lot from discussions on RCP with some eminent researchers of my field—Dr. Flavio Bonomi, Dr. David Clark, Prof. John Doyle, Dr. Sandy Fraser, Dr. Sally Floyd, Prof. Dina Katabi, Prof. Frank Kelly, Prof. Jim Kurose and Prof. Steven Low.

A very special thanks to my dear friend Rui Zhang-Shen for the many enjoyable times and discussions at our office. I would also like to thank the past and current members of the High Performance Networking Group at Stanford for the many lively discussions at group meetings and in making networking research so fun—Clay, Da, David, Gireesh, Glen,

Guido, Isaac, Jad, Jianying, John, Justin, Martin, Masa, Neda, Pablo, Sundar and Yashar.

I am grateful to Rui, Rong, Neda, Sundar, Da and Dr. Evelin Sull (from the English Department) for diligently proof-reading parts of my thesis. I thank Judy Polenta for meticulously taking care of all the administrative details and making it all look so seamless.

I am grateful to NSF for funding the research on RCP.

I would like to thank all my friends, both at Stanford and outside, for all their love and support throughout.

A very special thanks to my wonderful husband Subhachandra Chandra, not only for his immense support and understanding, but also for teaching me a lot of things about the Linux kernel which helped me when I was coding RCP. I have learnt so much from bouncing ideas off you. Thank you. Thanks to my family for their constant support, encouragement and enthusiasm: my parents, brother, grandparents, parents-in-law, Manju, Mahi, Naveen and Soujanya.

Finally, I have to thank all the coffee shops in and around Stanford and Palo Alto, where I routinely liked to work and had plenty of gulps of hot chocolate and caffeine.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet is a large distributed shared infrastructure. The network essentially consists of the following six crucial building blocks [5]:

- *Packets and Multiplexing*: A fundamental assumption of the Internet is that it is a packet-switched network. Data is broken down into packets which are independently routed on communication paths based on information in the IP header.

- *Naming, Addressing, and Forwarding*: These mechanisms are meant to get packets across the network. In the original Internet, every destination was assigned a global address, and any computer was allowed to send a packet to any other. Network devices, like routers and switches, look at the address in the packets to determine how to forward them.

- *Routing*: Computes the best paths from sources to destinations, and differs from forwarding in the sense that routing does the path computation whose results are then used by forwarding to take correct actions as packets arrive at the router.

- *Security*: Security mechanisms are meant to protect the infrastructure from malicious attacks as well as alleviate undesirable traffic including spam, denial-of-service attacks, and malicious traffic routed through zombie machines.

- *Network Management*: Includes the spectrum of functions ranging from managing and configuring low-level devices to setting high-level network-wide policies.

- *Congestion Control*: Fulfills the role of sharing network resources efficiently and fairly in some agreed-upon sense.

In this thesis we propose a new design for one of these crucial building blocks—a novel congestion control algorithm, called Rate Control Protocol (RCP). Before we go on to describe the motivation for RCP, we first explain how congestion control works today, what its shortcomings are, and in what ways it is a success.

1.1 Transmission Control Protocol

At the present time (2007), the Transmission Control Protocol, or TCP, is the most widely used congestion control mechanism. TCP fulfills two important functions. The first involves a reliable and in order delivery of bytes to the higher application layer. It builds on the unreliable, connectionless IP service, providing a service that is reliable by transmitting lost or corrupted data until the data is successfully received at the destination. It also delivers bytes in order (reorders out-of-order data and eliminates duplicates before delivering to the application process), multiplexes and de-multiplexes traffic from different processes on an end-host, and performs flow control (prevents a sender from overwhelming a receiver by specifying a limit on the amount of data that can be sent). TCP's second function is to perform congestion control and protect the network from a congestive collapse. We briefly describe TCP's congestion control mechanisms below.

TCP uses adaptive congestion control mechanisms that react to congestion events (such as packet loss or delay) by limiting the sender's transmission rate. These mechanisms allow TCP to adapt to heterogeneous network environments and varying traffic conditions, and keep the Internet from severe congestion events. TCP congestion control works on an end-to-end basis, where each connection, before starting, begins with a question [49]:

At what rate should the data be sent for the current network path?

It does not receive an explicit answer for this question, but each connection determines the sending rate by probing the network path and modulating its rate based on perceived congestion, through packet-loss and delay. The connection rate is proportional to TCP's sliding window (*swnd* is the limit on the amount of outstanding data in flight), which is set as the minimum of the receiver advertised window (*rwnd*) and of the congestion window (*cwnd* changes dynamically based on feedback of network conditions). To determine the congestion window, TCP employs the following mechanisms, shown in Fig. 1.1:

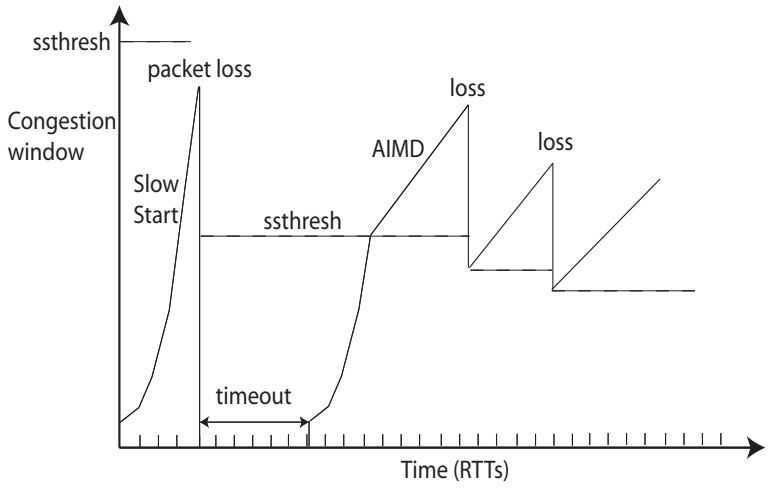


Figure 1.1: TCP's congestion control mechanisms: Slow-Start and Additive Increase Multiplicative Decrease (AIMD)

Each TCP connection starts with a pre-configured small initial congestion window (no larger than 4 Maximum Segment Size (MSS) [26]), and probes the network for available bandwidth using the *Slow-Start* procedure. The goal of Slow-Start is to keep a new sender from overflowing network buffers, while at the same time increasing the congestion window fast enough and avoiding performance loss while the connection is operating with a small window. Slow-Start increases the congestion window by one MSS for each new acknowledgment received, which results in the window doubling after each window's worth of data is acknowledged. With this exponential increase, $RTT\ log_2 W$ (where RTT stands for round-trip time) time is required to reach a window of size W. A connection enters Slow-Start on newly starting or on experiencing a packet retransmission timeout, and exits Slow-Start when it detects a packet loss or when the congestion window has reached a dynamically computed threshold, *ssthresh*. More specifically, *ssthresh* is set to half of the current congestion window when packet loss was detected.

TCP exits Slow-Start to enter the *Congestion Avoidance* phase, where it continues to probe for available bandwidth, but more cautiously than in Slow-Start. During periods when no packet losses are observed, TCP performs an *Additive Increase* of the window size, by 1 MSS each time a full window is acknowledged (i.e., increases the congestion window as $cwnd = cwnd + \frac{1}{cwnd}$ on receiving each acknowledgment packet). The reaction to congestion indication (packet loss) varies across different flavors of TCP, and we will briefly describe

the four most commonly used variants below.

TCP Tahoe: Tahoe congestion control, described in a paper by Jacobson [43], was introduced in 1987 in response to a series of "congestion collapse" events (a state where the network is live-locked, performing little useful work). On detecting a packet loss (either through retransmission timeout or three duplicate acknowledgment packets), a Tahoe sender records the *ssthresh* as $\frac{cwnd}{2}$ value (the initial value of *ssthresh* is usually set to the receiver window size), sets *cwnd* to 1 MSS and enters Slow-Start. It continues Slow-Start so long as $cwnd < ssthresh$, and is in Congestion Avoidance beyond that.

TCP Reno: This is the second version which differs from TCP Tahoe when detecting packet loss through three duplicate acknowledgment packets (an indication of a milder congestion). TCP Reno reduces the *cwnd* by half (as opposed to reducing the window to one like in Tahoe) to achieve a higher sending rate after loss recovery. The procedure implementing this is called *Fast Recovery*.

TCP NewReno: NewReno also reduces *cwnd* by half on detecting a packet loss through three duplicate acknowledgments, but NewReno improves upon Reno when retransmitting multiple packet losses. Reno fails to recover efficiently from multiple packet losses in a window. After transmitting the first lost segment, it typically waits for the retransmit timer to expire, in order to recover the remaining lost segments. When a received acknowledgment does not acknowledge all outstanding segments, NewReno retransmits the missing segment.

TCP SACK: A TCP extension called Selective Acknowledgment improves further on NewReno's retransmission mechanisms. SACK allows the receiver to indicate up to four non-contiguous blocks of sequence numbers received correctly, thus allowing the sender to retransmit lost data more efficiently. This is currently the most widespread TCP version in the Internet.

Different TCP versions and how they evolved over time are described extensively and surveyed in many papers [3], books [1], theses [2], and RFCs [4]. TCP's congestion control mechanisms of Slow-Start and Additive Increase Multiplicative Decrease have performed well in preventing extreme congestion even as the Internet has scaled up many times in link speeds, network size, traffic load, and heterogeneity. However as the bandwidth of the network has increased, in other words as the network bandwidth-delay product or the "pipe" size has increased, TCP has substantially lagged in utilizing network bandwidth, especially for large transfers and dynamic traffic patterns. The rest of this chapter first describes the problems with current TCP (Sec. 1.2). It then lists ideal properties we would

like in congestion control (Sec. 1.3), and which of these properties are achieved by existing mechanisms (High-speed TCPs - Sec. 1.4), by newly proposed mechanisms (eXplicit Control Protocol - Sec. 1.5), and by the mechanisms this thesis proposes (Rate Control Protocol - Sec. 1.6).

1.2 Problems with TCP

It has been demonstrated experimentally and shown mathematically that the most widely used TCP,[1] behaves badly in a large bandwidth-delay network. The following is a summary of its shortcomings (some of these are also summarized in [11]):

1. In a high bandwidth-delay product environment TCP's additive increase of one packet per round-trip time means flows will take a long time to acquire any spare capacity. For example, when the peak window size is 80,000 packets, which is needed to sustain 7.2 Gbps with an RTT of 100 ms and a packet size of 1500 Bytes, 40,000 RTTs or almost 70 minutes, is required to recover from a single packet loss [25].

2. TCP relies on packet-loss feedback to modulate its sending rate. The end-to-end loss probability needs to be impractically small for a TCP flow to be able to sustain a large equilibrium window [25], making it hard for a high-speed connection to obtain a large throughput in a high bandwidth-delay environment. This is another consequence of TCP's slow additive increase and drastic multiplicative decrease. For example, a simple model for the steady state TCP throughput for large bulk transfers (in the AIMD phase), with loss probability p, is [35]:[2]

$$\text{Throughput (Bytes/sec)} = \frac{MSS}{RTT}\frac{K}{\sqrt{p}}$$

where K is a constant that depends on the assumed loss pattern (for example, random or periodic) and whether acknowledgments are delayed or not. It can be easily seen that to sustain a throughput of 10 Gbps with a round-trip time of 100 ms, the loss probability cannot be more than 1 in 5 billion packets.

[1] TCP SACK [57] at the time of writing.

[2] More sophisticated models taking into account delayed acknowledgments, retransmission timeouts, periodic losses, receiver window limitations etc. are summarized in [2].

3. TCP gets confused by lossy links. It uses packet loss as a binary indicator of congestion, treating lossy links, such as wireless, as congested networks and under-utilizing them [58].

4. TCP shares bandwidth inversely proportional to the round-trip times. TCP flows with long round-trip times (RTT), such as those going through satellite links, have difficulty obtaining their fair share of bandwidth on a bottleneck link [59]. Once they enter the AIMD mode, long RTT flows open their windows much slower and lose out to short RTT flows.

5. TCP's Slow-Start makes short flows last much longer than necessary. Even if a flow is capable of completing within one round-trip time, TCP's Slow-Start makes it take multiple round-trip times to find its fair share rate. Often the flow completes before it reaches its fair share rate. As the bandwidth-delay product increases, more and more flows will be capable of completing within one RTT, and so this inefficiency will increase over time.

6. Finally, TCP deliberately fills up any amount of buffering available just before the bottleneck link. Extra buffers mean extra delay which adds to the duration of the flows.

Proposing band-aid solutions to these problems has its advantages in the short-term. We will however, take a fresh look at the wish list of properties we would like to achieve in congestion control, and propose a solution to get us there.

1.3 A Wish List for Congestion Control

What is the ideal list of qualities we would want in a congestion control mechanism?

Fig. 1.2 divides our list into two broad categories: A) Flow and network level properties and B) Properties related to implementation and ease of deployment. For A) the properties we desire are

1. Processor Sharing: We would like to share the bottleneck link equally among competing flows. Emulating Processor Sharing (PS) is a simple way to do this. Processor Sharing is a worthwhile goal to achieve: Even if its mean flow completion time is not quite the minimum achievable, it comes reasonably close to the minimum, and so

Wish List

A. Flow and Network level properties:

1. Processor Sharing
   - Performance invariant of flow-size distribution
   - Mix of flows: Short flow completion times
   - Long flows: 100% link utilization
   - Fair sharing
2. Stable
3. Close to zero queuing delay. Loss free network.
4. Efficient use of high bandwidth-delay links
5. Proportional bandwidth sharing
6. Any network conditions
7. Any traffic mix

B. Implementation and Deployment:

1. Police flows
2. No per-flow state or queue
3. No per-packet computation in routers

Figure 1.2: Wish-list of properties for a congestion control mechanism.

flows complete quickly (in fact, as we will see in Chap. 2, often an order of magnitude quicker than in TCP for typical Internet size flows). Furthermore, its mean completion time is invariant of flow size distribution for a single bottleneck. Even when flow completion time does not make sense (e.g., in long-lived bulk transfers), Processor Sharing results in flows getting high throughput and fair sharing of the bottleneck bandwidth.

2. Stability: Networks occasionally experience sudden large traffic surges (e.g., flash-crowds). We want the network to come back to a stable operating behavior quickly after such disruptions.

3. Queuing delay and packet losses: Ideally, we want close to zero buffer occupancy at all times. Queued up packets in buffers mean extra latency for every packet. This is a problem for short flow performance because queuing delay in buffers can be a significant portion of flows' completion time (as we will see in Chap. 3). If possible, we would like to achieve close to zero buffer occupancy or a loss-free network.

4. Efficiency: Naturally, at the same time we do not want to sacrifice the efficiency of high bandwidth-delay links such as the long haul fiber-optic links. These links, which often go through difficult terrains, are expensive, and service providers like to minimize unused capacity when the sources have traffic to send.

5. Differential bandwidth sharing: When need be, we would like to achieve some kind of differential bandwidth sharing among flows—for example, if we would like to give an important file transfer temporarily ten times the bandwidth share we give to a less urgent background movie download, congestion control should be able to achieve that.

6. Network and traffic conditions: We want to achieve the above under any network conditions such as different round-trip times, bandwidth-delay products—including challenging conditions like wireless links with long delays, and high loss rates—and similarly under any traffic conditions such as short flows, long flows, or any mix of flows, flash crowds and so on.

   For implementation and deployment (B) the properties we would like to achieve are

7. Policing: Today, it is hard to determine whether flows are following TCP congestion control and hence to enforce their doing so [60]. We would like a simple way to ensure that flows are adhering to congestion control and that a few bandwidth hoggers (presumably with broken or non-compliant algorithms) are not using up more than their fair share of the link-rate.

8. No per-flow state or queue and no per-packet computations: A scheme can be more easily deployed if it does not involve routers in congestion control and only requires upgrades to end-host software. While that would be ideal, short of that we would like to achieve these properties without any per-flow state and queue, or per-packet computations in routers.

Some of these qualities have inherent trade-offs. For example, to achieve short flow completion times we want to aggressively push the data into the network to get it across as soon as possible; on the other hand to achieve close-to-zero buffer occupancy at all times, packets need to be sent conservatively, which prolongs flow completion times. We describe below which of these qualities are achieved by existing and by our newly proposed congestion control algorithm.

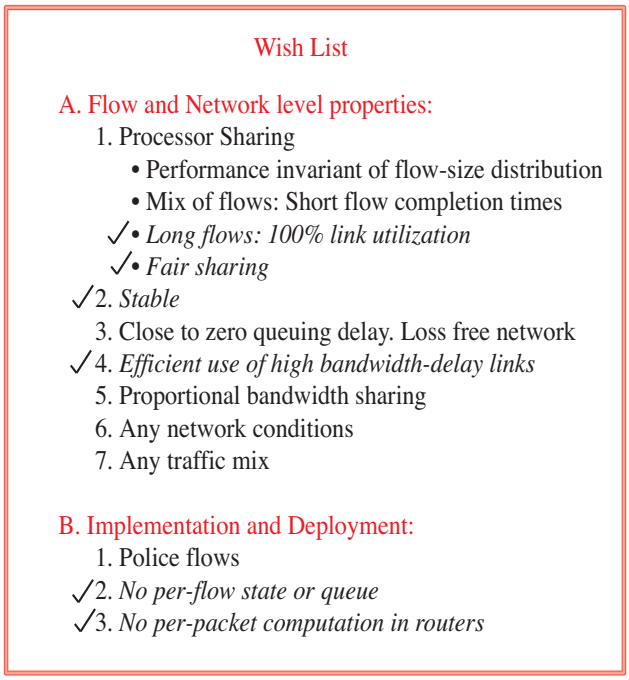


Figure 1.3: Wish-list properties that high-speed TCPs achieve (those shown in italics and check marks near the properties).

1.4 High-speed TCPs: Pros and Cons

Fig. 1.3 shows the goals achieved by high-speed TCPs, including the many recent TCP flavors, such as FAST TCP [42], BIC [51] and CUBIC TCP [52] (default in Linux from 2.6.19), H-TCP [53], Highspeed TCP [25], Scalable TCP [27], Westwood TCP [55], Compound TCP [54] (used in Windows Vista), and many more.

The biggest problem these TCPs are aiming to solve is: How can a few very high-bandwidth flows share a long fat pipe efficiently and fairly. This is a niche problem that arises in applications such as data-center backups and scientific computing. For example, the High Energy and Particle Physics community routinely needs to transfer a few terabyte physics data-sets between SLAC (Stanford) and CERN (Geneva) [56]—so their goals are primarily to maintain sustained high flow throughput by maximizing network efficiency and attaining some sense of fairness among flows. These TCPs are essentially trying to improve the steady-state performance of long-lived large bandwidth flows. They do a very good job—while maintaining network stability—so long as flows are long-lived and the network

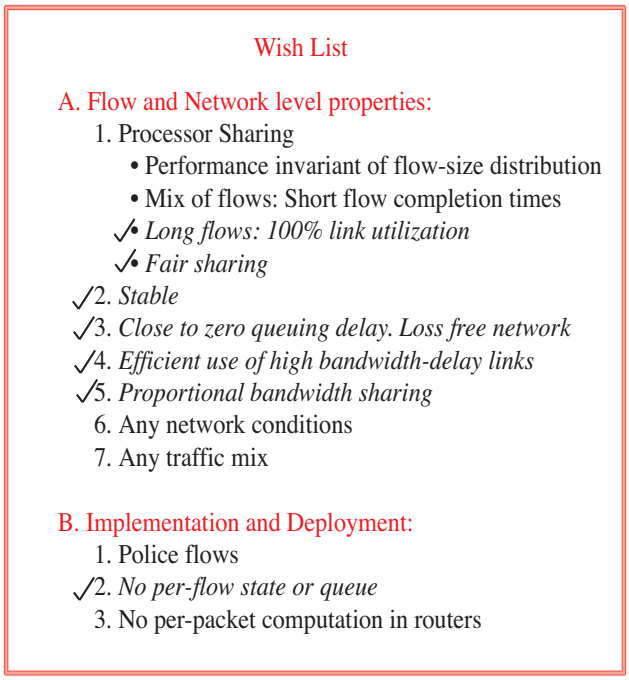


Figure 1.4: Wish-list properties that XCP achieves (those shown in italics and check marks near the properties).

and traffic conditions are not too challenging, i.e., do not have flows with a heterogeneous mix of round-trip times, or a traffic mix with different flow-sizes. In Chap. 2, we will see some specific examples of how current TCPs do not meet the wish-list properties.

But perhaps the biggest plus of these efforts is that they only require changes to the end-host, in fact most times only the sender side needs to be modified and needs no cooperation from the routers. The downside is that (in the absence of explicit information from the network) they are forced to resort to heuristics based on either packet loss or delay, as a consequence of which they generally find it hard to work well under traffic conditions that deviate from a set of long-lived flows or, as we will see in Chaps. 2 and 4, when flows are short.

1.5 eXplicit Control Protocol (XCP): Pros and Cons

While there have been many research efforts to solve TCP's problems of large data transfers in high bandwidth-delay networks, one of the boldest attempts so far is the eXplicit Control

Protocol [11].

XCP works by involving the routers in congestion control. The network explicitly tells the receiver the state of congestion and how to react to it. This allows senders to adjust their windows based on the precise feedback information. XCP carries the per-flow congestion state in packets, requiring no per-flow state in routers. XCP senders specify a desired throughput increase in packet congestion headers, which the routers modify to give a bandwidth increment or decrement based on the link congestion conditions. The novelty in XCP is the concept of decoupling the link efficiency control from the flow fairness control. XCP controls link utilization by adjusting its aggressiveness to the spare bandwidth and the feedback control delay, thus achieving stability and efficiency even for large bandwidth-delay product networks. It controls fairness by (conservatively) managing the bandwidth distribution among flows. More specifically, it reclaims bandwidth from flows with rates above their fair share and distributes it to flows with lower rates. New XCP flows start with a small window size and thereafter receive a window increment/decrement. At any time, XCP flows can have different window sizes, different round-trip times, and different rates. XCP continuously tries to converge to the point where the link is efficiently utilized and all flows have their fair-share rate.

Fig. 1.4 shows XCP's pros and cons. XCP essentially solves the same problem as other high-speed TCPs—how can a few high bandwidth transfers share a long fat pipe efficiently and fairly. When flows are long-lived, XCP will eventually converge so that every flow gets its fair share rate and there is 100% link utilization. But most importantly, XCP achieves this while keeping the buffer occupancies close to zero, and with almost no packet loss. The shortcomings of XCP for the general Internet are

- XCP solves the problems with TCP in a static scenario in which there are a fixed number of flows having an infinite amount of data to send. In this scenario it emulates Processor Sharing by giving each flow eventually an equal share of the bottleneck link. But in practice, flows arrive randomly and transfer a finite arbitrary amount of data. In this dynamic environment with a mix of flow sizes, XCP not only deviates far from Processor sharing but in fact does far worse than TCP. It is inefficient and unfair (more examples of this, are in Chap. 4).

- When there is a mix of flow sizes, XCP typically makes the flows last two orders of magnitude longer than necessary.

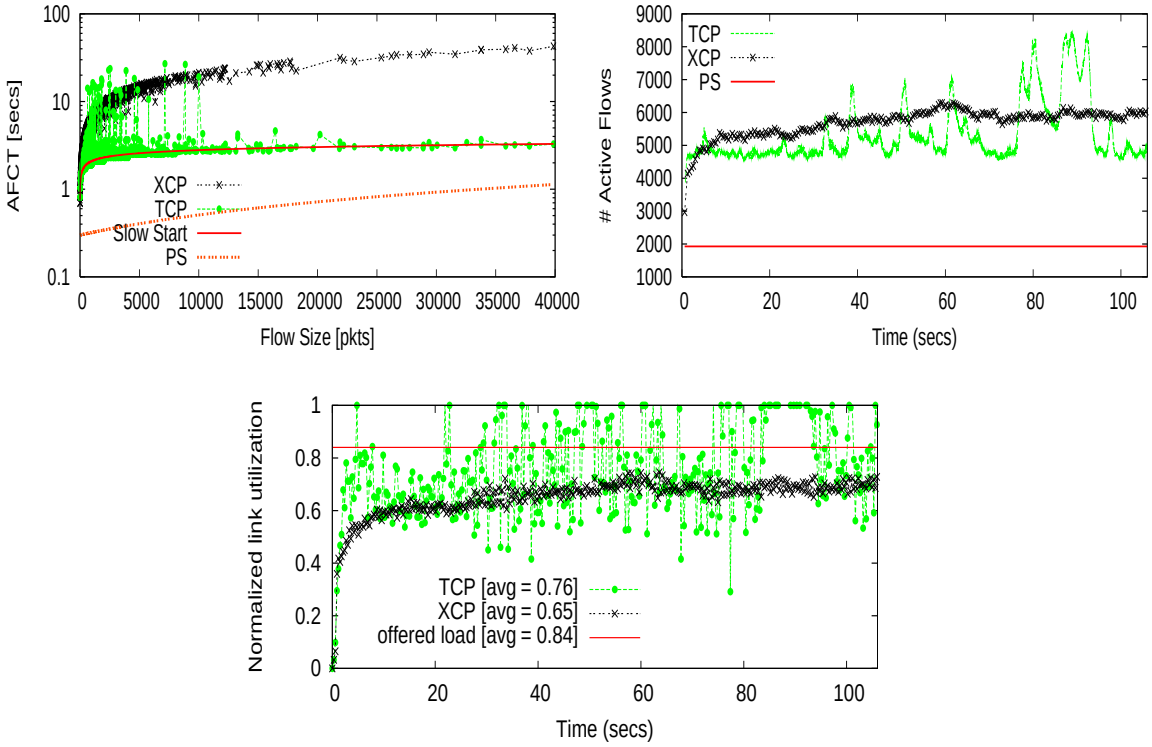


Figure 1.5: The top left plot shows the average flow completion time (AFCT) versus flow size under TCP and XCP from a simulation with Poisson flow arrivals. Flow sizes are Pareto distributed with mean = 50 pkts (1000 byte/pkt) and shape = 1.2, link-capacity = 2.4 Gbps, Round-Trip Time = 200 ms, offered load = 0.84. The top right plot shows the number of active flows versus time. The bottom plot shows the link utilization for the two protocols measured over every 100 ms.

- XCP requires detailed per-packet calculations in the routers: 4 floating-point multiplications and 10 additions for every packet [13], which will be too slow to be implemented in software for high-speed routers and a burden to implement in hardware, particularly since the algorithm is experimental.

An example, representative of the many simulations we performed, comparing XCP and TCP with Processor Sharing (PS) is shown in Fig. 1.5. The plot compares the mean flow durations, number of active flows over time and the link utilization, in a setup of high bandwidth-delay product with a mix of flow sizes. Often the metrics used in the literature for evaluating congestion control, such as link utilization or convergence to fairness, are network-centric and do not mean much from a user's view point. We instead decided to

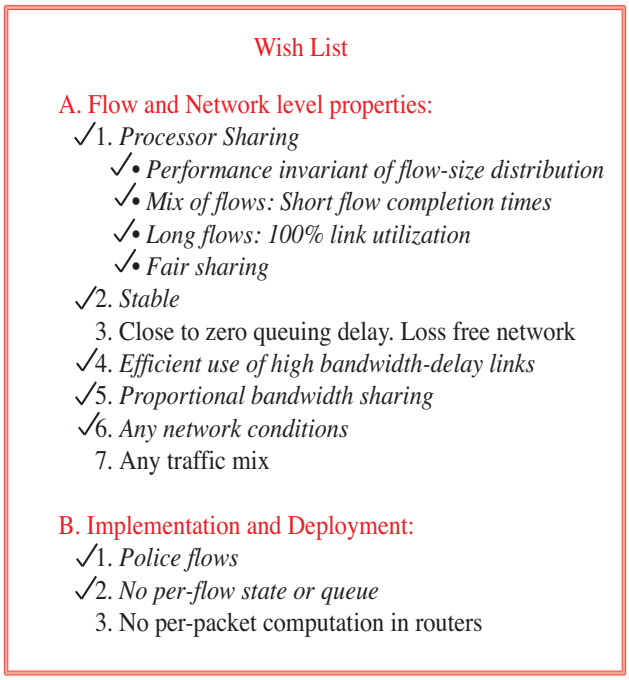


Figure 1.6: Wish-list properties that RCP achieves (those shown in italics and check marks near the properties).

focus on the metric Flow Completion Time (FCT), or how quickly a flow finishes, because that is what a user is mostly concerned about. Chap. 2 provides a more detailed reasons for why it is important to finish flows quickly.

When there is a mix of flows, TCP and XCP make flows take much longer to complete than they would take under ideal PS. The goal of RCP, conversely, is to achieve performance close to the Processor Sharing delay while doing very little work at the routers.

1.6 Rate Control Protocol (RCP): Pros and Cons

In this section we summarize Rate Control Protocol very briefly (it is described in detail in Chap. 3) only so as to make some early comparisons with TCP and XCP and to set the stage for what is to come later.

In the basic RCP algorithm that we propose, a router maintains a single rate, $R(t)$, for every link. The router "stamps" $R(t)$ on every passing packet (unless it already carries a slower value). The receiver sends the value back to the sender, thus informing it about

the slowest (or bottleneck) rate along the path. In this way, the sender quickly finds out the rate it should be using (without the need for Slow-Start). The router updates $R(t)$ approximately once per roundtrip time, and strives to emulate Processor Sharing among flows.

The foremost goal in RCP is short flow completion times, or in other words to make a flow complete as quickly—after all this is what most users experience and care about when they interact with the network in any way. Fig. 1.6 shows RCP's pros and cons. The biggest plus of RCP is the short flow completion times under a wide range of network and traffic characteristics—which are in fact quite close to what flows would achieve if they were processor-shared. It turns out that as a consequence of this, RCP also achieves the other goals that high-speed TCPs and XCP achieve, which is efficient and fair network usage in the presence of a few high-bandwidth transfers. In summary, there are four main features of RCP that make it an appealing and practical congestion control algorithm

- RCP is inherently fair (all flows at a bottleneck receive the same rate).
- RCP's flow completion times are often one to two orders of magnitude shorter than those of TCP-Sack and XCP, and close to what flows would have achieved if they were ideally processor-shared. This is because RCP allows flows to jump-start to the correct rate (because even connection set-up packets are stamped with the fair-share rate). Even short-lived flows that perform badly under TCP (because they never leave slow-start) will finish quickly with RCP. And equally importantly, RCP allows flows to adapt quickly to dynamic network conditions in that it quickly grabs spare capacity when available and backs off by the right amount when there is congestion, so flows do not waste RTTs in figuring out their transmission rate.
- There is no per-flow state or per-flow queuing.
- The per-packet computations at a RCP router are simple.

On the other hand, the biggest downsides of RCP are

- RCP involves the routers in congestion control, so it needs help from the infrastructure. Although they are simple, it does have per-packet computations.
- Although the RCP algorithm strives to keep the buffer occupancy low most times, there are no guarantees of buffers not overflowing or of a zero packet loss. This

becomes especially acute in situations such as sudden flash-crowds, where although RCP recovers quickly, there will be transient spikes in the queue and possibly packet losses.

The rest of this thesis, arranged as follows, describes how RCP works and studies in detail its pros and cons: Chap. 2 motivates why flow completion time is the right metric for congestion control; Chap. 3 describes Rate Control Protocol; Chap. 4 focusses on RCP's flow completion times, discussing how they compare with TCP, XCP, and PS, and what the impact of RCP's short FCTs would be for the general Internet; Chap. 5 explores whether RCP is stable when there are sudden traffic and network changes; Chap. 6 delves into RCP's practical considerations, such as how it is implemented on real systems, what its buffering requirements are at routers, and how it can be incrementally deployed in legacy networks; Chap. 7 discusses the most commonly asked questions on RCP; Chap. 8 is the conclusion.

Chapter 2

Why we should make flows complete quickly

In this chapter we explain why completing flows fast is an important goal for RCP, how the existing and newly proposed congestion control algorithms do not come close to minimizing download times, and why minimizing flow completion times is a hard problem in both theory and practice [16] [17].

When users download a web page, transfer a file, send/read email, or involve the network in almost any interaction, they want their transaction to complete in the shortest time; and therefore, they want the shortest possible flow completion time (FCT).[1] They care less about the throughput of the network, how efficiently the network is utilized, or the latency of individual packets; they just want their flow to complete as fast as possible. Today, most transactions are of this type and it seems likely that a significant amount of traffic will be of this type in the future [6] [7].[2] Short FCTs also reduce the control loop delay of distributed applications interacting over the network. So it is perhaps surprising that almost all work on congestion control focuses on metrics such as throughput, bottleneck utilization and fairness. While these metrics are interesting - particularly for the network operator - they are not very interesting to the user; in fact, high throughput or efficient network utilization is not necessarily in the user's best interest. Certainly, as we will show, these metrics are not sufficient to ensure a quick FCT.

[1] FCT = time from when the first packet of a flow is sent (in TCP, this is the SYN packet) until the last packet is received.

[2] Real-time streaming of audio and video are the main exceptions, but they represent a tiny fraction of traffic.

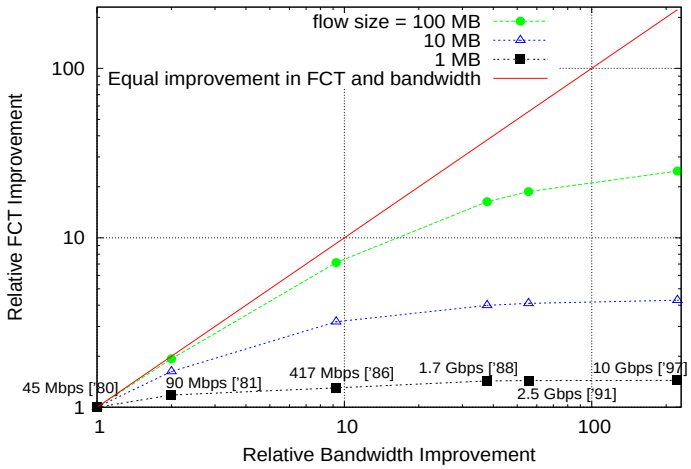


Figure 2.1: Improvement in flow completion time as a function of link bandwidth for the Internet, normalized to 45 Mbps introduced in 1980. Flows have an RTT of 40 ms and complete in TCP slow-start. Plot inspired by Patterson's illustration of how latency lags bandwidth in computer systems [8].

Intuition suggests that as network bandwidth increases flows should finish proportionally faster. For the current Internet, with TCP, this intuition is wrong. Fig. 2.1 shows how improvements in link bandwidth have not reduced FCT by much in the Internet over the past 25 years. With a 100-fold increase in bandwidth, FCT has reduced by only 50% for typical downloads. While propagation delay will always place a lower bound, FCT is dominated by TCP's congestion control mechanisms which make flows last multiple RTTs even if a flow is capable of completing within one round-trip time (RTT).

So can we design congestion control algorithms that make flows finish quickly? Unfortunately, it is not usually possible to provably minimize the FCT for flows in a general network, even if their arrival times and durations are known [9] [10]. Worse still, in a real network flows come and go unpredictably and different flows take different paths! It is intractable to minimize FCT. So instead congestion control algorithms are focused on efficiently using a bottleneck link (and only for long-lived flows) because this is easier to achieve. But we believe - and it is the main argument of this chapter - that instead of being deterred by the complexity of the problem, we should find algorithms that come close to minimizing FCTs, even if they are heuristic.

A well-known and simple method that comes close to minimizing FCT is for each router

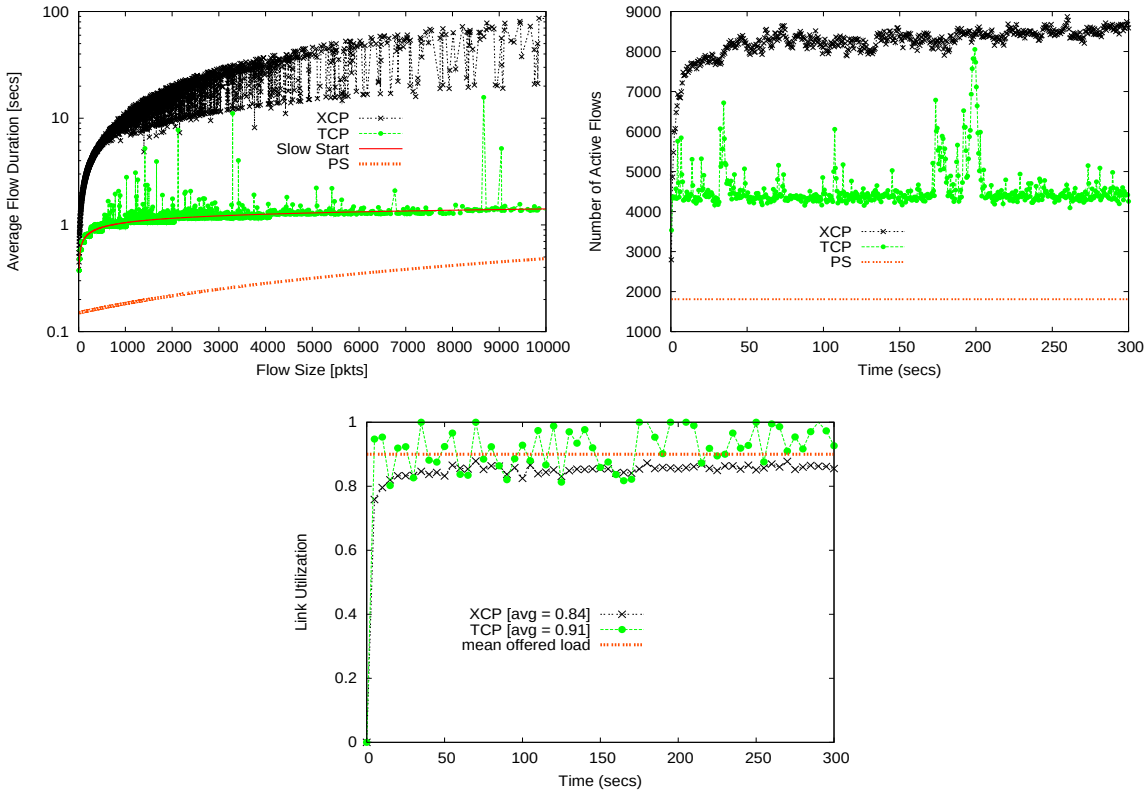


Figure 2.2: The top plot shows the average flow duration versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 30 pkts (1000 byte/pkt) and shape = 1.4, link-capacity = 2.4 Gbps, Round-Trip Time = 100 ms, offered load = 0.9. The middle plot shows the number of active flows versus time. In both plots the PS values are computed from analytical expressions. The bottom plot shows the offered load and the link utilization for the two protocols measured over 100 ms time intervals.

to use processor-sharing (PS) - a router divides outgoing link bandwidth equally among ongoing flows. On the face of it, TCP seems to approximate PS - if several infinitely long TCP flows with the same round-trip time share a bottleneck, TCP will eventually converge on a fair-share allocation. Similarly, eXplicit Control Protocol (XCP) [11] will converge on the fair-share allocation by gradually (explicitly) assigning excess bandwidth to slow flows and reducing the rate of fast flows. But because they react over many round-trip times, neither TCP nor XCP come close to processor-sharing for a mix of flow-sizes that represent the current usage of the Internet: both have expected FCTs typically one or two orders of magnitude larger than need-be.

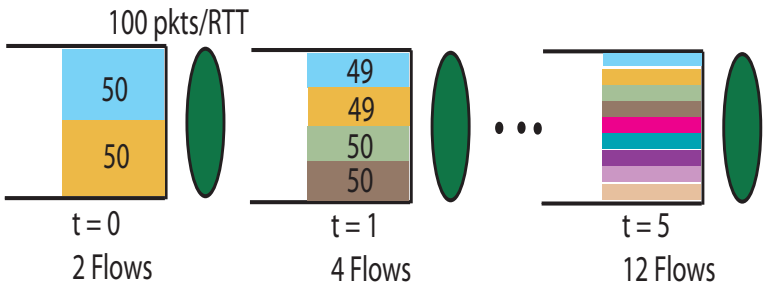


Figure 2.3: An example illustrating how flows in TCP slow-start accumulate over time.

To illustrate how much longer flows take to complete with TCP and XCP, when compared to ideal PS, we used ns-2.29 [24] to obtain the results shown in Fig. 2.2. The simulation conditions (explained in the caption) were chosen to be representative of traffic over a backbone link today, and this graph is representative of hundreds of graphs we obtained for a variety of network conditions and traffic models. The values for PS are derived analytically, and show that flows would complete an order of magnitude faster than for TCP. There are several reasons for the long duration of flows with TCP. First, it takes "slow-start" several round-trip times to find the fair-share rate. In many cases, the flow has finished before TCP has found the correct rate. Second, once a flow has reached the "congestion-avoidance" mode, TCP adapts slowly because of additive increase. While this was a deliberate choice to help stabilize TCP, it has the effect of increasing flow duration. A third reason TCP flows last so long is because of buffer occupancy. TCP deliberately fills the buffer at the bottleneck, so as to obtain feedback when packets are dropped. Extra buffers mean extra delay, which add to the duration of a flow.

Our plots also show eXplicit Control Protocol (XCP). XCP is designed to work well in networks with large per-flow bandwidth-delay product. The routers provide feedback, in terms of incremental window changes, to the sources over multiple round-trip times, which works well when all flows are long-lived. But as our plots show, in a dynamic environment XCP can increase the duration of each flow even further relative to ideal PS, and so there are more flows in progress at any instant.

So why do TCP and XCP result in such long flow durations? In the following, we will try to explain why both mechanisms prolong flows unnecessarily. There seem to be four main reasons: (1) Flows start too slowly and are therefore artificially stretched over multiple round-trip times, (2) Bandwidth is allocated unfairly to some flows at the expense of

others; either statically (e.g. TCP favors flows with short RTTs), or dynamically (XCP allocates excess bandwidth slowly to new flows), (3) Buffers are filled (TCP) and therefore delay all packets, and (4) Timeouts and retransmissions due to packet losses (TCP). We will examine each reason in turn, and use simple examples to clarify each factor.

1) Stretching flows to last many Round-Trip Times (RTT) even if they are capable of finishing within one/few RTTs

TCP: When a TCP flow starts, the source doesn't know the rate to use, and so it uses a small value to start with and ramps the rate over multiple RTTs. This is the well-known slow-start phase. Essentially, the source starts with a conservative rate, and forces the flow to last multiple RTTs. Hence, a typical flow today never leaves slow-start; and a flow of size L has a FCT given by $[log_2(L+1)+1/2]\cdot RTT + L/C$ (excluding the queuing delay). For example with a typical short flow size today of 15 packets, an RTT of 200 ms and a user connected via 1 Mb/s link, TCP will force the flow to last 800 ms - about 4 times longer than the minimum possible. Over time, as bandwidth-delay products increase, the discrepancy will get worse. For example, in Fig. 2.3 the link capacity is 100 packets/RTT. Two flows, each with size 50 packets, arrive at the start of every RTT beginning from t = 0. In PS, both flows would complete in one RTT, the equilibrium number of flows in system is 2 and the link utilization would be 100%. With TCP slow-start, the number of flows in the system evolves as shown in Fig. 2.3. In steady-state there are 12 flows - six times higher than for PS; consequently the flow duration is six times higher than need-be.

The problem is not limited to slow-start. As Fig. 2.2 shows even flows that have entered AIMD phase have long FCTs. This is because AIMD increases the window sizes even more slowly.

XCP: XCP can be even more conservative in giving bandwidth to flows than TCP, particularly to new flows. This is why there are always more active, incomplete flows. XCP gradually reduces the window size of existing flows and increases the window size of new flows, making sure the bottleneck is never over-subscribed. It takes multiple RTTs for most flows to reach their fair share rate (which is changing as new flows arrive). Many flows complete before they reach their fair share rate. In general, XCP stretches flows over multiple RTTs to avoid over-subscribing the bottleneck link. This prolongs flows and so the number of active/ongoing flows grows, which in turn reduces the rate of new flows, and so on. Our many simulations showed that new flows in XCP start slower than with slow-start in TCP.

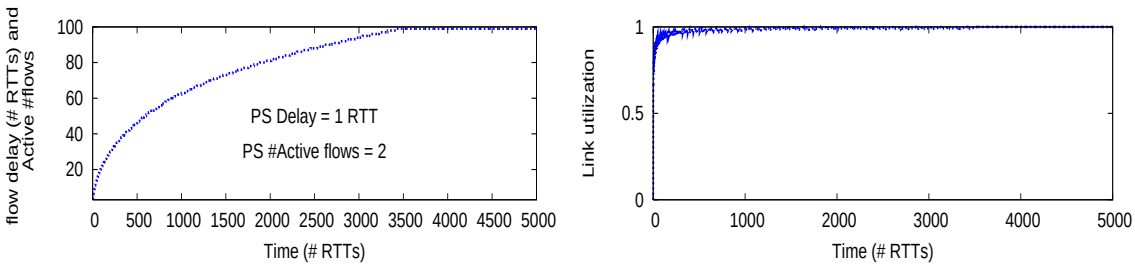


Figure 2.4: XCP flows can accumulate over time. Two flows arrive at the start of every RTT, with flow sizes 99 pkts (flow 1) and 1 pkt (flow 2); link of capacity = 100 packets/RTT. In the first RTT, the server asks both flows to send $\frac{C \times RTT}{2} = 50$ packets. Flow 2 completes; flow 1 waits for more RTTs competing for bandwidth with newly arriving flows. At the start of second RTT, there are 3 flows so the server gives out $\frac{C \times RTT}{3} = 33.33$ packets to each of 3 flows. This continues until the system reaches a steady-state of 100 flows.

Every control interval, XCP carefully hands out spare capacity to ongoing flows. While this works well if all flows are long-lived, it is inefficient for a typical mix of flow-sizes. Short flows - that are about to finish - cannot use the bandwidth they are given while other flows which deserve more bandwidth land up waiting more RTTs for their share. The following example shows how XCP leads to long FCTs even when the link utilization is high. Consider two new flows starting every RTT, flow 1 has 99 packets, flow 2 has 1 packet and the link-capacity equals 100 packets/RTT. In the k^{th} RTT each flow is told to send $(C \times RTT)/N(k)$ packets where $N(k)$ is the number of ongoing flows in RTT k.[3] Fig. 2.4 shows that even though the link utilization is eventually 100%, the number of flows keep growing until there are 100 competing flows each sending one packet per RTT. The FCT in this system is 50 times worse than ideal PS, even while the link utilization in both is 100%.

2) Bandwidth hogging

Both TCP and XCP allow flows to hog the bottleneck link, which increases the FCT for other flows. TCP does this by favoring flows with short RTTs; in fact, flows with long RTTs can be made to have arbitrarily small rates (i.e. a large FCT) even when the bottleneck link is fully utilized. In addition, both TCP and XCP allocate excess bandwidth slowly to new flows. So when there is a mix of flow-sizes (for example, the heavy-tailed mix of flow-sizes in today's Internet), the long flows will converge on their fair share, while the short flows don't have time to reach their fair share before they finish. This is illustrated in Fig. 2.5 in which a long flow keeps the link occupied, and three new flows start in the same RTT,

[3] Although not identical, this is similar to what XCP would do.

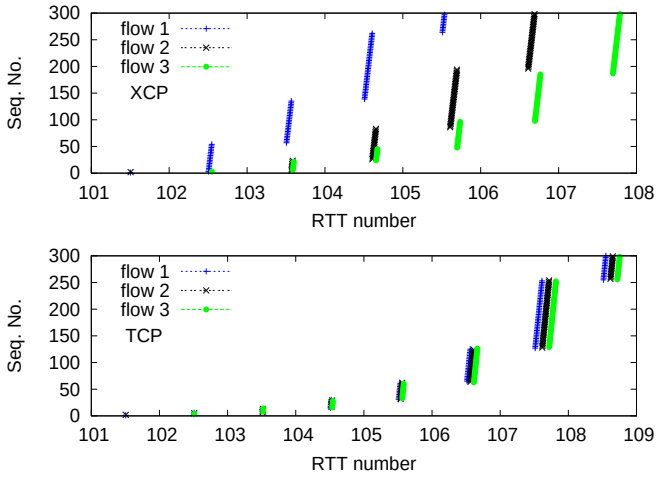


Figure 2.5: Example illustrating unfair bandwidth sharing. A long flow is keeping the link occupied, 3 flows of size 300 pkts each, start in RTT number 100. Before TCP and XCP get a chance to take bandwidth away from the long flow and give these new flows their fair share, the flows are done. Link capacity = 100 Mbps, RTT = 0.1s. Under PS, the three flows would have finished in 1 RTT.

each of size 300 packets. With PS, the three flows would finish in one RTT; but XCP and TCP make them last eight times longer.

The problem is in general more pronounced in XCP. When link is close to full utilization only 10% of link capacity is available for all newly arriving flows through bandwidth shuffling - bandwidth is slowly reclaimed from ongoing flows and distributed to new flows. Favoring early flows over new flows nudges XCP toward an M/G/1-FCFS discipline (for heavy-tailed jobs FCFS has the worst mean job-completion time among many known scheduling disciplines), instead of the more desirable M/G/1-PS.

3) Filling up buffers

TCP will try to fill the buffers at the bottleneck link - until a packet is dropped. While this leads to high utilization of the bottleneck link, it increases queuing delay, RTT and the FCT. Flows that arrive when queue occupancy is high will experience large and highly variable delays. In this regard, XCP is better than TCP because it always drives the buffer towards empty.

4) Retransmissions and Timeouts Flows experience packet losses when the buffer overflows. Eventually they are notified of the losses or time out and retransmit the lost packets. Part of the link utilization is contributed by the retransmitted packets. As the example in

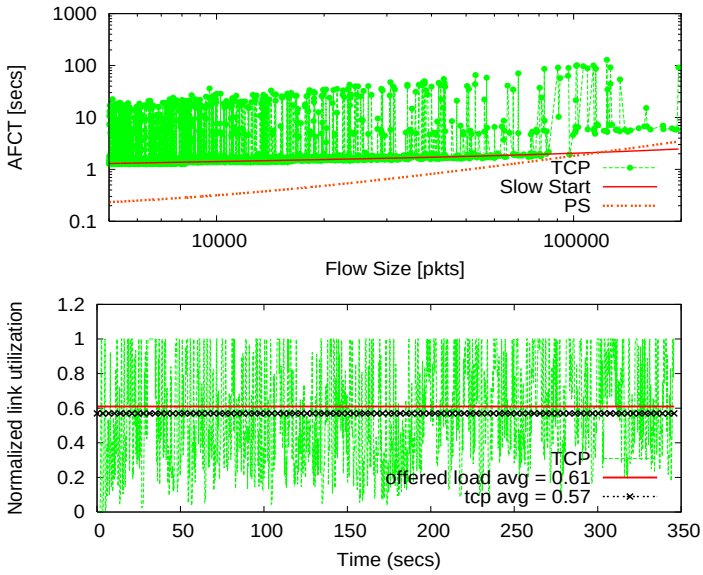


Figure 2.6: Experiment to illustrate that link utilization (bottom plot) could be high due to retransmissions, but that does not mean much for Flow Completion Times (top plot). Set up: $C = 2.4$ Gbps, RTT $= 0.1$s, Poisson flow arrivals, offered load $= 0.8$, Pareto distributed flow sizes, Mean flow length $= 30000$ pkts. Link utilization is measured over 100 ms intervals.

Fig. 2.6 shows, retransmitted packets contribute to 3.6% of the link utilization, and while the average link utilization is close to the offered load, the flow durations are two orders of magnitude higher than PS. Odlyzko gives an example of a real scenario [6] in which he notes that high utilization carries a penalty - during the hours of peak usage the average packet drop rate was around 5%, so service quality was poor, but the throughput figure was deceptively high since it included substantial retransmissions.

So far we have seen FCTs under TCP Sack, but there have been many proposals recently to improve TCP's AIMD behavior, such as HighSpeed TCP [25] and Scalable TCP [27]. Their main goal is to make TCP work well under low-multiplexed high bandwidth-delay networks. In particular, a single TCP flow should be able to achieve a large sustained equilibrium window size under realistic drop probabilities. For large window sizes, these schemes make the window increase more aggressive (as compared to additive increase of one packet per RTT) and the window decrease less drastic than halving on a packet drop. While these schemes work well in an environment of a few high bandwidth flows, they do not necessarily improve FCTs in a statistical mix of flows. For example, Fig. 2.7 shows the

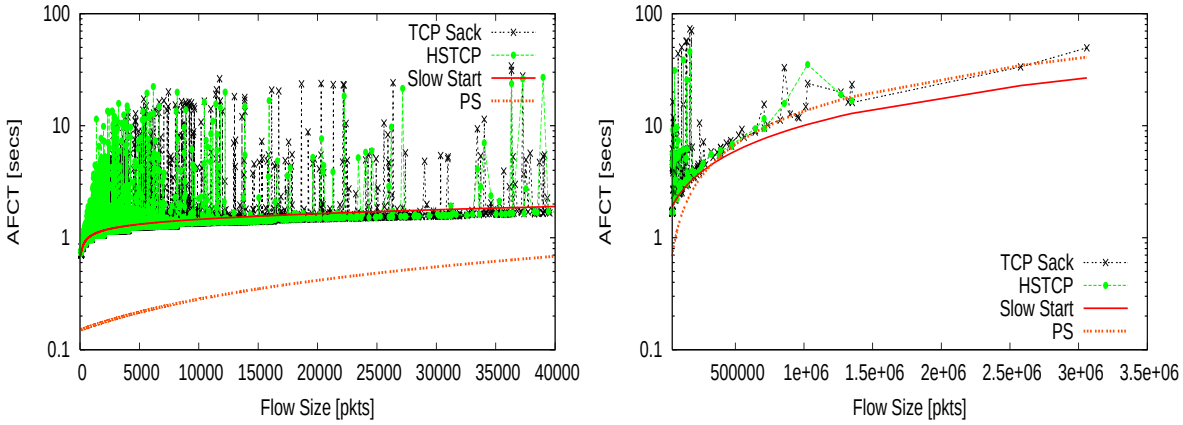


Figure 2.7: Experiment comparing flow completion times in HSTCP and TCP Sack, both with Drop Tail queues. HSTCP (default) parameters: Low window = 38 packets, High Window = 83000 packets, High $P = 10^{-7}$. Set up: $C = 1$ Gbps, RTT = 0.1s, Poisson flow arrivals, offered load = 0.4, Pareto distributed flow sizes, Mean flow length = 500 pkts. Buffer size = bandwidth times RTT.

FCTs for a heavy-tailed mix of flow sizes for High Speed TCP (HSTCP)[4] along with the regular TCP-Sack. Flows in HSTCP take about the same time to finish as regular TCP flows, and both achieve similar link utilization.

If RED or ECN is used, flow completion times are much longer as shown in Fig. 2.8. RED/ECN indicate the onset of congestion much sooner than Drop Tail, making flows exit slow-start early and therefore longer to finish. While this helps in keeping queues very small (bottom plot of Fig. 2.8), it comes at the cost of longer FCTs.

2.1 Why minimizing flow completion time is a hard problem

Flow completion time (or transaction time) is what users and distributed applications really care about, and so it is surprising that in-spite of the many papers on congestion control, there isn't a single paper which asks the fundamental question: How would we design congestion control to minimize flow completion times?

The problem is that it is a hard question to answer. In the case of a single link, we know that the scheduling discipline Shortest Remaining Processing Time (SRPT) [21] minimizes

[4]HSTCP uses TCP's standard increase and decrease parameters for packet drop rates up to 0.0015 or roughly for window sizes up to 38 packets. Beyond that it uses a more aggressive increase and smaller decrease to achieve a different response function, for example a congestion window size of 83000 packets for a packet drop rate 10^{-7}.

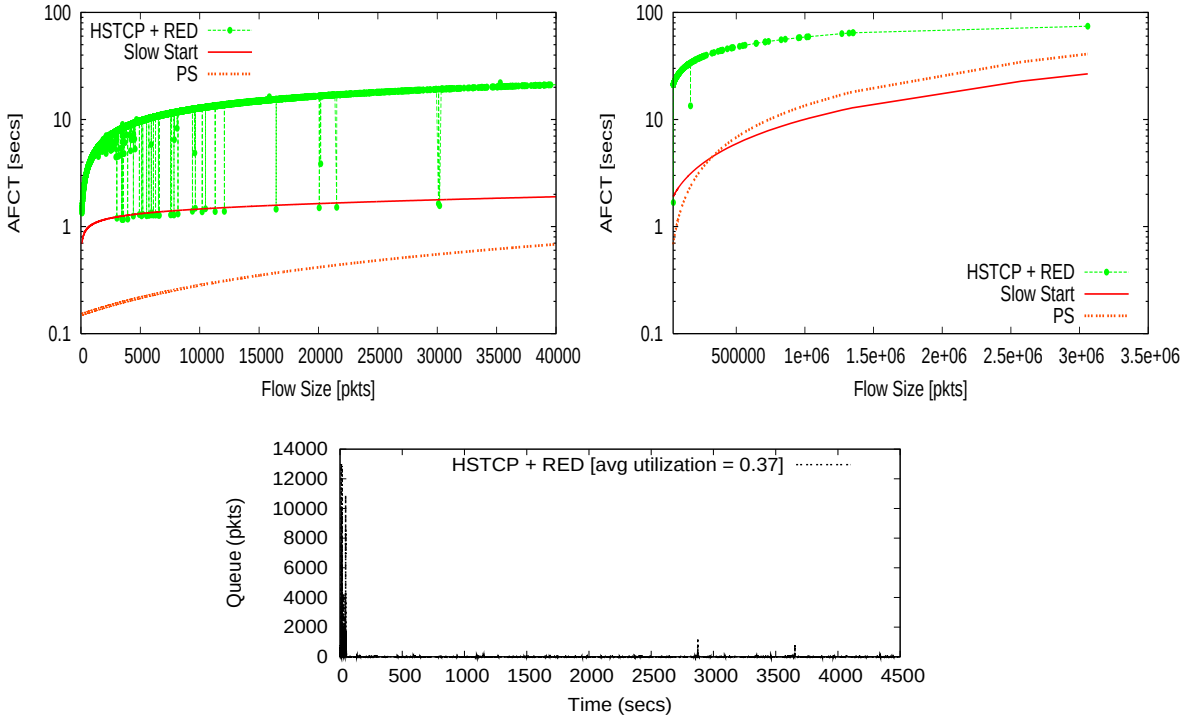


Figure 2.8: Flow completion times (top plots) and queue occupancy (bottom plot) in HSTCP with RED and ECN. HSTCP (default) parameters: Low window = 38 packets, High Window = 83000 packets, High $P = 10^{-7}$. RED is configured to (recommended settings [28]) gentle and adaptive with target delay as 0.005s. Set up: $C = 1$ Gbps, RTT = 0.1s, Poisson flow arrivals, offered load = 0.4, Pareto distributed flow sizes, Mean flow length = 500 pkts. Buffer size = bandwidth times RTT.

the mean Flow Completion Time [22]. The SRPT rule states that the server should at all times serve that flow of those available, which has the smallest remaining size. SRPT assumes that when a new flow (or a job in the terminology of queuing theory) arrives, the entire flow's packets are present at the bottleneck queue and the router knows the flow-size. It services jobs in the order of shortest processing time, preempting old flows when new shorter flows arrive.

This simple discipline is provably optimal is terms of minimizing the mean flow completion time, irrespective of the flow arrival and size distributions. However, using SRPT is not practical in the Internet for two main reasons: 1) It requires the flow-size information ahead of time, which often is not available to the end-host itself when the flow starts, 2) It assumes the a flow is available all at once ready to be served at the bottleneck when

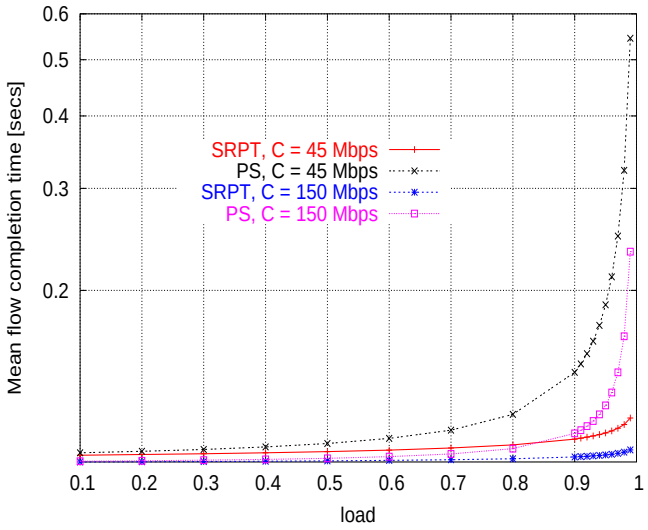


Figure 2.9: Plot comparing mean flow completion times under SRPT and PS for different offered loads. The flow sizes are Pareto distributed with a mean of 25 KB. RTT = 100 ms.

its turn comes; however in reality a flow can be queued at many different points along the network, including at the source host. Furthermore, in the case of a network scenario it isn't even clear what an optimal solution would be that minimizes flow completion times in the presence of flow-arrivals and departures, a variant of the job-shop scheduling problem known to be NP-hard [9] [10].

Our approach is to identify practical and close approximations to SRPT, and create a congestion control algorithm that comes close to minimizing FCT. For example, consider the Foreground-Background (FB) policy [29], proposed as an approximation to SRPT in the absence of flow size information. FB gives service to the flow with the least attained service at any time instant. If more than one flow has the least attained service, they get served equally. FB policy has a bias towards small flows, and attempts to complete the short jobs as quickly as possible. The upside of FB is it minimizes mean flow completion time (among disciplines which have no knowledge of remaining sizes), for a certain class of heavy-tailed flow-size distributions (such as the Pareto distribution) [80]. The downside is routers still need information on the so-far-completed flow length.

Another practical way to achieve short FCTs is to emulate an even easier discipline - Processor Sharing (PS) - which is not quite the minimum achievable but for most practical purposes comes reasonably close to it. PS shares bandwidth equally among flows queued

at a link. The expressions for mean FCTs under PS and SRPT[5] for Poisson flow arrivals are given below, and an example comparing them is shown in Fig. 2.9:

$$\begin{aligned} E(FCT)^{PS} &= RTT + \frac{E(p)}{1-\lambda E(p)} \\ E(FCT)^{SRPT} &= RTT + E(\text{Waiting time}) + E(\text{Residence time}) \\ &= RTT + \frac{\lambda}{2}\int_0^\infty \left\{ \frac{\int_0^p t^2 dF(t) + p^2[1-F(p)]}{[1-\rho(p)]^2} \right\} dF(p) + \int_0^\infty \frac{1-F(t)}{1-\rho(t)} dt. \end{aligned} \tag{2.1}$$

where RTT is the round-trip propagation delay between the source and destination, λ is the Poisson flow arrival rate, p is a flow's original service requirement, $F(\cdot)$ is the flow service requirement distribution, and $\rho(p) = \lambda \int_0^p t dF(t)$.

As seen in the figure, the PS and SRPT flow completion times are quite close to each other for low and medium loads, and diverge when the link is highly loaded such as 0.95 and above. Even though PS does not minimize FCTs and its mean completion time is at least twice that of SRPT as the offered load approaches 1 [79], yet trying to emulate PS is a good goal to start with because, a) FCTs in PS come close to SRPT for most practical purposes under low to medium network loads; in addition PS has the following advantages - b) Unlike SRPT or FB policies, PS does not need information on flow sizes - which is often unavailable to the routers, c) Flow durations in PS are invariant of the flow size distribution and finally, d) PS is inherently fair among flows.

The Rate Control Protocol is about emulating these practical disciplines under a broad range of network and traffic conditions, using very little information from network and end-hosts. In the next chapter, we will describe how RCP's mechanisms and algorithm can approximate Processor Sharing and Foreground-Background based policies.

[5]Mean FCT under SRPT is the sum of waiting time (the interval between the job's moment of arrival and the instant at which its first service occurs) and residence time (the remaining flow time including all the processing time plus any time spent standing aside because of pre-emptions from shorter jobs) [21].

Chapter 3

Rate Control Protocol

In this chapter we describe a simple and practical congestion control algorithm that emulates processor sharing for a broad range of flow size distributions and network conditions [14] [15]. The approach is different from TCP and XCP. Instead of incremental window changes in each round-trip time, the routers ask flows to transmit at a rate so as to emulate processor sharing. Furthermore, we aim to achieve this without per-flow state, per-flow queues, or per-packet calculations at the routers.

3.1 Rate Control Protocol (RCP): An Algorithm to Achieve Processor Sharing

3.1.1 The Basic Mechanism

RCP involves explicit feedback from routers along the path. In the basic RCP algorithm a router maintains a single rate, $R(t)$, for every link. The router "stamps" $R(t)$ on every passing packet (unless it already carries a slower value). The receiver sends the value back to the sender so that it knows the slowest (or bottleneck) rate along the path. In this way, the sender quickly finds out the rate it should be using (without the need for slow-start). The router updates $R(t)$ approximately once per roundtrip time (RTT), and strives to emulate processor sharing among flows. More formally, the rate feedback mechanism involves the following steps:

1. Every router maintains a single fair-share rate, $R(t)$, for all flows. Routers update $R(t)$ once per control interval which is approximately the average round-trip time of

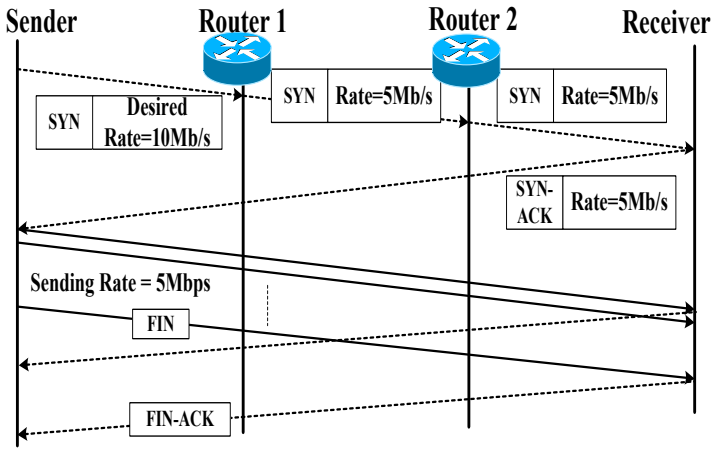


Figure 3.1: The sender sets the desired rate in the SYN packet, and a router can overwrite this rate if the current rate R that it can give out is lower then the desired value. The receiver then sends back the rate in the SYN-ACK packet. If the flow lasts longer than one RTT, the subsequent rates are piggy-backed on the data and ACK packets.

traffic passing through the queue.

2. Every packet header carries a rate field, R_p. When transmitted by the source, $R_p = \infty$. When a router receives a packet, if $R(t)$ at the router is smaller than R_p, then $R_p \leftarrow R(t)$; otherwise it is unchanged. The destination copies R_p into the acknowledgment packets, so as to notify the source. The packet header also carries an RTT field, RTT_p, where RTT_p is the source's current estimate of the RTT for the flow. When a router receives a packet it uses RTT_p to update its moving average of the RTT of flows passing through it, d_0.

3. The source transmits at rate R_p, which corresponds to the smallest offered rate along the path.

4. Each router periodically updates its local $R(t)$ value according to the algorithm described below.

An example of the RCP startup mechanism is illustrated in Fig. 3.1. We assume that - as with TCP - flows continue to have the connection set-up phase to establish state at both ends of the connection. This allows the initial rate to be calculated during the initial

handshake by piggy-backing on the SYN and SYN-ACK messages. This is very important for short-lived flows, which could last less than one RTT. Current feedback-based algorithms don't work well for short-lived flows, yet most flows in the Internet are of this type [30]. The SYN message sent by the source indicates the rate at which it wants to send the flow (which could be infinite). Each router maintains a single rate, $R(t)$, that it assigns to all flows. As the message passes through the network, if the current rate $R(t)$ at a router is lower than the value in the SYN packet, the router overwrites it. When the SYN packet reaches its destination, it has the lowest rate corresponding to the most congested link along the path. This value is sent back to the source in the SYN-ACK message to set the starting rate. When the flows last longer than an RTT then they are periodically and explicitly told a new rate by the network. This rate is piggy-backed on the data and the ACK messages.

3.1.2 Picking the Flow Rate

We're going to address the following question:
Is there a rate that a router can give out to all flows, so as to emulate processor sharing?
If the router has perfect information on the number of ongoing flows at time t, *and* there is no feedback delay between the congested link and the source, then the rate assignment algorithm would simply be:

$$R(t) = \frac{C}{N(t)} \tag{3.1}$$

where $R(t)$ is the rate given out to the flows by the router at time t, and C is the link capacity. But the router does not know $N(t)$ and it is complicated to keep track of. And even if it could, there is a feedback delay and so by the time $R(t)$ reached the source, $N(t)$ would have changed. And further, different flows have different amount of traffic to send and they are all bottlenecked at different links - so the exact value of $N(t)$ at a link may not even mean much. So, we propose that the routers have an adaptive algorithm that updates the rate assigned to the flows, to approximate processor sharing in the presence of feedback delay, without any knowledge of the number of ongoing flows. RCP is a particular heuristic designed to approximate PS. It has three main characteristics that makes it simple and practical:

1. The flow rate is picked by the routers based on very little information (the current queue occupancy and the aggregate input traffic rate).

2. Each router assigns a single rate for all flows passing through it.

3. The router requires no per-flow state or per-packet calculations.

Intuitively, to emulate processor sharing the router should offer the same rate to every flow, and try to fill the outgoing link with traffic. To keep delays small it should keep the queue occupancy close to zero. The following rate update equation is based on this intuition:

$$R(t) = R(t-d) + \frac{\left(\alpha(C - y(t)) - \beta\frac{q(t)}{d}\right)}{\hat{N}(t)} \tag{3.2}$$

where d is a moving average of the round-trip time (RTT) measured across all traffic passing through the RCP queue, $R(t-d)$ is the last updated rate, C is the link rate, $y(t)$ is the measured input traffic rate during the last update interval (d in this case), $q(t)$ is the instantaneous queue size, $\hat{N}(t)$ is the router's estimate of the number of ongoing flows (i.e., number of flows actively sending traffic) at time t and α, β are parameters chosen for stability and performance.

The basic idea is: If there is spare capacity available (i.e., $C - y(t) > 0$), then share it equally among all flows. On the other hand, if $(C - y(t)) < 0$, then the link is oversubscribed and the flow rate is decreased evenly. Finally, we should decrease the flow rate when the queue builds up. The bandwidth needed to drain the queue within an RTT is $\frac{q(t)}{d}$. The expression $\alpha(C - y(t)) - \beta\frac{q(t)}{d}$ is the desired aggregate change in traffic in the next control interval, and dividing this expression by $\hat{N}(t)$ gives the change in traffic rate needed per flow.

RCP doesn't exactly use the equation above for three reasons:

- First, the router can't directly measure the number of ongoing flows, $N(t)$, and so estimates it as $\hat{N}(t) = \frac{C}{R(t-d)}$.

- Second, we would like to make the update rate interval (i.e., how often $R(t)$ is updated) a user-defined parameter, T. For example, if the queue is filling up, it is unnecessary to wait for an entire round-trip time to reduce the rate in order to drain the queue. The update interval is actually $\min(T, d)$ since we want it to be at least equal to RTT. However, to maintain system stability we scale the traffic change we bring about in each update interval - the desired aggregate change in traffic over one average RTT is $\alpha(C - y(t)) - \beta\frac{q(t)}{d}$, we scale this aggregate change by T/d. And, $\hat{N}(t) = C/R(t-T)$.

- Third, we would like the flexibility of operating the link at a peak utilization of less then 100%, to give some headroom for sudden traffic surges to drain away before a queue starts to build. For this, we introduced a parameter, $\eta(0 < \eta \leq 1)$; for example choosing $\eta = 0.95$, gives us a peak utilization of 95%.

Then the equation becomes:

$$R(t) = R(t - T)\left(1 + \frac{\frac{T}{d}\left(\alpha(\eta \cdot C - y(t)) - \beta\frac{q(t)}{d}\right)}{\eta \cdot C}\right) \tag{3.3}$$

As with any congestion control algorithm, there are many questions to address: How do flow completion times compare with processor sharing, and with TCP, XCP? Is RCP stable when there are sudden changes in the network? How do we choose its parameters for good performance and stability? How is RCP implemented in routers and end-hosts? How much buffering does it require in routers? How can it be incrementally deployed in real networks? And so on. The remaining chapters address these questions in detail, for the rest of this chapter we will focus on gaining a better understanding of the RCP algorithm.

3.2 Understanding the RCP Algorithm

3.2.1 How good is the estimate $\hat{N} = C/R$?

When the router updates the rate, it knows precisely the spare capacity and the queue size it needs to drain. So the accuracy of the algorithm depends on how well C/R estimates $N(t)$.

In the simplest scenario with only long-lived flows, C/R converges to the correct number of flows, N. An example is shown in Fig. 3.2 where 20 flows start at time $t = 0$ and 20 more flows start at time 40, and 20 flows complete at time 100. In each case, C/R converges to $N(t)$. RCP provably converges to its correct fair-share rate, as we will see in Chap. 5, and the convergence is fast. The values of α and β only affect the rate of convergence; we'll examine the stability region for α and β shortly.

But, what if some of the flows are bottlenecked elsewhere and cannot send in traffic at rate R. For example if N_1 flows are bottlenecked at a certain node and N_2 flows are bottlenecked elsewhere and are arriving at rate, R_2, less than their fair share i.e. $R_2 < C/(N_1 + N_2)$. In this case, the rate R in RCP will be such that $R \cdot N + R_2 \cdot N_2 = C$ i.e.

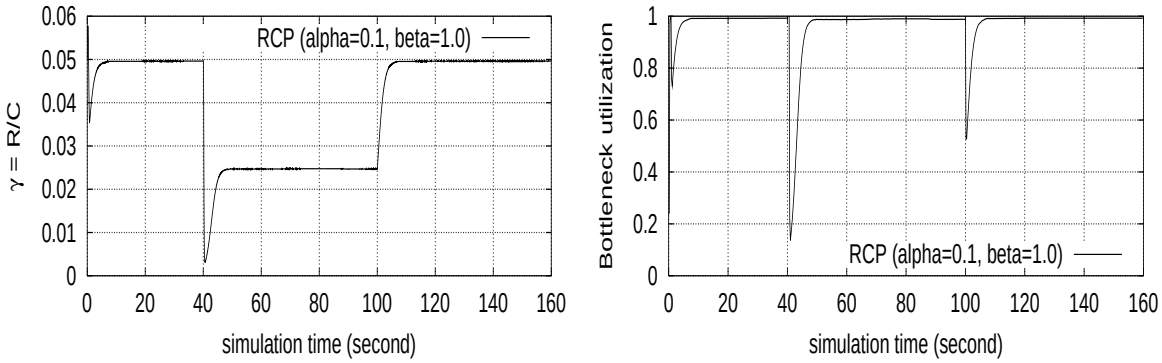
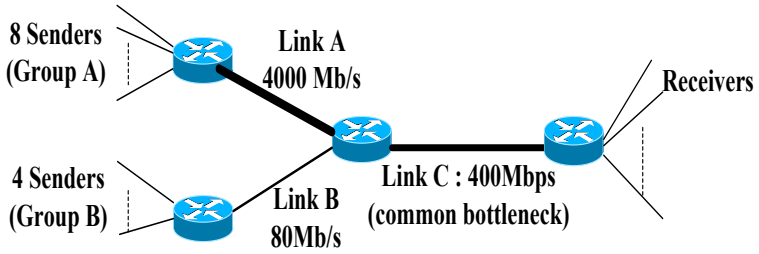


Figure 3.2: The time evolution of RCP rate factor $\gamma(t) = R(t)/C$ and measured bottleneck utilization under long-lived flows. At $t = 0$, 20 flows start; at $t = 40$s, 20 more flows start; at $t = 100$s, 20 flows finish. In each case, $C/R(t)$ converges to $N(t)$.



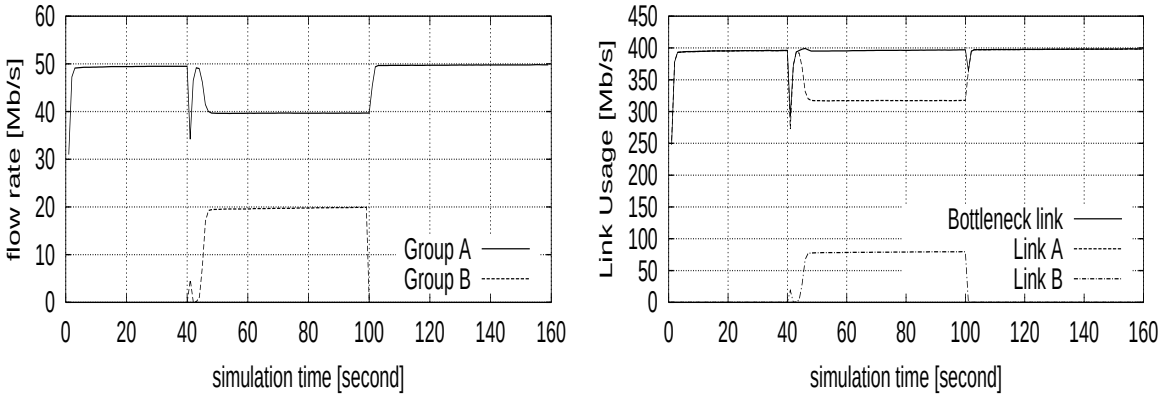


Figure 3.3: Long flows achieve max-min fair under RCP: 8 flows (Group A) start at $t = 0$, and 4 flows (Group B) join at $t = 40$ and leave at $t = 100$. Group A is bottlenecked at Link C and Group B is bottlenecked at Link B. The flows achieve their max-min fair rates.

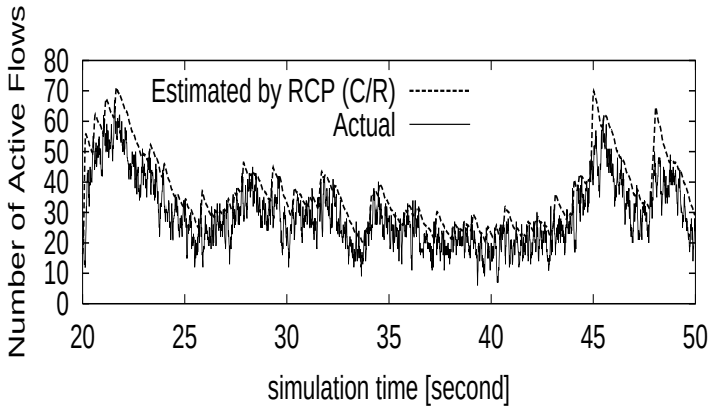


Figure 3.4: Comparison of the number of measured active flows and the estimate (C/R). Bottleneck capacity, C = 10Mb/s, RTT = 50 ms, flow arrival rate = 400 flows/sec, and flow sizes are Pareto with mean = 25 pkts (1000 byte/pkt) and shape parameter is 1.2.

$R = \frac{C-R_2N_2}{N_1}$. In other words, RCP achieves max-min fairness. An example is shown in Fig. 3.3. From time, t = 40 to 100, N_1 = 8 flows (Group A) and N_2 = 4 flows (Group B) share the bottleneck of 400 Mbps (Link C). Group B flows are bottlenecked at Link B of 80 Mbps. As seen in the bottom figure, the flows achieve their max-min fair rates. In this case, C/R is an estimate of $N_1 + \theta N_2$ where $\theta = R_2/R$.

In the past, algorithms similar to RCP have been proved to achieve max-min fairness under general topologies [33]. The same proof shows that RCP also achieves max-min fairness in a general network.

When flows are not long-lived, C/R can still be a good estimate of the number of active flows. In particular, when the mean flow size, $E[L]$, is close to or greater than the bandwidth-delay product (in other words the flow sizes are large compared to the pipe size), then C/R is a good estimate. It is a smoothing estimate since flows arrive and depart quickly and $N(t)$ changes rapidly. An example of this case is shown in Fig. 3.4.

When $E[L] \ll bandwidth \times RTT$, most flows fit in the bandwidth-delay "pipe" and most do not have sufficient data to send for an entire round-trip time. In this case $C/R(t)$ represents an "effective" number of lows, $N_e(t) < N(t)$, where each flow has at least a round-trip time worth of data to send. Even though $C/R(t)$ underestimates the number of flows (and hence increases the rate for each flow), it is actually the right thing to do because when most flows have less than an RTT of data to send, giving exactly $C/N(t)$ to each flow means the pipe will never be filled. $C/R(t)$ correctly so has no connotation to the exact

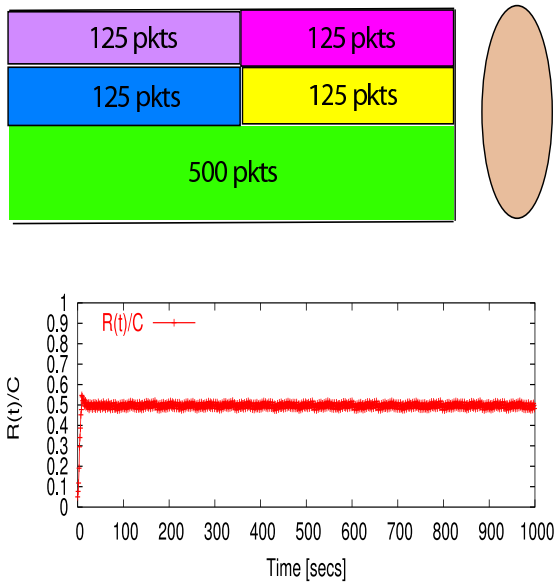


Figure 3.5: Plot illustrating the $R(t)/C$ value in the presence of short flows. Bandwidth-delay product = 1000 packets, four short flows (125 pkts each) and one long flow (500 pkts) arrive at the start of every RTT. The RCP rate (bottom plot) converges to the correct fair-share rate of long flow.

number of ongoing flows. $R(t)$ represents the max-min share of the flows. As an example, Fig. 3.5 illustrates a link with a bandwidth-delay product of 1000 packets and 5 flows: 4 short flows having 125 packets each, arrive every RTT (they are capable of finishing within a round-trip time) while the fifth flow is a high bandwidth long-lived flow that has a lot of data to send. Max-min fairness gives the non-bottlenecked short flows what they need and gives the remaining link bandwidth to the long flow; i.e. $R(t)/C = 0.5$. RCP converges to this value, as shown in the figure. The fair share rate in this case is:

$$R = \frac{C \cdot RTT - \sum_{f \in A} S(f)}{n_B \cdot RTT}$$

where $C \times RTT$ is the pipe-size, A denotes the set of flows capable of finishing within a RTT under the current workload (i.e. the non-bottlenecked flows), $S(f)$ denotes the flow size of f, and n_B is the number of bottlenecked flows. Given any general traffic pattern, the algorithm to compute the above rate is quite similar to the one used for max-min fair rate calculation [48].

3.3 What is the role of the term, $\beta \cdot \frac{q(t)}{d}$?

In the last section we have seen the role played by $C/R(t)$ in RCP. While the role of other terms such as $C - y(t)$, is simple to understand ($C - y(t)$ adapts RCP rate to link under or over subscription), the part played by $\beta \cdot \frac{q(t)}{d}$ in achieving short completion times is less clear. It is worth asking questions such as: Isn't it better to do away with the $\beta \cdot \frac{q(t)}{d}$ term and always have a full queue because that way, the link is kept busy; this should increase the link utilization and therefore the end-to-end throughput? If RCP reduces its rate aggressively in response to a growing queue, aren't we in effect reducing the flow transmission rate and thereby increasing completion times? On the other hand won't not responding to a growing queue increase queuing delay and therefore the completion times? This section answers these questions.

We need to understand how the flow completion time varies with R. Let's define the flow duration, τ, to be the time from when the first packet of a flow enters the network to when the last packet of the flow leaves the network, i.e.,

$$\tau = \frac{L}{R} + d_l(R) \tag{3.4}$$

where L is the flow size, R is the flow rate (assuming for now that it remains constant for a flow's lifetime), and $d_l(R)$ is the propagation plus queuing delay of the flow's last packet (as a function of R). We want to minimize flow-duration and for that we'll consider a hypothetical but interesting question:

Is there a rate R that the router can ask the flows to transmit at, so as to minimize $E[\tau]$?

Assume that a router picks the same rate for all flows passing through it. So what rate should it choose? Is there a rate that will minimize flow duration? It turns out that there isn't in general; but, when flow lengths are heavy-tailed (which they are in the Internet), there is an optimal rate (it's just hard to compute). Let's see why there is an optimal rate.

Notice that $\frac{L}{R}$ (the flow transmission time) *decreases* as R increases, whereas $d_l(R)$ (the fixed propagation delay plus the variable queuing delay) increases with R. It all depends on whether $d_l(R)$ increases faster, or slower, than $\frac{L}{R}$ decreases with R. Clearly, $d_l(R)$ depends on the flow size distribution. The more deterministic the flow size, the lower the queuing delay, and so the slower $d_l(R)$ will grow with R. For example, if all flows are the same size, and that new flows arrive as a Poisson process to a router (which can be modeled

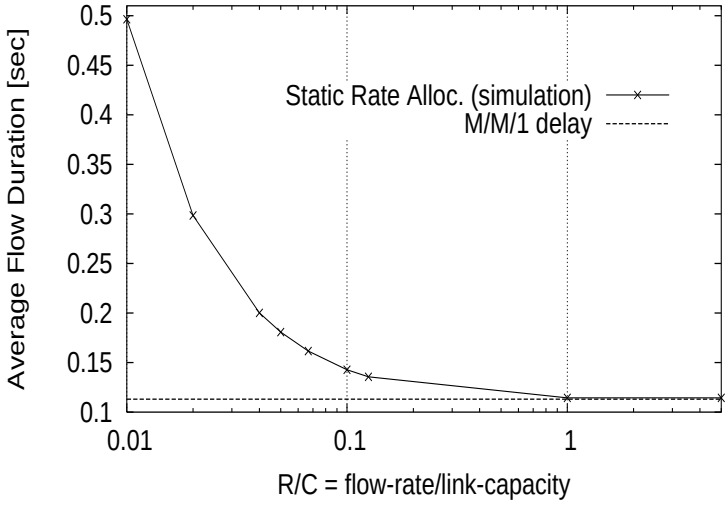


Figure 3.6: Average flow duration vs. normalized flow-rate for exponentially distributed flow sizes. Flow arrivals are Poisson with rate 175 flows/s; server capacity $C = 150$ Mb/s; mean flow size = 50 pkts (1000 byte/pkt), RTT = 0.2s. As $R \to 0$, $E[\tau] \to \infty$ and when $R \geq C$, $E[\tau] \to E[D]_{M/M/1-FCFS}$ (shown by dotted line).

as a single server FCFS queue with capacity C), then as $R \to \infty$ the flows become single entities and the router becomes an M/D/1 queue. Since all the flows are the same size, FCFS is equivalent to the optimal SRPT discipline. In other words, if flows are fixed size, the routers should not limit their rate; it should set $R = \infty$. Intuitively, it is best to push as much work into the network, so it makes forward progress towards its destination, even if it means the routers need huge buffers.

It turns out that $R = \infty$ is also optimal when flow sizes are exponentially distributed. A plot of the average flow duration vs. flow rate is shown in Fig. 3.6. As we'd expect, the flow transmission time, $\frac{L}{R}$, (and hence the flow duration) tends to infinity as $R \to \infty$. As R increases, the average flow duration decreases, and converges to the average delay of an M/M/1-FCFS system (as shown by the dotted line in the Fig. 3.6).

In both examples, the increase in queuing delay is dominated by the decrease in transmission time, and the expected flow duration decreases monotonically as R increases, converging to the delay of an M/D/1 or M/M/1 system, respectively. These results suggest that to minimize flow duration we should transmit flows at as high a rate as possible. If that was always true, or at least was true for flow sizes in the Internet, there would be no

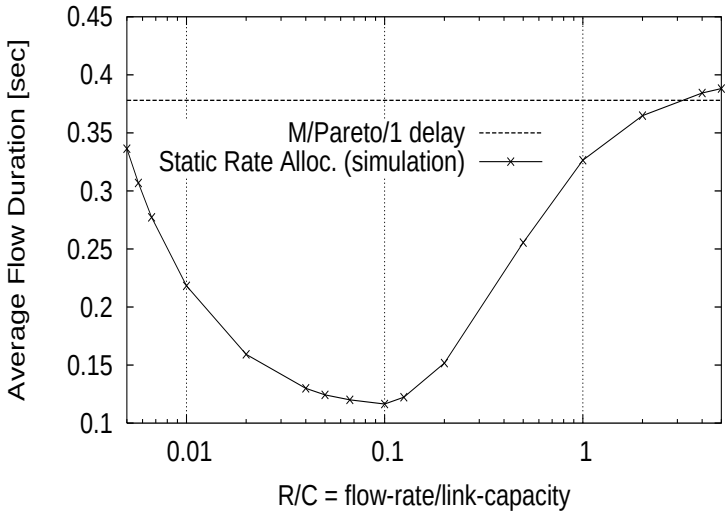


Figure 3.7: Average flow duration vs. normalized flow-rate for Pareto distributed flow-sizes with mean 25 pkts (1000 byte/pkt) and shape 1.2. Flow arrivals are Poisson with rate 600 flows/s, Server capacity $C = 150$ Mb/s, RTT $= 0.2$ s. As $R \to 0, E[\tau] \to \infty$ and when $R >> C, E[\tau] \to E[D]_{M/Pareto/1-FCFS}$.

need to set an explicit starting rate – the sources should just send as fast as they can.

Flow-sizes in the Internet are neither fixed, nor exponentially distributed: they have a heavy-tailed distribution, and are modeled well by the Pareto distribution with complementary distribution function $F^c(x) = (\frac{k}{x})^\alpha, 0 < \alpha \leq 2, k > 0, x \geq k$ [30]. Simulations of a single-server FCFS queue with Pareto distributed flow sizes, shown in Fig. 3.7, indicate that in fact there is a rate that minimizes the expected flow duration.

The intuitive reason why there is an optimal rate is that an FCFS queue fed by an input process with heavy tailed flow sizes also has heavy-tailed queue occupancy; i.e. $lim_{b\to\infty} P\{Q > b\}$ is heavy-tailed [47]. It is no longer the case that as R increases, the decrease in the flow transmission time dominates the increase in the queuing delay. At some point the increase in the queuing delay starts to dominate, and as $R \to \infty$, the expected flow duration approaches the M/G/1 delay [48]:

$$E[\tau]_{M/G/1} = \frac{\lambda E[L^2]}{2C^2(1-\rho)} + \frac{E[L]}{C} \tag{3.5}$$

where λ is the flow arrival rate, $E[L]$, $E[L^2]$ are the first and second moments of the flow size distribution and $\rho = \frac{\lambda E[L]}{C}$ is the offered load. In the case of heavy-tailed distributions,

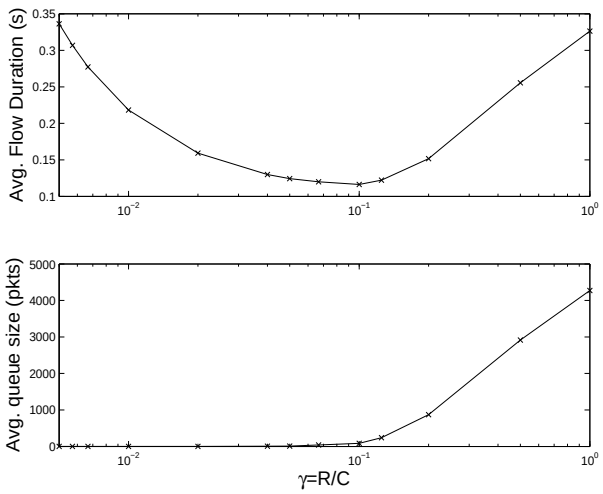


Figure 3.8: Average flow duration and average queue size vs. flow rate for Pareto distributed flows. $C = 150$ Mbps, $RTT = 100$ ms. The data is from the same experiment as in Fig. 3.7

$E[L^2]$ is much larger than $(E[L])^2$. For example, when the flow sizes are Pareto with the shape parameter $1 < \alpha \leq 2$, the mean flow size is finite while the second moment is infinite, and the expected flow duration is infinite.[1]

The key point here is as the router gives out higher rates, the queue size increases and the flow durations increase. Our observation in several simulations is that at the rate where delay is minimized, the average queue occupancy is small (as shown in the Fig. 3.8), and the link utilization is equal to the offered load, ρ. At rates below the optimal, although the queue occupancy is still small, the link utilization is less than the offered load (meaning that the work given to the system is unnecessarily stretched), while at rates above the optimal, although the link utilization is high the mean queue occupancy is high as well. We modeled the 'U' curve analytically; the details are in Appendix A.

In summary our findings on the $q(t)$ term are: Under flow arrivals and departures, the queue term is important to emulate completion times under processor sharing, especially in the presence of heavy-tailed distributed flow-size. If the queue is allowed to grow then it mostly contains packets from long flows, short flows get stuck behind long flows, resulting in increased queuing delay and completion times. Put another way, if the queue is allowed

[1] Of course in simulations which run for a finite amount of time, the second moment of file size is not infinite. The value of $\overline{L^2}$ increases with the length of simulation. Substituting $\overline{L^2}$ for $E[L^2]$ in Eqn. 3.5 gives the asymptotic value for large R, shown in Fig. 3.7.

to grow then it roughly emulates the FCFS discipline, which has large completion times when flow sizes have a large variance. An exception to the above argument, as we will see in Chap. 5, is when α is chosen large enough (such as 0.9) in which case the large negativity of '$\alpha \cdot (C - y(t))$' compensates for the absence of queue term. The queue term is not crucial for non-heavy tailed distributions such as exponential or uniform, for which the FCFS discipline is not too bad as compared to processor sharing. For example in the case of exponentially distributed flows, mean flow duration under FCFS equals that under PS.

3.4 Is RCP stable?

An important question to ask is whether RCP's short flow completion times come at the cost of network instability when there are sudden changes in network traffic or there are bad traffic patterns. After all there is a good reason why the congestion control mechanisms of TCP have been so deliberately made conservative. A desirable characteristic of any congestion control scheme is that it is stable, in the sense that – if there exists an equilibrium point it always converges to the equilibrium operating behavior, and even when perturbed it should return to this stable state. We want to know whether RCP exhibits such a desirable stable behavior when there are sudden changes such as flash crowd scenarios or link failures. Chap. 5 studies this topic in great detail, and we only present the high order bit here.

Stability of RCP depends on its parameters α and β. We thought about system stability under the following two very different regimes:

1) *Deterministic scenario of long-lived flows*: In this scenario, N long-lived high bandwidth flows start up at the same time and we want to know if the system reaches equilibrium (equilibrium rate is $R_e = C/N$ and queue is zero) and remains there. The good news is that α and β can not only be chosen such that the system is stable independent of the network link rates, round-trip times, and the number of flows, but the stable region is also very broad. In Chap. 5, we use tools in control theory to characterize this region precisely and detail on how long RCP takes to converge to equilibrium.

2) *Stochastic scenario with random flow arrival times and flow sizes*: In this case, convergence of $R(t)$ in the same sense as for long-lived flows is less meaningful because the input conditions are changing. Further, as discussed before we do not always want $R(t)$ to be equal to $C/N(t)$ exactly: If $N(t)$ is very large but each of the flows has very little traffic to send (less than a RTT) then we actually want to underestimate $N(t)$ and thereby give

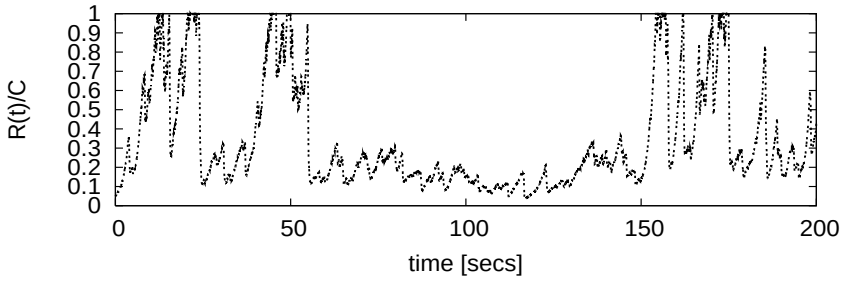


Figure 3.9: The figure shows the normalized rate, $R(t)/C$ versus time for Poisson flow arrivals with pareto distributed flow sizes. Bottleneck capacity, $C = 150$ Mb/s, RTT = 100 ms, offered load = 0.7, mean flow size = 30 pkts (1000 byte/pkt) and shape parameter is 1.2. Initial rate, $R(0) = 0.05C$. $R(t)$ sweeps over it's entire range depending on the input conditions.

a higher rate to each flow, since if we give $C/N(t)$ exactly to each flow we will never fill up the link.

What would be more meaningful is convergence in the stochastic sense like $E[N(t)]$ (mean number of flows) and $E[D(l)]$ (mean flow completion time for flow of length l) converge to finite equilibrium values. Proving such a result rigorously is a notoriously hard problem specially for non-linear delayed feedback systems such as RCP. The same is true for TCP, XCP and other algorithms. A large number of simulations indicate that under a variety of dynamic situations (like different flow arrival distributions, different flow size distributions, offered load, link capacities, and round-trip times) RCP's performance in terms of $E[N(t)]$ and $E[D(l)]$ converges to that under ideal processor sharing for a wide range of (α, β). These simulations are shown in Chap. 4.

The convergence of RCP's performance measures $E[D(l)]$ (and $E[N(t)]$) to that of processor sharing is independent of the initial value of $R(t)$ chosen. Our simulations support this. For any particular simulation we observe that $R(t)$ sweeps over the entire space (ranging from minimum-rate to a maximum of link-rate), depending on the conditions on the link. Any point could have been the starting point of the experiment. An example to illustrate this is shown in Fig. 3.9. Notice that $R(t)$ takes a wide range of values depending on the input conditions. Starting with different initial values of $R(t)$ will give different sample paths of the stochastic processes $N(t)$ and $D(l)$, but the key point is the underlying statistical properties $E[N(t)]$ and $E[D(l)]$ converge to that in processor sharing.

Given that the algorithm is stable for a wide range of (α, β), we picked those values for

the RCP system to maximize performance for a wide range of traffic and network conditions.

3.5 Estimating the average round-trip time

RCP uses an estimate of average round-trip time to determine how quickly a standing queue should be drained ($q(t)/d$), and how often it should update $R(t)$. Just as with any control system updating the rate less (or more) often than necessary can result in a sluggish (or unstable) system.

Every packet passing through the router carries the source's estimate of its RTT. The router uses this to update its moving average, d, as follows:

$$d = \theta \cdot RTT_p + (1 - \theta) \cdot d$$

where RTT_p is value carried in the packet, and θ is the moving average gain. Note that this gives an estimate of average RTT across all *packets* passing through the RCP queue, as opposed to an average over number of flows. Even though this skews the RTT estimate towards flows with larger number of packets, that's what is desired because it's the flows with a large number of packets that last multiple RTTs and determine the control loop stability. Stability depends less on short flows which finish within one or just a few RTTs.

In reality, the router's RTT averaging above is achieved in two different steps as shown below:

1. The data-path of an RCP router simply averages the RTT over all packets seen in the control interval, T:

$$d_T = \frac{\sum_i rtt_i}{n_T} \tag{3.6}$$

   where d_T is the RTT average over time-interval T, rtt_i is the value carried in packet $i's$ header, and n_T is the number of packets carrying a valid RTT in interval T .

2. The control path (which also performs the RCP rate computation periodically) takes d_T as input to keep track of a smoothed estimate, d. It determines what the moving average gain should be. In particular, if d_T is much smaller than the smoothed version (d), it is better to age d and bring it down gracefully as opposed to dropping it to a sudden low value. This is achieved by deciding the moving-average gain as follows:

if $(d_T > d)$

$\theta = \frac{T}{d}$

else

$\theta = \frac{R}{C} \cdot \frac{T}{d} \cdot \frac{d_T}{d}$

The smoothed RTT is then updated as:

$$d = \theta \cdot d_T + (1 - \theta) \cdot d$$

We will see in Chap. 5 why such an RTT averaging results in a system that remains stable even in networks where flows with vastly heterogeneous round-trip times coexist.

3.6 Achieving differential bandwidth sharing

In this section we will explore how RCP is useful beyond one rate for all flows to achieve differential bandwidth sharing among flows. Differential bandwidth sharing is useful in a variety of contexts: for example, service providers frequently like to share a link in different proportions among different flows depending on which customer they belong to and what applications they are carrying. Another use is to emulate practical flow-size based disciplines like those described in Chap. 2, Feedback-Based (FB) or Least Attained Service (LAS), or SIFT described in [50]. In the absence of flow size information, these disciplines have been proposed as an approximation to the optimal Shortest Remaining Processing Time. They do not require knowledge of the entire flow-size but just the flow's age (or how much it has been served so far). The RCP algorithm can be used to emulate these disciplines.

We will illustrate with a simple example where the router would like to allocate two different rates: one rate for the *short* flows and another rate for the *long* flows, in ratio of $1 : \theta$. A single rate is computed just as before in Eqn. 3.3:

$$R(t) = R(t - T)\left(1 + \frac{\frac{T}{d}(\alpha(\eta \cdot C - y(t)) - \beta\frac{q(t)}{d})}{\eta \cdot C}\right)$$

In addition to RTT value, packets also carry size of the flow transmitted so far; we will call this as the *age* of a flow. This avoids the router to have to maintain any per flow state as to how *old* each flow is. If the age of the flow is $\leq$ some threshold, Th, (flow is short so

far) the router stamps in the packets with the rate $R(t)$, otherwise the router classifies it as a long flow and stamps with rate $\theta \cdot R(t)$ where $0 < \theta < 1$. Note above that now C/R is no longer an estimate of the number of ongoing flows, N, but is (correctly so) an estimate of $N_s + \theta \cdot N_l$, where N_s are the number of short flows and N_l, are the number of long flows.

$R(t)$ is allowed to grow up to C/θ, and the rate assigned to a short flows equals $\min(R(t), C)$ while that to a long flow equals $\min(\theta \cdot R(t), C)$. It is important that $R(t)$ be allowed to grow up to C/θ, so that if there are only long flows in the system they will be able to claim the link rate. As an example, if there is only one long-lived flow, $R(t)$ will eventually grow up to C/θ, and the long flow is given a rate equal to $\theta \cdot R(t) = C$.

The above example can be generalized to achieve differential sharing for any number of levels.

3.7 Comparison with XCP's mechanisms

Both XCP and RCP try to emulate processor sharing among flows, which is why their control equations are similar. However, the manner in which they converge to PS is very different; the main difference between XCP and RCP is in the kind of feedback that flows receive. XCP gives a window increment or decrement over the current window size of the flow (which is small for all newly starting flows). At any time each XCP flow can have a different window size, different RTT and a different rate. XCP continuously tries to converge to the point where all flows have the fair-share rate, by slowly reducing the window sizes of the flows with rates greater than fair-share and increasing windows of the flows with rates less than fair-share (while avoiding over-subscription). New flows start with a small window, and the convergence could take several RTTs especially if there is little or no spare capacity. If the flows arrive as a Poisson process with heavy-tailed flow sizes, then most of the flows finish by the time they reach their fair share. The upside is XCP achieves a stable network and avoids congestion related packet losses. In RCP, all flows (new and old) receive the same rate feedback which is close to their equilibrium rate. This helps flows finish quickly. We will see in Chap. 4 that how these different mechanisms of RCP and XCP contribute to a large performance difference.

XCP is computationally more complex than RCP since it gives different feedback values to each flow, and involves multiplications and additions for every packet. RCP maintains a

single rate for all flows and involves very few simple per-packet computations.[2]

[2]The router uses the RTT information in the packets to update its RTT estimate - our stability analysis and simulations indicate that it is sufficient for the router to have a "rough" estimate of the feedback delay, and so it can even just sample a few packets and update its estimate of RTT.

Chapter 4

Flow completion times under RCP

Our goal in this chapter is to study RCP's flow completion times for a broad range of network and traffic conditions [14] [15]. We are particularly interested in how the *flow completion times* compare with ideal processor sharing, with existing (TCP), and with newly proposed mechanisms (XCP); and the impact they would have on users' experience.

4.1 Simulation Setup

To get an idea of how RCP behaves, we first measure RCP's flow completion times using simulations.[1] Later (in Chap. 6), we will repeat some of these results in experiments using real implementations.

We compare the performance of RCP with Processor Sharing (PS), TCP and XCP. We are primarily interested in the average flow completion time (AFCT).[2] Flow completion time (FCT) is defined as the time from when the sender sends a SYN packet until the receiver receives the last data packet of the flow, i.e. FCT = 1 RTT for the connection set-up plus the duration of the data transfer. We will use RTPD to abbreviate round-trip propagation delay (i.e. the round-trip time minus the queuing delay). AFCT is the average of FCT over all flows for the simulation run. Note that AFCT $\geq$ 1.5 RTPD + $\frac{E[L]}{C}$. This is because (ignoring queuing delay) the minimum FCT for any flow of size L is: 1 RTPD for SYN/SYN-ACK and (0.5 RTPD + L/C) for the data transfer. The analytical expression

[1] using ns-2 [24] (Version 2.29) augmented with RCP end-host and router modules.

[2] We will use the term 'flow' here to represent the packets corresponding to a particular TCP connection.

for FCT of a flow of size L under processor sharing is [29]:

$$FCT_{PS} = 1.5\ RTPD + \frac{L}{C(1-\rho)} \tag{4.1}$$

where ρ is the offered load and C is the link capacity. We will use Eqn. 4.1 to compute the PS values for our simulation setups. As secondary measures, we are also interested in the link utilization, and the average number of ongoing or active flows – which in PS can be simply computed by Little's Law: $E[N] = \lambda \times FCT_{PS}$ where λ is the flow arrival rate.

We will assume for now the usual rule-of-thumb that the size of a router's queue equals the bandwidth-delay product, i.e., link capacity multiplied by maximum RTPD of flows passing through it [44]. Today router buffers are sized according to the rule-of-thumb, however recent research has challenged it and has shown buffers of several orders of magnitudes smaller suffice for TCP to work well in high bandwidth networks with large statistical multiplexing [45][46]. We assume here that packets are dropped from the tail of the queue. Our simulations are run until the performance measures converge.

Eqn. 3.3 is the rate update equation used in the RCP router. The RCP parameters are: Control period, $T = \min(10\text{ ms}, RTT)$ and $\alpha = 0.1, \beta = 1.0$. For TCP, we have tested with TCP Reno, TCP New Reno, and TCP Sack modules in *ns-2*, with an initial window size of two packets. These flavors of TCP have the same underlying congestion control algorithms for window increase and decrease. They only differ in their behavior, in the use of retransmission timeouts and recovery mechanisms, when there are multiple packet drops within a window [57]. The *ns-2* implementation of XCP (Version 1.1) is publicly available [12], and the parameters are set as in reference [11].

All data packets are 1000 bytes and the control packets (SYN, SYN-ACK, FIN) are 40 bytes. It has been observed that the session arrivals can be accurately modeled as Poisson and that the flows within a session can be bursty in nature [32]. However, it is reasonable to suppose that flows arrive as a Poisson process, which would be the case when they correspond to a large number of independent sessions [31]. Unless otherwise mentioned, we will assume that flows arrive as a Poisson process with rate λ and flow sizes are Pareto distributed [30, 32]. The offered load on a link is $\rho = \lambda E[L]/C$. In our simulations we vary each of the following parameters – $\rho, E[S], C, RTPD$, flow size distribution, arrival process distribution – while keeping the rest constant, and observe how RCP, TCP and XCP compare with PS when a network or traffic parameter is varied from one extreme to

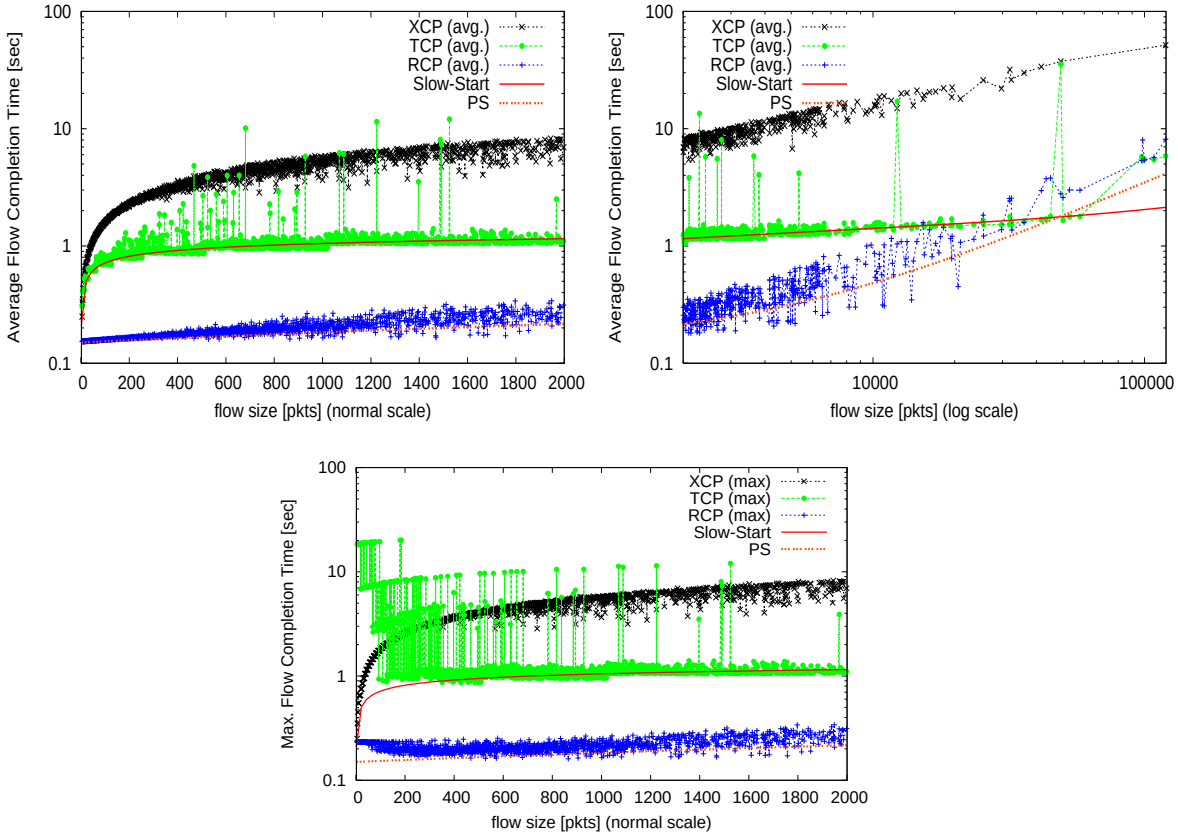


Figure 4.1: AFCT for different flow sizes when $C = 2.4$ Gb/s, RTPD $= 0.1$s, and $\rho = 0.9$. Flows are Pareto distributed with $E[L] = 25$ pkts, shape $= 1.2$. The top left plot shows the AFCT for flow sizes 0 to 2000 pkts; the right side plot shows the AFCT for flow sizes 2000 to 10^4 pkts; the bottom plot shows the maximum flow completion time among all flows of the particular size.

the other.

4.2 When Traffic Characteristics Vary

In this section our goal is to find out if RCP's performance is close to PS under different traffic characteristics. All simulations in this section are done with a single bottleneck link in the network.

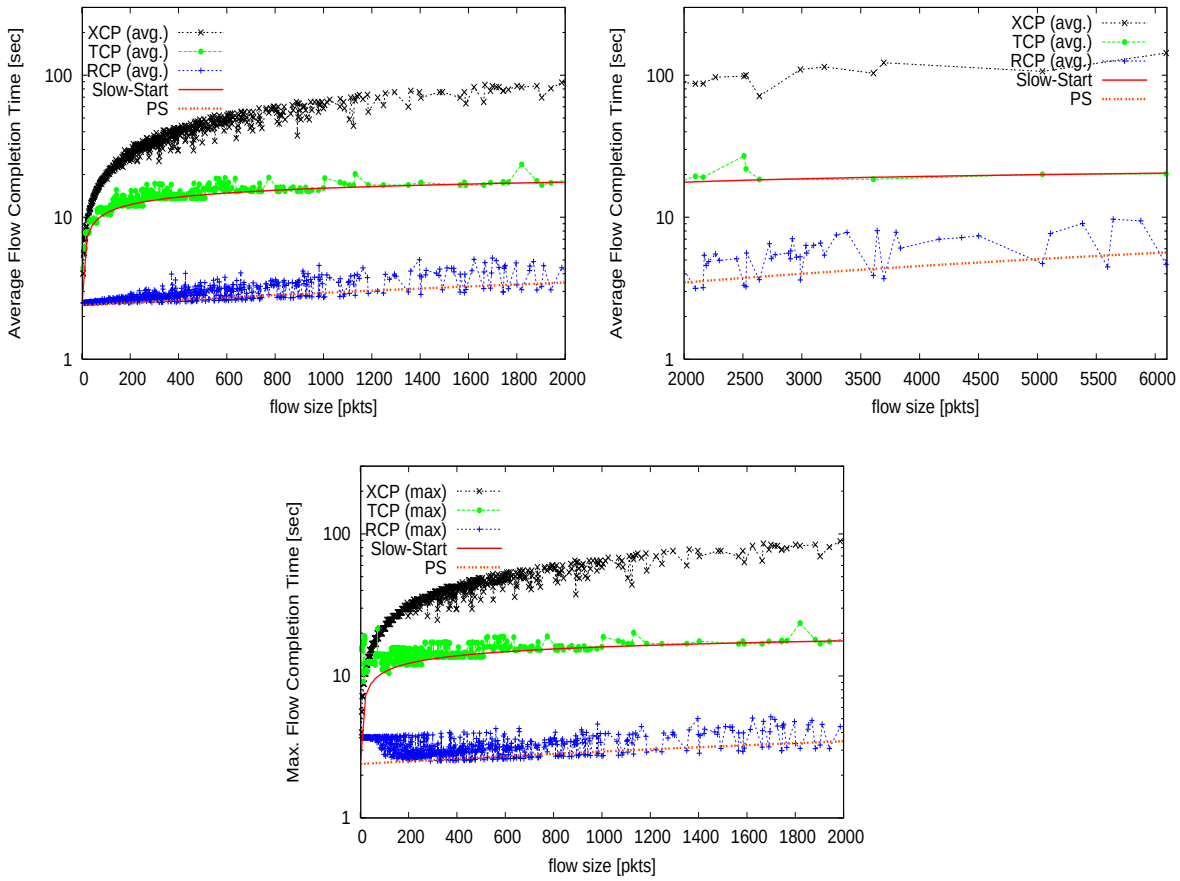


Figure 4.2: AFCT for different flow sizes when C=0.15 Gb/s, RTPD=1.6s, and $\rho = 0.9$. Flows are Pareto distributed with $E[L] = 25$ pkts, shape $= 1.2$. The top plots show the AFCT for flow sizes 0 to 2000 pkts (left) and 2000 to 6000 pkts (right); the bottom plot shows the maximum flow completion time among all flows of the particular size.

4.2.1 Average Flow Completion Time vs. Flow Size

In this section we will observe the AFCT of RCP, XCP and TCP for an entire range of flow sizes in two particular simulation setups. These setups are chosen to represent high bandwidth-delay product ($C \times$ RTPD) networks, since this is the scenario that often differentiates the performance of protocols. In both setups flow sizes are Pareto distributed.

- Setup 1: $C = 2.4$ Gbps, RTPD $= 100$ ms, $\rho = 0.9$

AFCT is plotted against flow size in the top two graphs of Fig. 4.1. The AFCT of RCP is close to that of PS and it is always lower than that of XCP and TCP. For flows up to

2000 pkts, TCP delay is 4 times higher than in RCP, and XCP delay is as much as 30 times higher for flows around 2000 pkts. Note the logscale of the y-axis.

With longer flows ($>$ 2000 pkts), the ratio of XCP and RCP delay still remains around 30, while TCP and RCP are similar. For any fixed simulation time, not only was RCP better for the flows that completed, but it also finished more flows (and more work) than TCP and XCP.

The third graph in Fig. 4.1 shows the maximum delay for a given flow size. Note that in RCP the maximum delay experienced by the flows is also very close to the average PS delay. With all flow sizes, the maximum delay for RCP is smaller than for TCP and XCP. TCP delays have high variance, often ten times the mean.

- Setup 2: $C = 150$ Mbps, RTPD = 1.6 s, $\rho = 0.9$

AFCT is plotted against flow size in the top two graphs of Fig. 4.2. Once again, the AFCT for RCP is always lower than for TCP and XCP and is close to the PS delay for all flow sizes. TCP's delay is about five times higher than RCP, while XCP is 20 times higher. The third figure shows the maximum delay for all three algorithms, and again TCP has the highest delay variance. RCP's mean and maximum are close.

The results above are representative of the large number of simulations we performed. Now let's see why these protocols have such different delays.

RCP vs. TCP: In both Figs., 4.1 and 4.2, the TCP delay for most flows follows the *Slow-start* curve. The delay in TCP slow-start for a flow of size L is $[log_2(L+1)+1/2] \times RTPD + L/C$ (excluding the queuing delay). With RCP the same flows get a jump-start because the routers set a higher initial rate close to what they would have gotten with PS. Hence their delay is close to PS. This is clear from the time evolution of a typical flow, as shown in Fig. 4.3 (top plot).

Next, consider the TCP flows which deviate from the Slow-start curve. These flows experienced at least one packet drop in their lifetime and entered the additive increase, multiplicative decrease (AIMD) phase. Once a flow is in the AIMD phase, it is slow in catching up with any spare capacity and therefore lasts longer than it needs to. RCP on the hand is quick to catch up with any spare capacity available and flows finish sooner. An example of the time evolution of a flow is shown in Fig. 4.3 (bottom plot).

RCP vs. XCP: The time evolution of XCP for two sample flows is shown in Fig. 4.3. XCP is slow in giving bandwidth to the flows, giving a small rate to newly starting flows. It gradually reduces the window sizes of existing flows and increases the window sizes of the

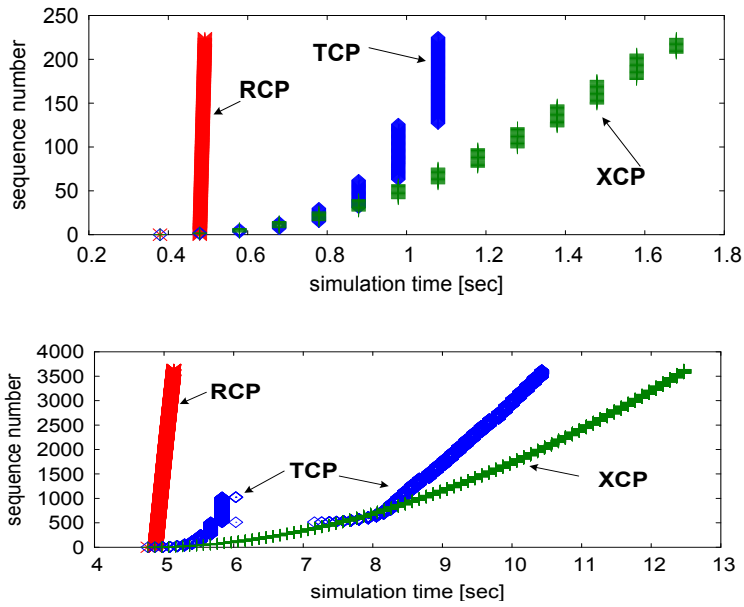


Figure 4.3: Time evolution of the sequence numbers of two flows under TCP (Reno), XCP and RCP, chosen from the simulation set up of Fig. 4.1. The flow size in the top plot is 230 pkts, and in the bottom plot is 3600 pkts.

new flows, making sure there is no bandwidth over-subscription. It takes multiple RTTs for most flows to reach their fair share rate (which is changing as new flows arrive). Many flows complete before they reach their fair share rate. In general, XCP stretches the flows over multiple RTPDs, to avoid over-subscribing the link, and so keep buffer occupancy low. On the other hand, RCP tries to give the equilibrium rate to every flow based on the information it has so far, at the expense of temporary bandwidth over subscription.

4.2.2 When mean flow size increases

Fig. 4.4 compares AFCT when mean flow size gets longer. Flow sizes are Pareto distributed and the mean flow size is varied from 30 pkts (equals $\frac{1}{1000} \cdot C \cdot RTPD$) to 30,000 pkts (equals $C \cdot RTPD$). The top plot shows the AFCT averaged over flows with $< 7,000$ pkts and the bottom one is for flows $\geq 7,000$ pkts.[3] There are two points to take away from the graph:

[3]We consider these two different ranges because, with a Pareto distribution, there are many more short flows than long flows. Just taking the average AFCT over all flows is more representative of the short flows than the long flows.

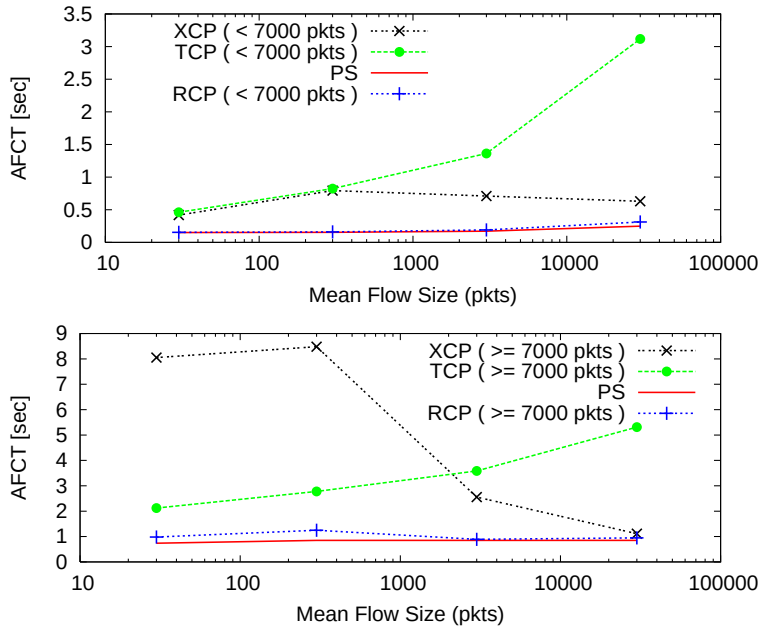


Figure 4.4: Comparison of AFCT as the mean flow size increases. Flows are Pareto distributed with shape 1.2 and the mean flow size varies as shown on x-axis. $C = 2.4$ Gb/s, RTPD $= 0.1$s and $\rho = 0.8$. The top plot shows the AFCT for flows with < 7000 pkts vs. mean flow size; the bottom plot shows the AFCT for larger flows (> 7000pkts) vs. mean flow size.

1. The AFCT of RCP is close to PS irrespective of the mean flow size.

2. The performance of XCP and TCP is reversed as the mean flow size increases: For small flows, XCP performs far worse than TCP – see bottom plot of Fig. 4.4. As flows get larger, XCP's performance gets closer to PS while TCP deviates further from it.

XCP vs. TCP: The reversal in performance of XCP and TCP is also clearly illustrated in Fig. 4.5. The top plot shows a snap shot of the AFCTs for $E[L] = 30$ pkts and the bottom two plots are for $E[L] = 30000$ pkts. In the bottom plot the AFCT of TCP flows is more than an order of magnitude higher than in PS – this is due to the well known problem with TCP in high bandwidth delay product environments [25] i.e., long flows are unable to catch up with spare bandwidth quickly after experiencing a loss. XCP and RCP are both close to PS. On the other hand, for small flows, XCP's performance is worse than TCP's because XCP is conservative in giving bandwidth to flows, especially newly starting flows. This unnecessarily prolongs flows and so the number of active/ongoing flows begins to grow.

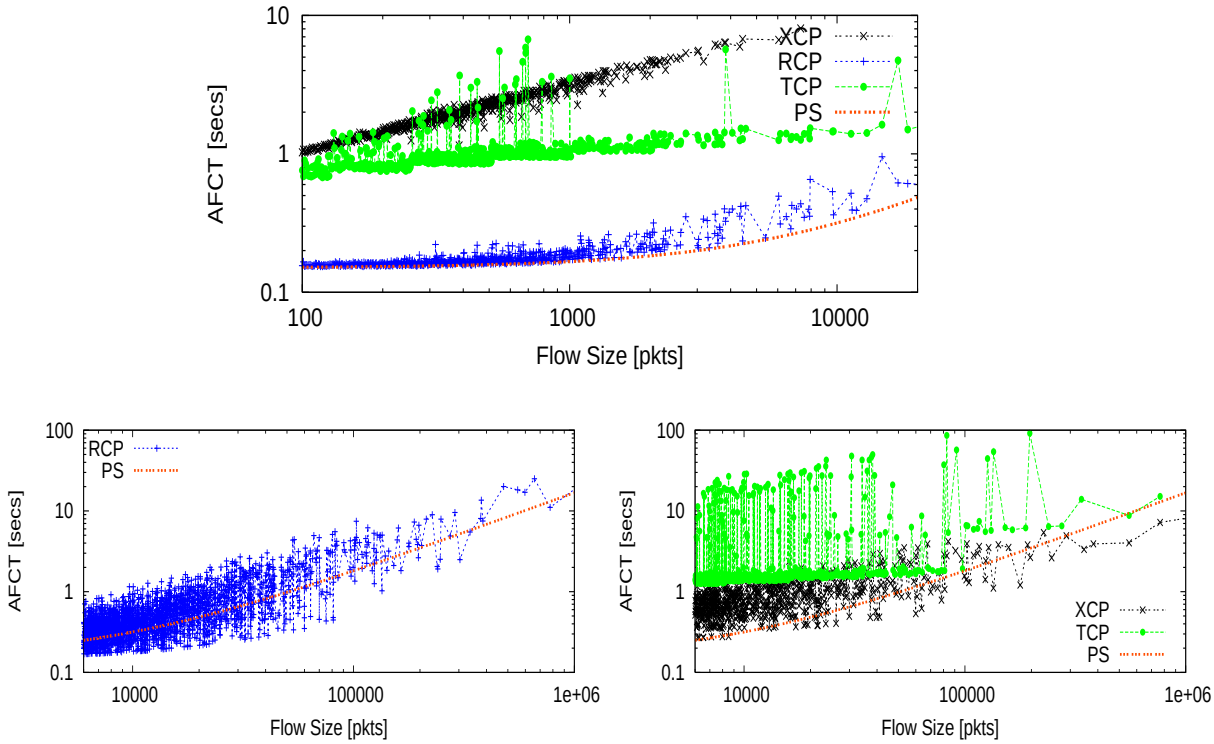


Figure 4.5: Comparison of AFCT as the mean flow size increases. The simulation set up is the same as in Fig. 4.4. The top graph shows the AFCT vs. flow size when $E[L] = 30$; the bottom left (RCP) and bottom right graph (TCP, XCP) show the AFCT vs. flow size when $E[L] = 30000$ pkts. RCP does close to PS irrespective of mean flow size. The performance of XCP and TCP are reversed with the increase in the mean flow size.

This in turn reduces the rate of new flows, and so on. Our many simulations showed that new flows in XCP start slower than with Slow-Start in TCP.

We will see next that this phenomenon is not specific to Pareto distributed flows, but happens with other distributions as well.

4.2.3 Different flow size distributions

We simulated RCP, TCP and XCP under several different flow size distributions *and* for different parameters of each distribution. We will present representative examples here. We chose three distributions – uniform, exponential and Pareto – to represent super exponential-tailed, exponential-tailed and heavy-tailed distributions. Figs. 4.6, 4.7, and 4.8 show the AFCT and number of active flows for uniform, exponential and Pareto respectively. The

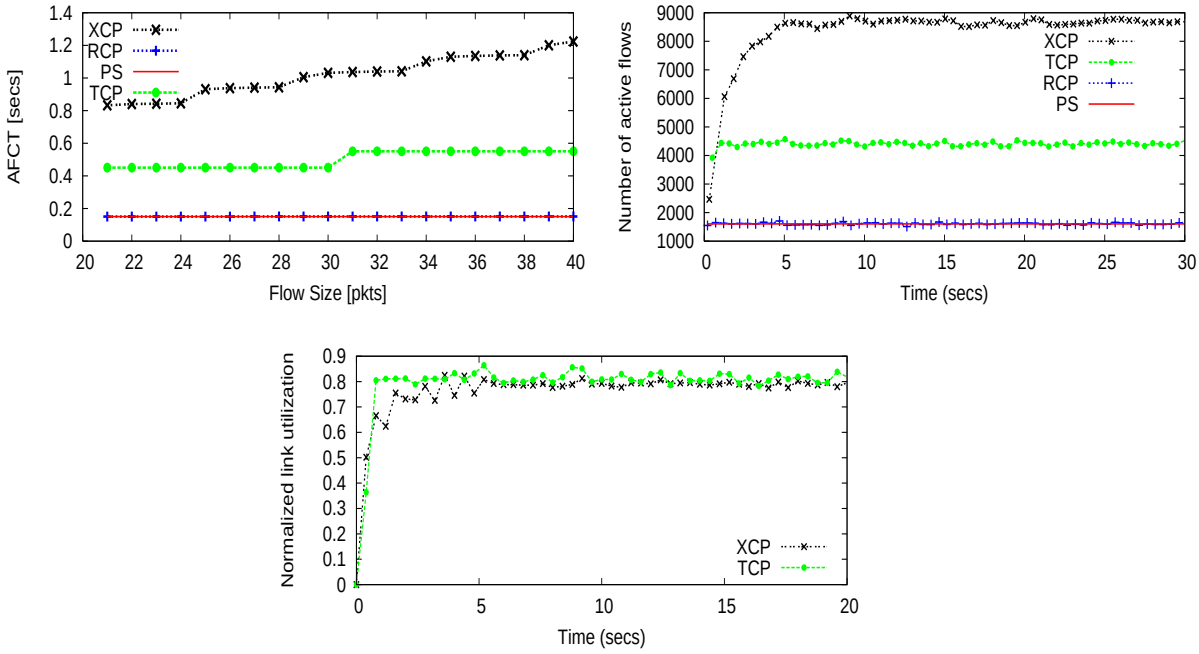


Figure 4.6: Comparison under uniform distributed flow sizes $[20, 40]$. $C = 2.4$ Gbps, RTPD $= 0.1$ sec, $\rho = 0.8$. The top left graph shows the AFCT vs. flow size, the right side graph shows the number of active flows versus time and the bottom graph shows the normalized link utilization (RCP utilization is omitted for sake of clarity). Note how XCP builds up work in the system due to it's conservative nature of giving out bandwidth to flows.

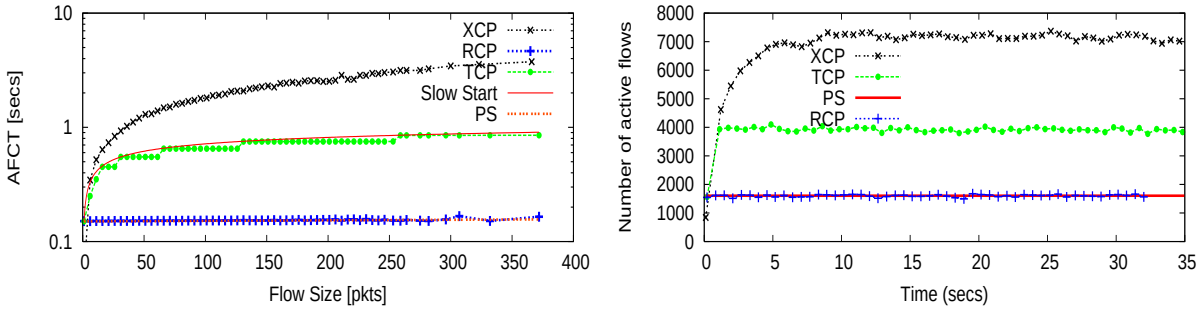


Figure 4.7: Flows sizes have exponential distribution with $E[L] = 30$ pkts. The rest of the set up is same as in Fig. 4.6.

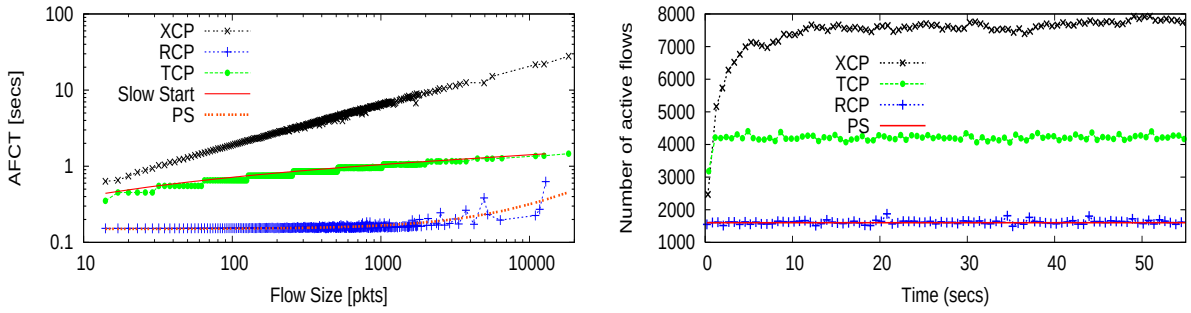


Figure 4.8: Flows have Pareto distributed flow sizes with $E[L] = 30$ pkts and shape parameter $= 1.8$. The rest of the set up is same as in Fig. 4.6.

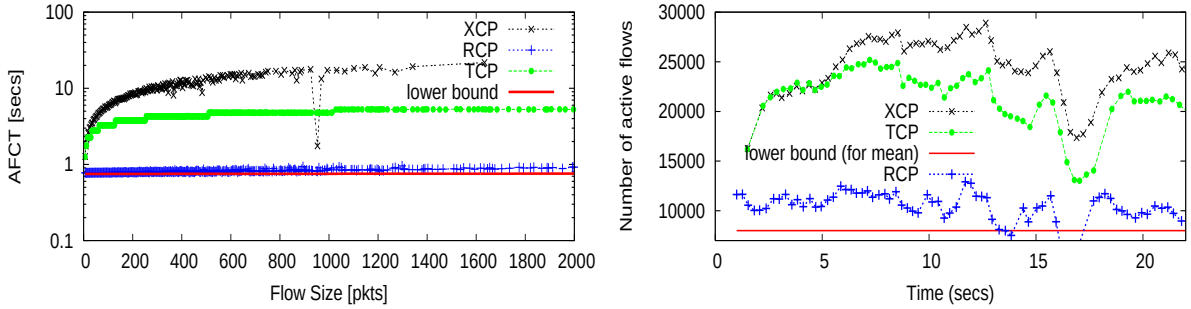


Figure 4.9: Comparison when flows arrival times are Pareto distributed, shape $= 1.2$. $E[L] = 30$ pkts, $C = 2.4$ Gbps, RTPD $= 0.5$ sec, $\rho = 0.9$. RCP, TCP and XCP are compared with the minimum possible AFCT $= 1.5RTPD + \frac{L}{C}$. The lower bound for the mean number of active flows is given by Little's Law.

take away point from all these plots is that RCP follows PS closely regardless of the flow-size distribution.

As noted before, because the mean flow size is small compared to the bandwidth delay product, XCP has a large number of active flows and an AFCT higher than TCP.[4]

4.2.4 Non-Poisson flow arrivals

Up until now we have only considered the case when new flows arrive according to a Poisson process. In practice, "flash crowds" can happen due to a sudden interest in a website, time of day, a particular set of data, or just bad luck. While we can expect a sudden burst of

[4]In some of the simulations, not shown here, the number of active flows in XCP was still growing when the simulation ended and did not converge to any equilibrium point. This may either be because XCP does not have any equilibrium point i.e. the system is stochastically unstable in the sense of $E[N] \to \infty$ or that it has an equilibrium but which is a larger number than the simulation reached.

new flows to upset RCP - in particular, to temporarily upset RCP's estimate of $N(t)$ - RCP should recover and converge to the new situation within a few RTTs.

Our simulations suggest that this is the case. We simulated RCP, TCP and XCP with bursty flow arrivals using a Pareto distribution. Fig. 4.9 shows one such example. Qualitatively, our results are the same as we observed with Poisson flow arrivals. Since we do not have an expression for PS under a $G/G/1$ model, we compare the AFCT with its lower bound i.e. $1.5RTPD + L/C$. RCP compares well with the lower bound in all cases we considered.

These simulations only show the typical behavior. The worst case behavior of RCP is when there is a sudden flash crowd of large elephant flows starting at nearly the same time. In such scenarios, we are interested in finding out (as we will see in Chap. 5) whether and how fast the RCP system converges to a stable behavior.

4.2.5 As load increases

The results so far were for the case when average offered load is at or below 90%. Fig. 4.10 shows the AFCT as the offered load is varied. The main observations are:

1. The AFCT of RCP follows that of PS closely.

2. Under low loads and for large flows, TCP's AFCT is as much as fourteen times higher than PS (and RCP) but improves with increased load. XCP on the other hand has sufficient spare capacity under low loads so as not to make all flows last many RTTs. On the other hand XCP's performance gets worse with an increase in load – because with increased load, the system maintains more active flows and hence increases flow duration.

We have seen such a switch in performance between XCP and TCP once before in Sec. 4.2.2 as the mean flow size increases. We see that here again as the offered load increases. A detailed plot is shown in Fig. 4.11 under a different setting, for a low offered load of 0.2 and in Fig. 4.12 for a high offered load of 0.94. Notice that XCP performs better than TCP on average under a low load regime and vice versa under a high load regime. RCP's performance is fairly independent of the offered load.

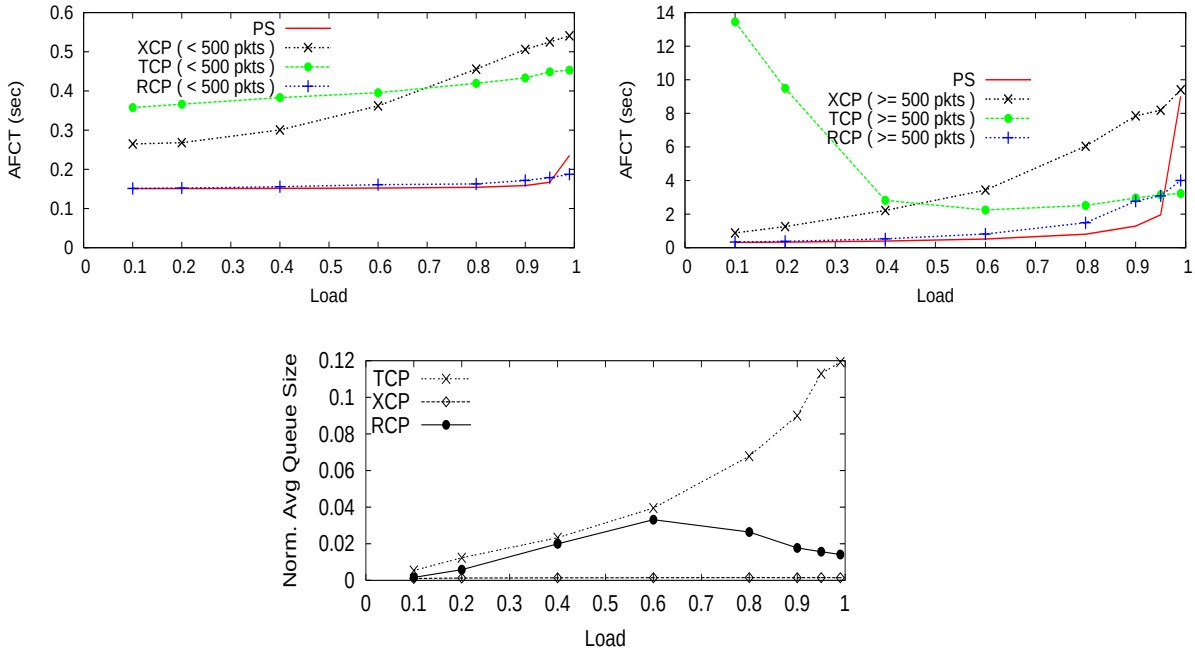


Figure 4.10: Comparison of RCP, TCP and XCP under different loads with $C = 150$ Mb/s and RTPD $= 0.1$s. The top two plots shows the AFCT vs. load and the bottom plot shows the normalized average queue length vs. load.

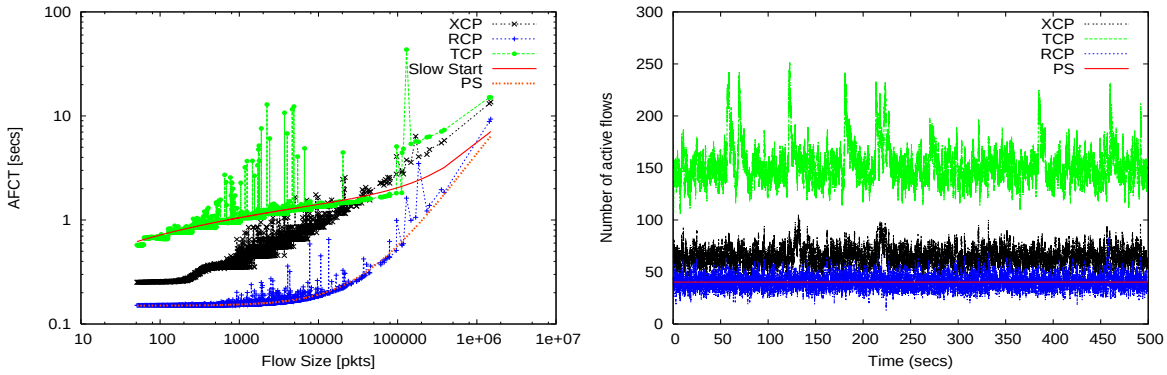


Figure 4.11: Comparison of RCP, TCP and XCP under low offered load of $\rho = 0.2$, $C = 2.4$ Gb/s, RTPD $= 0.1$s, Pareto flows $E[L] = 300$ pkts. The left plot shows the AFCT vs. flow size and the right plot shows the number of active flows vs. time. XCP finishes flows faster than TCP under this low load regime.

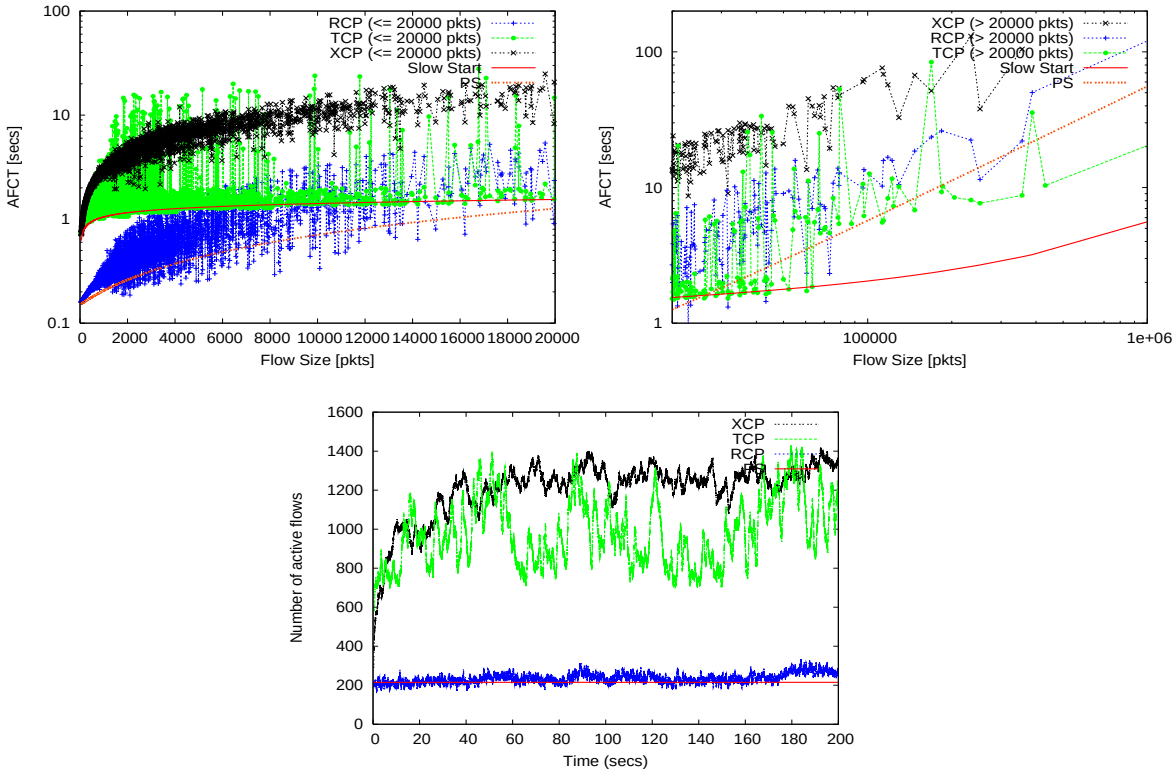


Figure 4.12: Comparison of RCP, TCP and XCP under a high offered load of $\rho = 0.94$, $C = 2.4$ Gb/s, RTPD = 0.1s, Pareto flows $E[L] = 283.35$ pkts. The top and middle plot show the AFCT versus flow size and the bottom plot shows the number of active flows versus times. TCP finishes flows quicker than XCP under the high load regime.

4.3 When Network Conditions Vary

In this section we explore how well the congestion control algorithms match PS under different network conditions. We will vary the link capacity, round-trip times, and increase the number of bottlenecks. In each case, we find that RCP matches PS closely. In the simulations that follow, flows arrive as a Poisson process with Pareto distributed flow sizes, $E[L] = 25$ pkts, shape $= 1.2$.

4.3.1 When link capacity increases

As link capacity increases, the bandwidth-delay product increases, and so more flows can potentially complete in fewer RTTs. Fig. 4.13 shows the AFCT for flows < 500 pkts and

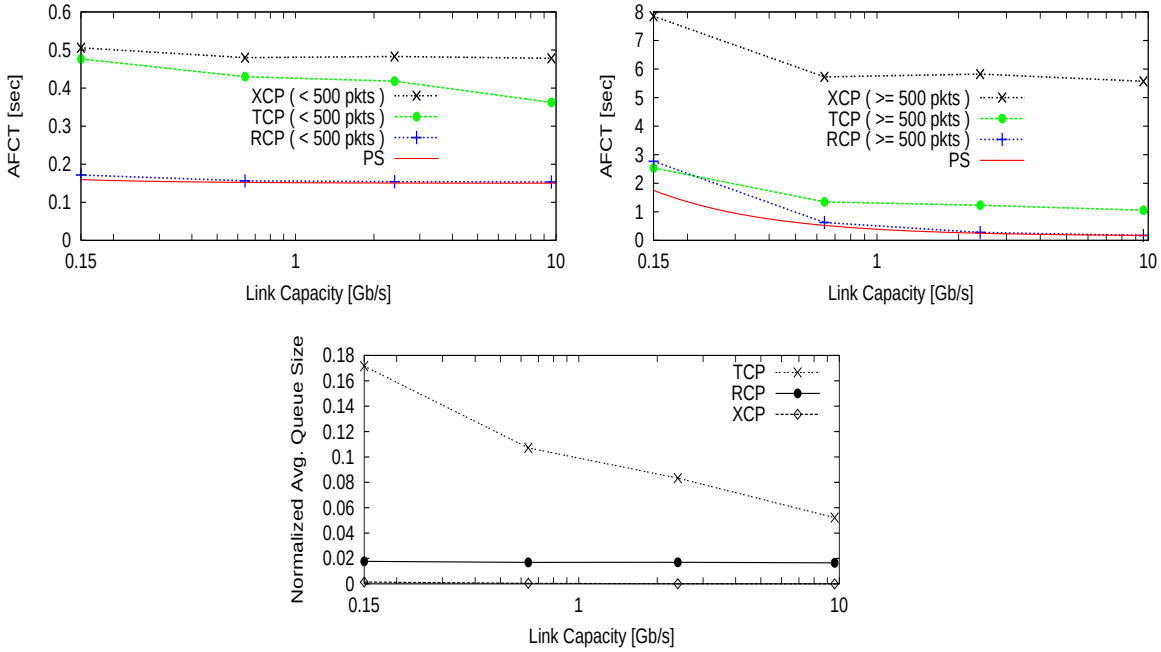


Figure 4.13: The comparison of RCP, TCP and XCP under different capacities; RTPD = 0.1s and ρ =0.9. The top left plot shows AFCT of flows with size $<$ 500 pkts, the right plot shows AFCT of flows with size $\geq$ 500 pkts. The bottom plot shows the normalized average queue size.

$\geq$ 500 respectively,[5] for link capacities varying from 100 Mbps to 9.6 Gbps.[6] AFCT for RCP is always lower than for TCP and XCP, and is close to PS for any link capacity. TCP and XCP require multiple RTTs to find the rate for a flow, and so short flows are forced to last longer than necessary. As link rates increase, the problem gets worse for short flows.

For flows with fewer than 500 packets, flows last three times longer with XCP than with RCP; and twice as long for TCP than for RCP. For flows with more than 500 packets, flows last 30 times longer with XCP, and six times longer with TCP than with RCP. Fig. 4.13 also shows the normalized queue sizes i.e. average queue size divided by the product $C \times$ RTPD. The normalized queue size of RCP is 2% of the bandwidth-delay product for all the link capacities and is smaller than that of TCP. XCP always maintains a very small queue size.

[5] 99.5% of flows have size $<$ 500 pkts

[6] RTPD and load are fixed at 100ms and 0.9 respectively for all simulations.

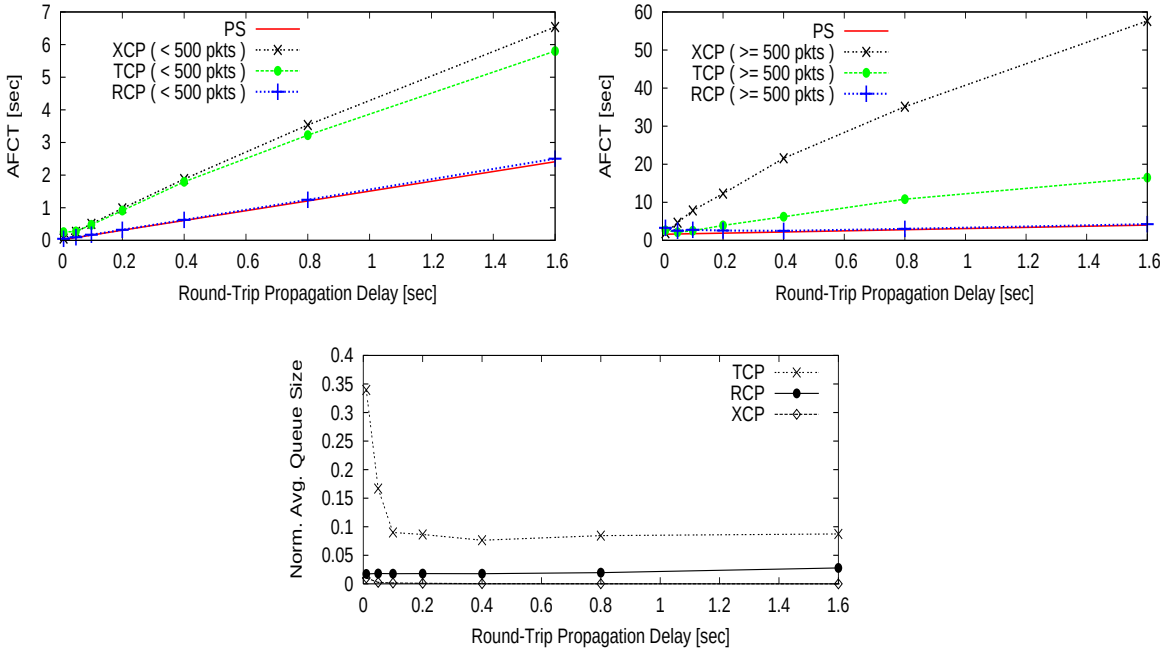


Figure 4.14: The comparison of RCP, TCP and XCP under different RTPDs, C =150Mb/s, ρ =0.9. The top left plot shows AFCT of flows with sizes $<$ 500 pkts and the right plot shows AFCT of flows with sizes $\geq$ 500 pkts. The bottom plot shows the normalized average queue size.

4.3.2 When Round-Trip Propagation Delay increases

Increasing the path length also increases the bandwidth-delay product, making it possible for flows to complete in fewer RTTs. Fig. 4.14 shows the performance for different RTPDs ranging from 10 ms to 1.6 s, with a fixed C = 150 Mb/s and ρ = 0.9.

Again, RCP flow duration is lower than for TCP and XCP, and follows the PS line closely for all RTPDs. Flows with fewer than 500 pkts last about 2.5 times longer with TCP and XCP than for RCP. And flows with more than 500 pkts last four times longer with TCP and thirteen times longer with XCP than with RCP.

As with PS, RCP is robust to variations in propagation delay: As RTPD increases, the AFCT of flows increases as 1.5 RTPD – the first term in Eqn. 4.1; the queuing + transmission delay is independent of RTPD in RCP, just as with PS.

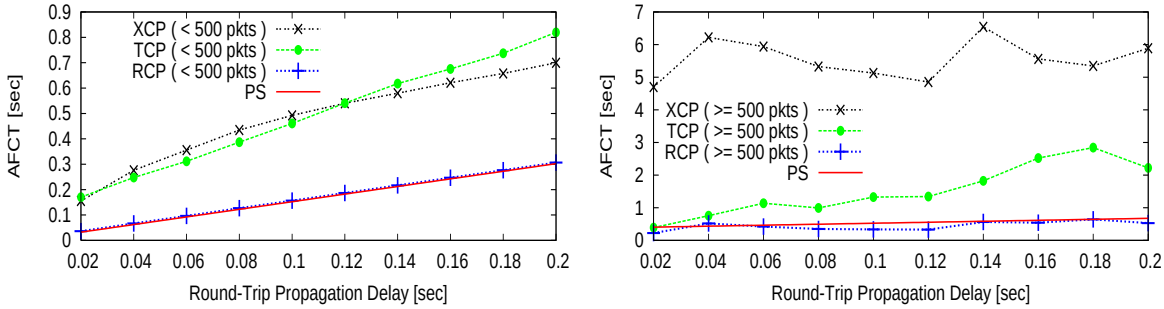


Figure 4.15: Comparison of RCP, TCP and XCP when flows with different RTPDs coexist on a single bottleneck of $C = 0.64$ Gb/s. RTPD of flows vary from 0.02s to 0.2s. The left plot is the AFCT of flows with flow size ≤ 500 pkts and the right plot shows the AFCT for flows with size > 500 pkts.

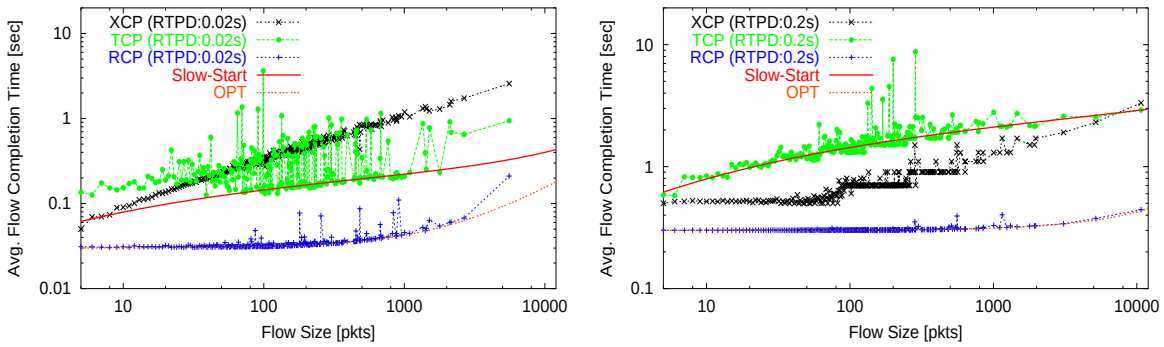


Figure 4.16: Comparison of RCP, TCP and XCP when flows with different RTPDs coexist on a single bottleneck of $C = 0.64$ Gb/s. RTPD of flows vary from 20 ms. to 200 ms. The left plot is the AFCT of flows with RTPD = 20ms and the right plot shows the AFCT for flows with RTPD = 200 ms.

4.3.3 Flows with different round-trip times

All three congestion control schemes depend on feedback to adjust the window size and/or sending rate. If different flows have shorter round-trip times, we do not want them to benefit at the expense of others.

To explore this effect we simulated flows that share a common bottleneck link, but with different RTPDs. The round-trip delay of the common bottleneck link is 0.01 sec. and its capacity is 640 Mb/s. Arriving flows are classified into ten groups. Flows in the same group have the same end to-end RTPD, and each group has an RTPD of 0.02, 0.04, ..., 0.18, or 0.2 sec. All groups have the same flow arrival rate and total $\rho = 0.9$.

Fig. 4.15 shows the AFCT for these different groups of flows. The x-axis is each group's

RTPD. For each RTPD, RCP is close to PS, suggesting that RCP is not biased in favor of flows with shorter RTPD. For a more explicit illustration of this fact, Fig. 4.16 shows the AFCT for two different groups of flows. The top figure shows the AFCT for flows with RTPD of 0.02s (smallest value) while the bottom figure is for flows with RTPD 0.2s (largest value). RCP seems robust to variations in RTT and achieves close to the minimum AFCT for each of the group of flows.

4.3.4 When the reverse link is congested

So far the flows encountered a single bottleneck link in the forward direction. In this section the reverse link is also bottlenecked, allowing us to observe any effect on flow completion times when the ACKs encounter congestion. Fig. 4.17 shows the performance of the three protocols when the forward and reverse links are both loaded at 0.75. For comparison purposes Fig. 4.18 shows the corresponding performance when there is no reverse traffic. The only way we can expect a reverse congested link to affect the forward traffic is through the ACKs experiencing queuing delay or being dropped. Since the queue sizes in RCP and XCP are under control and are deliberately driven to zero, we can expect the reverse congestion to have little or no effect on the forward traffic. The results in Figs. 4.17 and 4.18 confirm this. TCP on the other hand maintains large buffer occupancies resulting in delayed or dropped ACKs. This results in increased timeouts at the sender and eventually increased flow completion times.

TCP's performance deterioration under reverse congestion is further illustrated in Fig. 4.19 where now there is a high offered load of 0.95 on both forward and reverse links. We saw in Sec. 4.2.5 that under high loads TCP out-performs XCP. But in the presence of reverse congestion, TCP performs just as poorly as XCP. In fact in this particular simulation with a high load on both forward and reverse link, TCP and XCP appear to be stochastically unstable in the sense that $E[N(t)]$ does not converge. Unlike with TCP, XCP's instability is probably just due to the heavy offered load and has nothing to do with the reverse traffic. In case of RCP, the performance is slightly worse than PS, nevertheless comparable to PS and stable.

A final example is shown in Fig. 4.20 and 4.21 with a heavy load of 0.95 on the forward link and a light load of 0.2 on the reverse link. The results reiterate the observations made before — for the heavily loaded link, the active number of flows in XCP continually increases, TCP flows see a large variance in the flow completion times, while RCP's performance is

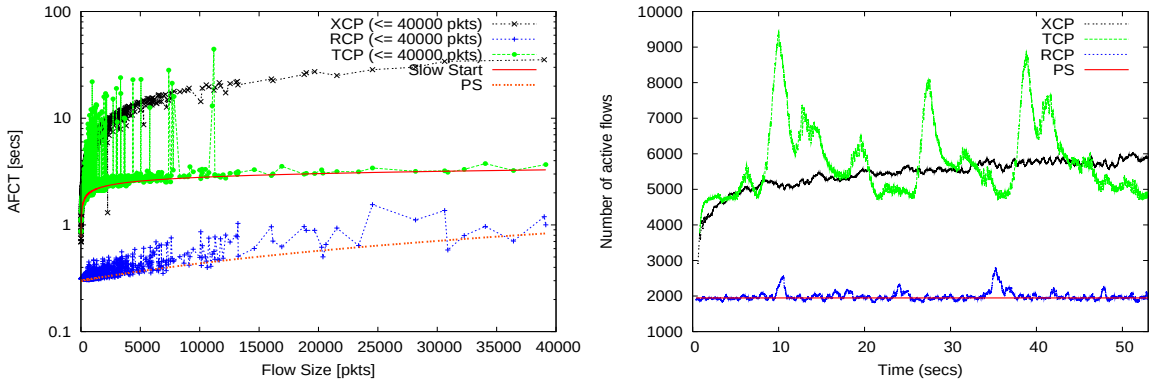


Figure 4.17: Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Offered load = 0.75, $E[L]$ = 46 pkts, C = 2.4 Gbps, RTT = 0.2 s. The left plot shows the AFCT versus flow size and the right plot shows the number of active flows over time. Compare the metrics here with Fig. 4.18 and notice the increased variance in TCP's delay and the number of active flows. The reverse traffic metrics (not shown here) are qualitatively similar.

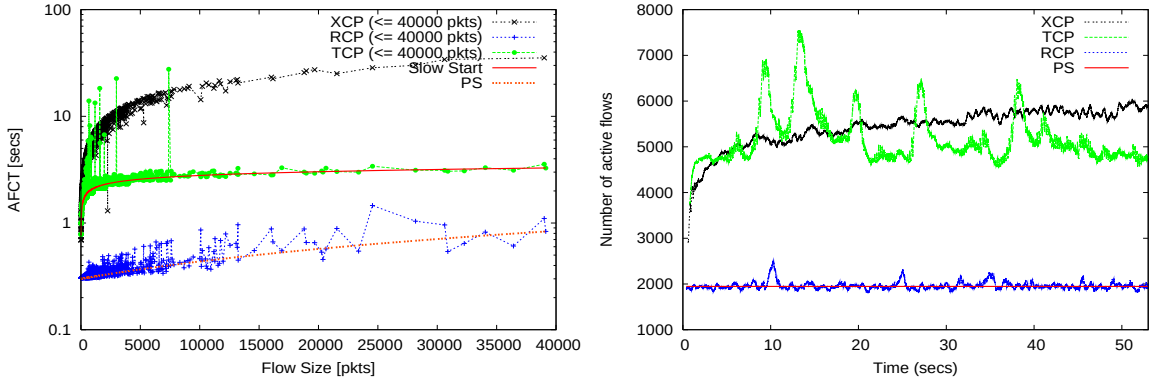


Figure 4.18: Comparison of RCP, TCP and XCP when only forward link is congested. Offered load = 0.75, $E[L]$ = 46 pkts, C = 2.4 Gbps, RTT = 0.2 s.

comparable to PS. In case of the lightly loaded link, as was seen in Sec. 4.2.5, XCP performs better than TCP.

4.3.5 When there are multiple bottlenecks

While flows typically encounter only one bottleneck, it is possible for a flow to encounter two or more. Fig. 4.22 shows a topology with n cascaded bottleneck links. Flows in group 0 traverse all the bottleneck links, and flows in group $i (i = 1, .., n-1)$ use only the bottleneck link i. All the bottleneck links have the same capacity C and the same delay, and all flow

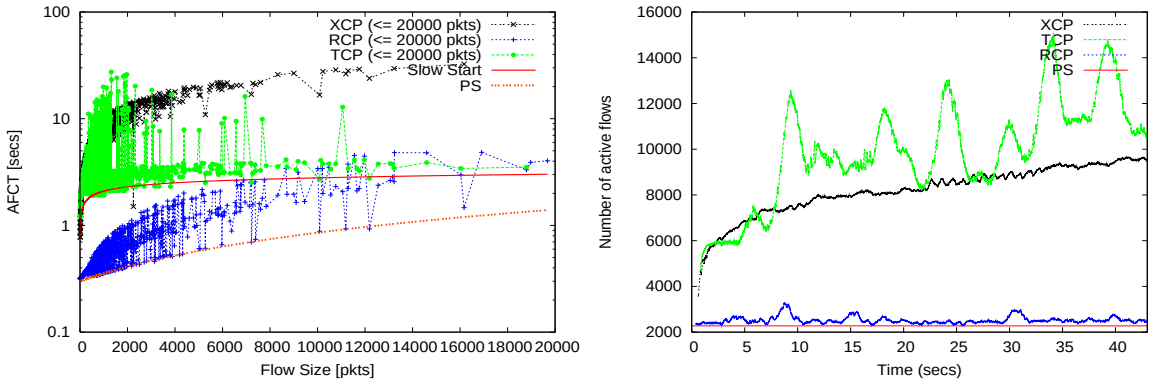


Figure 4.19: Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Offered load = 0.95, $E[L]$ = 46 pkts, C = 2.4 Gbps, RTT = 0.2 s. The top plot shows the AFCT versus flow size and the bottom plot shows the number of active flows over time. XCP and TCP appear to be stochastically unstable. RCP's performance deviates slightly from PS but is stable.

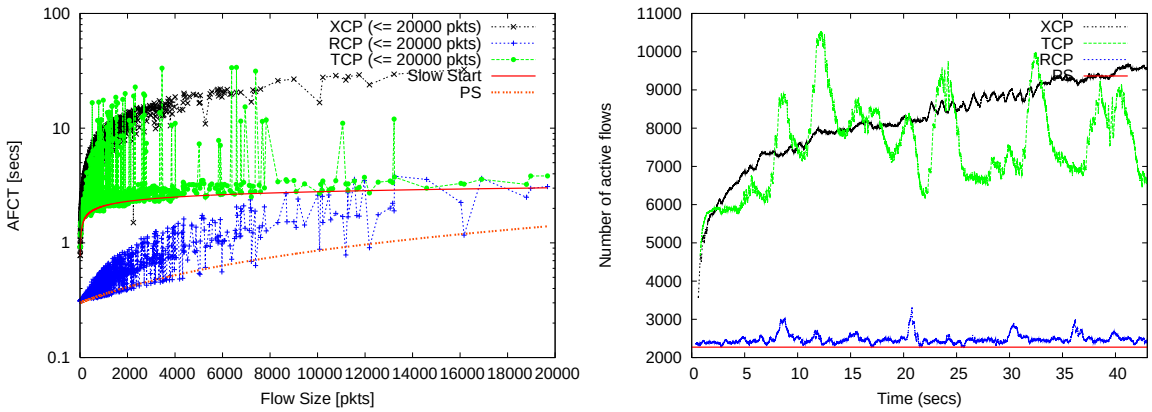


Figure 4.20: Comparison of AFCTs when both forward and reverse link are bottlenecked. Forward link: offered load = 0.95, $E[L]$ = 46 pkts, C = 2.4 Gbps, RTT = 0.2 s. The left plot shows the AFCT versus flow size and the right plot shows the number of active flows over time. XCP appears to be stochastically unstable. TCP flows have a large variance in the flow completion times. RCP's performance deviates slightly from PS but is stable.

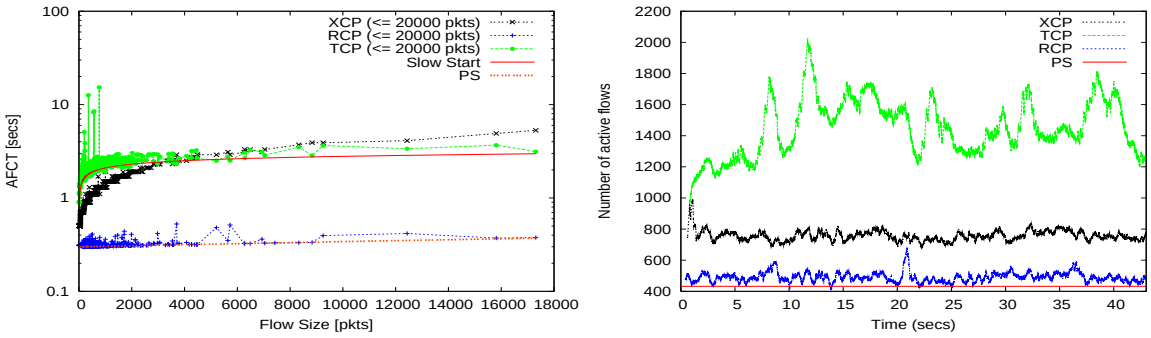


Figure 4.21: Comparison of RCP, TCP and XCP when both forward and reverse link are bottlenecked. Reverse link: offered load = 0.2, $E[L] = 46$ pkts, $C = 2.4$ Gbps, RTT = 0.2 s. The left plot shows the AFCT versus flow size and the right plot shows the number of active flows over time. As expected under low load, XCP performs better than TCP, while RCP is close to PS.

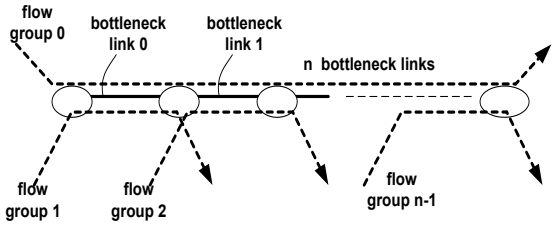


Figure 4.22: Multiple bottleneck topology. Flow group 0 traverses all the bottleneck links and the other group of flows (group 1,..,n-1) use only one bottleneck link.

groups have the same arrival rate λ.

Here, we investigate the effect of increasing the number of bottlenecked hops on flows in group 0. While fixing the RTPD of flow group 0 at 0.2s, we increase the number of hops to one, two, four, eight and observe how AFCT changes. The AFCT for flow group 0 is plotted in Fig. 4.23. RCP always achieves a delay close to the lower bound irrespective of the number of hops. The performance of TCP degrades as the number of hops increase. We noticed that TCP flows have a greater chance of experiencing losses when traversing through multiple bottlenecks, and in general have a lower throughput.

The detailed plots of AFCT vs. flow-size as the number of bottlenecked links increase are shown in Fig. 4.24. Notice how the performance of TCP degrades as the number of

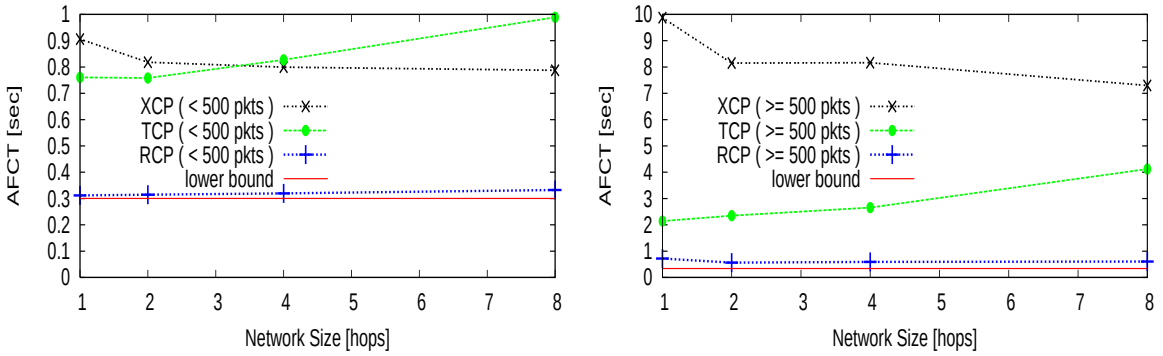


Figure 4.23: The comparison of RCP, TCP and XCP under 1,2,4 8 bottlenecks, C=0.64 Gb/s, RTPD of flow group 0 is 0.1 s, $\rho = 0.9$. The left plot shows the AFCT of flow group 0 for flow sizes < 500; the right plot shows the AFCT of flow group 0 for flow sizes ≥ 500.

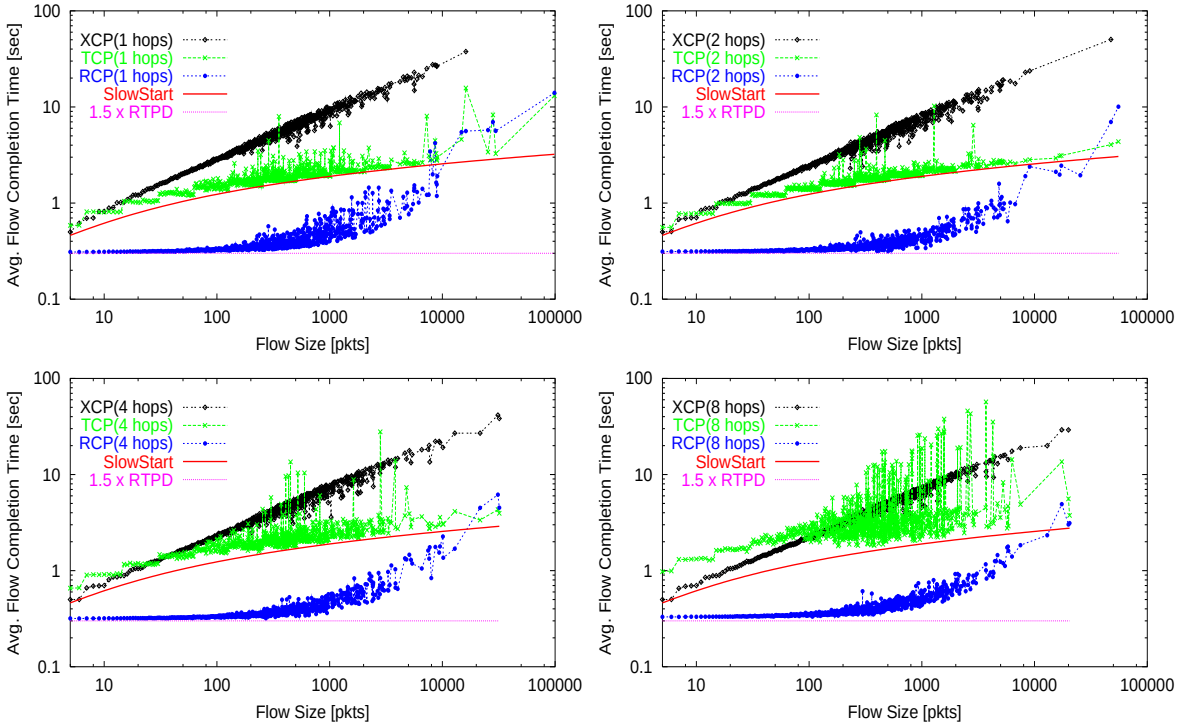


Figure 4.24: The comparison of RCP, TCP and XCP under 1,2,4 8 bottlenecks, C = 0.64 Gb/s, RTPD of flow group 0 is 0.1 s, $\rho = 0.9$. Top through bottom are plots showing AFCT of flow group 0 for number of bottleneck links = 1, 2, 4 and 8 respectively.

bottlenecks increase; the AFCT of TCP gets farther away from the Slow-Start curve as the flow passes through more bottlenecks. This is because the probability of a loss increases as a flow traverses through multiple bottlenecks, and hence having a greater chance of entering into AIMD phase.

RCP achieves max-min bandwidth sharing in the multiple bottleneck case, and with multiple bottlenecks, flows complete an order of magnitude faster with RCP than with TCP or XCP.

4.4 Impact of RCP's short flow completion times on network users

We have hundreds of simulations illustrating RCP's short flow completion times under different network and traffic conditions, but we also want to understand how this will impact end users' experience. To that end, we articulate three distinct examples below on how RCP will improve our day-to-day interactions over the network.

Example 1: Downloading short back-to-back files (e.g. pictures, video-clips)

A single short download often only takes such a short time, that improving upon it may not be even discernible to a human eye. However latency is additive, and doing multiple such back-to-back short downloads makes the difference more obvious, as is the case in the following example: a network user, Zoe, with an access to 100 Mbps link is looking to replace her desktop wallpaper. She goes to the popular photo sharing site, Flickr [61], and searches for all images showing sunsets by the Golden Gate bridge. A page with thumbnail images comes up, she then clicks on individual images to get larger versions. Suppose each image is 1 MB each and she settles down on the one she likes after looking at 10 images.

With TCP, even in the very best case, with no packet losses and no congestion, each image takes at least 10 round trips due to TCP's slow-start. With an average round-trip time of 80 ms., it will be at least 8 s. before Zoe finds the image she likes. With RCP, each image downloads within one round-trip time and Zoe can find what she is looking for in less than a couple of seconds.

An alternative way of looking at it is in the time it takes Zoe to download 10 of these images with TCP, she can download 50 of them with RCP.

Example 2: When users join and leave the network dynamically

We often use the network dynamically, one moment we are downloading email, the next instance we are following a link in an email to browse through online pictures, and so on. Naturally, we also want congestion control to keep pace by adapting just as quickly. TCP cannot always keep up with the sudden changes or availability in link bandwidth, as is the case in the example where a network user, Jessica, is watching videos from YouTube when other network users start up different applications. The video quality degrades over time as there are 20 other users sharing the link - some downloading email, some watching television through the Internet, others downloading iTunes music, and yet others the latest release of Linux kernel. Suppose, these users are all approximately sharing the the 100 Mbps link in equal shares, and after a while they are done downloading what they have to and Jessica is expecting to see her video quality jump up to as it was before any of these had started. But the reality is, a TCP flow in congestion avoidance will take close to 1100 round-trip times or a 118 seconds (with a round-trip time of 80 ms.) to fill the pipe with traffic. On the other hand, RCP is capable of making quick use of available bandwidth within a reasonable value of 10 RTTs i.e. within a second, and Jessica sees the video quality improve in a blast.

Example 3: Crisscrossing flows across the globe

With individual user content popping up in every continent, it is now quite common for flows originating from half the way across the world to share links with flows originating in the same city. In such situations, we want congestion control to treat all flows equally. For example a network user at Stanford, Rui, is watching a video clip from University of Melbourne with a round trip of 200 ms. Neda, another user at Stanford, is transferring a file from a nearby server in Palo Alto with a round trip of 10 ms. Because TCP flows with long round trips have a difficult time getting their fair-share of the bottleneck link, Rui's video from Australia is barely a trickle, while Neda's flow ramps up fast and hogs the links.

RCP, on the other hand, ensures a fair allocation to all without distinguishing based on RTTs, so all users are happy.

Chapter 5

Stability of RCP

In the last chapter we saw how RCP behaves under "normal" traffic conditions. However, sudden and large changes in traffic can and will occasionally happen in real networks. They are more likely to happen more often in access networks which typically have a low degree of statistical multiplexing. Even in backbone networks, where traffic is usually smooth and highly multiplexed, sudden changes can happen if there is a link or a router line-card failure. After such an impulse change, the network can transition to one of two states - it either settles down to a new stable equilibrium point, or it oscillates between under-flowing and overflowing queues (depending on router buffer sizes). Naturally, network operators and users would both like the system to reach equilibrium quickly and remain there. Over the past years there has been a significant amount of understanding on the stability and equilibrium properties of TCP under different network and traffic conditions [34, 37]. In this chapter we are interested in the dynamics and equilibrium of RCP under sudden changes, and particularly questions such as: can we choose RCP's parameters such that it is stable for a broad range of network conditions? Will it be stable even when flows with a large difference in round-trip times share bottleneck links? How long will it take to converge to a stable state? How well does its stability compare to TCP and other congestion control mechanisms?

We will first derive the stable region of RCP's parameters for a single bottleneck link using a simple linearized model (Sec. 5.1), then see how the stable region changes when we take the system non-linearities into consideration (Sec. 5.2) and when flows with heterogeneous round-trips share multiple bottleneck links (Sec. 5.3), and finally determine how to choose RCP's parameters not just for stability but also for fast convergence (Sec. 5.4).

5.1 Stability analysis

For a single bottleneck, the RCP system can be expressed as:[1]

$$\begin{aligned}
\dot{R}(t) &= R(t)\left(\frac{\alpha(C - y(t)) - \beta\frac{q(t)}{d(t)}}{C \cdot d(t)}\right) && (5.1)\\
y(t) &= N \cdot R(t - d_0) && (5.2)\\
d(t) &= d_0 + \frac{q(t)}{C} && (5.3)\\
\dot{q}(t) &= [y(t) - C]; \qquad q(t) > 0 && (5.4)\\
&= \max[y(t) - C, 0]; \quad q(t) = 0
\end{aligned}$$

where $R(t)$ is RCP rate, C is link-rate, $y(t)$ is aggregate incoming traffic rate, $q(t)$ is queue occupancy, N is number of flows, d_0 is the round-trip propagation delay, and $d(t)$ is the round-trip time at t. If (R, q) represents the system state, then the equilibrium point (defined by $\dot{R} = 0, \dot{q} = 0$) is, $(R_e, q_e) = (\frac{C}{N}, 0)$. The next step is to linearize the rate equation (Eqn. 5.1) around the equilibrium point. The queue equation (Eqn. 5.4) is non-linear as well, however the discontinuity is at the equilibrium point ($q_e = 0$) making it hard to deal with the non-linearity. In this section we first analyze stability assuming the queue evolves as $\dot{q}(t) = y(t) - C$, and in the next section we will see how the discontinuity in real queue changes the stable region. The linearized system (the linearization steps are shown in appendix B) is shown below:

$$\begin{aligned}
\delta\dot{q}(t) &= N\ \delta R(t - d_0) && (5.5)\\
\delta\dot{R}(t) &= -\frac{\alpha}{d_0}\delta R(t - d_0) - \frac{\beta}{Nd_0^2}\delta q(t)
\end{aligned}$$

where $\delta R \doteq R - R_e$ and $\delta q \doteq q - q_e$ are perturbations around the equilibrium point. The Laplace transform of the linearized system (block diagram in Fig. 5.1) is:

$$\begin{aligned}
s\delta R(s) &= -\frac{\alpha}{d_0}e^{-sd_0}\delta R(s) - \frac{\beta}{Nd^2}\delta q(s) && (5.6)\\
s\delta q(s) &= Ne^{-sd_0}\delta R(s)
\end{aligned}$$

[1] We are ignoring the "non-critical" parameter, η. Recall that under over-loaded conditions, η allows us to target a peak link utilization of less than 100%. In all of our simulations and analysis we have chosen η to be 1. If η is chosen < 1 we expect that it will only affect the equilibrium point and not the system stability.

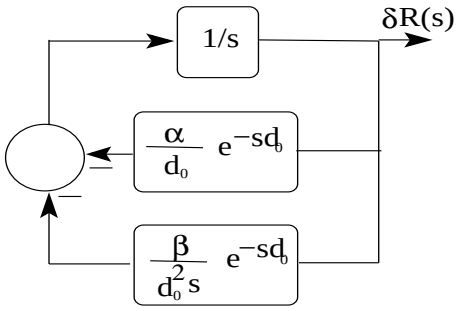


Figure 5.1: Block diagram of the linearized RCP system

5.1.1 Bode Plot and Nyquist Analysis

Using Bode plot analysis, we can obtain the stable region of the system described by Eqn. 5.6. The open loop transfer function is:

$$G(s) = e^{-sd_0}\frac{\alpha s d_0 + \beta}{s^2 d_0^2} \tag{5.7}$$

Note that the variables N and C do not appear in the transfer function, meaning the stability is *independent* of number of flows and link capacity. This is a consequence of RCP's equation having the appropriate scaling with respect to the link rate and number of flows. XCP's equation is also stable independent of C and N. On the other hand, TCP is shown to be unstable as the window size becomes large (i.e., under high bandwidth-delay environments when C is large and N is small [38]). Let's see if there is a region of (α, β) that makes the RCP system stable for *any* d_0.

From Eqn. (5.7) the magnitude and phase of $G(s)$ are given by:

$$\begin{aligned} |G(j\omega)| &= \frac{\sqrt{\beta^2 + (\omega d_0 \alpha)^2}}{(\omega d_0)^2} \\ \angle G(j\omega) &= -\omega d_0 + \arctan(\frac{\omega \alpha d_0}{\beta}) - \pi \end{aligned} \tag{5.8}$$

The stability criterion for the closed loop system is [62]:

$$|G(j\omega)| < 1 \quad \text{at} \quad \angle G(j\omega) = -\pi \tag{5.9}$$

The criterion is true for systems where $|G(j\omega)|$ crosses the magnitude = 1 line only once, which holds for our system. If ω_z is the frequency at which $|G(j\omega)| = 1$ and ω_c is the frequency at which $\angle G(j\omega) = -\pi$, then the criterion is equivalent to $\omega_z < \omega_c$ i.e.

$$\omega_z = \frac{1}{d_0}\sqrt{\frac{\alpha^2 + \sqrt{\alpha^4 + 4\beta^2}}{2}} < \omega_c \tag{5.10}$$

where ω_c is the solution of the equation:

$$\frac{\omega\ d_0\ \alpha}{\beta} = \tan(\omega d_0) \tag{5.11}$$

Unfortunately, there is no closed form solution. We need the following condition for existence of a non-zero solution to Eqn. 5.11 (details in appendix C):

$$\frac{\alpha}{\beta} > 1 \tag{5.12}$$

What remains is to solve for α and β satisfying Eqns. 5.10, 5.11 and 5.12. Using Matlab, the region (α, β) for which RCP is stable for any N, C and d_0 is shown in Fig. 5.2.

A more rigorous way of obtaining the stable region is to use the Nyquist stability criterion [62], the details and steps of which are in appendix D. As Fig. 5.2 shows, the Nyquist criterion confirms the stable region obtained through Bode analysis.

The good news from our analysis is that we can choose RCP's parameters to make the system stable *independent* of link-rate, number of flows and round-trip times; and picking the parameters is easy because the stable range of (α, β) is large.

5.1.2 Stable Region

Let's step back and understand why the stable region looks the way it does. α is the gain parameter that determines how quickly the system ramps up to grab unused capacity, so understandably it cannot be arbitrarily large (in Fig. 5.2 the stable region is capped at $\alpha < 1.6$). On the other hand β determines how aggressively a queue is drained and it isn't clear why the stable region is capped by $\beta \leq 0.55$. On further thought, it is perhaps because the analysis is an approximation and does not take into account the discontinuity in the queue, allowing the queue length to take negative values. This is a standard problem in linearized systems. It is particularly problematic here because RCP's equilibrium point lies on the queue discontinuity.

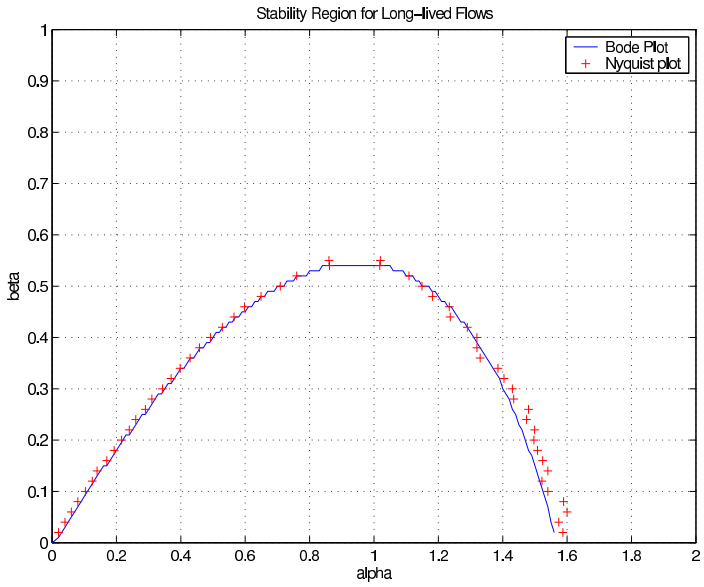


Figure 5.2: Stable region obtained from Bode analysis and Nyquist analysis

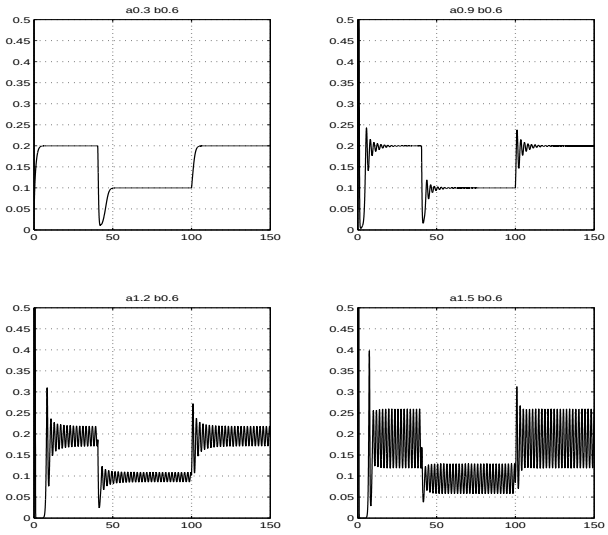


Figure 5.3: Plots showing RCP rate versus time for the following set-up: Five long flows, all with the same round-trip time, start at time, $t = 0$. Their equilibrium rate is $R/C = 0.2$. Five more flows then start at time, $t = 40$, changing the equilibrium rate to 0.1. The original five flows then depart at time 100. This experiment is done for every value of $0.1 \leq \beta \leq 1, 0.1 \leq \alpha \leq 2$ in steps of 0.1, to plot the stable region. The particular plots above show sample values for $\beta = 0.6$, for which the system is stable so long as $\alpha \leq 1.2$.

We hypothesize the actual stable region is larger and includes the one obtained by the linearized model. Initial evidence of this is from simulations, wherein for particular set-ups we obtained the stable region by testing for (α, β) values in the range $0.1 \leq \beta \leq 1, 0.1 \leq \alpha \leq 2$ in steps of 0.1. In fact simulations suggest that the region to the left of the line in Fig. 5.4 is stable. Fig. 5.3 goes on to show an example of a stable point in the non-linear system that is not captured by the linear system.

Packet level simulations can take a long time, so we will use a method known as *phase-plane analysis* (widely used in the study of non-linear systems) to simulate the RCP system for a broader range of network and traffic conditions. The RCP system described by Eqns. 5.1-5.4, can be written as:

$$\begin{aligned} \dot{R}(t) &= f(R(t-T), R(t-d_0), q(t), N, C) \\ \dot{q}(t) &= g(R(t-d_0), N, C) \end{aligned} \tag{5.13}$$

where $f(.)$ and $g(.)$ are non-linear functions. If we take R and q as coordinates of a plane, then to each state of the system there corresponds a point in this plane. As t varies, this point describes a curve in the R-q plane, indicating the history of the system dynamics. Such a geometrical representation of the system behavior in terms of trajectories is called a phase-plane. The initial condition determines the initial location of a representative point on the trajectory, and as time progresses, the representative point moves along the trajectory. A family of such trajectories is called a phase-plane portrait. We can determine whether a system is stable from the phase portraits.

We simulated the system for $C = \{56$ Kbps, 0.15 Gbps, 2.5 Gbps, 10 Gbps, 1000 Gbps$\}$, $d_0 = \{0.01, 0.5, 1, 2\}$ seconds, and $N = \{10, 100, 1000, 5000\}$, and in every case we found the larger region shown in Fig. 5.4 holds good for stability. Appendix E has examples of phase-plane plots for RCP.

5.2 Stability of non-linear system

The success of linear analysis depends on how well the system dynamics can be approximated by its first-order behavior about the equilibrium point. In particular, when the equilibrium point lies on a discontinuity in the system dynamics (which it does for RCP and XCP because their equilibrium queue size is zero), stability of the linearized system

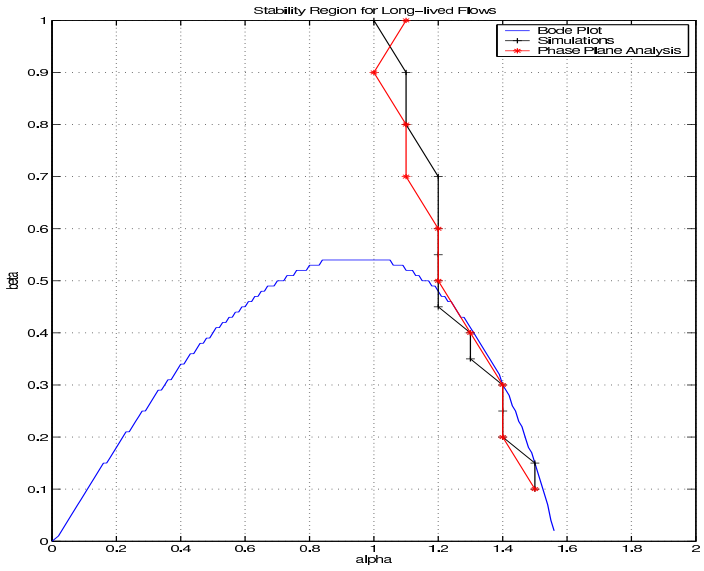


Figure 5.4: Stable Region obtained from linearization model (enclosed by the bell shaped curve), Simulations and Phase Plane analysis (region to the left of solid lines).

gives *no* guarantees on the stability of the system, even for simple network topologies with a single bottleneck links.

When we compare stability of the simulated switched RCP system (the system switches behavior at $q = 0$ and is represented in Eqn. 5.14; henceforth referred to as *SYSTEM 1*) with that obtained through linear analysis (represented in Eqn. 5.15), Fig. 5.5 suggests that a potentially larger region, that to the left of the dotted line is stable [63]. If we simulate both the linearization and the switched systems for two sets of parameter values ($\alpha = 0.8$, $\beta = 0.55$ and $\alpha = 1.4$, $\beta = 0.3$), we notice that the first set of parameters results in a stable system; while for the second set of parameters, linear analysis predicts a stable system, while simulations indicate that the system is unstable [64]. The considerable difference in the stable region predicted by linearization and the actual stable region motivates a more careful analysis. We notice that *SYSTEM 1* is a switched system with two modes of operation, one when the queue length is positive, and one when the queue length is zero, and that the equilibrium point lies on the line ($q(t) = 0$) when switching between the two systems occurs. It is known that for a switched system, linearizing about an equilibrium point at which the system dynamics are discontinuous could lead one to erroneous conclusions, even

on its local stability [64].

$$\begin{aligned} \dot{y}(t) &= -\tfrac{\alpha}{d_0}(y(t-d_0)-C) - \tfrac{\beta}{d_0^2}q(t-d_0) \\ \dot{q}(t) &= \begin{cases} y(t)-C, & q(t)>0 \\ \max(0, y(t)-C), & q(t)=0 \end{cases} \end{aligned} \tag{5.14}$$

$$\begin{aligned} \dot{y}(t) &= -\tfrac{\alpha}{d_0}(y(t-d_0)-C) - \tfrac{\beta}{d_0^2}q(t-d_0) \\ \dot{q}(t) &= y(t)-C. \end{aligned} \tag{5.15}$$

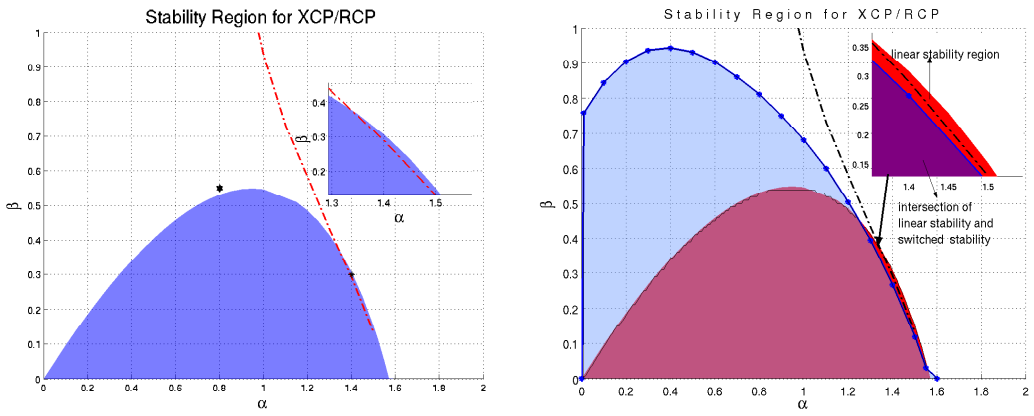


Figure 5.5: (Left) Comparison of linearized stability region (shaded area) with simulated stability region (area to the left of the dotted line), for the *SYSTEM 1*. (Right) Provably safe regions of α and β (for $d_0 = 200$ ms).

Ref. [63] proposes a method for taking discontinuities in the system dynamics into account by modeling the protocol as a switched system. The approach involves choosing particular Lyapunov functions that prove the stability of switched linear time-delay systems. We present the key results here along with a few examples below, the details of the analysis are in [64, 63].

The outer boundaries of the provably stable regions of parameters for a round-trip delay of 200 ms are plotted in Fig. 5.5 (right). The smaller (dark) region corresponds to the stable region predicted by linear analysis, which ignores the switch. The inset shows a closer look at the region where the switched Lyapunov results are conservative (which is to be expected, since they are derived from a sufficient condition for stability) – while the linear analysis results predict a stable system, the switched system is unstable. Fig. 5.5 (right) also shows

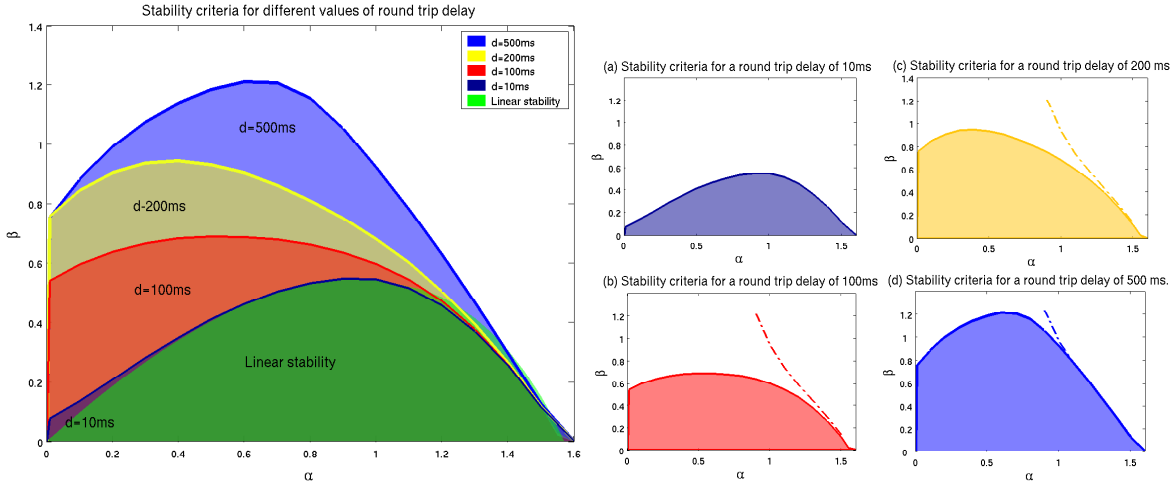


Figure 5.6: Provably safe boundaries of α and β (for d_0 = 10 ms to 200 ms). The dotted lines in the figures on the right correspond to the simulated stability boundaries.

that the actual stable region is much larger than that predicted by the linearization.

The proposed Lyapunov functionals for switched systems provide us with sufficient conditions for delay-dependent stability. Since studies have shown that 85% of Internet traffic has round-trip delays between 15-500 ms., we analyze the stability for this range of round-trip delays. The provable stability boundaries, in terms of α and β are shown in Fig. 5.6. We find that for small delays, it is more difficult to prove the stability of the switched system. We should bear in mind that these results are based on sufficiency conditions, and therefore our not being able to prove stability does not imply instability. For values of delay more than 100 ms, we can prove stability for a substantially large range of parameters. Even for small values of delay, we note that the region stays larger than previously derived using linearization. These techniques can also be extended to the case of heterogeneous delays on bottleneck links [64].

5.3 Stability under multiple bottlenecks and heterogeneous round-trip times

So far we have seen RCP's stability in the case of a single bottleneck link. But a real network often carries flows with vastly heterogeneous round-trip times sharing multiple bottleneck

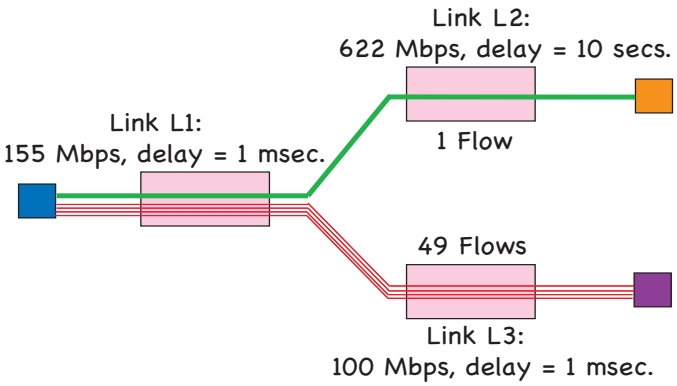


Figure 5.7: A configuration where flows with heterogeneous round-trip times have different bottleneck links. There are 50 flows, of which 49 flows with short round-trip times of 4 ms. are bottlenecked at $L3$, while the 50^{th} flow has a long round-trip time of 20 seconds and is bottlenecked at $L1$.

links. We do not have a proof of general stability region for arbitrary networks, however simulations of specifically chosen topologies and RTTs suggest that the same stable region as we had seen for the single bottleneck holds true in the case of a general network.

Fig. 5.7 illustrates an example which is intuitively one of the worst case scenarios for RCP's stability [65]. Of the three links, $L1$ and $L3$ have a propagation delay of 1 ms while $L2$ has a delay of 10 s.[2] There are 50 flows, 49 with short round-trip times of 4 ms are bottlenecked at $L3$, while the 50^{th} flow has a long round-trip time of 20 seconds and is bottlenecked at $L1$. The situation is interesting because of what's happening at $L1$: the link is shared by flows with four orders of magnitude difference in RTTs with most traffic from short RTT flows. These flows are not bottlenecked at this link but nevertheless affect the router's estimate of average round-trip time (d). The control dynamics are that of the long RTT flow which is bottlenecked here but its traffic is not a significant portion of the link. It turns out the same stable region we have seen in the last section for single bottleneck with homogeneous delay still holds true. Figs. 5.8, 5.9 show examples of some extreme points in the stable region. For the sake of brevity only the results with the extreme parameter values are included, and the same property holds true for all the in-between α, β values as well.

RCP's RTT estimation algorithm (first described in Sec. 3.5 and recapped below) plays a

[2]It is unlikely to have paths with such high propagation delays, however if there is a lot of buffering then it is possible for delays to be this high.

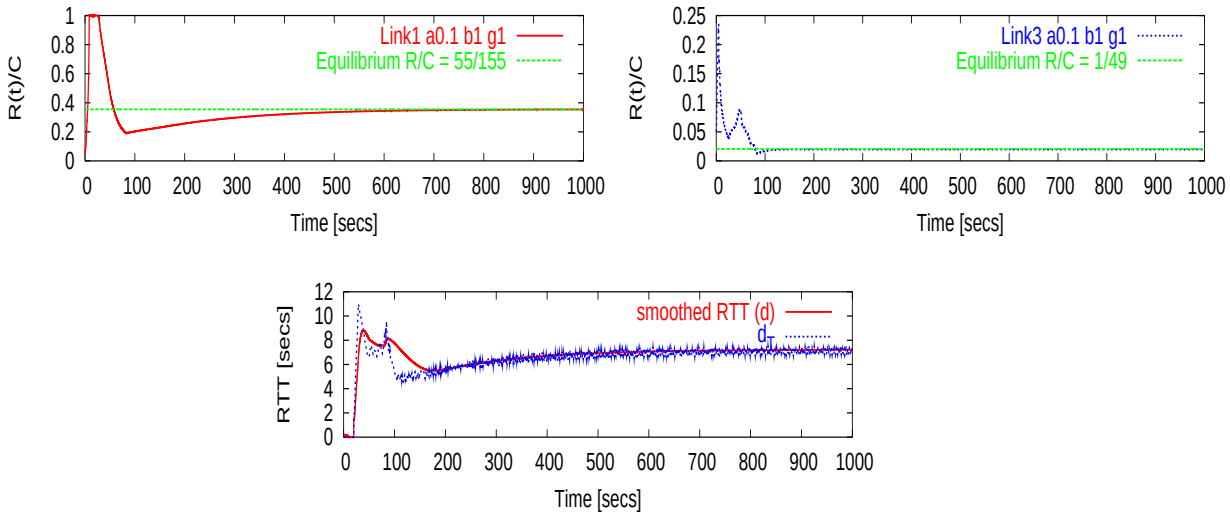


Figure 5.8: Plots showing the RCP rate at link $L1$ (top left) and $L3$ (top right) and average RTT estimate at the $L1$ (bottom) in the configuration of Fig. 5.7. RCP parameters: $\alpha = 0.1, \beta = 1, \eta = 1$.

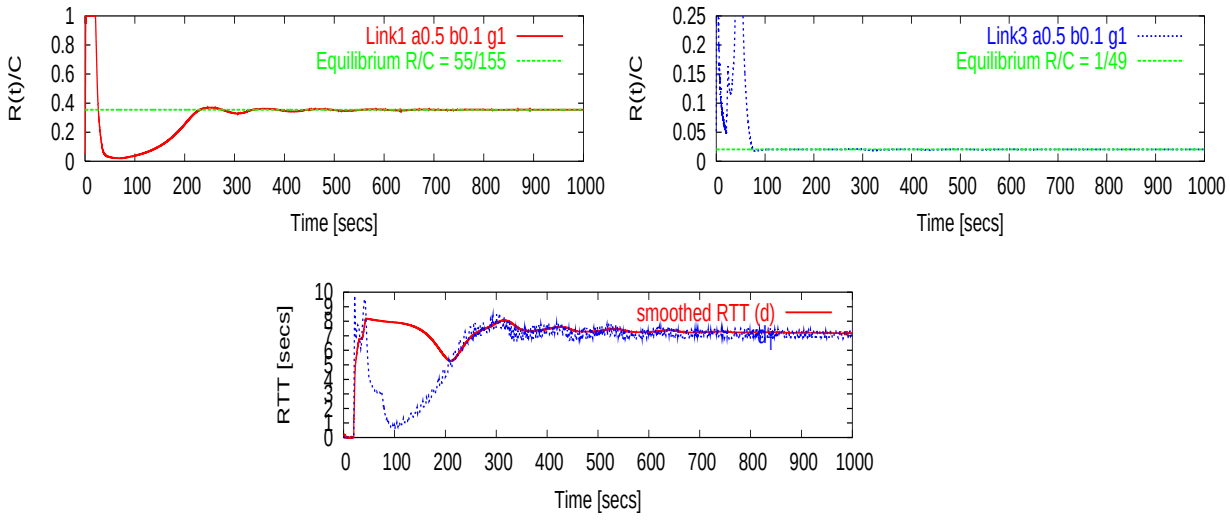


Figure 5.9: Plots showing the RCP rate at link $L1$ (top left) and $L3$ (top right) and average RTT estimate at the $L1$ (bottom) in the configuration of Fig. 5.7. RCP parameters: $\alpha = 0.5, \beta = 0.1, \eta = 1$. Notice the difference in the smoothed RTT estimate (d) versus the average RTT taken for every time interval, d_T.

key role in maintaining stability under heterogeneous RTTs. RCP's original RTT estimation algorithm ($d = \theta \cdot rtt_p + (1-\theta) \cdot d$, where θ is the fixed gain of the moving average and rtt_p is the round-trip time sample carried in the RCP packet) lead to instability in the system under heterogeneous delays because the RTT estimate itself was unstable. A more robust way of maintaining an RTT estimate has two parts:

1. The data-path of the RCP router simply averages the RTT over all packets seen in the control interval:

$$d_T = \frac{\sum_i rtt_i}{n_T} \tag{5.16}$$

   where d_T is the rtt average over time-interval T, rtt_i is the RTT value carried in the packet header and n_T is the number of packets carrying a valid RTT.

2. The control path (which performs the RCP rate computation periodically) takes d_T as input and keeps track of a smoother RTT estimate, d. In particular, if d_T is much smaller than the smoothed version (d), it is better to age d and bring it down slowly instead of dropping it suddenly. This is achieved by deciding the moving-average gain as follows:

   if $(d_T \geq d)$

   $\theta = \frac{T}{d}$

   else

   $\theta = \frac{R}{C} \cdot \frac{T}{d} \cdot \frac{d_T}{d}$

   The smoothed RTT is updated as:

$$d = \theta \cdot d_T + (1-\theta) \cdot d \tag{5.17}$$

The basic intuition is that the gain should be at most T/d since if T is small we have many more RTT samples over a period of an average RTT, so we correspondingly give a smaller weight to each of the samples. However, if the rtt sample (d_T) is smaller than d we want to be cautious in decreasing d suddenly, and so the gain is made smaller by weighing it with d_T/d.

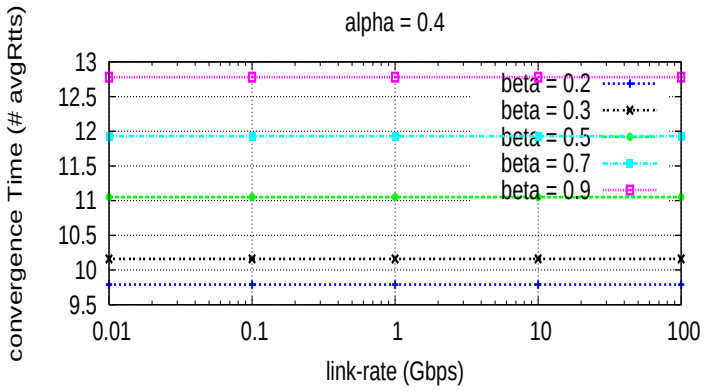


Figure 5.10: Plot shows the convergence times of RCP versus link-rates. Convergence time is defined as the time taken for R and q to be within 98% and 99% of their equilibrium values. Simulations were done for $0 \leq \alpha, \beta \leq 1$ in increments of 0.1 for link-rates varying over four orders of magnitude. The plot shows that the link-rate does not affect the RCP convergence time.

5.4 Picking values for α and β

Now that we know there is a broad range of (α, β) values for a stable system, we want to pick values to maximize performance. To see what we mean by this, let's consider the following two distinct cases:

Case 1: Fast convergence times on sudden network changes

We are interested in the following question: *How long does RCP take to converge for the (α, β) values in its stable region?*

The RCP equation is designed to scale well with the link-rate, number of flows, and round-trip time, so it is perhaps not surprising that its convergence time to equilibrium is *independent* of the link-rate (Fig. 5.10), the number of flows (Fig. 5.11) and value of the round-trip time (Fig. 5.12). It does however depend on α, β, as well as on the amount of buffering at the bottleneck link.

Fig. 5.13 shows how long it takes RCP to converge when α and β vary. RCP converges slowly if α is too small or too big. When small, RCP is slow to use spare capacity; when large, the rate overshoots many times before settling down on the equilibrium. The best values for quick convergence will be the range: $\alpha \in (0.4, 0.6)$ and $\beta \in (0.2, 0.6)$, for example when $(\alpha, \beta) = (0.5, 0.5)$, the system converges within 10 round-trip times.

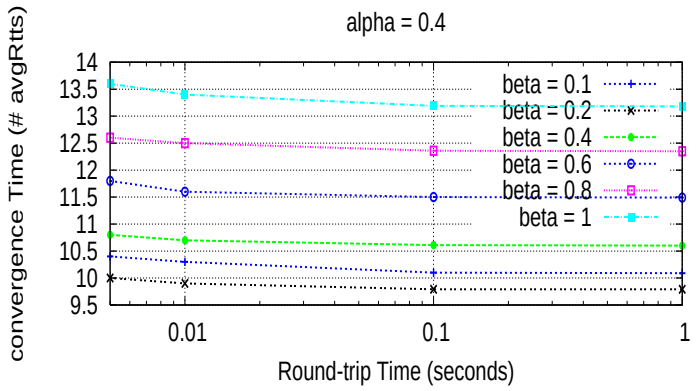


Figure 5.11: Plot shows the convergence time of RCP versus increasing round-trip time. The convergence time is defined as the time taken for R and q to be within 98% and 99% of their equilibrium values. The plot shows that the RTT value does not affect the RCP convergence time (measured in terms of #RTTs).

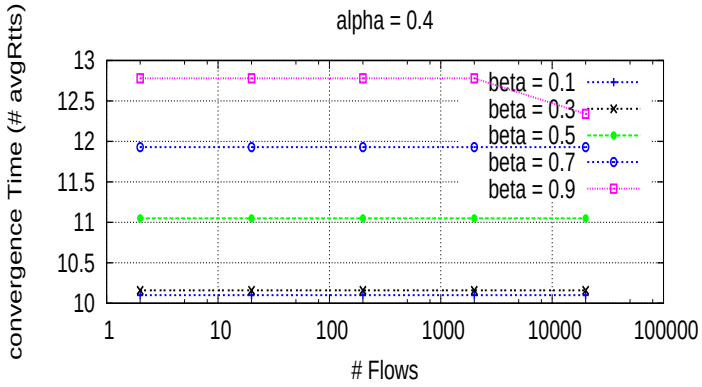


Figure 5.12: Figure showing the convergence time (in #RTTs) of RCP versus the number of flows. In each case, the initial conditions correspond to half the equilibrium load, for example: when $N = 200$, the equilibrium is $R_e/C = 0.005, q_e = 0$ and the initial conditions are chosen as $R(0)/C = 0.01, q(0) = 0$. While the number of flows is different in every experiment, the increase in load is the same in each case (equilibrium load is twice the initial load). Convergence time is defined as the time taken for R and q to be within 98% and 99% of their equilibrium values respectively.

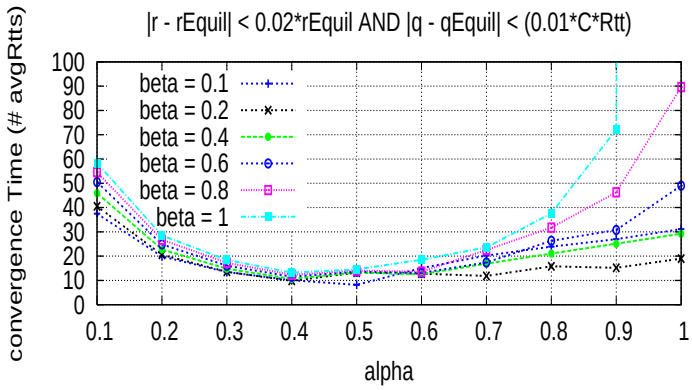


Figure 5.13: Plot showing the convergence time of RCP for different α and β values.

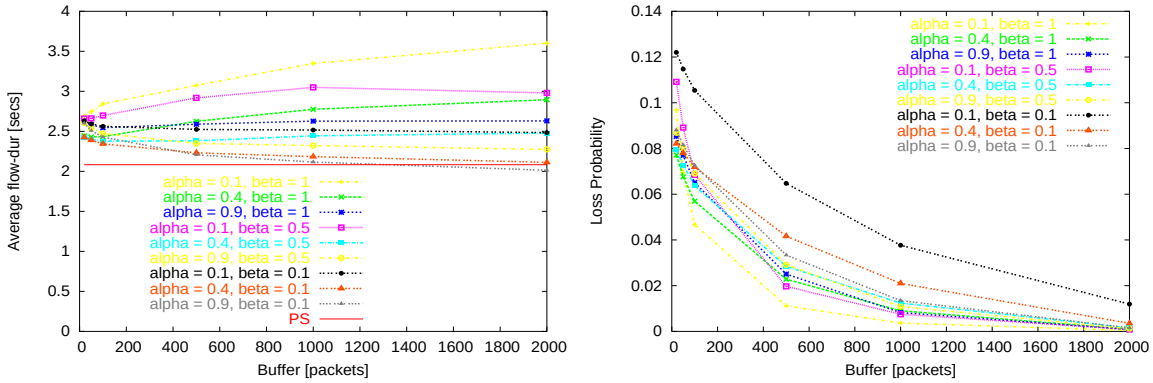


Figure 5.14: Plot illustrating the effect of α and β on FCT and Loss probability when flow-sizes are large. The set-up is: $C = 0.1$ Gbps, RTT $= 80$ ms, $\rho = 0.68$, mean flow-size is 10000 packets (1000% BW-RTT). Buffer size is varied from 2% to 200% of bandwidth-delay.

Case 2: Fast flow completion times for a broad range of traffic conditions

We want to know if the same values of α and β for fast convergence will also work well from the view point of how close RCP's flow completion times are to processor sharing. It turns out the optimal (α, β) depends on how the mean flow-size compares with the bandwidth-delay product or the pipe size. When the flow-sizes are large (Fig. 5.14) it is best to choose a large α and small β (example: when $\alpha = 0.9$ and $\beta = 0.1$), as long flows care more on maximizing their individual throughputs and less on minimizing the queuing delay. On the other hand, a small α and large β (example: $\alpha = 0.1$ and $\beta = 1$) work best for small flow-sizes (Fig. 5.15), because this combination helps in keeping their queuing delay small.

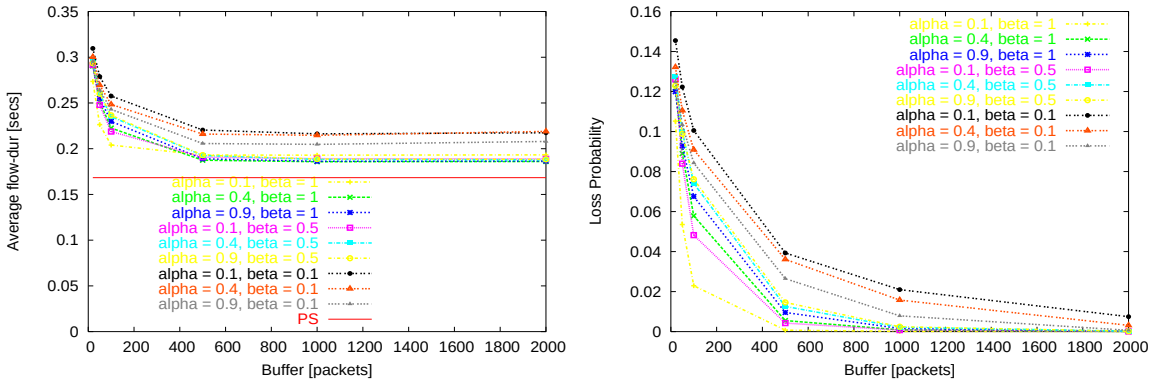


Figure 5.15: Plot illustrating the effect of α and β on FCT and Loss probability when flow-sizes are small. The set-up is: $C = 0.1$ Gbps, RTT $= 80$ ms, $\rho = 0.8$, mean flow-size is 20 packets (2% BW-RTT). Buffer size is varied from 2% to 200% of bandwidth-delay.

To strike a middle-ground between the two extreme set of parameters, we recommend choosing the values of $\alpha \in (0.4, 0.6)$ and $\beta \in (0.2, 0.6)$.

Chapter 6

Practical considerations in building an RCP network

We want to enable deployment of RCP in real networks. In prior chapters we studied RCP through simulations and modeling; in particular we described the motivation behind RCP and why flow completion time is the appropriate metric for congestion control (Chap. 2), the RCP protocol, mechanisms, and algorithm (Chap. 3), simulations suggesting it is promising under a broad range of conditions (Chap. 4), and control theoretic analysis to show that RCP is stable independent of link-capacities, number of flows and network round-trip times (Chap. 5). Although we have many thousands of promising *ns*2 simulations, we want to find out how feasible it is to deploy RCP in real networks.

Deploying RCP requires solving many practical problems, such as: How is RCP congestion information carried in presence of tunneling? What happens when the link-rate is not a constant any more and instead varies with time and/or is unknown (e.g., in wireless links)? How does RCP interact with middle-boxes, such as NATs and firewalls? How complex is it to implement RCP? How can it be incrementally deployed? And how much buffering would it require in routers and switches? Our goal is to enable a network, similar to that shown in Fig. 6.1, where RCP coexists along-side legacy protocols, mechanisms and middle-boxes. In this chapter we are interested in the last three questions: How is RCP implemented in real systems and what is the additional complexity it introduces into end-hosts and routers? How does RCP coexist in a network where a significant portion of traffic is non-RCP and a large portion of queues do not yet implement RCP? How do we size router buffers for RCP congestion control?

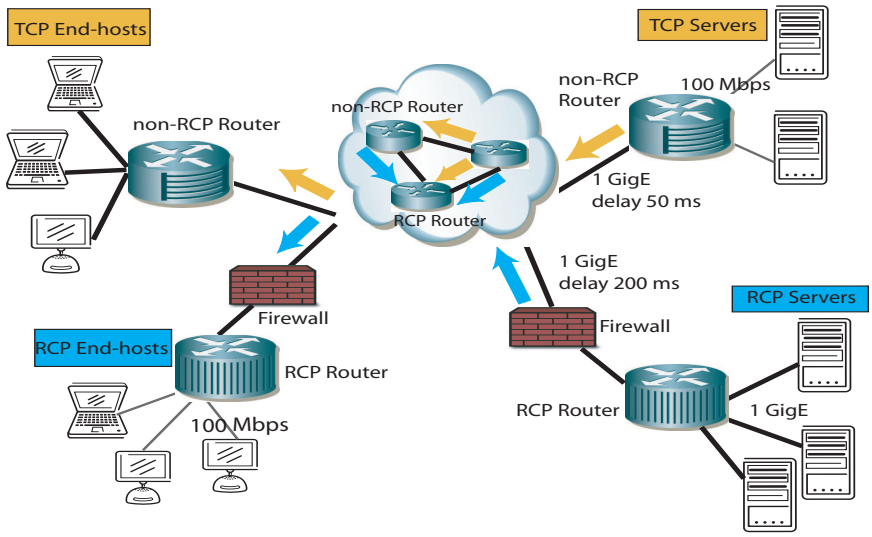


Figure 6.1: An example network where RCP coexists alongside with non-RCP traffic and a variety of network devices like firewalls and NATs, not all of which necessarily understand RCP.

We are particularly interested in how complex the changes are to the routers. Router vendors are, understandably, very reluctant to add new features to the forwarding path of their routers, particularly if they involve complex calculations. Routers are already overloaded with many features, and are limited by the power they consume. Great care needs to be given to bloating the requirements further. So in Sec. 6.1, we describe our implementations of RCP for end-hosts and routers, and try to analyze the additional complexity in routers.

Any research on congestion control that requires network participation will be irrelevant if we cannot find a viable deployment path. Because the Internet is not controlled by any single entity and there will be no fork-lift upgrades, we want to find out how RCP can be deployed even when a significant number of network routers are not RCP-enabled and a large portion of traffic is non-RCP. In Sec. 6.2 we describe heuristics, of varying degrees of complexity, that enable RCP flows to share bandwidth equitably with non-RCP flows, and that allow RCP flows to detect non-RCP bottlenecks and fall back to TCP congestion control. And finally, in Sec. 6.3, we characterize the amount of buffering that routers will require for RCP.

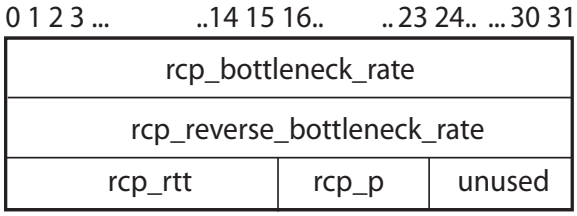


Figure 6.2: The 12-Byte RCP header: the *rcp_bottleneck_rate* (4 Bytes) carries the rate (in Bytes/ms) of the most congested link along the path; *rcp_reverse_bottleneck_rate* (4 Bytes) is the bottleneck rate (in Bytes/ms) echoed by the receiver, so the sender can adapt its rate; *rcp_rtt* (2 Bytes) is the sender's estimate of its round-trip time (in ms); *rcp_proto* (1 Byte) is the protocol number of the higher transport layer.

6.1 Implementing and experimenting with RCP

We will first describe Linux-based implementations of RCP end-host and router, then a hardware implementation of RCP router on Stanford's NetFPGA system [68], analyze the additional complexity that RCP introduces into end-hosts and routers, and finally describe experiments that validate the implementation.

The descriptions of Linux and NetFPGA based implementations first appeared in [18], and the experiments in [19].

6.1.1 RCP End-host

We have implemented the RCP end-system in Linux 2.6.16. An RCP sender maintains a congestion-window which it modulates based on explicit feedback information from the network. It also maintains a round-trip time estimate of the path, and spreads the transmission of a window's worth of packets (or "paces") over a RTT. An RCP receiver echoes the network rate feedback it receives to the sender by piggybacking it in the DATA/ACK packets flowing from receiver to sender. We describe below the key pieces of an RCP end-system.

Placement and Format of RCP Header

RCP is implemented as its own protocol layer between IP and transport layers, as shown in Fig. 6.3. Other places to carry RCP information would be IP or TCP options, each

having its pros and cons.[1] The advantages of having RCP as a shim layer between IP and transport are: a) routers which don't understand RCP will let RCP packets pass through, and b) the RCP rate information can be used by any transport protocol including TCP for file-transfers, as well as by UDP for streaming content.

Fig. 6.2 illustrates the 12-Byte RCP congestion header, with the following four fields:

1. *rcp_bottleneck_rate* carries the bottleneck rate of the most congested link along the path. The RCP sender puts in a zero here - meaning the sender would like as high a rate as the network gives it. Alternatively, it can also put in the local interface rate. The routers overwrite this rate as the packet passes through the network.

2. *rcp_reverse_path_bottleneck_rate* is filled by RCP receiver to communicate the bottleneck rate to an RCP sender.

3. *rcp_rtt* carries the sender's round-trip time estimate and is used by routers to update their traffic-averaged RTT estimate.

4. *rcp_p* is higher layer protocol number such as that of TCP or UDP.

All rates are expressed in Bytes/ms, and RTT in ms.

RCP End-host Functions

This section describes the RCP implementation at sender and receiver end-hosts. We will focus here on the case of TCP transport protocol running over RCP. Fig. 6.3 shows the placement of RCP in the network stack; there are two parts: the RCP layer between transport and IP layer, which carries congestion information from network to the end-system, and the congestion control component in transport layer which adapts the flow-rate based on network feedback. One can think of congestion control consisting of two broad parts: a) modulating the flow-rate (and congestion window), and b) deciding which packets to send among the three pools of packets—those which have not yet been transmitted, those which have been sent but not yet acknowledged, and finally packets which are known to be lost.

[1] One could argue that it belongs in any layer involved in packet-switching—in L2, L3, or between the two. Recent studies [69] have shown that 70% of connections are not established when SYN segments have a new IP option X, and a third of connections are not established even for known IP options. Further, packets with IP options take the slow-path on routers. TCP options on the other hand are more widely used and the same study showed only 0.2% of connections failed on introduction of new TCP option. The downside is that routers would need to modify TCP header and be aware of every new transport protocol that uses RCP information.

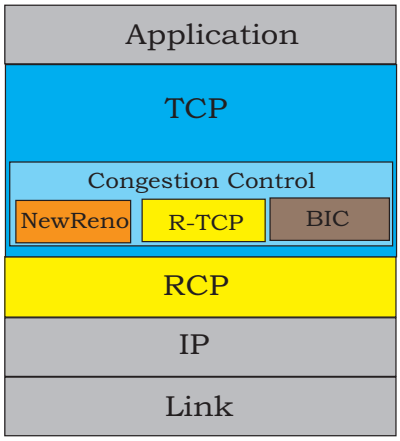


Figure 6.3: RCP is a protocol between the IP and transport layers.

RCP only modifies the first of these functions in TCP, i.e., modulating the flow-rate, and we call this part *R-TCP*. Starting from Linux 2.6.13, the TCP code was re-written to make it more modular [70], as a result of which the specific TCP congestion control mechanism, e.g., BIC TCP, HTCP, Scalable TCP, HighSpeed TCP can be chosen dynamically either using sysctl or on a per-socket basis. *R-TCP* can also be chosen dynamically. The rest of TCP functionality such as the state-machine and mechanisms for in-order packet delivery remain unchanged.

The sender maintains the following variables: a) bottleneck rate of the forward path, b) bottleneck rate of the reverse path, c) round-trip time estimate for the current path, and d) the packet pacing interval. These are maintained in TCP's *tcp_sock* structure. The sender fills in RCP fields of an outgoing packet: a) sender's desired throughput, *rcp_bottleneck_rate*, which can be the speed of the local interface, b) bottleneck rate of the reverse path, *rcp_reverse_bottleneck_rate*, which is zero if the host is not aware of the rate yet, c) round-trip time estimate, which is zero for the first packet of the connection (SYN) when the sender does not have an estimate yet, and d) the protocol number of TCP. Fig. 6.4 (left plot) shows the paths that different packets take from TCP to the lower RCP layer. SYN, RESET, DATA, and ACK packets take different paths, and RCP intercepts all calls from TCP to IP, to attach the 12-Byte RCP header. The incoming RCP packets from the network have only one path up the stack, where RCP intercepts the function calls from IP to TCP, to strip off the 12-Byte header and pass the segment to TCP, as shown in Fig. 6.4 (right plot).

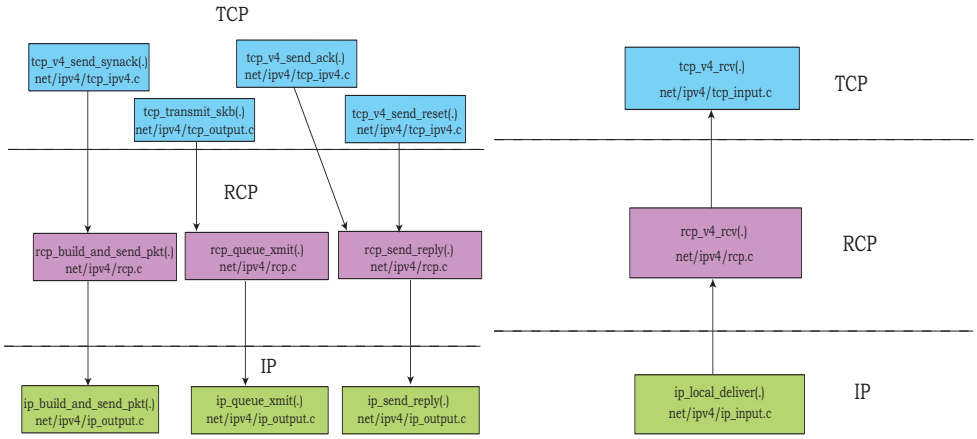


Figure 6.4: Left plot shows data path of outgoing packets, where RCP intercepts the function calls from TCP to IP, to introduce its 12-Byte RCP header and fill in the rate and RTT fields. Right plot shows data path of incoming packets, where RCP intercepts the function calls from IP to TCP, to strip of the 12-byte header and pass the segment to TCP.

An RCP receiver echoes the network rate feedback to the sender by copying the *rcp_bottleneck_rate* value into *rcp_reverse_bottleneck_rate*, and usually piggybacking on DATA/ACK packets. For a pure ACK packet, the bottleneck rate and RTT fields are set to zero.

On receiving valid rate feedback, an *R-TCP* sender modulates its congestion window as shown below, overriding the existing TCP slow-start and congestion avoidance window changes:

```
snd_wnd = (rcp_bottleneck_rate * rcp_rtt)/(MSS + RCP_HEADER_SIZE
                                              + IP_HEADER_SIZE)
```

where MSS is the maximum segment size. Just as other flavors of TCP, *R-TCP* uses a window which is the minimum of the above window calculation and window size advertised by the receiver. It also maintains the window size to be least equal to one Maximum Segment Size. *R-TCP* keeps track of the smoothed round-trip time estimate for the connection.

The sender paces packets from TCP's send queue at the following pacing interval:

```
packet_pacing_interval = MSS/rcp_bottleneck_rate;
```

The retransmission mechanisms of TCP are left unchanged.

6.1.2 RCP Software Router

A software router is one that is completely implemented in software (e.g. in Linux), and requires no special hardware support. This section outlines the implementation of an RCP router in Linux, based on the RCP description in Chap. 3, and specification in App. F. Such an implementation allows us to easily verify the properties of RCP, as well as demonstrates the feasibility and simplicity of supporting RCP within a software-based router. We begin by describing the operation of a non-RCP router design before elaborating on our RCP design.

Vanilla router functionality

The operation of a router [81] can be subdivided into two parts – the data path and the control path.

The data path processes incoming packets and routes them towards their destination. Tasks performed on each packet include verifying the IP header checksum, extracting the destination address from the IP header, performing a longest prefix match look-up of the address in the routing table, decrementing the packet's time-to-live, updating the checksum, and forwarding the updated packet out the correct interface. High-performance Internet routers implement the data path in fast hardware since it needs to process every packet.

The control path is responsible for a set of tasks that are performed infrequently such as maintaining the routing tables and providing a control interface to the router. Since operations on the control path occur relatively infrequently, they are usually implemented in software and executed on a CPU inside the router.

RCP router enhancements

An RCP-enabled router must additionally compute the fair-share rate (as per Eqn. 3.3) and stamp that rate into the header of every RCP packet. The rate computation requires the router to maintain an average round-trip time estimate for outgoing traffic on each interface using the RTT information carried in RCP packets. The rate is computed once every control interval, approximately once per round-trip time. Statistics (aggregate incoming traffic and average RTT) are gathered during each control interval which are then used for the rate computation. When an RCP packet arrives, the router adds RTT value of packet header to the running sum it maintains and before departure the packet is stamped with the RCP

rate.

The RCP functionality is split between the data path and the software control path. The additions to the data and control paths are summarized below:

1. Per-packet data-path processing:
   - Identifying whether an incoming packet is an RCP packet
   - Updating a running RTT sum of the outgoing interface, if the packet carries a valid RTT
   - Updating the aggregate traffic destined to the outgoing interface
   - Stamping the RCP rate in the outgoing packet
2. Periodic control path computations:

   The following are calculated approximately once per average RTT of traffic transiting the router:
   - The bandwidth, $R(t)$, allocated to the average data flow as per Eqn. 3.3.
   - The moving round-trip time average, as detailed in Chap. 3.

We run our RCP router implementation on a standard Linux system, implemented as a Linux Kernel Module (LKM), namely *rcp-router-driver.ko*. This approach avoids the complication of applying patches to Linux source distribution and recompiling the whole Linux kernel.

The control plane is a timer driven function to compute the RCP rate, moving RTT average on each outgoing interface, and the next wake-up interval for this timer. The timer is maintained per network interface. The Data plane is built based on Linux's NetFilter feature, which allows customized per-packet operations in the packet processing chain of the kernel. RCP requires only a small amount of per-packet processing—in the worst case 3 integer additions, 2 comparisons, and 1 write operation. No multiplications or divisions are performed on the data path. Data plane operations on Ingress and Egress path are described below:

- The Ingress function is registered with the *IP_FORWARDING* hook in NetFilter. When a packet arrives at the NIC driver and is destined to one of the outgoing interfaces, this function updates the running RTT sum of the outgoing interface (if the packet carries valid RTT); it also updates the aggregate traffic rate to the outgoing interface.

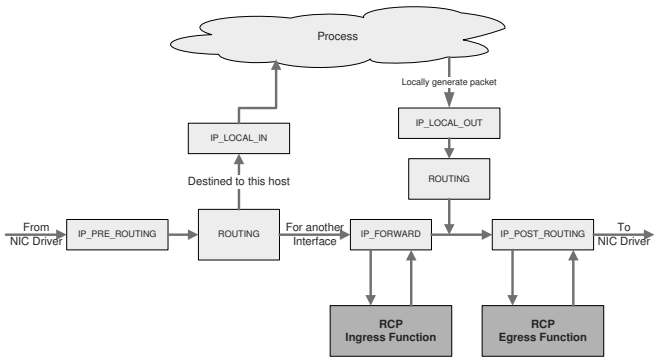


Figure 6.5: Linux NetFilter: Packet processing paths and the points where RCP is hooked.

• The Egress function is registered with the *IP_POSTROUTING* hook in NetFilter. When a packet is ready for one of the outgoing interfaces, this function stamps the RCP rate in the header and updates the TX queue occupancy for that interface.

Fig. 6.5 shows the packet processing chain in NetFilter and where exactly the RCP functions are hooked up.

Validation Experiments

To verify the RCP implementation, we conducted our experimental study on a real test network under different topologies and number of nodes [19]; we will restrict our discussion here to the simple set-up in Fig. 6.6. We built the RCP software router from a regular desktop PC (AMD Athelon 64 X2 Dual-Core 3600+ and 1 GB of DRAM) and a 2-port Gigabit Ethernet card. The end-hosts are built from similar desktop PCs running a Linux kernel with our RCP end-host modifications.

The RCP router in kernel mode can forward MTU size packets at 1 Gbps. However, as the packet size becomes smaller, the router cannot keep pace with interrupt overheads due to limited FPS (frame per second) processing power on this platform. To avoid such performance bottlenecks, we used lower link speeds across the topology: on one side of the RCP router, for the 192.168.1.0/24 network segment, we used a 100 Base-T switch to connect end-hosts and a router, while for the 192.168.2.0/24 network segment, we used a 10 Base-T Ethernet switch. The bottleneck is the 10 Mbps TX queue at the RCP router's Eth1 interface. The RCP kernel module operates at this interface to regulate flow throughput. We set η at 0.9, which means the maximum rate an RCP router can advertise is 9 Mbps. Taking

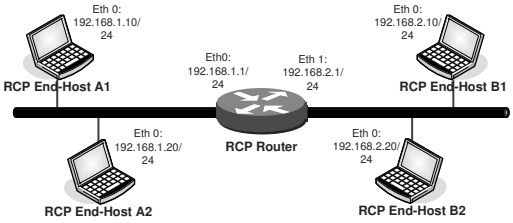


Figure 6.6: A simple topology used for experiments

into account other protocol and application overhead, the maximum iperf[2] throughput on this link is about 7.5 Mbps.

The end-to-end delay is emulated using NetEmulation [72] on router's Eth1 interface, through which we can vary the delay, packet loss-rate or even packet corruption rate. We set the end-to-end delay to 50 ms. On the end-host PCs, we use an iperf [71] client to generate multiple continuous traffic flows and iperf server as the traffic sink. Statistics are gathered at both the end-host PCs as well as at RCP router.

The RCP system is deterministic, and so with a fixed set of initial conditions (initial rate-value $R(0)$, parameters of the algorithm, and initial link state such as an empty queue) and a deterministic input traffic pattern (fixed start times of flows and the amount of data they transfer), the system output will always be the same.

The first step of validation is to determine if the RCP rate, $R(t)$, converges to C/N, as the number of ongoing flows, N, varies and the congestion window on the end-host can be modulated accordingly.

The experiment is as follows: End-hosts A1, A2, A3 and A4 start an iperf flow of length 100 s at times 0, 20, 40, 60 respectively, destined to the iperf sink on host B. At $t = 0$, A1 begins with a high RCP rate in the TCP SYN packet (the interface capacity is configured to be 100 Mbps). Since the router's bottleneck queue can only handle 8 Mbps, it overwrites this rate with a smaller value. At $t = 20$, A2 also begins with 100 Mbps in its TCP SYN packet, which is overwritten with the current RCP rate (8 Mbps). The new flow arrival creates a sudden backlog on the Eth1 TX queue at $t = 20$ s (Fig. 6.7), in response to which RCP reduces $R(t)$ to about half the bottleneck rate. The dynamics are similar at $t = 40$, 60, when more new flows join in. The flows depart at $t = 100$, 120 and 140, freeing up a fraction of the bottleneck link and causing the TX queue length to drop suddenly. The

[2]iperf is a tool to measure TCP and UDP throughput performance.

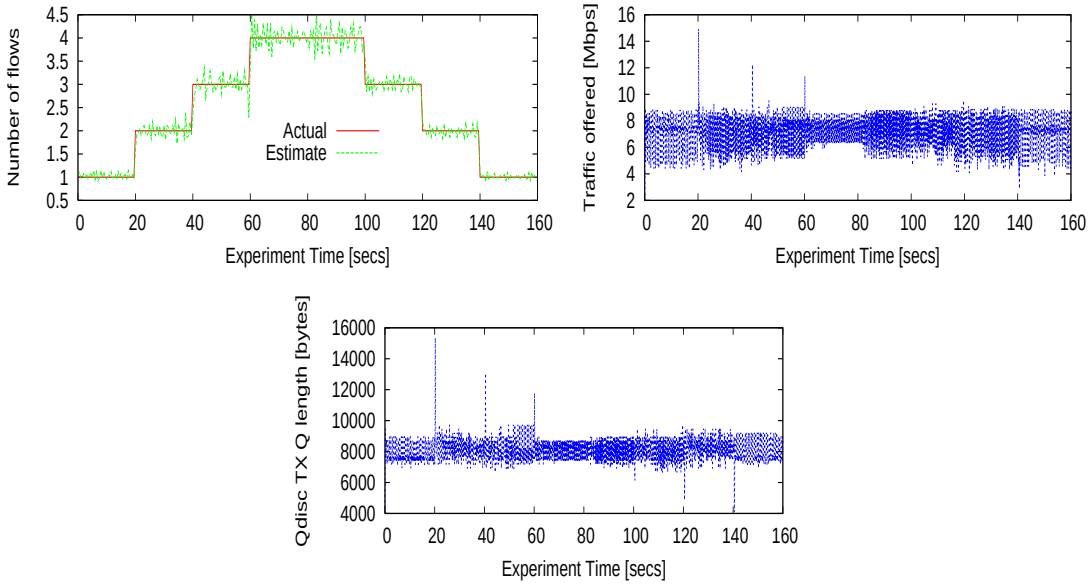


Figure 6.7: Top left plot: Comparing RCP's estimate of the number of flows, $C/R(t)$, with the true value. Top right plot: Aggregate incoming traffic at router's output port. Bottom plot: Qdisc TX Queue length on outgoing interface of RCP router.

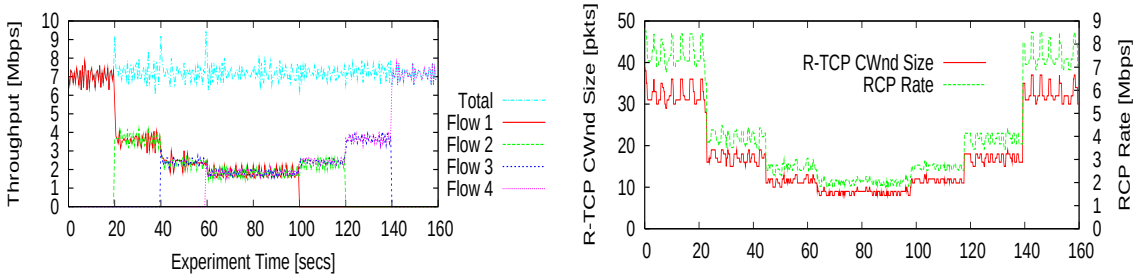


Figure 6.8: Left plot: Per-flow and aggregate throughput. Right plot: R-TCP modulates its congestion window based on RCP's rate feedback.

RCP control plane increases the rate such that $R = C/N$ still holds. Fig. 6.7 compares the actual number of flows in the system with the N that RCP estimates. Fig. 6.8 shows how the end-host A1 modulates its *cwnd* in response to the rate feedback information.

We have verified several key properties of RCP end-hosts and router implementations [19]. In the process we faced a few technical challenges that we had not thought about in our simulations: for example, the calculation of average RTT and RCP rate in the router's control plane requires 64-bit arithmetic operations support, which is not available

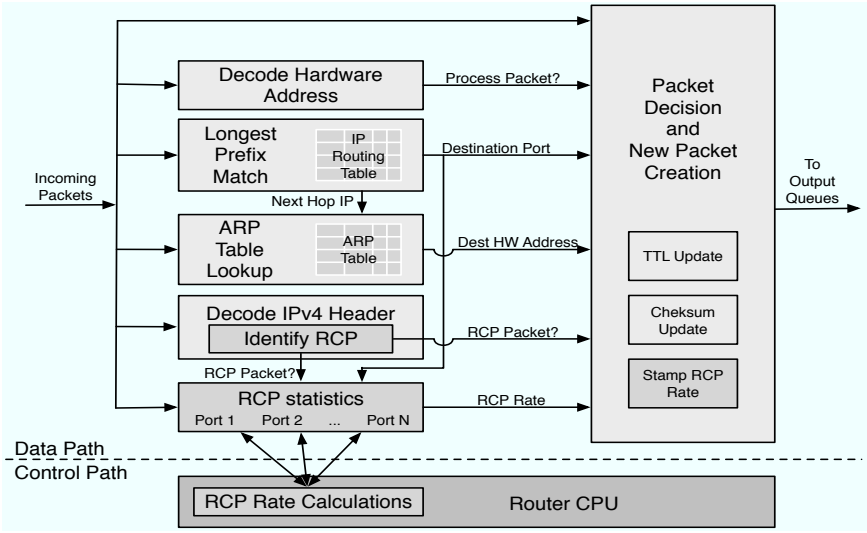


Figure 6.9: A generic hardware router with RCP support

on the 32-bit kernel we are running. We have added special handling to overcome this problem.

6.1.3 RCP Router based on NetFPGA

Real routers implement the data path functionality in hardware for speed. And so to demonstrate the feasibility and simplicity of supporting RCP within a router, we describe an implementation of RCP in hardware, running at 1 Gbps.

Just as before, the RCP functionality is split between the data path and the software control path. Fig. 6.9 shows the RCP enhancements to a vanilla router implementation. Per-packet data-path processing includes *RCP Identification* (identifying whether an incoming packet is an RCP packet), *RCP stats per-port* (updating a running RTT sum of the outgoing interface if the packet carries a valid RTT, and updating the aggregate traffic destined to the outgoing interface), and *Stamp RCP Rate* (stamping the RCP rate in the outgoing packet). The periodic control path computations include calculating the rate, $R(t)$, allocated to the average data flow as per Eqn. 3.3, and the moving round-trip time average, as detailed in Chap. 3.

Our hardware implementation utilizes the NetFPGA programmable hardware platform. NetFPGA is a programmable hardware platform for networking research and teaching [66, 67, 68]. The platform consists of a PCI card, which hosts a user-programmable FPGA,

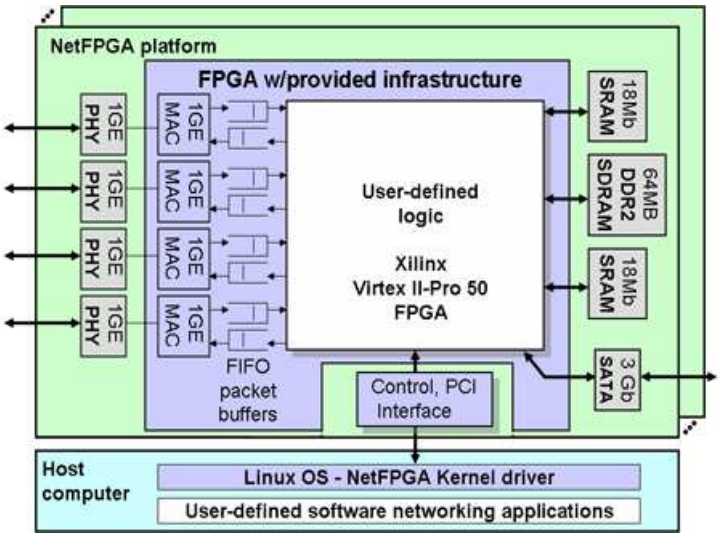


Figure 6.10: A block diagram of the NetFPGA hardware platform. The platform consists of a PCI card which hosts a user-programmable FPGA, SRAM, DRAM, and four 1 Gbps Ethernet ports.

SRAM, DRAM, and four 1 Gbps Ethernet ports, together with associated software for programming and control. A block diagram of the platform is shown in Fig. 6.10.

The remainder of this section describes the per-packet processing and the periodic computations in more detail.

Per-packet processing in hardware

RCP requires only a small amount of per-packet processing – in the worst case 3 integer additions, 2 comparisons, and 1 write operation. No multiplications or divisions are performed in the data path hardware. Pseudo-code of the data path operations, listed below, illustrates the modest processing requirements. Upon packet arrival the router must update counts for the corresponding output port of the running RTT sum, the number of arriving bytes, and the number of packets carrying a valid RTT. On packet departure the router overwrites the bottleneck rate carried in the packet if need be.

Processing performed upon packet arrival

```
input_traffic_Bytes += packet_size_Bytes
if (this_packet_RTT < MAX_ALLOWABLE_RTT)
```

```
  sum_rtt_Tr += this_packet_RTT
  num_pkts_with_rtt += 1
```

Processing performed upon packet departure

```
if (packet_BW_Request > rcp_rate)
  packet_BW_Request = rcp_rate
```

A quantification of the implementation complexity as well as achievable clock rates can be found in Sec. 6.1.4.

The control path in software

The control path performs periodic rate and RTT computations, and since these are performed infrequently they are implemented in software. In our implementation this software runs on the host in which the NetFPGA board is installed—in practice this would run on the CPU of the router.

The control path only performs work at initialization when each port of the RCP router is brought online and at the expiration of each time-slot. The initialization performed by the control path simply sets the various RCP parameters within the router to their default values. Upon expiration of a timer the control path reads the hardware registers to retrieve the RCP statistics, performs the necessary rate and RTT computations, writes the updated RCP rate and time-slot interval to the hardware registers, and then restarts the time-slot timer.

It should be noted that *all* multiplication and division operations required for rate calculations are performed within the control path. This allows RCP to take advantage of the multiplication/division operations of the router's CPU.

The control path software interfaces with the hardware via the registers provided by the hardware. The number of registers required per port is small—we provided 7 registers per-port for input-traffic, queue occupancy, current time-slot duration, elapsed time within the current time-slot, RCP rate to be stamped into the packet, sum of RTTs in packets, and input traffic in bytes carrying valid RTT.

6.1.4 Quantifying the Implementation Complexity

Complexity of RCP NetFPGA implementation

RCP takes 31,000 gates of the 4.1 $\times 10^6$ gates used for the basic IPv4 router implementation on the NetFPGA platform, amounting to 0.75% of the total logic used [18]. This translates to a die area of $0.1 - 0.2\ mm^2$ in a $90nm$ ASIC. Clearly, RCP is simple to implement in high-speed routers.

Our reference router (running on a Xilinx Virtex II Pro 30) used a core clock frequency of 62.5 MHz. The RCP implementation was not the bottleneck in this design.
Hardware resources:

- 5 × 32-bit software accessible registers per port
- 2 × 16-bit software accessible registers per port
- 2 × 16-bit counters per port (used by timers)
- 2 × 32-bit adders per port (packet/queue statistics)
- 1 × 32-bit counter per port (packet/queue statistics)

Lines of code (including comments, data structure declarations, etc.):

- Verilog (data path): approximately 600 lines
- C (control path): approximately 350 lines

Complexity of RCP End-host and RCP software router

The RCP end-host has 250 lines of C code (including commenting and declarations), and does not involve any floating point computations.

For the software RCP router, the computations on the control plane (for periodic RCP rate and average RTT calculations) and the data plane for each transit packet through the router are shown in Table 6.1.

To compare the complexities between a standard linux router without RCP support and that with RCP-enabled, we measured the time it takes a packet to traverse through the IP forwarding path in both cases. For a non RCP-enabled kernel, each packet processing takes 9.7368 jiffies, and when RCP-enabled it takes 9.9998 jiffies. Therefore, RCP-related processing is only 2.6% of the IP packet forwarding in the kernel.

Table 6.1: Complexity of RCP software router

	Control Plane	Data Plane		
		Ingress	Egress	Total
LOC	28	11	13	24
U32 Comparison	3	3	2	5
U32 Additions	5	4	0	4
U32 Multiplication	26	0	0	0
U32 Assignments	9	5	6	11

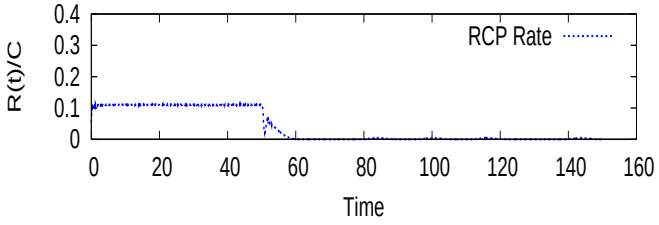


Figure 6.11: Demonstrating the problem when RCP and TCP coexist: Bottleneck link implements the RCP algorithm. Nine RCP flows start at time $t = 0$, and one TCP flow joins in at $t = 50$. RCP rate is throttled immediately.

6.2 Incrementally deploying RCP

For RCP to see widespread deployment, we need to understand how it can be incrementally deployed in the current Internet. Even if RCP received widespread adoption it would need to be introduced into a network with two hindrances: (1) RCP will need to operate alongside existing non-RCP traffic, such as TCP and UDP, without adversely affecting or being adversely affected by the other traffic; and (2) RCP will need to operate in a network where some routers are not RCP-enabled. In this section, we will explore both hindrances in turn.

6.2.1 Hindrance 1: RCP must coexist with non-RCP traffic

Let's first understand how severe the hindrance can be. Imagine a simple network in which all routers are RCP-enabled, but must carry both RCP and TCP traffic. Fig. 6.11 illustrates an example, where nine long-lived RCP flows share a link from time 0, and a TCP flow joins the network at time 50. The RCP flows are throttled. This is because TCP fills up router buffers until they overflow, while RCP attempts to keep the buffer occupancy low and only makes use of the spare capacity. On seeing queued-up TCP packets and increasing

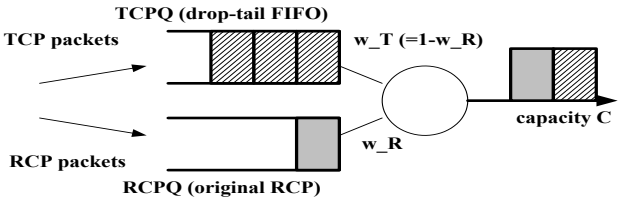


Figure 6.12: RCP and TCP traffic is isolated into separate queues that are served using WFQ, with weights w_R and w_T respectively. TCPQ and RCPQ are both FIFO drop-tail queues. RCPQ additionally uses the RCP algorithm to give rate feedback to RCP flows.

incoming traffic, RCP backs off and eventually stifles its own transmission rate.

To solve the above problem, we describe below some simple modifications to RCP routers. Our approaches do not affect the end-host implementation. We restrict the discussion to bandwidth sharing with TCP traffic and expect the same mechanisms to hold true for other kinds of traffic as well.

The most obvious solution, illustrated in Fig. 6.12, is to isolate the RCP and TCP traffic into two different queues. Both RCP and TCP queues are FIFO drop-tail queues. The RCP queue uses the RCP algorithm to give a rate feedback to the RCP flows, with a key difference that the link-rate C now varies with time, denoted as $C_R(t)$. The two queues are served using schedulers such as Weighted Fair Queuing (WFQ), or Deficit Round Robin (DRR). Our goal is for an RCP flow to receive its fair share of the link, regardless of how other types of traffic behave. We estimate the average flow-rate of RCP and TCP and periodically adapt the weights with which the two queues are served (w_R for RCP queue and w_T for TCP queue), so as to equalize the average RCP and TCP flow-rate. These weights are also used to determine the link-rate, $C_R(t)$, that RCP uses in its equation. More formally:

- $y_R(t)$ denotes the measured incoming RCP traffic-rate, $C_R(t)$ is the link-rate that RCP rate computation uses, and $q_R(t)$ is its queue length.

- **STEP C**: Once every control update period T_c, do the following:
  (C.1) Estimate the approximate number of active RCP and TCP flows, denoted by $N_R(t)$ and $N_T(t)$.
  (C.2) Find the weight, w_R, to adaptively equalize the average flow-rates of RCP

$(r_R(t))$ and TCP $(r_T(t))$:

$$w_R = w_R - \theta \frac{r_R(t) - r_T(t)}{r_R(t) + r_T(t)}$$

where $r_R(t) = \frac{y_R(t)}{N_R(t)}, r_T(t) = \frac{y_T(t)}{N_T(t)}$, and θ is a parameter defined in $(0, 1)$ controlling how smooth the update is.

(C.3) Update the RCP link share, $C_R(t)$, as follows:

$$C_R(t) = \max(C - y_T(t), C \cdot w_R)$$

- **STEP R**: Once every rate-update interval, compute the RCP rate:

$$R(t) = R(t - T)\left(1 + \frac{\frac{T}{d}(\alpha(\eta \cdot C_R(t) - y_R(t)) - \beta\frac{q_R(t)}{d})}{\eta \cdot C_R(t)}\right)$$

- **STEP S:** Serve the RCP and TCP queues using WFQ, with weights w_R and w_T respectively.

In Step C.1, we estimate the number of flows $N_R(t)$ and $N_C(t)$ using a randomized algorithm with Zombie list proposed in [74]. A zombie list is a *fixed* size array recording the flow identifiers of recently seen packets. On each packet arrival, the flow Id either writes to an empty entry or, with probability p, overwrites a stored entry if the list is full. The number of flows, $N_R(t)$ and $N_T(t)$, is estimated by the reciprocal of the hit probability—the probability that an incoming packet belongs to a flow in the list. There are several other algorithms in the literature to achieve the same goal, such as techniques using Bloom Filters [73].

This algorithm introduces another parameter, (the control update interval) T_c, whose value impacts the system stability. For example, if T_c is too small, RCP reclaims the link-share it relinquished to TCP sooner than TCP has had a chance to use it. This causes oscillations in $C_R(t), C_T(t)$ values ($C_T(t)$ is TCP's share of the link bandwidth). On the other hand, a large T_c value makes the algorithm sluggish, and RCP does not achieve its fair share. For stability, T_c should at least be equal to the amount of time needed for TCP to fill up the newly allocated link capacity to it. In TCP's Congestion Avoidance mode, the window size increases by one packet size per round-trip time (assuming TCP-Sack). In general however, suppose the window size changes by $b(\delta t)$ bytes in time δt, and there are

n_T TCP flows, then the input traffic rate of TCP, $y_T(t)$, evolves as follows:

$$y_T(t + RTT) = y_T(t) + \frac{n_T \cdot b(RTT)}{RTT}$$

Assume it takes K round-trip times for TCP to increase its sending rate from $y_T(t)$ to a link capacity share C_T allocated at time t. That is,

$$C_T = y_T(t + K \cdot RTT) = y_T(t) + \frac{n_T \cdot b(RTT) \cdot K}{RTT}$$

Then, the minimum T_c can be derived as:

$$T_c^* = K \cdot RTT = \frac{C_T - y_T(t)}{n_T \cdot b(RTT)} \cdot RTT^2$$

Routers generally are not able to obtain the round-trip times of each TCP flow, but they can estimate the average RTT by noting that:

$$\dot{y}_T(t) = \frac{n_T \cdot b(RTT)}{RTT^2}$$

where $\dot{y}_T(t)$ is the change in traffic-rate. Then T_c^* becomes:

$$T_c^* = K \cdot RTT = \frac{C_T - y_T(t)}{n_T \cdot b(RTT)} \cdot \frac{n_T \cdot b(RTT)}{\dot{y}_T(t)} = \frac{C_T - y_T(t)}{\dot{y}_T(t)}$$

For stability, the choice of T_c should be larger than T_c^*. Note that we do not require knowing the particular TCP flavor or a mix of flavors that RCP is coexisting with.

Fig. 6.13 shows an example of the above scheme where RCP and TCP flows share bandwidth equally.

In some extreme situations a purpose-built RCP-enabled switch/router might not be able to afford one more queue for all traffic. In this case, a variant of the above scheme may be used: all traffic share a single queue that is served in a FIFO manner. The weights computed in the scheme above are simply used to compute RCP's ($C_R(t)$) and TCP's ($C - C_R(t)$) link-shares, achieving a "soft" isolation of traffic. RCP only fills up its share, $C_R(t)$, leaving the remaining bandwidth for TCP flows. The algorithm has the same steps as the scheme described above (except for **STEP S**, that schedules queues, is not necessary here) and we omit restating the scheme.

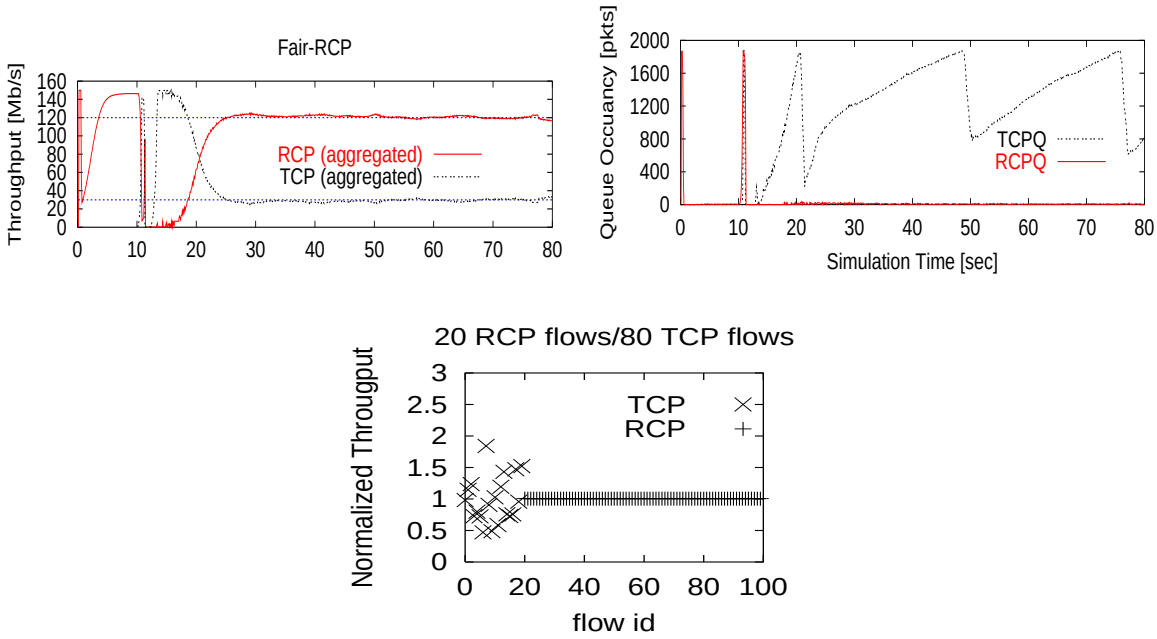


Figure 6.13: The scheme achieves fair bandwidth sharing among 80 RCP (existing on the link from time 0) and 20 TCP (starting at time 10) flows sharing a bottleneck link. The top left plot shows the aggregate throughput of RCP and TCP flows; the top right plot shows their queue occupancies; the bottom plot shows the normalized fair-share rate of individual flows.

An example of the simplified scheme in action is demonstrated in Fig. 6.14, in which there are equal number of RCP and TCP flows, and both receive their fair shares. Note that while this scheme achieves fair sharing of the bottleneck bandwidth, there are fluctuations in RCP's rate ($R(t)$), link-share ($C_R(t)$) and its queue, corresponding to TCP's saw-tooth behavior in the shared queue.

Under Different Network Conditions

The above schemes achieve the fair link-share property, under various network conditions. In this section, we show that these algorithms:

- Apply to multiple bottleneck links and still achieve equal bandwidth-sharing.
- Work for all flow sizes and RCP retains its property of short flow completion times.

The scenario of multiple bottleneck links can occasionally occur in real networks and causes throughput degradation in TCP. We need to ensure that RCP does not have this

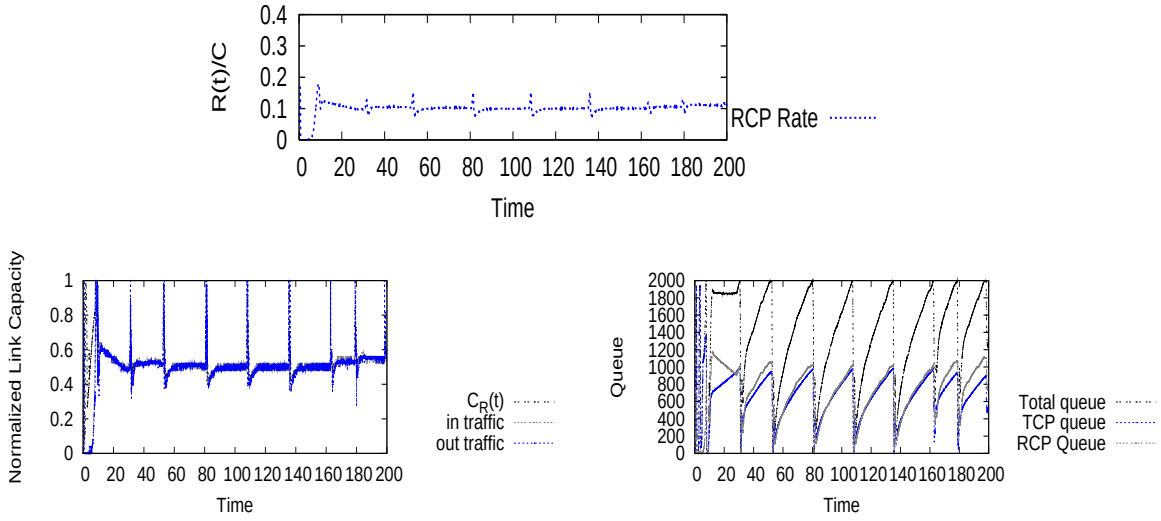


Figure 6.14: Five long RCP and TCP flows share a bottleneck link. The top plot shows RCP's rate; bottom left plot shows RCP's computed link-share; bottom right plot shows the queue occupancy of RCP and TCP flows. The average link shares are 0.5 each for RCP and TCP. The fluctuations follow from TCP's saw-tooth behavior.

problem and will not be affected by the coexisting TCP traffic. Fig. 6.15 illustrates an example of this scenario. While TCP is unfavorable to flows passing through multiple bottlenecks, RCP still achieves fair sharing in the sense that each RCP long flow achieves the same share as single-hop TCP flows.

So far we have tested the schemes' fairness properties under long-lived flows. To be more realistic, we generate RCP and TCP flows arriving in a Poisson process and having Pareto distributed flow-sizes. Results show that on average, long RCP flows not only share the link bandwidth equally with long TCP flows, but RCP also retains the feature of short flow completion times, as shown in Fig. 6.16.

6.2.2 Hindrance 2: Coexisting with non-RCP bottlenecks

Obviously, we cannot hope that all routers and other network devices will be equipped with RCP functionality overnight. RCP end-hosts need to be able to communicate with non-RCP enabled network devices. If after the initial handshake, an RCP flow does not receive a valid rate, by default it switches to TCP congestion control since none of the routers along the path understand RCP. This situation is the simplest to resolve.

When an RCP flow receives a valid rate, it needs to find out if the router suggesting

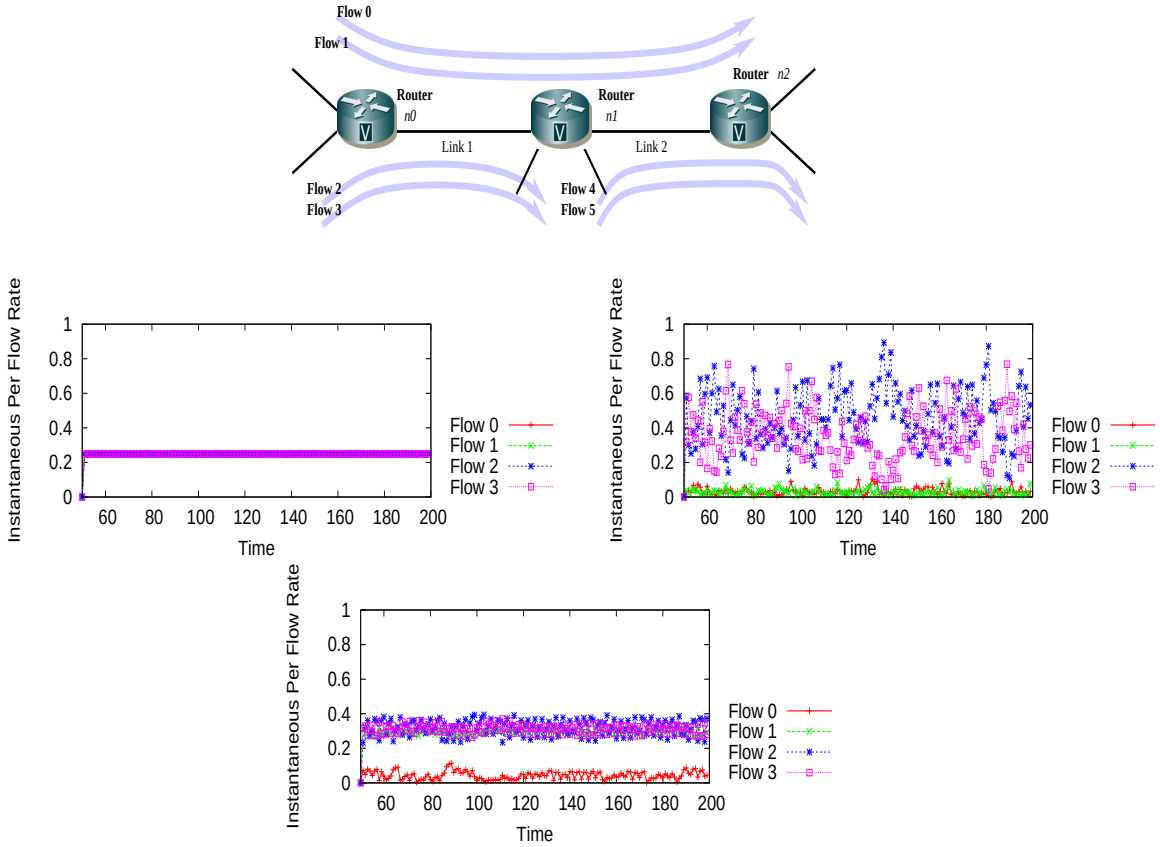


Figure 6.15: Example of RCP and TCP flows coexisting in the presence of multiple bottleneck Links (top plot shows the topology). Simulation results are shown for Link 1; Link 2 gives a symmetric outcome. Middle left plot: When all six flows are RCP, each receives the same rate. Middle right plot: When all flows are TCP, the two-hop flows (flows 0 and 1) are throttled by the 1-hop flows. Bottom plot: In the case of RCP-TCP traffic mix, flows 1, 3 and 5 are RCP while flows 0, 2 and 4 are TCP. Each RCP flow and single-hop TCP flows achieve the same rate.

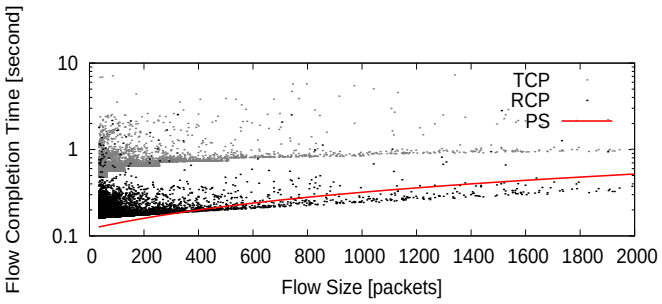


Figure 6.16: Flow completion times when RCP and TCP coexist.

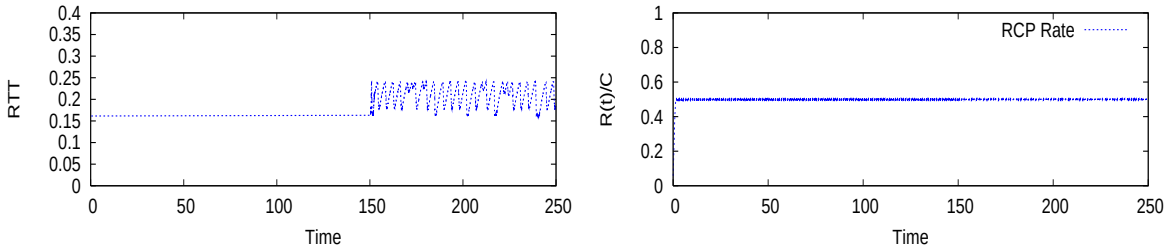


Figure 6.17: Example demonstrating the switching point for an end-host from RCP to TCP: At the start, an RCP flow goes through a bottlenecked RCP-enabled link and a non-bottlenecked non-RCP link. At time $t = 150$, a TCP flow joins in the non-RCP router, which shifts the bottleneck to the non-RCP link. At this point, the end-host observes an increase in RTT (left plot) as TCP fills up the buffer, while the received RCP rate feedback remains unchanged (right plot).

this rate is in fact the bottleneck. In this case, a flow optimistically starts by transmitting at the received RCP rate and uses a heuristic to figure out if in fact this is the bottleneck rate. Once RCP determines that the suggested rate is not the bottlenecked rate, it switches to TCP congestion control immediately to avoid overwhelming the bottleneck. We would like a heuristic with the following properties:

- *Zero* false-negative rate (we do not want an RCP sender to use RCP if the bottleneck is non-RCP) and,
- A small false-positive rate (i.e., we want the sender to use RCP if the bottleneck router runs RCP).

Our heuristic is based on an invariant property of the RCP algorithm: When a queue builds up, RCP will react by reducing its rate so as to drain the queue. An end-host switches to TCP if it observes either of the following, without a corresponding decrease in the received RCP rate:

- Estimated round-trip time at end-host is doubled, or
- Experiences packet-losses.

The rationale behind the heuristic is: If the non-RCP router has a buffer size that equals the bandwidth-RTT product, then as the buffer fills up the RTT will increase to at least twice the base RTT value, and if there is no corresponding reduction in RCP rate,

this is an indication to switch over. On the other hand, if the buffering is smaller than bandwidth-RTT product, then packet losses occur before the RTT doubles, indicating the need to switch.

Fig. 6.17 illustrates an example of this heuristic in action. The end-host switches to TCP within 10 RTTs after the bottleneck shifts to a non-RCP router, and in this set-up we also checked that there are no false positives.

6.3 Sizing router buffers for RCP congestion control

RCP will also exist in a network where switches and routers have very small or large buffers. Our focus in this section is to find the amount of buffering RCP requires, and particularly questions such as: How does its buffering scale with network bandwidth and round-trip times? What happens when flows arrive and depart randomly, and have a mix of flow-sizes? How does buffering depend on the offered network load? Does it depend on the flow-size distribution?

With RCP, we find that if there are a fixed number of long-lived flows in the system (Fig. 6.18 shows an example), the amount of buffering required is trivially $O(1)$, i.e., the required buffering is a constant *independent* of the link capacity, RTT and the number of flows. Since the router can give an explicit rate feedback, all flows eventually converge to the equilibrium fair-share rate with a steady state link utilization of 100% and zero losses, independent of number of flows. Our simulations suggest that 100% link utilization can be achieved with as few as 10-20 packets.

[75] goes a step further and characterizes the amount of buffering needed by explicitly modeling flow arrivals and departures. The analysis indicates that in a dynamic environment, $O(1)$ buffering is no longer sufficient. To maintain consistent performance (in terms of flow completion times), buffers need to be scaled linearly with link-capacity, as $O(C)$. The good news is our simulations indicate that RCP can function very well even with buffers as small as 5% of the bandwidth delay product. Because RCP is designed for short flow completion times, we choose AFCT (Average flow completion time) to be the primary performance metric and successfully capture the trade-off between the use of small buffers and the increase in the flow completion times.

[75] derives buffer sizing results for two extreme cases:

- When the mean flow size is large compared to the bandwidth delay product, $E[L] \gg$

$C \cdot RTT$

- When the mean flow size is small compared to the bandwidth delay product, $E[L] \ll C \cdot RTT$

The first case is a natural extension of the static scenario in which a fixed number of flows, N, stay in the system for an infinitely long-time. In this setting, the number of flows in the network changes very slowly when compared to the convergence times of RCP. Therefore, RCP can track the number of flows in the network quite accurately, thereby making efficient use of link capacity.

The second case represents a network dominated only by short-flows. The motivation for studying short-flows is the large number of short flows in the Internet today. Since short-flows cannot be controlled, one could argue that large buffers must be employed to eliminate large packet loss. [75] shows that even when *no* congestion control is imposed on the short-flows, small buffers do not degrade the performance of RCP significantly. The Internet consists of a large number of short-flows that contribute to a small fraction of the traffic volume, and a small number of long-flows that contribute to a large fraction of the traffic volume. The analysis in [75] considers the two extreme cases (short-flows and long-flows) while the mixture is studied in simulations below. [75] also analytically characterizes the impact of losses on the AFCT of flows, in particular calculates the AFCT of flows as a function of packet loss probability p.

The analytical results indicate that - for both cases - the buffer size scales linearly with the capacity. The simulations, described below, indicate that buffer sizes as small as 5-10% of the bandwidth delay product are sufficient to maintain good performance.

6.3.1 Simulation Results

Our simulations use ns-2.29 and we have augmented the RCP end-host algorithm described in Chap. 3 (code available at RCP web page [76]) with a simple retransmission scheme, which works as follows: The receiver informs the sender of the cumulative number of packets it has received so far, through the acknowledgment packets. After the sender completes transmitting all packets of a flow, it waits for a certain amount of time (for example, 2×RTT) for the remaining packets in network to reach the destination host. Beyond that, it assumes that the difference in the number of packets transmitted (the flow-size) and the number of packets acknowledged by the destination host are lost. It transmits the difference

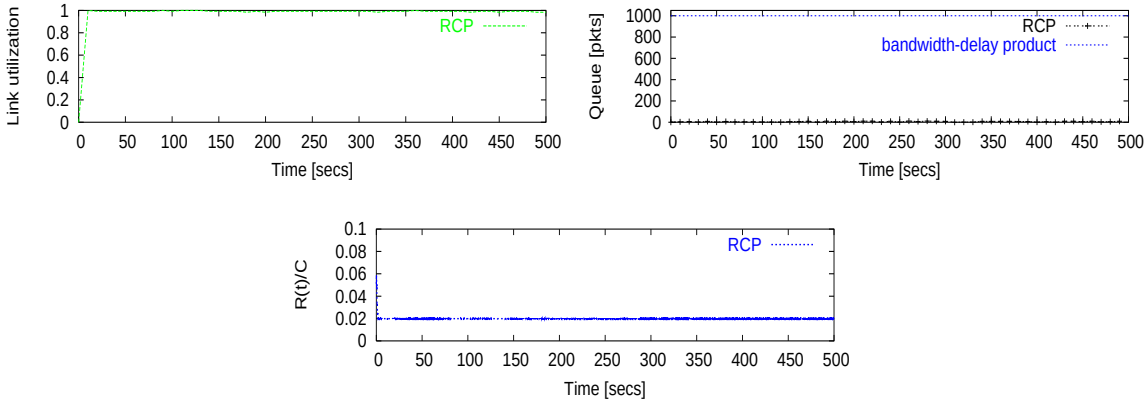


Figure 6.18: Simulation illustrating that RCP achieves PS under small buffers for a static scenario of N long-lived flows. The set-up is: $C = 0.1$ Gbps, RTT = 80 ms, 50 long flows start at time 0, Buffer size = 20 packets (2% of BW-RTT). The top left plot shows the normalized link utilization, the right plot shows the queue occupancy over time and the bottom plot shows the normalized RCP rate.

and the flow completes when the sender knows that the receiver has received all the flow's packets. This abstraction simplifies significantly the retransmission mechanisms by which the receiver informs the sender of exactly which packets were lost, at the same time it is detailed enough to study the effects of small buffers on flow completion times.

Let's first consider the simplest case with N static, long-lived flows. Because RCP does not rely on loss as an indication of congestion, we expect it to settle down to its equilibrium of $(R_e = C/N, q_e = 0)$ irrespective of the buffer size at the congestion point. This is shown in Fig. 6.18, where the buffer is 2% of the link-capacity[3]—the link-utilization is 100%, steady-state queue is close to zero and every flow receives its fair-share.

We are interested in the case when flows arrive and depart with a mix of flow sizes. Our metric of interest is average flow completion time (AFCT) (or equivalently the mean number of active flows) and our reference is the Processor Sharing (PS) system. If RCP implements PS accurately, then the mean FCT is given by,

$$2 \cdot RTT + \frac{E[L]}{C \cdot (1 - \rho)} \tag{6.1}$$

The expression takes into account the time required for the connection setup.

Simulation Set-up: In the following we will see how RCP performs w.r.t. Eqn. 6.1 when

[3]This property holds true for smaller and larger buffers as well.

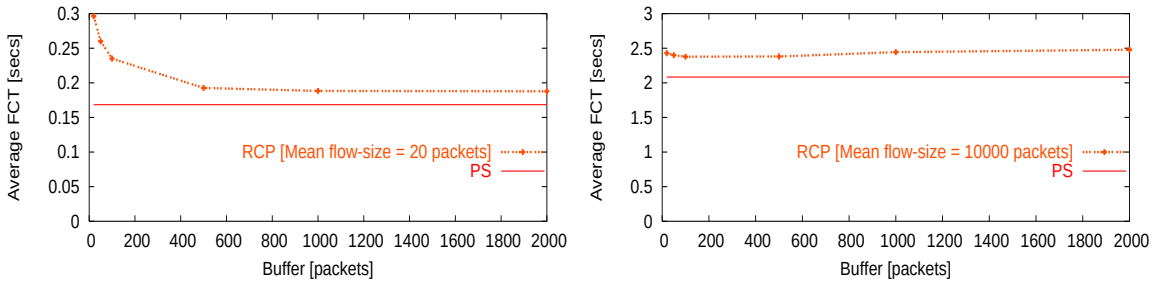


Figure 6.19: Plot illustrating that small buffers increase AFCTs for small flows and do not affect AFCTs as much for large flows. The set-up is: $C = 0.1$ Gbps, RTT $= 80$ ms, $\rho = 0.68$, mean flow-size is 20 packets (2% BW-RTT) for left plot and 8200 packets (820% BW-RTT) for the right plot. Buffer size is varied from 2% (20 pkts) to 200% (2000 pkts) of bandwidth-delay.

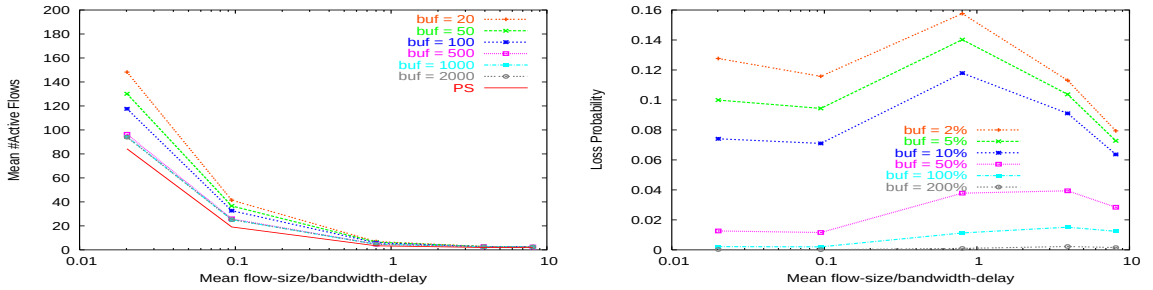


Figure 6.20: Plot illustrating mean number of active flows (left) and loss probability (right) versus mean flow-size under different buffer sizes. The set-up is the same as the last Figure.

buffer sizes are varied. In each case we will vary one of $C, E[L], \rho$ and RTT while keeping all else constant. The RCP parameters are chosen as $(\alpha, \beta) = (0.4, 0.5)$. Unless mentioned otherwise, we will assume the flow arrival process is Poisson and the flow sizes are Pareto distributed with shape parameter 1.2.

As the mean flow-size increases

The relative mean flow-size (compared to the bandwidth-delay product) is the single biggest deciding factor on how close RCP is to PS with small buffers. To understand this, consider two extremes - a small mean flow-size and extremely large mean flow-size. For infinitely large flow-sizes, small buffers make little/no difference to how well RCP emulates PS, like the example in Fig. 6.18. For smaller/medium flow-sizes the network dynamics are changing more rapidly (due to higher λ for any fixed load $\rho = \frac{\lambda E[L]}{C}$), and we would expect RCP to

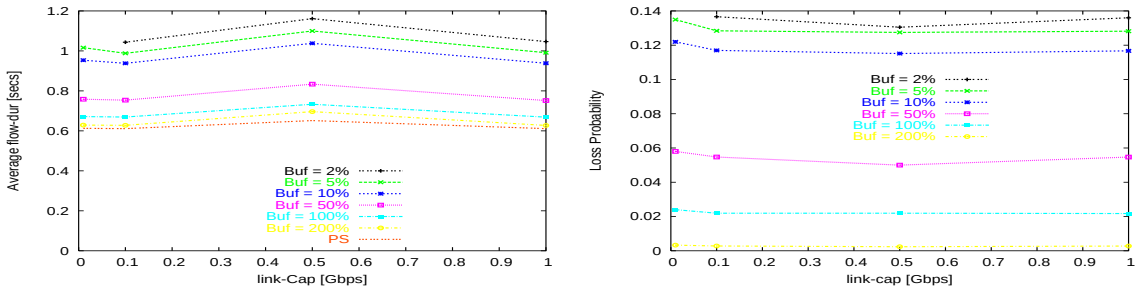


Figure 6.21: Plot illustrating mean FCT (left) and loss probability (right) versus link-capacity under different buffer sizes. The set-up is: RTT = 80 ms, $\frac{\rho}{C} = 0.68$, C is varied over three orders of magnitude and $E[L]$ is varied so that $\frac{E[L]}{C \cdot RTT}$ equals 1.7 in every case.

deviate from the PS behavior for small buffers.

In the following simulations, new flows arrive according to a Poisson process and have Pareto flow-size distributions. We fixed the link-capacity (C), offered load (ρ) and RTT while varying $E[L]$ from 2% to 1000% of the bandwidth-delay product for buffer sizes ranging from 2% to 200% of the bandwidth-delay product. Fig. 6.19 shows the mean flow completion time for the two extremes of mean flow-sizes. As expected, RCP clearly emulates PS well for large flows even under small buffers. Small flows on the other hand, have larger completion times as buffers become smaller, with the increase being at most 50% for a buffer as small as 2% of bandwidth-delay product.

Fig. 6.20 shows the mean number of active flows and loss probability across a wide range of mean flow-sizes. Note that the difference between RCP and PS gets negligible as the mean flow-size increases. When the flow-size is small the RCP system is nothing but a M/M/1-FCFS system where loss probability for any given buffer-size goes to zero as mean flow-size goes to zero. For very large flows too the loss probability goes to zero (in the extreme case N long-lived flows). So as the mean flow-size increases, we expect the loss probability to first increase and then decrease. Fig. 6.20 confirms this.

As the link-capacity increases

There are two ways of looking at results with increasing link-capacities - 1) Vary C while keeping all else constant; this amounts to varying the mean flow-size relative to bandwidth-delay which we have already seen in Sec. 6.3.1 and 2) Vary C and scale $E[L]$ to keep the relative mean flow-size $\frac{E[L]}{(C \cdot RTT)}$ a constant, which we will see in this section.

Fig. 6.21 shows the mean FCT and the loss probability for different buffer sizes while

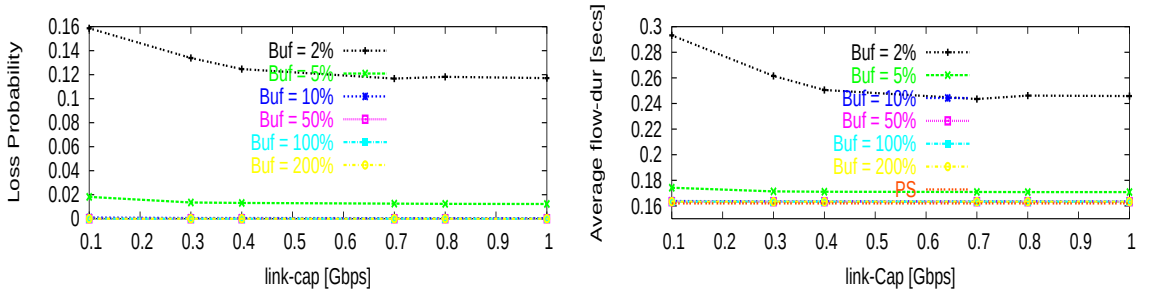


Figure 6.22: Plot illustrating mean FCT (left) and loss probability (right) versus link-capacity under different buffer sizes. The set-up is: RTT = 80 ms, $\rho = 0.8$, C and $E[L]$ are varied so that $\frac{1}{\mu(C \cdot RTT)}$ equals 0.005 in every case.

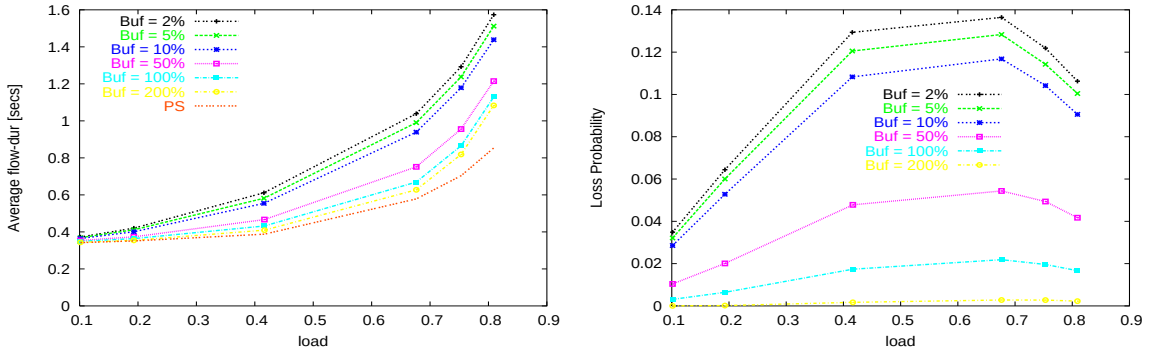


Figure 6.23: Plot illustrating mean FCT (left) and loss probability (right) versus offered load under different buffer sizes. The set-up is: C = 0.1 Gbps, RTT = 80 ms, $E[L]$ is 1700 packets ($\frac{E[L]}{(C \cdot RTT)} = 1.7$).

keeping $\frac{\rho}{C}$, RTT, $\frac{E[L]}{C}$ constant and varying C over three orders of magnitude. In each case, $E[L]$ is chosen to be twice the bandwidth-delay product. As we would expect from Eqn. 6.1 the FCTs in RCP remain constant for fixed $\frac{E[L]}{C}$ even while both $E[L]$ and C vary for any given buffer size. The increase in FCT is at most 50% under very small buffers. The loss probability is constant over different link-speeds so long as the buffers are scaled with C: a result that agrees well with the analytical results in [75].

While Fig. 6.21 is for large mean flow-sizes, Fig. 6.22 shows the loss probability versus link-rates when the mean flow-size is small. The loss probability remains constant so long as buffer sizes scale with link-rates, agreeing well with the analysis in [75].

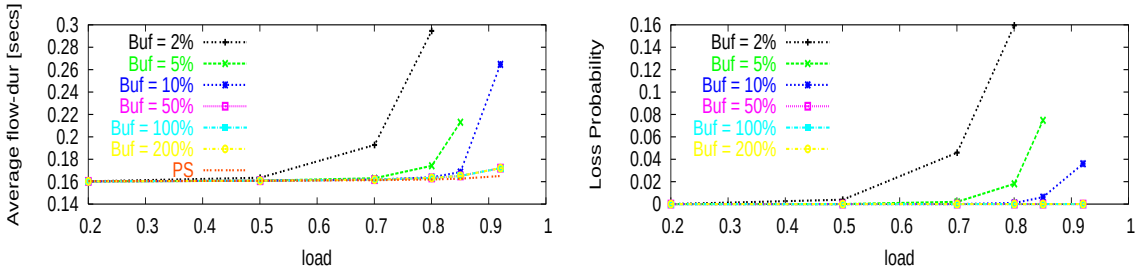


Figure 6.24: Plot illustrating mean FCT and loss probability versus offered load under different buffer sizes. The set-up is: C = 0.1 Gbps, RTT = 80 ms, $E[L]$ is 5 packets ($\frac{E[L]}{(C \cdot RTT)} = 0.005$). Flow sizes are constant.

As the offered load increases

The offered load ρ is varied from 0.1 to 0.8 (varying λ), while keeping everything else fixed. Fig. 6.23 (large mean flow-size) and Fig. 6.24 (when mean flow-size is small) show the mean FCT versus the offered load. The FCTs under RCP follow the PS curve well for large buffers and are shifted by 50% for the very small buffer sizes. The shift is more pronounced for higher loads as opposed to low loads. The loss probability versus load is shown in the right hand plots of Fig. 6.23 (when mean flow-size is large) and Fig. 6.24 (when mean flow-size is small). The nature of the curves (in complete agreement with the results in [75]) is as follows: For small flows, the loss probability increases with the offered load - similar to a M/M/1/B-FCFS system. For large-flows, the loss probability first increases and then decreases with the offered load. The intuitive reasoning is as follows: For large flows RCP is analogous to a M/M/1/B-PS system where losses are caused by newly arriving flows and because the system has a time-delay to adjust to a new rate. Beyond a certain load, the processor sharing rate received by the new flows decreases faster than the rate at which new flows are arriving and hence the decrease in losses.

There is good news and bad news about sizing buffers for RCP. The bad news is that the amount of buffering scales with the bandwidth-delay product. The good news is that we can reap the benefits of RCP with buffers as small as 5-10% of bandwidth-delay product. Ultimately, the deployment of any congestion control mechanism, especially one involving the network infrastructure, requires solving many practical problems. In this section, we have convinced ourselves that RCP retains its appealing properties even with small buffers and therefore its buffering requirements are unlikely to be a show stopper when it comes to practical deployment.

6.4 Other practical considerations in an RCP network

There are a few other practical considerations, not addressed in this chapter, such as: How is RCP information carried in the presence of tunneling? How does RCP coexist with middle-boxes such as NATs and firewalls? What happens when the link-rate is not a constant and varies with time, in some manner that is hard to predict? The latter of these questions arises in at least two different contexts. First, in wireless environments where the link-rates vary over time because of interference and mobility, and second, in routers and switches with input queues that are arbitrated by some scheduling discipline.

An obvious solution to the above problem, is to observe how the link-rate, C, is used in the RCP equation, and look to replace C with a term that would be a close approximation to it. C appears at two places in the RCP rate equation: $(C - y(t))$ is the spare bandwidth and $\frac{C}{R(t)}$ is the router's estimate of the equivalent number of flows. Note that when queue length is positive, $(C - y(t))$ represents the change in queue length. Further, we also find that an equally good estimate of the equivalent number of flows is $\frac{y(t)}{R(t)}$, where $y(t)$ is the aggregate incoming traffic rate. With these changes, the rate equation is as follows:

$$R(t) = R(t-T)\left(1 + \frac{\frac{T}{d}\left(-\alpha\dot{q}(t) - \beta\frac{q(t)}{d}\right)}{y(t)}\right) \tag{6.2}$$

where $\dot{q}(t)$ is the change in queue-length and $y(t)$ is the aggregate incoming traffic rate over the interval T. If $q(t) = 0$, then $\dot{q}(t) := \theta$—a user-defined value which determines the increase in $R(t)$ when there is no queue built up. It remains to be seen how the above rate equation will perform in practical wireless networks and input-queued switches.

Chapter 7

Most Commonly Asked Questions About RCP

The design, analysis and deployment of RCP happened over several years. In the many talks I have given along the way, and from other interactions, I have collected a number of commonly asked questions. Suspecting that the reader might still have open questions, I am repeating them here, along with a response to each.

Question 1: *Won't RCP take a long time to converge after a sudden change in the number of flows?*

Like all feedback algorithms, RCP sets a rate that is out of date when it is used (a round-trip time later). While most congestion control algorithms act conservatively (or even timidly) to start with (e.g., the Slow-Start algorithm of TCP Sack that starts out with a window size of just two packets; and the slow additive increase of XCP), RCP starts up immediately at the best-estimate of the current fair-share rate. This means flows finish quickly; but it also runs the risk of overshooting—particularly when there are rapid changes in network traffic, for example because of a sudden flash crowd. A sudden increase in traffic is the worst-case scenario for RCP.

It is a deliberate design decision to optimize RCP for fast flow completion times in the common case as opposed to avoiding packet-losses in the worst-case network scenarios [77]. RCP is not optimized to prevent bad things from happening in the network, but instead, to recover quickly if and when sudden things happen. In scenarios such as sudden traffic changes, RCP will recover, and *provably* so in the control theoretic sense as we have seen in Chap. 5. The good news is that in the recommended range of RCP's parameters, $\alpha \in$

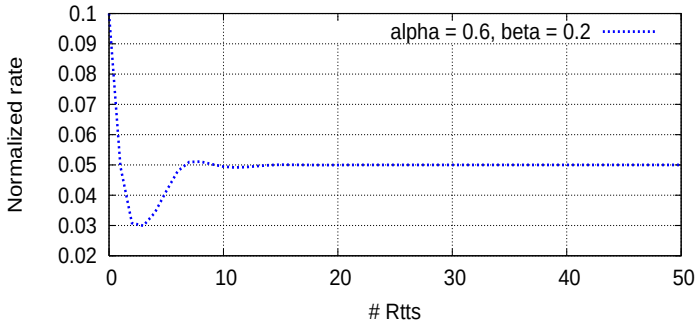


Figure 7.1: Example illustrating that RCP converges within 10 round-trip times on a sudden traffic change. There are 10 flows in equilibrium at time 0 (normalized RCP rate is 1/10), when the traffic load doubles suddenly. RCP reduces its rate in response to this change and finds the new equilibrium within 10 round-trip times.

$(0.4, 0.6)$ and $\beta \in (0.2, 0.6)$, RCP also recovers quickly—10 round-trip times *independent* of the link-rate, number of flows and round-trip time value, as we have seen in Chap. 5. Fig. 7.1 illustrates an example. I hypothesize that in the future this will be proved to be a non-problem.

[20] describes a variant of RCP for readers who are looking for a more conservative algorithm, with even shorter convergence times.

Question 2: *Do I have to pick an alpha and beta based on my network conditions, or can the router ship with a default?*

Fortunately, the range of parameters for which RCP works well is broad and the algorithm is not very sensitive to the exact choice within the range. The router can ship with default values of $\alpha \in (0.4, 0.6)$ and $\beta \in (0.2, 0.6)$. These values have been chosen among values that : a) RCP is known to be stable, b) flow completion times are close to processor sharing for a broad range of network and traffic conditions and, c) RCP recovers quickly after sudden traffic changes, such as flash-crowds.

Question 3: *What do you see as the deployment path for RCP? Will it appear first in routers or switches?*

RCP can first be deployed in special networks (often under the control of a single administrative domain) such as Data Centers (flow completion time is an important metric for distributed applications running in Data Centers) and Satellite networks (the unusually long round-trip times make end-host based schemes sluggish). As for the general Internet, it will first appear in routers and switches connecting bottlenecked links followed by end-host

stacks which will adopt RCP to take advantage of the more explicit information from the network.

Question 4: *If the problem is that TCP is too timid, why not increase the initial Slow-Start size to a larger value and to possibly more than double the rate of increase in Slow-Start. Minor changes seems preferable to major changes especially minor changes to end nodes that needs no coordinated change in routers.*

Slow-start is not the only reason why TCP's flow completion times can be long. TCP's conservative AIMD and the unfair way of bandwidth sharing among flows is also a penalty on FCTs. That said, the proposal above is a fair one considering that it may benefit the vast majority of flows. However, some problems with this proposal are:

- Even in the best-case scenario, flows will still take $O[log_k(S/wInit)]$ rounds to finish, where S is flow-size, $wInit$ is the initial window size and k is the increase factor of window size in Slow-Start mode—a logarithmic number of round-trip times even in the best of the conditions, as opposed to $O(1)$ round-trips for RCP.

- The above scheme sometimes makes problems much worse than a conservative Slow-Start—it can increase packet losses in Slow-Start mode making FCTs even worse. It increases unfairness among flows, especially with drop-tail buffers—flows can be in different stages of Slow-Start, so those in the later rounds which are increasing their windows rapidly, benefit at the expense of those in the earlier rounds.

- For more challenging conditions such as small buffers, the above proposal may turn out to be even worse than the traditional Slow-Start.

In essence, just increasing the initial window size and/or the rate of increase in Slow-Start while having only loss as feedback could be either be too aggressive or conservative, depending on the network conditions.

Question 5: *Apart from short flow completion times, what are the other consequences of RCP?*

Following are other consequences of RCP, some of which are useful from a network operator's point of view:

- RCP makes efficient use of high bandwidth-delay product networks such as the long haul optical links, and does not have the associated problems with TCP.

- It is easy in RCP to police flows and ensure they adhere to congestion control (which is generally not possible with just end-host based TCP schemes).
- Network operators can give preference (or weighted preference) to some sessions or aggregates of sessions.

Question 6: *Isn't RCP similar to the ATM ABR style algorithms [39] [40] [41]?*

RCP is similar to ATM ABR only in the sense that it receives an explicit congestion feedback from the network. The RCP equation shares similarities to the UT algorithm [84] developed by Sanqi Li for ATM networks. That aside, I am not aware of any ATM ABR algorithm that has been demonstrated to achieve flow completion times close to processor sharing for a broad range of traffic and network conditions in the presence of flow arrivals and departures, without maintaining per-flow state or per-flow queues.

Question 7: *Is Slow-Start (and the Slow-Start that XCP does) under certain circumstances, better than fair, better in the sense that XCP holds off the new flows and lets the old flows complete, but always hands out any capacity actually unused, so the link loading should remain high. But simulations suggest that this intuition is exactly backwards.*

The Slow-Start that XCP does is good when the mean flow-size is large enough, so what happens at the start is amortized over the duration of the flow. It is definitely not good when most flows are capable of finishing within a few round-trip times. It is especially poor when flow sizes are heavy-tailed in which case XCP's Slow-Start—in holding off new flows and letting old ones complete—is akin to emulating the FCFS discipline, whose average flow completion time is worse compared to processor sharing.

Question 8: *Suppose the router keeps track of the exact number of flows going through a router, $N(t)$, can RCP use this information?*

There are two reasons why knowing the exact number of flows is not directly useful to RCP.

- The skewed distribution of traffic in flows makes the knowledge of the exact value of $N(t)$ somewhat irrelevant.
- Different flows have different bottleneck links—there can be a large number of ongoing flows at a link but most bottlenecked elsewhere. Knowing $N(t)$ does not help much in this case.

Question 9: *Does RCP achieve max-min fairness in a network setting? Can RCP's mechanisms be used to achieve proportional bandwidth sharing in a network?*

RCP provably achieves max-min fairness. [83] proposes a variant of RCP that achieves α-fair rate allocations [83], including proportional fairness when $\alpha = 1$.

Question 10: *If the recently proposed end-host based scheme—PCP: Efficient Endpoint Congestion Control [82]—becomes widespread, would there still be a need for explicit feedback congestion control such as RCP?*

PCP cannot be a replacement for explicit feedback congestion control, as it fails to achieve many of their desired properties, of which I give four distinct examples below:

1. Short flow completion times: In the absence of any prior history information, a new PCP connection takes $O(logN)$ round-trips at start-up, and so short and medium sized flows will not experience any smaller response times as compared with TCP. On the other hand, explicit feedback schemes can finish flows in $O(1)$ round-trips, even in the absence of any history.

2. Short high-bandwidth flows: If a new flow on starting up wants to blast at a high rate (e.g., 1 Gbps) for a short period (e.g., 2 round-trip times) and stop, PCP (and TCP or any other end-host based schemes) cannot achieve it while explicit feedback schemes can.

3. Policing misbehaving flows: Like TCP, PCP provides no means to counter misbehaving hosts; network-based enforcement such as fair queuing is still the only way to achieve this. On the other hand, policing flows to ensure they adhere to congestion control is a simple byproduct in explicit feedback schemes such as RCP.

4. Fairness and stability for a broad range of network conditions: PCP's operating regime for good performance is a network with low load. Stability and fairness under high loads are PCP's secondary concerns, as opposed to the explicit feedback schemes which operate well over a broader range of network and traffic conditions, including extremely high loads.

In summary, if we would like our network to have the above properties, PCP is not the solution.

Chapter 8

Conclusion

In this thesis we argue that flow completion time is a very important metric for congestion control. We propose Rate Control Protocol, whose main goal is fast download times or flow completion times. In RCP, routers maintain a single fair share rate for all flows passing through a link. They update the rate periodically based on the congestion conditions, and strive to emulate processor sharing among flows. We studied RCP through simulations, modeling and experiments, and showed the following:

1. The flow completion times in RCP are close to those in ideal processor sharing for a broad range of network and traffic conditions.

2. RCP is provably stable irrespective of the link-rates, round-trip times, and the number of flows. Further, simulations indicate that it can be stable even under multiple bottleneck links with vastly heterogeneous round-trip times.

3. RCP is easily implementable in high-speed routers and end-hosts. The simplicity of operations on the data-path are particularly appealing.

4. We characterized the size of buffers for RCP in routers and switches, to understand how they scale with link-rates, offered loads, and flow sizes.

5. We described how RCP can be incrementally deployed in real networks, in particular how it can coexist with non-RCP traffic and routers that are not RCP enabled.

While RCP's performance is convincing in simulations and experiments with synthetic traffic, it is natural to ask if it works as well in real networks with real traffic, real users,

and real applications. Will users be able to perceive a noticeable improvement with RCP? While simulations and experiments definitely suggest so, ultimately running it on networks such as GENI [78] will tell us if that will in fact be so in reality.

If deployed, RCP is likely to have a tremendous impact on applications including web-browsing, watching videos, long file transfers (e.g. movies, uploading pictures), gaming, and ultimately on users' experience. Given that the impact is so huge, why isn't it then already widely prevalent in today's networks? The biggest impediment is incremental deployment in real networks. Today's networks are patchy with many local optimizations, and don't allow new network technologies to be easily integrated. The biggest challenge by far is not to come up with the next "better" algorithm, but instead explore how to seamlessly deploy RCP (or RCP like) congestion control in today's kludged networks.

On sudden changes in the network, we have seen in Chap. 5 that RCP provably converges to a stable state. While simulations show us that RCP converges as fast as 10 round trips, it turned out to be hard to prove this. It would be quite a relief to provably show that RCP recovers to a stable state in a reasonable amount of time.

Appendix A

A model for the 'U' curve

In this section we propose an approximate model that predicts the expected flow duration when the flow sizes have bounded Pareto distribution and the traffic in each flow is arriving at a rate, R. The cumulative distribution function of a bounded Pareto distribution is:

$$F(x) = \frac{1-(m/x)^{\alpha}}{1-(m/M)^{\alpha}};\ m \leq x \leq M,\ 0 < \alpha \leq 2$$

Following is the notation used in this section:

- C: Link capacity
- $R = \frac{C}{K}$: Flow rate (assume that K is an integer)
- $E[L]$ and $E[L^2]$: First and second moments of flow size
- $E[Q_l]$: Expected queue length as seen on the arrival of the last packet of a flow
- λ: Poisson flow arrival rate (flows/s)
- $\rho = \frac{\lambda E[L]}{C}$: Offered load
- $E[\tau]$: Expected flow duration

From Eqn. 3.5, the expected flow duration is given as

$$E[\tau] = \frac{E[L]}{R} + \frac{E[Q_l]}{C}. \tag{A.1}$$

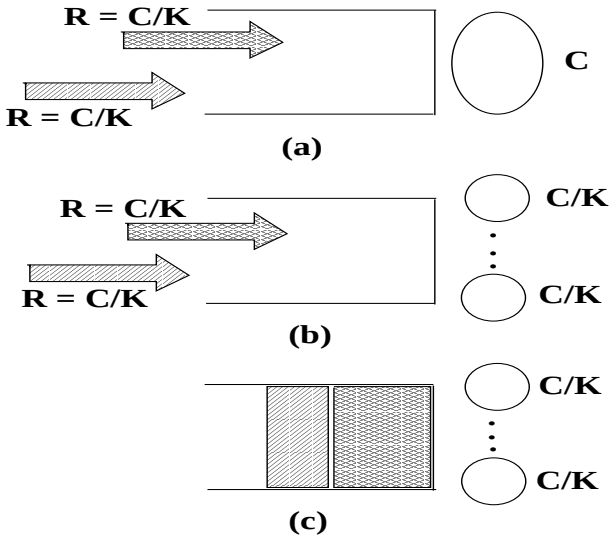


Figure A.1: The three systems: a) Our system of interest b) Flow-wise FCFS c) M/Pareto/K system.

So for a given rate, in order to the find the expected flow duration, it suffices to find the expected queue length seen by the last packet of a flow, which we will approximate as the expected queue length. To find the expected queue length we will consider two cases:

Case 1: R < C

Consider the three systems shown in Fig. A.1. Fig. A.1(a) represents our system of interest where flows arrive according to a Poisson process and each flow transmits data at rate R where $R = C/K$, $K > 1$. Let us assume for now that K is an integer. The buffer is served by an FCFS server of rate C. Fig. A.1(b) represents what we call a *Flow-wise FCFS* system. This system has K servers, each of capacity C/K and the flow arrival process is the same as that of system (a). Flows are queued according to when the first packet of a flow arrives. Whenever a server becomes free, it picks the first flow in queue that is not being served (if there are any) and serves the entire flow to completion. Note that a server will not serve a flow if all flows in the queue have started being served by other servers. Fig. A.1(c) represents the M/Pareto/K system, which is the same as (b) except each flow arrives as a single entity.

Systems (a) and (b) have the same arrival process for all time instants, and systems (b) and (c) have the same departure process for all time instants. Let $A_j(t)$ denote the amount of traffic that arrived into system j by time t and $D_j(t)$ denotes the amount of traffic served

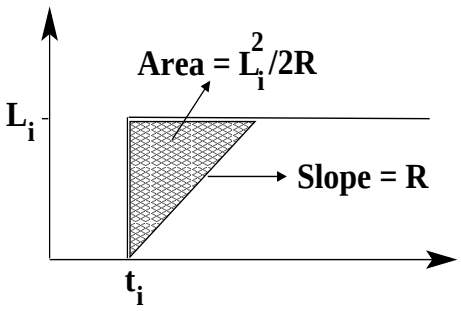


Figure A.2: Difference in queue occupancy in systems (b) and (c)

by system j up to time t. Assume that all systems are empty at time 0, then the expected queue size in system j is given by

$$E[Q_j] = \lim_{t\to\infty} \frac{\int_0^t (A_j(t) - D_j(t))dt}{t}. \tag{A.2}$$

Systems (a) and (b) have the same arrival process, but (a) is work-conserving while (b) is not, because an idle server in (b) cannot serve anything if all the flows in queue have started being served. Thus, we have $D_b(t) \leq D_a(t)$, $\forall t$. So, it follows from (A.2) that:

$$E[Q_a] \leq E[Q_b]. \tag{A.3}$$

Systems (b) and (c) have the same departure process, while $A_b(t) \leq A_c(t)$, $\forall t$ by setup. Thus:

$$E[Q_b] \leq E[Q_c]. \tag{A.4}$$

Since the systems (b) and (c) differ only in their arrival processes, we can find the difference between the expected queue lengths as the difference between the arrival processes:

$$E[Q_c] - E[Q_b] = \lim_{t\to\infty} \frac{\int_0^t (A_c(t) - A_b(t))dt}{t}.$$

Suppose $N(t)$ flows have arrived to the system till time t, then the contribution of flow i to the above difference, as shown in Fig. A.2, is $\frac{L_i^2}{2R}$. Summing over all flows gives:

$$E[Q_c] - E[Q_b] \quad = \quad \lim_{t\to\infty} \frac{\sum_{i=1}^{N(t)} \frac{L_i^2}{2R}}{t}$$

$$
\begin{aligned}
&= \lim_{t\to\infty} \frac{N(t)}{t} \frac{\sum_{i=1}^{N(t)} \frac{L_i^2}{2R}}{N(t)} \\
&= \lambda \frac{E[L^2]}{2R}. \qquad \text{(A.5)}
\end{aligned}
$$

From Eqns. A.3, A.4 and A.5, we have:

$$E[Q_a] \leq E[Q_b] = E[Q_c] - \lambda \frac{E[L^2]}{2R}. \qquad \text{(A.6)}$$

We will now obtain an approximation for $E[Q_c]$, the expected queue length for an M/Pareto/K system, using the results from [23]. In [23], the authors propose a simple model that accurately predicts the expected response time of a flow in a M/Pareto/K system, where the flows sizes have a bounded Pareto distribution. The expected flow response time in this system is approximated as:

$$E[D] \approx \frac{K\ E[L]}{C} + \frac{\rho}{1-\rho} \frac{E[L^2]}{2C\ E[L]} P(\text{blocking}), \qquad \text{(A.7)}$$

where

$$P(\text{blocking}) = 1 - F_{P(\rho_l K)}(K(1-\rho_s) - 1), \qquad \text{(A.8)}$$

$F_{P(\rho_l K)}(\cdot)$ denotes the value of the cumulative density function of a Poisson distribution with parameter $\rho_l K$,

$$
\begin{aligned}
\rho_s &= \frac{\alpha A}{E[L]}\rho, \\
\rho_l &= \frac{(1-\alpha)B}{E[L]}\rho, \\
A &= E[L] - \sqrt{(E[L^2] - (E[L])^2)\frac{1-\alpha}{\alpha}}, \\
B &= E[L] + \sqrt{(E[L^2] - (E[L])^2)\frac{\alpha}{1-\alpha}},
\end{aligned}
$$

and α is the percentage of flows with sizes between m and $M/20$.

Now, in order to obtain $E[Q_c]$, suppose $E[N]$ is the expected number of flows in the M/Pareto/K system. The probability the system is busy is approximately ρ.[1] If we assume

[1]The probability that system (a) is busy is ρ. Since system (c) is not work conserving the probability that the system is busy is higher than ρ. For simplicity, we shall assume that it is approximately ρ

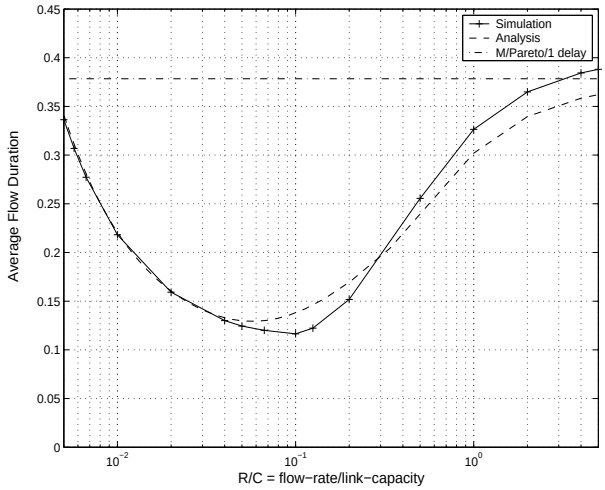


Figure A.3: Analysis and simulation plots for average delay of bounded Pareto distributed flow-sizes. Pareto distributed flow-sizes with mean 25pkts and shape 1.2, Flow arrivals is Poisson with rate 600 flows/sec, server capacity C=150Mbps.

that when the system is busy all the K servers are busy, we get an approximation for $E[Q_c]$:

$$E[Q_c] \approx \rho[(\frac{E[N]}{\rho} - K)E[L] + K\ E[L_r]], \tag{A.9}$$

where $E[L_r]$ is the expected residual file size and is equal to $\frac{E[L^2]}{2E[L]}$. Using Little's Law, $E[N]$ in the above expression is equal to $\lambda E[D]$ where an approximation to $E[D]$ is given in Eqn. A.7. Then, substituting for $E[D]$ in (A.9) and using (A.6) we have:

$$E[Q_a] \approx \lambda \frac{\rho}{2(1-\rho)} \frac{E[L^2]}{C} P(\text{blocking}), \tag{A.10}$$

where $P(\text{blocking})$ is given in (A.8).

Now we have the expected queue occupancy of our system. If we approximate the expected queue occupancy seen by the last packet of a flow, $E[Q_l]$, to be the expected queue occupancy, $E[Q_a]$, then substituting A.10 in A.1, we get:

$$E[\tau] \approx E[L]\frac{K}{C} + \lambda \frac{\rho}{2(1-\rho)} \frac{E[L^2]}{C^2} P(\text{blocking}). \tag{A.11}$$

Case 2: R $\geq$ C

Now let us consider the case when $R \geq C$. Consider the three systems as in Fig. A.1, but with $K = 1$ and $R \geq C$. Then, systems (a) and (b) have the same arrival and departure processes, therefore $E[Q_a] = E[Q_b]$. For systems (b) and (c), we again have $D_b(t) = D_c(t)$ and $A_b(t) \leq A_c(t)$. Thus, $E[Q_b] \leq E[Q_c]$. In the same way as in Case 1, the difference $E[Q_c] - E[Q_b]$ is found to be $\lambda \frac{E[L^2]}{2R}$. Thus,

$$E[Q_a] = E[Q_c] - \lambda \frac{E[L^2]}{2R},$$

where $E[Q_c]$ is the expected queue occupancy in an M/G/1 system and is equal to $\frac{\lambda E[L^2]}{2(1-\rho)C}$. So,

$$E[Q_a] = \frac{\lambda E[L^2]}{2} \left(\frac{1}{(1-\rho)C} - \frac{1}{R} \right).$$

Substituting $E[Q_a]$ in the delay Eqn. A.1 gives

$$E[\tau] = \frac{E[L]}{R} + \frac{\lambda E[L^2]}{2C} \left(\frac{1}{(1-\rho)C} - \frac{1}{R} \right). \tag{A.12}$$

Eqns. A.11 and A.12 model the expected flow duration, when $R < C$ and $R \geq C$ respectively. Plotting these delay equations verses R show that the expected flow duration has a unique minimum. Fig. A.3 shows an example comparing the average flow duration given by the model and the simulations.

Appendix B

Linearization of RCP rate equation

In this appendix we will linearize the RCP rate update equation. Equations describing the system are:

$$\begin{aligned} \dot{q}(t) &= NR(t-d_0) - C \\ d(t) &= \frac{q(t)}{C} + d_0 \\ \dot{R}(t) &= R(t-T)\left(\frac{\alpha(C - NR(t-d_0)) - \beta\frac{q(t)}{d(t)}}{Cd(t)}\right) \end{aligned} \tag{B.1}$$

We define:

$$\begin{aligned} f(R_T, R_d, q) &\doteq R_T(1 + \frac{\alpha(C - NR_d) - \beta\frac{q}{d}}{Cd}) \\ g(R_d) &\doteq NR_d - C \end{aligned} \tag{B.2}$$

where $R_T \doteq R(t-T)$, $R_d \doteq R(t-d_0)$, $d \doteq d(t)$ and $q \doteq q(t)$. Recall that the equilibrium point is given by:

$$\dot{q}(t) = 0 \Rightarrow NR_e = C \Rightarrow R_e = \frac{C}{N}$$

$$\dot{R}(t) = 0 \Rightarrow R_e(\frac{\alpha(C - NR_e) - \beta\frac{q_e}{d_e}}{Cd_e}) = 0 \Rightarrow q_e = 0$$

From above, the equilibrium value of $d(t)$ is $d_e = d_0$. Evaluating partials of f and g at the

equilibrium point $(R_e, q_e, d_e) = (\frac{C}{N}, 0, d_0)$ gives:

$$\begin{aligned}
\frac{\partial f}{\partial R_d} &= -\frac{NR_T\alpha}{Cd}\Big|_{\ R_T=\frac{C}{N},\ d=d_0} \\
&= -\frac{\alpha}{d_0} \\
\frac{\partial f}{\partial R_T} &= -\frac{\alpha}{d} - \frac{N\alpha R_d}{Cd} - \frac{\beta q}{d^2 C}\Big|_{\ R_d=\frac{C}{N},\ d=d_0,\ q=0} \\
&= 0 \\
\frac{\partial f}{\partial q} &= -\frac{\beta R_T(\frac{q}{C}-d)}{C(\frac{q}{C}+d)^3}\Big|_{R_T=\frac{C}{N},\ q=0,\ d=d_0} \\
&= -\frac{\beta}{Nd_0^2} \\
\frac{\partial g}{\partial R_d} &= N
\end{aligned}$$

The linearized equations are:

$$\begin{aligned}
\delta\dot{R}(t) &= \frac{\partial f}{\partial R_d}\delta R(t-d_0) + \frac{\partial f}{\partial R_T}\delta R(t-T) + \frac{\partial f}{\partial q}\delta q(t) \\
&= -\frac{\alpha}{d_0}\delta R(t-d_0) - \frac{\beta}{Nd_0^2}\delta q(t) \\
\delta\dot{q}(t) &= \frac{\partial g}{\partial R_d}\delta R(t-d_0) \\
&= N\delta R(t-d_0)
\end{aligned} \tag{B.3}$$

where

$$\begin{aligned}
\delta R &\doteq R - R_e \\
\delta q &\doteq q - q_e
\end{aligned} \tag{B.4}$$

Appendix C

Bode Plot Analysis

In this appendix we will see why we need the condition $\frac{\alpha}{\beta} > 1$, in order for Eqn. 5.11 to have a non-zero solution. Recall that if Eqn. 5.11 has a solution, ω_c, then this is the frequency at which the phase plot of $G(s)$ crosses the $-\pi$ line. In other words, at ω_c we have:

$$\angle G(j\omega_c) = -\omega_c d_0 + \arctan(\frac{\omega_c \alpha d_0}{\beta}) - \pi = -\pi$$

Notice that $\angle G(j\omega) = -\pi$ at $\omega = 0$. And for large ω, $\angle G(j\omega)$ is much smaller than $-\pi$. So, unless $\angle G(j\omega)$ first increases and then decreases, as ω increases from 0, it will not cross the $-\pi$ line. Thus, the condition is that there should exist a maxima for $\angle G(j\omega)$. Differentiating $\angle G(j\omega)$ and setting it to 0 gives:

$$\frac{d}{d\omega}\angle G(j\omega_m) = -d_0 + \frac{\alpha d_0}{\beta}\frac{1}{1 + (\frac{\omega_m \alpha d_0}{\beta})^2} = 0$$

$$\Rightarrow \omega_m = \frac{\beta}{\alpha d_0}\sqrt{\frac{\alpha}{\beta} - 1}$$

Obviously, the above maxima exists only if $\frac{\alpha}{\beta} > 1$.
Thus, if the condition $\frac{\alpha}{\beta} > 1$ is satisfied then the the phase plot crosses the $-\pi$ line. Examples: Fig. C.1 shows a Bode Plot for $(\alpha, \beta) = (0.2, 0.4)$. Notice that the phase always decreases starting from $-\pi$, and never crosses the $-\pi$ line for any non-zero ω. Hence ω_c does not exist. Fig. C.2 shows a Bode Plot for the case $(\alpha, \beta) = (0.4, 0.2)$ and in this case, since $\frac{\alpha}{\beta} > 1$, ω_c exists.

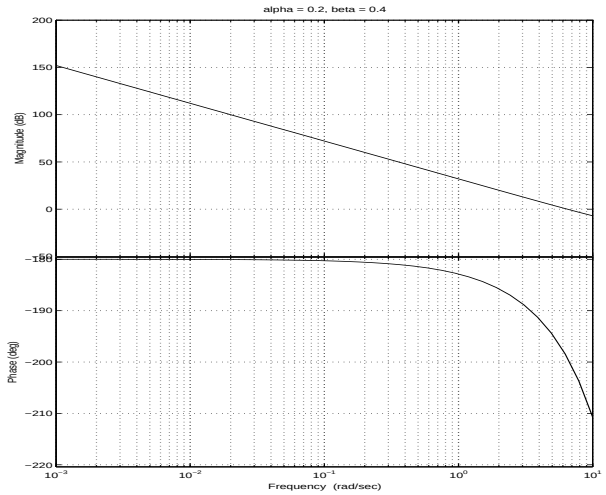


Figure C.1: $(\alpha, \beta) = (0.2, 0.4)$: ω_c does not exist

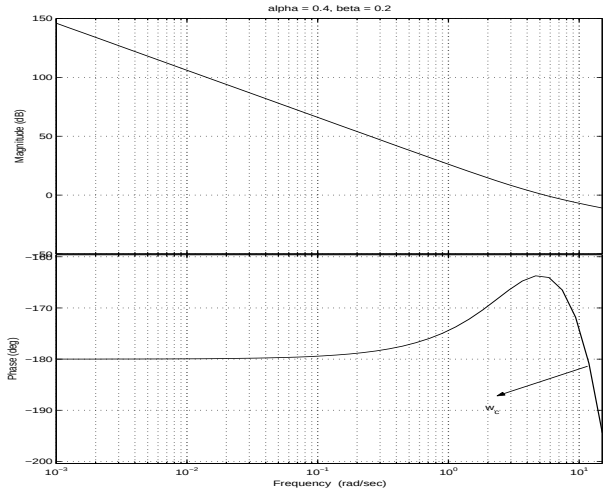


Figure C.2: $(\alpha, \beta) = (0.4, 0.2)$: ω_c exists

Appendix D

The Nyquist Stability Analysis

In the last Section we obtained the stability region from the Bode plot analysis. There are some conditions to be satisfied for the Bode analysis to hold. Specifically, the stability criterion, $|G(j\omega)| < 1$ at $\angle G(j\omega) = -\pi$, holds for systems where $|G(j\omega)|$ crosses the magnitude $= 1$ line once, the most common situation. However, there are systems when the $|G(j\omega)|$ crosses magnitude $= 1$ more than once. A rigorous way to resolve these ambiguities is to use the Nyquist stability criterion. So, in this Section we will use the Nyquist criterion and confirm the stability region obtained before.

Recall that the open loop transfer function of our system is given by $G(s) = e^{-sd_0}(\alpha s d_0 + \beta)/(sd_0)^2$. The closed loop transfer function is $G(s)/(1 + G(s))$. Therefore, the closed loop roots are the solutions of $1 + G(s) = 0$ i.e.:

$$d_0^2 s^2 + \alpha d_0 e^{-sd_0} s + \beta e^{-sd_0} = 0 \tag{D.1}$$

We will write $d_0^2 s^2 + \alpha d_0 e^{-sd_0} s + \beta e^{-sd_0} = 0$ in the form $1 + \alpha \frac{b(s)}{a(s)} = 0$. This is given by:

$$1 + \alpha \frac{d_0 e^{-sd_0} s}{d_0^2 s^2 + \beta e^{-sd_0}} = 0$$

i.e. $b(s) = d_0 e^{-sd_0} s$ and $a(s) = d_0^2 s^2 + \beta e^{-sd_0}$. Let $G_1(s)$ denote $\frac{d_0 e^{-sd_0} s}{d_0^2 s^2 + \beta e^{-sd_0}}$. Then, the procedure for obtaining the stability region can be summarized as follows:

1. Fix a value of β. Plot the nyquist plot of $G_1(s)$

2. Determine the number of unstable (i.e. Right Hand Plane) poles of $G_1(s)$ and call that number P.

3. Determine the region on the real axis where $-\frac{1}{\alpha}$ should lie such that if N denotes the number of encirclements of $-\frac{1}{\alpha}$, then $N + P$ should be equal to zero. $Z = N + P$ are the number of unstable closed loop roots, and therefore we want Z to be equal to 0. Note that, N is negative if the encirclement of $-\frac{1}{\alpha}$ is in anti-clockwise direction and positive if in clockwise direction.

To draw the nyquist plot of $G_1(s)$, we will use the Pade approximation for e^{-sd_0}.[1] Substituting this in $G_1(s)$ we get:

$$G_1(s) = \frac{\frac{d_0^3}{12}s^3 - \frac{d_0^2}{2}s^2 + d_0 s}{\frac{d_0^4}{12}s^4 + \frac{d_0^3}{2}s^3 + (d_0^2 + \beta\frac{d_0^2}{12} - \frac{d_0\beta}{2})s + \beta}$$

Let's go through the three steps above with an example. Let us take $\beta = 0.3$. Now, we want to find the range of α for which the system is stable. The nyquist plot of $G_1(s)$ is shown in Fig. D.1 for $\beta = 0.3$. The value of d_0 taken does not matter, the plot does not change with d_0. $G_1(s)$ has two unstable poles and so $P = 2$. Therefore, in order for Z to be equal to 0 we need $N = -2$, which means $-\frac{1}{\alpha}$ must be encircled twice in anti-clockwise direction in the plot. The only region in the plot where there are two anti-clockwise encirclements is between points A and B shown in the figure. So the stable region is $-2.91 < -\frac{1}{\alpha} < -0.696$ i.e. $0.3436 < \alpha < 1.436$ when $\beta = 0.3$. And thus continuing, we can obtain the stable range for α for every value of β. When β is larger than about 0.55 there does not exist any value of α for which the system is stable. An example of such a nyquist plot is shown in Fig. D.1 (right plot). As can be seen, there is no region on the plot where there are two anti-clockwise encirclements, and hence we cannot get Z to be equal to 0, and therefore the closed loop system will have at least one unstable pole for this value of β.

The stable region obtained from the procedure above is shown in Fig. 5.2. The region obtained from the nyquist analysis is shown in + signs, while that obtained from the Bode plot analysis is shown in solid line. As can be seen, the nyquist analysis confirms the region that we obtained.

[1] $e^{-sd_0} \approx (1 - \frac{sd_0}{2} + \frac{(sd_0)^2}{12})/(1 + \frac{sd_0}{2} + \frac{(sd_0)^2}{12})$

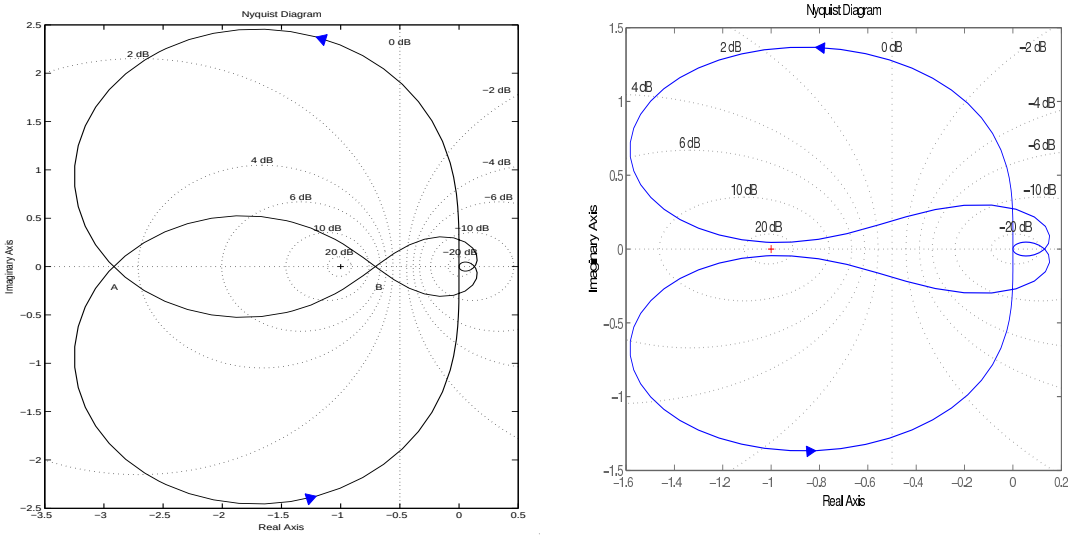


Figure D.1: Left plot: Nyquist plot of $G_1(s)$ when $\beta = 0.3$. Right plot: Nyquist plot of $G_1(s)$ when $\beta = 0.6$.

Appendix E

Phase Plane Analysis

In this section we will obtain the stability region of the RCP parameters, taking into account the non-linearity in the queue length equation. We will use a method called *phase-plane analysis*, which is used to solve and understand nonlinear control problems. We will first give a brief introduction to this method. The RCP system described by Eqns. 5.1-5.4, can be written as

$$\begin{aligned} \dot{R}(t) &= f(R(t-T), R(t-d_0), q(t), N, C) \\ \dot{q}(t) &= g(R(t-d_0), N, C) \end{aligned} \tag{E.1}$$

where $f(.)$ and $g(.)$ are non-linear functions. If we take R and q as coordinates of a plane, then to each state of the system there corresponds a point in this plane. As t varies, this point describes a curve in the R-q plane, indicating the history of the system dynamics. Such a curve is called a trajectory. The geometrical representation of the system behavior in terms of trajectories is called a phase-plane representation of the system dynamics. The initial condition determines the initial location of a representative point on the trajectory. As time increases, the representative point moves along the trajectory. A family of such trajectories is called a phase-plane portrait.

We will now plot the phase-plane portraits of the RCP system, and determine the stability region from these plots. The phase portraits are generated by a C program which given an initial state (R, q) does a step-by-step evolution of the RCP system state from Eqns. 5.1-5.4. Following is the set up for obtaining the phase portraits:

$C = 0.15$ Gbps, $d_0 = 0.2$s, $T = 0.01$s and $N = 10$ flows

The initial conditions are chosen to be: $\gamma_{init} = \{0.5, 0.1, 0.01\}$ where γ is the normalized flow rate i.e. $\gamma = \frac{R}{C}$, and $q_{init} = \{2 \cdot C \cdot d_0, 0\}$. The trajectories are drawn for all combinations of $(\gamma_{init}, q_{init})$. α and β vary from $(0.1, 2)$ in steps of 0.1

For each value of (α, β) we get a family of trajectories corresponding to each of the initial conditions chosen. Note that the equilibrium state of the system is $(\gamma, q) = (0.1, 0)$. So, for a given (α, β), if for any of the initial conditions the trajectory does not finally end at the equilibrium state we can conclude that this point does not lie in the stable region. Figs. E.1, and E.2 show some sample phase portraits. Fig. E.1 shows the phase portrait for $(\alpha, \beta) = (0.6, 0.2)$. This point is well within the linearized stability region and as seen in this figure, all the trajectories converge to the equilibrium point. Fig. E.1 also shows the phase portrait when $(\alpha, \beta) = (0.6, 0.8)$, which is a point outside the linearized stability region but within our hypothesized stable region. Here too all the trajectories converge to the equilibrium point. The left plot in Fig. E.2 shows phase portrait for $(\alpha, \beta) = (1.4, 0.8)$, a point well outside the stable region. The right plot in Fig. E.2 shows that for the initial condition $(\gamma_{init}, q_{init}) = (0.01, 2 \cdot C \cdot d_0)$, the trajectory never settles down at the equilibrium. It continuously keeps oscillating around the equilibrium point. The magnitude of these oscillations only grow larger as we get farther away from the stable region. Finally, Fig. E.3 shows the stable region obtained from the phase-plane analysis. The stable regions obtained from the linearized analysis and simulations are also shown for the purposes of comparison. As can be seen, the region obtained from the phase-plane analysis matched well with that obtained via the simulations.

The stable region in Fig. E.3 holds true for varying C, d_0 and N. The range that we tried are:

$C = \{5.6Kbps, 0.15Gbps, 2.5Gbps, 10Gbps, 1000Gbps\}$

$d_0 = \{0.01, 0.5, 1, 2\}$

$N = \{10, 100, 1000, 5000\}$

Under all these varying conditions the stable region that we obtained remained the same, as shown in Fig. E.3.

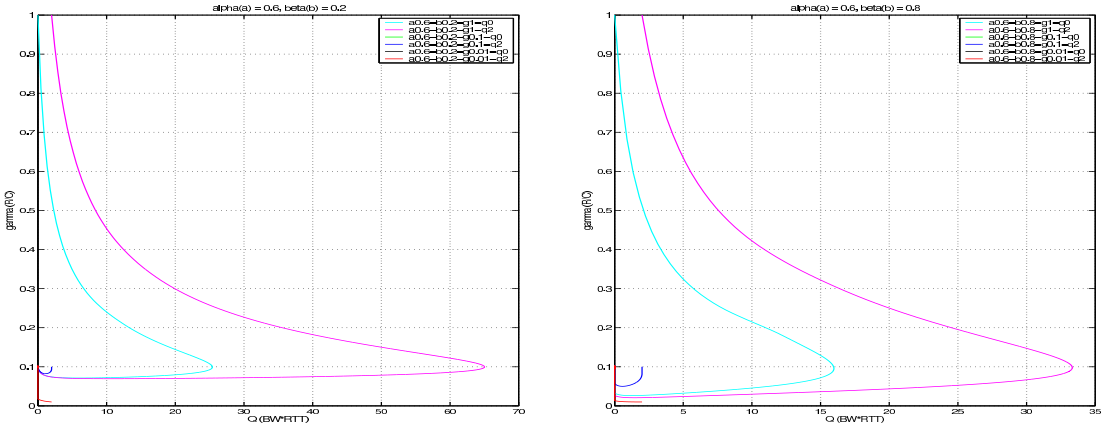


Figure E.1: Left plot: Phase Portrait for $\alpha = 0.6, \beta = 0.2$. Right plot: Phase Portrait for $\alpha = 0.6, \beta = 0.8$.

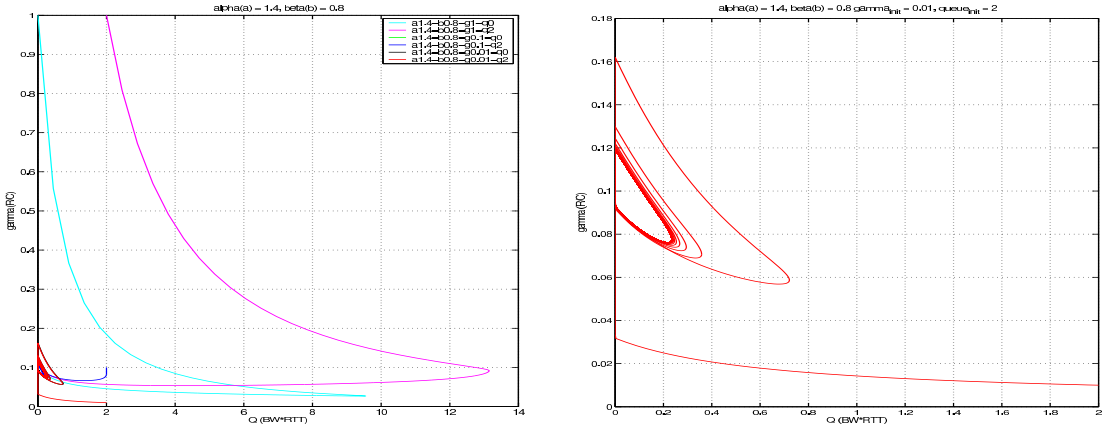


Figure E.2: Left plot: Phase Portrait for $\alpha = 1.4, \beta = 0.8$. Right plot: Phase Portrait for $\alpha = 1.4, \beta = 0.8$. $\gamma_{init} = 0.01, queue_{init} = 2 \cdot C \cdot d_0$

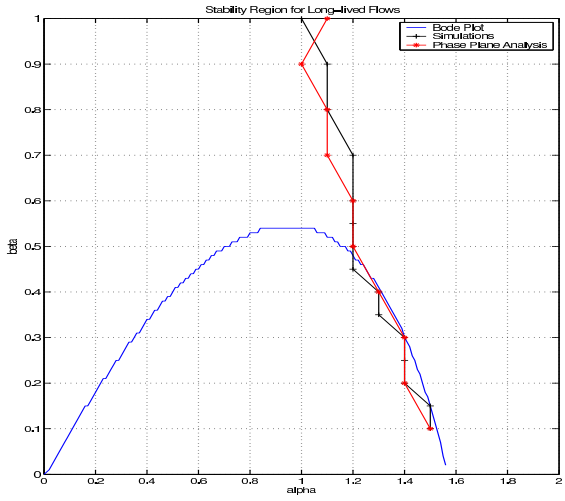


Figure E.3: Stable Region obtained from linearization model, Simulations and Phase Plane analysis

Appendix F

RCP Router Specification

F.1 Router calculations to be performed periodically

The router is intended to periodically (approximately once per average RTT of sessions passion through it, but more frequently if appropriate) calculate how much bandwidth it can allocate to the average data flow. The way it accomplishes this is described in Chap. 3.

When the Rate Estimation Timer expires, the router updates the rate as follows:

```
1. input_traffic_rate = input_traffic_Bytes/Tr
2. avg_rtt_Tr = sum_rtt_Tr/num_pkts_with_rtt
3. if (avg_rtt_Tr >= avg_rtt)
4.   rtt_sample_weight = (Tr/avg_rtt)
5. else
6.   rtt_sample_weight = (rcp_rate/link_rate) * (avg_rtt_Tr/avg_rtt)
                                              * (Tr/avg_rtt)
7. avg_rtt = rtt_sample_weight*avg_rtt_Tr + (1 - rtt_sample_weight) * avg_rtt
8. rcp_rate = rcp_rate * (1 + ((Tr/avg_rtt) * (ALPHA * (ETA * link_rate
     - input_traffic_rate) - BETA*Q_Bytes/avg_rtt))/(ETA*link_rate))
9. if (rcp_rate < MIN_RATE)
10.   rcp_rate = MIN_RATE
11. else if (rcp_rate > ETA*link_rate)
12.   rcp_rate = ETA*link_rate
13. Tr = min(avg_rtt, MAX_RATE_ESTIMATION_INTERVAL)
14. input_traffic_Bytes = 0
```

```
15. num_pkts_with_rtt = 0
16. sum_rtt_Tr = 0
17. schedule_rate_timer(Tr)
```

Before we explain what the code does, following is a glossary of the notation used above. All variables below are for a particular outgoing interface. Variables that are measured in the units of time - in milliseconds unless mentioned otherwise.

1. Tr: The Rate Estimation Interval or how often the router updates the bandwidth offered to the flows for an outgoing interface. It is also the interval over which the input statistics for the RCP algorithm (the aggregate incoming traffic rate for an interface and average of the round-trip time values carried in RCP packet headers) are gathered.

2. sum_rtt_Tr: The sum of round-trip time values seen in an interval Tr.

3. avg_rtt_Tr: The average of round-trip time values seen over all RCP packets in an interval Tr.

4. avg_rtt: The moving average of the round-trip time maintained by the router, that is updated periodically once every Tr with an appropriate weight attached to the latest sample (avg_rtt_Tr).

   Variables measured in units of traffic-volume (Bytes, number of packets) or bandwidth (Bytes/ms):

5. input_traffic_Bytes: is the aggregate amount of incoming RCP traffic (in Bytes) for an output interface in an interval Tr. This includes packets that are dropped due to a full buffer at the output interface.

6. input_traffic_rate: The aggregate amount of incoming traffic bandwidth (in Bytes/ms) over the interval Tr .

7. num_pkts_with_rtt: The number of packets in interval Tr that carry valid round-trip time values. Ideally, this should include packets that are dropped due to a full buffer at the output interface.

8. rcp_rate: This is the bandwidth offered to a flow and is updated periodically once every Tr.

9. Q_Bytes: The buffer occupancy at the output interface in Bytes.

   Constants and dimensionless variables:

10. rtt_sample_weight: In updating avg_rtt, this is the weight given to the most recent RTT sample (avg_rtt_Tr).

11. ALPHA, BETA, ETA: Constants whose recommended values are discussed later in this document.

12. MIN_RATE: The minimum value for rcp_rate.

13. MAX_RATE_ESTIMATION_INTERVAL: The maximum value for Tr - the time interval between any two updates for rcp_rate.

14. link_rate: Link bandwidth measured in Bytes/ms.

Line 1: The aggregate incoming traffic rate is computed by dividing the amount of traffic (in bytes) that arrived in an estimation interval by the interval length, Tr.

Line 2: The average round-trip time of traffic that arrived in the current estimation interval is the sum of the RTTs carried in the packets divided by the number of packets carrying a valid RTT.

Lines 3, 4, 5 and 6: The round-trip time sample (computed in Line 2) is used to update the router's estimate of the average round-trip time using a simple exponentially moving average. These lines decide how much weight the sample should receive in the moving average. Because there are approximately avg_rtt/Tr RTT samples in an average round-trip time, the current sample should receive a weight of at most Tr/avg_rtt. On the other hand when the current RTT sample is smaller than the router's RTT estimate we want to conservatively age the RTT estimate by additionally weighing the current sample by avg_rtt_Tr/avg_rtt . This ensures stability in links shared by flows with vastly different round-trip times.

Line 7: Updates the router's RTT estimate (avg_rtt) using an exponentially moving average.

Line 8: This line updates the RCP rate that will be offered to flows over the next rate estimation interval. The theoretical basis for this equation is described in Chap. 3. The values of ALPHA and BETA for a stable network are derived in Chap. 5. The recommended

values for stability and performance are in the range: ALPHA $\in (0.4, 0.6)$ and BETA $\in (0.2, 0.6)$.

ETA controls the target link-utilization and can be any value in the range $0.95 <$ ETA $<$ 1. Choosing a value leass than 1 gives some headroom to drain excess traffic before building up a queue.

Lines 9, 10, 11 and 12: These lines cap the RCP rate above to ETA * link_rate and below to MIN_RATE. The recommended value of MIN_RATE is (0.01 * MTU size of the link)/avg_rtt. This value is usually only reached in scenarios of extreme congestion.

Line 13: Decides the length of the next rate estimation interval. The RCP rate should be updated at least once per average round-trip time. When updated more often it provides a faster response to congestion. The recommended value of MAX_RATE_ESTIMATION_INTERVAL is 10 ms.

Lines 14, 15 and 16: Reset the variables. The data path will update them over the next interval.

Line 17: Restart the rate estimation timer.

F.2 Router calculations to be performed per-packet

The following computations are performed on every packet that carries an RCP header. Some calculations are done on packet arrival and some on packet departures. In the worst-case the total number of per-packet computations are three additions and two comparisons.

F.2.1 Router calculations to be performed on packet arrival

```
18. input_traffic_Bytes += packet_size_Bytes
19. if (this_packet_RTT < MAX_ALLOWABLE_RTT)
20.   sum_rtt_Tr += this_packet_RTT
21.   num_pkts_with_rtt += 1
```

The notation used above is defined below.

1. this_packet_RTT: The RTT value carried in the RCP header.
2. MAX_ALLOWABLE_RTT: A constant that this_packet_RTT is compared to.
3. pkt_size_Bytes: The IP datagram size in Bytes.

Line 18: Updates the total amount of incoming traffic in the current interval. The packet-size is taken from the IP header.

Line 19, 20 and 21: If the packet carries a valid round-trip time, its RTT value is added to the running sum and the number of packets carrying a valid RTT estimate is incremented. MAX_ALLOWABLE_RTT serves a couple of purposes: a) packets with unrealistic round-trip times should not skew the router's avg_rtt arbitrarily and b) As mentioned before, a packet with an unknown or unspecified RTT has all ones in its RTT field (this is a value > MAX_ALLOWABLE_RTT). The recommended value for MAX_ALLOWABLE_RTT is 20 seconds.

F.2.2 Router calculations to be performed before packet departure

```
22. if (packet_BW_Request > rcp_rate)
23.   packet_BW_Request = rcp_rate
```

Lines 22 and 23: If the rate being requested in any given packet is unspecified or exceeds the bandwidth being granted to data flows in the specified class, then it should be set to the RCP rate.

Bibliography

[1] W. R. Stevens, "TCP/IP Illustrated, Volume 1: The Protocols.," Addison Wesley, 1994. 4

[2] W. Noureddine, "Improving the Peformance of TCP Applications Using Network-Assisted Mechanisms," Ph.D. Thesis, Stanford University, June 2002. 4, 5

[3] E. He, P. V. Primet, M. Welzl, M. Goutelle, Y. Gu, S. Hegde, R. Kettimuthu, J. Leigh, C. Xiong, M. M. Yousaf, "A Survey of Transport Protocols other than Standard TCP," *Global Grid Forum Document GFD.55*, Data Transport Research Group, 23 November 2005. 4

[4] M. Duke, R. Braden, W. Eddy, E. Blanton, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents," *http://www.ietf.org/rfc/rfc4614.txt*, RFC 4614, September 2006. 4

[5] D. D. Clark, S. Shenker, A. Falk, "GENI Research Plan," *http://www. geni. net/GDD/GDD-06-28.pdf GENI Design Document 06-28*, Research Coordination Working Group, April 2007. 1

[6] A. M. Odlyzko, "The Internet and other networks: Utilization rates and their implications," *Information Economics and Policy*, 12 (2000), Pages 341-365. 16, 23

[7] A. M. Odlyzko, "Internet TV: Implications for the long distance network," *Internet Television*, E. Noam, J. Groebel, and D. Gerbarg, eds., Lawrence Erlbaum Associates, 2003, pp. 9-18. 16

[8] D. A. Patterson, "Latency Lags Bandwidth," *Communications of the ACM*, Volume 47, Number 10 (2004), Pages 71-75. 17

[9] J. Du, J. Y. Leung, G. H. Young, "Minimizing mean flow time with release time constraint," *Theoretical Computer Science archive*, Volume 75, Issue 3, October 1990. 17, 26

[10] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, Volume 1, Pages 343-362, 1977. 17, 26

[11] D. Katabi, M. Handley, C. Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002. 5, 11, 18, 47

[12] http://www.ana.lcs.mit.edu/dina/XCP/ 47

[13] A. Falk, D. Katabi, Y. Pryadkin, "Specification for the Explicit Control Protocol (XCP)," *draft-falk-xcp-03.txt*, July 2007. 12

[14] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, "Processor Sharing Flows in the Internet," *Thirteenth International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005. 28, 46

[15] N. Dukkipati, N. McKeown, "Processor Sharing Flows in the Internet," *Stanford HPNG Technical Report TR04-HPNG-061604*, http://yuba.stanford.edu/tr.html, June 2004. 28, 46

[16] N. Dukkipati, N. McKeown, "Why Flow-Completion Time is the Right metric for Congestion Control and why this means we need new algorithms," *Stanford HPNG Technical Report TR05-HPNG-112102*, http://yuba.stanford.edu/tr.html, November 2005. 16

[17] N. Dukkipati, N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control," *ACM SIGCOMM Computer Communication Review*, Volume 36, Issue 1, January 2006. 16

[18] N. Dukkipati, G. Gibb, N. McKeown, J. Zhu, "Building an RCP (Rate Control Protocol) Test Network," in *Hot Interconnects 15*, Stanford, August 2007. 87, 99

[19] C. H. Tai, J. Zhu, N. Dukkipati, N. McKeown, "Making large-scale deployment of RCP practical for real networks," *High Performance Networking Group Technical Report TR07-HPNG-062207*, Stanford University, June 2007. 87, 93, 95

[20] N. Dukkipati, N. McKeown, A. G. Fraser "RCP-AC: Congestion Control to make flows complete quickly in any environment," *High-Speed Networking Workshop: The Terabits Challenge (In Conjunction with IEEE Infocom 2006)*, Barcelona, Spain, April 2006. 117

[21] L. E. Schrage, L. W. Miller, "The Queue M/G/1 with the Shortest Remaining Processing Time Discipline", *Operations Research*, Volume 14, Number 4 (July. - Aug., 1966), Pages 687-690. 24, 27

[22] L. E. Schrage, "A proof of the optimality of the shortest remaining processing time discipline", *Operations Research*, Volume 16, Pages 687690, 1968. 25

[23] K. Psounis, P. Molinero-Fernandez, B. Prabhakar, F. Papadopoulos, "Systems with multiple servers under heavy-tailed workloads," *Journal of Performance Evaluation*, Elsevier, Vol. 62, Number 1-4, pp. 456-474, 2005. 126

[24] The Network Simulator, http://www.isi.edu/nsnam/ns/ 19, 46

[25] S. Floyd, "HighSpeed TCP for Large Congestion Windows," *RFC 3649*, http://www.icir.org/floyd/hstcp.html, December 2003. 5, 9, 23, 52

[26] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window," *RFC 3390*, http://www.ietf.org/rfc/rfc3390.txt, October 2002. 3

[27] T. Kelly, "Scalable TCP: improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, Volume 33 , Issue 2, April 2003. 9, 23

[28] S. Floyd, "Setting Parameters for RED," http://www.icir.org/floyd/red.html. 25

[29] R. W. Wolff, "Stochastic Modeling and the Theory of Queues," Prentice Hall, 1989. 26, 47

[30] M. E. Crovella, A. Bestavros, "Self Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, December 1997. 30, 38, 47

[31] S. Ben Fredj, T. Bonald, A. Proutiere, G. Regnie, J.W. Roberts, "Statistical Bandwidth Sharing: A Study of Congestion at Flow Level," *Proceedings of ACM Sigcomm 2001*, San Diego, August 2001. 47

[32] V. Paxson, S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, June 1995. 47

[33] B. Wydrowski, M. Zukerman, "MaxNet: A congestion control architecture," *IEEE Communications Letters*, Volume 6, Issue 11, Nov 2002, Page(s): 512 - 514. 34

[34] R. Srikant, "The Mathematics of Internet Congestion Control," University of Illinois, Urbana, IL. 69

[35] J. Paddy, V. Firoiu, D. Towsley, J. Kurose "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *Proceedings of ACM Sigcomm 1998*, British Columbia, Canada, August 1998. 5

[36] S. H. Low, D. E. Lapsley, "Optimization Flow Control, I: Basic Algorithm and Convergence," *IEEE/ACM Transactions on Networking*, 7(6):861-75, Dec. 1999.

[37] S. H. Low, F. Paganini, J. Wang, S. Adlakha, J. C. Doyle, "Dynamics of TCP/RED and a Scalable Control," *Proceedings of IEEE Infocom 2002*, New York, June 2002. 69

[38] S. H. Low, F. Paganini, J. Wang, J. C. Doyle, "Linear stability of TCP/RED and a Scalable Control," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Volume 43, Issue 5, December 2003. 71

[39] E. Altman, T. Basar, R. Srikant, "Robust Rate Control for ABR Sources," *Proceedings of IEEE Infocom 1998*, San-Fransisco, March 1998. 119

[40] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, B. Vandalore, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks," *IEEE/ACM Transactions on Networking*, Vol. 8, No. 1, February 2000, pp. 87-98. 119

[41] A. Charny, "An Algorithm for Rate Allocation in a Packet Switching Network With Feedback," MS Thesis, MIT, April 1994. 119

[42] C. Jin, D. X. Wei, S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *Proceedings of IEEE Infocom 2004*, Hong Kong, March 2004. 9

[43] V. Jacobson, "Congestion Avoidance and Control," *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 314-329, 1988. 4

[44] C. Villamizar, C. Song, "High Performance TCP in ANSNET," *ACM Computer Communication Review*, Vo1. 24, No. 5, October 1994. 47

[45] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing Router Buffers," *ACM SIGCOMM 2004*, Portland, August 2004. 47

[46] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, "Routers with very small buffers", *Proceedings of the IEEE INFOCOM '06*, Barcelona, Spain, April 2006. 47

[47] M. Parulekar, "Buffer Engineering for Self Similar Traffic," Ph.D. Thesis, Electrical Engineering Department, University of Maryland, College Park, 1999. 38

[48] D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, 1992. 35, 38

[49] S. Floyd, M. Allman, A. Jain, P. Sarolahti, "Quick-Start for TCP and IP," *RFC 4782*, http://www.icir.org/floyd/papers/rfc4782.txt, January 2007. 2

[50] K. Psounis, A. Ghosh, B. Prabhakar, G. Wang, "SIFT: a simple algorithm for tracking elephant flows and taking advantage of power laws," *43rd Allerton Conference on Communication, Control, and Computing*, September 2005. 43

[51] L. Xu, K. Harfoush, I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," *Proceedings of IEEE Infocom 2004*, Hong Kong, March 2004. 9

[52] I. Rhee, L. Xu, S. Ha, "CUBIC for Fast Long-Distance Networks," http://www.ietf.org/internet-drafts/draft-rhee-tcp-cubic-00. txt, February 27, 2007. 9

[53] R.N. Shorten, D.J. Leith, "H-TCP: TCP for high-speed and long-distance networks," *Proceedings of PFLDnet 2004*, Argonne, 2004. 9

[54] K. Tan, J. Song, Q. Zhang, M. Sridharan, "Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks," *Proceedings of PFLDnet 2006*, Nara (Japan), 2006. 9

[55] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *Proceedings of ACM Mobicom 2001*, pp 287-297, Rome, Italy, July 16-21 2001. 9

[56] http://www-iepm.slac.stanford.edu/monitoring/bulk/sc2004/ 9

[57] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, Vol. 26 No. 3, July 1996, pp. 5-21. 5, 47

[58] Y. Tian, K. Xu, N. Ansari, "TCP in Wireless Environments: Problems and Solutions," *IEEE (Radio) Communications Magazine*, Vol. 43, No. 3, pp. S27 - S32 , March 2005. 6

[59] T.R. Henderson, R.H. Katz, "TCP Performance over Satellite Channels," UCB Computer Science Technical Report 99-1083, December 1999. 6

[60] S. Floyd, K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, August 1999. 8

[61] http://flickr.com 67

[62] G. Franklin, J. D. Powell, A. Emami-Naeini, "Feedback Control of Dynamic Systems," 5/E Prentice Hall, 2006. 71, 72

[63] H. Balakrishnan, N. Dukkipati, N. McKeown, C. Tomlin, "Stability Analayis of Explicit Congestion Control Protocols," *IEEE Communications Letters*, 2007. 75, 76

[64] H. Balakrishnan, N. Dukkipati, N. McKeown, C. Tomlin, "Stability Analayis of Explicit Congestion Control Protocols," Technical Report SUDAAR 776, http://yuba.stanford.edu/rcp/SUDAAR_776.pdf, September 2005. 75, 76, 77

[65] *Private communication with Lachlan Andrew*, Senior Research Engineer, Department of Computer Science, California Institute of Technology (Caltech), September 2006. 78

[66] M. Casado, G. Watson, N. McKeown, "Reconfigurable Networking Hardware: A Classroom Tool," *Hot Interconnects 13*, Stanford, August 2005. 96

[67] M. Casado, G. Watson, N. McKeown, "Teaching Networking Hardware," *ITiCSE, Monte de Caparica*, Portugal, June 2005. 96

[68] "NetFPGA Web Page," *http://NetFPGA.org*. 87, 96

[69] A. Medina, S. Floyd, M. Allman, "Measuring Evolution of Transport Protocols in the Internet," *ACM Computer Communications Review*, April 2005. 88

[70] I. McDonald, R. Nelson, "Congestion Control Advancements in Linux," *linux.conf.au*, January 2006. 89

[71] "Network Performance Measuring Tool: iperf," *http://dast.nlanr.net/Projects/Iperf/*. 94

[72] "Network Emulation," *http://linux-net.osdl.org/index.php/Netem/*. 94

[73] C. Estan and G. Varghese. "New directions in traffic measurement and accounting," *Proceedings of the ACM SIGCOMM 2002*, October 2002. 102

[74] T.J. Ott and T. V. Lakshman and L.H. Wong, "SRED: Stabilized RED" *Proceedings of INFOCOM '99*, 1999. 102

[75] A. Lakshmikantha, N. Dukkipati, R. Srikant, N. McKeown, C. Beck, "Performance Analysis of RCP," Technical Report available at http://www.ifp.uiuc.edu/lkshmknt, 2006. 108, 109, 113, 114

[76] RCP Web Page. http://www.yuba.stanford.edu/rcp 109

[77] N. Dukkipati, Y. Ganjali, R. Zhang-Shen, "Typical versus Worst Case Design in Networking," *Fourth Workshop on Hot Topics in Networks*, College Park, Maryland, November 2005. 116

[78] Global Environment for Network Innovations (GENI) Web Page. http://geni.net/ 122

[79] N. Bansal, M. Harchol-Balter, "Analysis of SRPT scheduling: investigating unfairness," *Proceedings of the ACM SIGMETRICS 2001*, Cambridge, Massachusetts, 2001. 27

[80] M. Nuyens, A. Wierman, "The foreground-background queue: a survey," *Performance Evaluation*, 2007. 26

[81] F. Baker, "Requirements for IP Version 4 Routers," *RFC 1812*, http://www.faqs.org/rfcs/rfc1812.html, June 1995. 91

[82] T. Anderson, A. Collins, A. Krishnamurthy, J. Zahorjan, "PCP: Efficient Endpoint Congestion Control," *NSDI*, 2006. 120

[83] F. Kelly, G. Raina, T. Voice, "Stability and fairness of explicit congestion control with small buffers," *http://www.statslab.cam.ac.uk/~frank/PAPERS/KRV/krvpaper.pdf*, 2007. 120

[84] C. Fulton, S.-Q. Li, "UT: ABR feedback control with tracking," *Proceedings of IEEE Infocom '97*, Kobe, Japan. 119