# Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks

by

Dina Katabi

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

March 2003



Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
March 7, 2003

Certified by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
David Clark
Senior Research Scientist
Thesis Supervisor

Accepted by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

**Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks**

by

Dina Katabi

Submitted to the Department of Electrical Engineering and Computer Science
on March 7, 2003, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

## Abstract

In this dissertation, we propose a new architecture for Internet congestion control that decouples the control of congestion from the bandwidth allocation policy. We show that the new protocol, called XCP, enables very large per-flow throughput (e.g., more than 1 Gb/s), which is unachievable using current congestion control. Additionally, we show via extensive simulations that XCP significantly improves the overall performance, reducing drop rate by three orders of magnitude, increasing utilization, decreasing queuing delay, and attaining fairness in a few RTTs. Using tools from control theory, we model XCP and demonstrate that, in steady state, it is stable for any capacity, delay, and number of sources. XCP does not maintain any per-flow state in routers and requires only a few CPU cycles per packet making it implementable in high-speed routers. Its flexible architecture facilitates the design and implementation of quality of service, such as guaranteed and proportional bandwidth allocations. Finally, XCP is amenable to gradual deployment.

Thesis Supervisor: David Clark
Title: Senior Research Scientist

## Acknowledgments

I have been fortunate to have Dave Clark as my adviser. Dave simultaneously provided the freedom to work on what I wanted and the guidance that enabled me to succeed in my work. My PhD committee: Charles Rohrs, John Guttag, and Hari Balakrishnan provided valuable counsel on both research and writing. Mark Handley collaborated on part of this work and contributed much insight. Tim Shepard and Mangesh Kasbekar read this dissertation and provided important comments. John Wroclawski, Karen Sollins, Barbara Liskov, Frans Kaashoek have supported me at various stages of my graduate career. I am very grateful to all my friends on the 5th floor of LCS who made my graduate years at MIT exciting and pleasurable.

# Contents


**1 Overview** **13**

1.1 Decoupling Congestion Control from the Bandwidth Allocation Policy . . . . . . . 14

1.2 Congestion Control in Large Bandwidth-Delay Product Networks . . . . . . . . . . 15

1.3 Contributions & Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

1.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

**2 Background & Related Work** **18**

2.1 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

2.2 Internet Congestion Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

2.2.1 End-System Congestion Control Protocols . . . . . . . . . . . . . . . . . 20

2.2.2 Router Congestion Controllers . . . . . . . . . . . . . . . . . . . . . . . 22

2.3 Expressiveness & State . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.3.1 ECN . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

2.3.2 Packet Pair & Fair Queuing . . . . . . . . . . . . . . . . . . . . . . . . . 26

2.3.3 Core Stateless Fair Queuing (CSFQ) . . . . . . . . . . . . . . . . . . . . 26

2.3.4 ABR Congestion Control . . . . . . . . . . . . . . . . . . . . . . . . . . 27

2.4 The Challenges for Internet Congestion Control . . . . . . . . . . . . . . . . . . . 28

2.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

**3 The Problem** **30**

3.1 TCP's Performance in High Bandwidth-Delay Product Networks . . . . . . . . . . 30

3.2 Simulation Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**4 XCP: eXplicit Control Protocol** **37**

4.1 Design Rationale . . . 37

4.2 The XCP Protocol . . . 39

4.3 Framework . . . 40

4.4 The Congestion Header . . . 40

4.5 The XCP Sender . . . 40

4.6 The XCP Receiver . . . 42

4.7 The XCP Router: The Control Laws . . . 42

4.7.1 The Efficiency Controller (EC) . . . 43

4.7.2 The Fairness Controller (FC) . . . 44

4.7.3 Notes on the Efficiency and Fairness Controllers . . . 46

4.8 Stability Analysis . . . 48

4.9 Summary . . . 49

**5 Performance** **51**

5.1 Simulation Setup . . . 52

5.2 Comparison with TCP and AQM Schemes . . . 53

5.2.1 Impact of High Capacity . . . 53

5.2.2 Impact of Feedback Delay . . . 55

5.2.3 Impact of Number of Flows . . . 56

5.2.4 Impact of Short Web-Like Traffic . . . 57

5.2.5 Fairness . . . 58

5.2.6 A More Complex Topology . . . 59

5.3 The Dynamics of XCP . . . 61

5.3.1 Convergence Dynamics . . . 62

5.3.2 Robustness to Sudden Increase or Decrease in Traffic Demands . . . 62

5.4 Robustness to Large RTT Variance . . . 63

5.5 Impact of Error-Based Drops . . . 65

5.6 Large Scale Simulations . . . 68

5.7 Summary . . . 69

**6 Quality of Service (QoS)** **70**

6.1 QoS Models . . . 70

6.2 Providing QoS Within the XCP Framework . . . . . . . . . . . . . . . . . . . . . . 72
6.3 Relative Bandwidth Allocation in XCP . . . . . . . . . . . . . . . . . . . . . . . . . 73
6.3.1 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
6.4 Providing Bandwidth Guarantees in XCP . . . . . . . . . . . . . . . . . . . . . . 77
6.4.1 Service Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
6.4.2 Modifying the Congestion Header . . . . . . . . . . . . . . . . . . . . . . 78
6.4.3 Behavior of the Sender . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
6.4.4 Control Plane: Distributed Admission Control With No Per-Flow State . . 79
6.4.5 Data Plane . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
6.4.6 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
6.5 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

**7 Non-Compliant Flows** **90**

7.1 Flow Monitoring . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90
7.2 Networks with No Flow Protection . . . . . . . . . . . . . . . . . . . . . . . . . . 91
7.3 Senders' Misconduct . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
7.3.1 The T-liar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
7.3.2 The D-liar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
7.3.3 Responsive Abusers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
7.3.4 Unresponsive Abuser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
7.4 Receivers' Misconduct . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
7.5 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104

**8 Gradual Deployment** **105**

8.1 XCP-based Core Stateless Fair Queuing . . . . . . . . . . . . . . . . . . . . . . . 105
8.2 A TCP-Friendly XCP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106
8.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109

**9 Conclusion & Future Work** **110**

9.1 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110
9.1.1 Good Performance in Networks with Large Bandwidth-Delay Product . . . 110
9.1.2 XCP Outperforms Other Protocols With No Per-Flow State . . . . . . . . . 111
9.1.3 Integrating Control Theory with Internet Protocols . . . . . . . . . . . . . 111

9.1.4 New Concepts for Internet Bandwidth Management . . . . . 112

9.1.5 A Simple Approach to Quality of Service . . . . . 113

9.2 Limitations . . . . . 114

9.2.1 Deployment . . . . . 114

9.2.2 Security Without Per-Flow State . . . . . 114

9.3 Outstanding Problems & Future Work . . . . . 115

9.3.1 Multi-Access Links . . . . . 115

9.3.2 Increasing Robustness to RTT Variance . . . . . 116

9.3.3 Specifying the Bit Format of the Congestion Header . . . . . 116

9.4 Final Remarks . . . . . 117

**A XCP Implementation** **118**

**B Analysis of XCP Stability** **121**

B.1 Model . . . . . 121

B.2 Stability . . . . . 122

# List of Figures


2-1 TCP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3-1 A single bottleneck topology. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3-2 TCP's average utilization decreases with increased bottleneck bandwidth. . . . . . 34

3-3 TCP's average utilization decreases with increased delay. . . . . . . . . . . . . . . 35

4-1 Congestion header. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

5-1 A single bottleneck topology. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

5-2 A parking lot topology. Arrows represent traffic directions. . . . . . . . . . . . . . 53

5-3 XCP significantly outperforms TCP in high bandwidth environments. The graphs compare the efficiency of XCP with that of TCP over RED, CSFQ, REM, and AVQ, as a function of capacity. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

5-4 XCP significantly outperforms TCP in high delay environments. The graphs compare bottleneck utilization, average queue, and number of drops as round trip delay increases when flows are XCPs and when they are TCPs over RED, CSFQ, REM, and AVQ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

5-5 XCP is efficient with any number of flows. The graphs compare the efficiency of XCP and TCP with various queuing schemes as a function of the number of flows. 56

5-6 XCP is robust and efficient in environments with arrivals and departures of short web-like flows. The graphs compare the efficiency of XCP to that of TCP over various queuing schemes as a function of the arrival rate of web-like flows. . . . . 57

5-7 XCP is fair to both equal and different RTT flows. The graphs compare XCP's Fairness to that of TCP over RED, CSFQ, REM, and AVQ. Graph (b) also shows XCP's robustness to environments with different RTTs. . . . . . . . . . . . . . . 59

5-8 Simulation with multiple congested queues. Utilization, average queue size, and number of drops at nine consecutive links (topology in Figure 5-2). Link 5 has the lowest capacity along the path. . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
5-9 XCP's smooth convergence to high fairness, good utilization, and small queue size. Five XCP flows share a 45 Mb/s bottleneck. They start their transfers at times 0, 2, 4, 6, and 8 seconds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
5-10 XCP is more robust against sudden increase or decrease in traffic demands than TCP. Ten FTP flows share a bottleneck. At time $t = 4$ seconds, we start 100 additional flows. At $t = 8$ seconds, these 100 flows are suddenly stopped and the original 10 flows are left to stabilize again. . . . . . . . . . . . . . . . . . . . . . . . . . . 62
5-11 The RTT distribution at an OC3 access link at the Supercomputer center in San Diego, (a reproduction of Figure 13-a-top in [42]). . . . . . . . . . . . . . . . . . 63
5-12 The performance of XCP and TCP+RED+ECN when the RTT is distributed according to Figure 5-11 (values are logged every second). . . . . . . . . . . . . . . . . 64
5-13 The performance of XCP and TCP+RED when the RTT is uniformly distributed over [5 ms, $\tau$], where $\tau$ is the value on the x-axis. . . . . . . . . . . . . . . . . . . 65
5-14 The throughput of a single XCP flow as a function of packet error rate. Capacity is 4 Gb/s, RTT = 80 ms. For a reference, the figure also shows the steady state throughput of a TCP flow as a function of the packet error rate. . . . . . . . . . . 66
5-15 A large simulation with 50 nodes, 153 links, and 1000 flows. . . . . . . . . . . . . 67
5-16 A comparison between the efficiency of XCP and TCP over RED. . . . . . . . . . 67

6-1 Comparison between XCP's relative bandwidth allocation and MulTCP [24], CSFQ [75], and Weighted Fair Queuing (WFQ) [25]. Three flows each transferring a 10 MB file over a shared 10 Mb/s bottleneck. Flow 1's weight is 5, Flow 2's weight is 10, and Flow 3's weight is 15. Throughput is averaged over 200 ms (5 RTTs). . . . 75
6-2 The range of relative bandwidth allocation provided with XCP, MulTCP, CSFQ, and WFQ. The figure shows the ratio of the throughputs of two flows sharing a bottleneck as a function of the ratio of their weights. Each point is the average of 10 runs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

6-3 $R_g$ tracks the reservation made by the guaranteed bandwidth flows. The guaranteed bandwidth flows arrive separated by 5 sec. Each flow requests 15% of the bottleneck bandwidth. The rest of the bandwidth is filled up by best-effort flows. Since G = 0.8 capacity, we can accommodate at most 5 guaranteed bandwidth flows. At $t = 50$, active guaranteed flows are stopped, which frees up the reserved bandwidth. The cycle repeats with new arrivals of guaranteed flows. . . . . . . . . . . . . . . . . 83

6-4 Admission control with on-off flows. The guaranteed service flows all arrive around $t = 10s$ and stop at $t = 90s$. The router accepts them as long as $R_g < G$. The figure shows that although the on-off flows do not use all of their reserved bandwidth ($R_u < R_g$), the router maintains the reservations. . . . . . . . . . . . . . . . . . . 84

6-5 The service perceived by the guaranteed bandwidth flows. 6-5-a shows that the flows obtain their reserved throughput. 6-5-b shows that the delay perceived by guaranteed service flows is almost always the RTT. . . . . . . . . . . . . . . . . . 85

6-6 The service perceived by best-effort traffic. A 10 Mb/s bottleneck is shared by 10 best-effort flows and 5 guaranteed service flows. The guaranteed service flows are consuming 75% of the bandwidth. The rest is used by the best-effort traffic. . . . . 86

6-7 A large simulation topology with 153 links and 50 nodes. . . . . . . . . . . . . . 87

6-8 Guaranteed bandwidth flows experience negligible worst case queuing delays. The figure shows the maximum G-queue size at each of the links in Figure 6-7, which is the worst case queuing delay experienced by guaranteed bandwidth flows at that link. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89

7-1 The impact of T-lying on fairness. The throughput of a single T-liar that is sharing a bottleneck with 19 compliant flows. . . . . . . . . . . . . . . . . . . . . . . . . 95

7-2 The impact of T-lying on efficiency. The utilization and queue size of a bottleneck traversed by 20 T-liars. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

7-3 The impact of D-lying on fairness.The throughput of a single D-liar that is sharing a bottleneck with 19 compliant flows. . . . . . . . . . . . . . . . . . . . . . . . . 97

7-4 The impact of D-lying on efficiency. The utilization and queue size of a bottleneck shared by 20 flows. The graphs are for the cases when all flows are D-liars, 75% of the flows are D-liars, and 50% of the flows are D-liars. . . . . . . . . . . . . . . . 98

7-5 The impact of a responsive abuser on fairness. The graphs on the right use TCP, whereas the graphs on the left use XCP. . . . . . . . . . . . . . . . . . . . . . . . 100
7-6 A comparison between the efficiency of XCP and TCP, when all users increase aggressively by multiplying their fair positive increase by a gain $> 1$. . . . . . . . . 101
7-7 A comparison between the efficiency of XCP and TCP, when all users decrease leniently by multiplying their fair negative decrease by a gain $< 1$. . . . . . . . . . 101
7-8 The impact of a non-responsive abuser in an XCP and a TCP network. The graphs on the right use TCP whereas the graphs on the left use XCP. . . . . . . . . . . . . 103

8-1 XCP is TCP-friendly. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

9-1 A black box model of the efficiency controller and the fairness controller. . . . . . 112

B-1 The feedback loop and the Bode plot of its open loop transfer function. . . . . . . 123
B-2 The Nyquist plot of the open-loop transfer function with a very small delay. . . . . 123

# List of Tables

6.1 QoS Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

6.2 The throughputs of the guaranteed bandwidth flows that got admission into the network in Figure 6-7. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88

7.1 Various sender's misconducts, their impact on efficiency and fairness, and whether a strategic sender has an incentive to perform such a misconduct. IG and DG are the increase and decrease gains respectively, $r_i$ is the rate of unresponsive traffic, and $C$ is the capacity. To evaluate the incentive of the user, we assume that the sender is a strategic agent who misbehaves to increase its throughput. . . . . . . . . . . . . 93

# Chapter 1

# Overview

In this dissertation, we take a step back and re-examine the design of Internet congestion control. Although congestion control is a two-decade old problem, which has been studied in a large number of papers and theses, we believe that it is time to re-visit the principles underlying the current design for two reasons. First, current TCP-based congestion control exhibits poor performance in environments with a large per-flow bandwidth-delay product and cannot deliver a large end-to-end throughput (i.e., more than Gb/s) even when the path has enough spare bandwidth [5, 32, 28]. Currently, this problem occurs only over a small number of paths that enjoy a significantly large end-to-end bandwidth [3, 28, 2]. However, technology trends indicate that we are moving quickly toward an Internet where tens of Gb/s end-to-end paths are common [7, 21]. Demands for using this bandwidth are also increasing at an exponential rate [21], creating a need for protocols that can achieve a very large per-flow throughput. Second, congestion control remains a hard and open problem. As a community, we have made many advances since the installation of congestion control in the Internet in 1989 [39]. However, there are still many open problems such as the need for a less oscillatory and more predictable behavior [54] and better fairness over shorter time scales [12]. Thus, this dissertation proposes a new architecture for managing Internet bandwidth that is based on decoupling congestion control from the bandwidth allocation policy. It shows that this approach stays efficient as the per-flow bandwidth delay product increases, and also improves the overall performance in current networking environments.

## 1.1 Decoupling Congestion Control from the Bandwidth Allocation Policy

The management of Internet resources involves two objectives:

1. Efficiency: The bandwidth should be used efficiently but without congesting the network. This means maintaining high link utilization, small queue size, and almost no packet drops.

2. Bandwidth Allocation Policy: The resources should be divided among the users to satisfy some desired bandwidth allocation policy. For example, bandwidth may be divided among users according to max-min fairness, where users who share the bottleneck link and have enough demands are allocated equal bandwidth shares.[1] Also, bandwidth may be allocated to users proportionally to the price they pay, or according to their priorities, etc.

Internet resources are managed by end-system congestion control protocols and router queuing mechanisms. The vast majority of these [13, 66, 34, 35, 11] do not distinguish the problem of congestion control and efficient network usage from the problem of apportioning the bandwidth between individual users; they use a single control rule to regulate both. For example, TCP, which is the dominant congestion control protocol in the Internet, uses a single control law called Additive-Increase Multiplicative-Decrease (AIMD) to control both congestion and fairness [20]. Thus, while controlling congestion TCP enforces on its users a particular bandwidth allocation.[2]

The coupling of efficient network utilization and a particular bandwidth allocation in congestion control protocols results in two adverse effects. First, it becomes difficult to change the bandwidth allocation policy without affecting the dynamics of congestion/efficiency control. Second, imposing two objectives on a single control law (i.e., efficiency and a particular allocation) makes it hard to find a control law that excels in achieving either goal. For example, in the case of TCP, AIMD neither converges to optimal utilization [78, 51, 58] nor achieves high fairness [12, 37, 58].[3]

We propose a new approach for bandwidth regulation that decouples congestion control from the process that enforces the bandwidth allocation policy. This decoupling is done by recognizing that efficiency and the occurrence of congestion are determined by *aggregate traffic* on a link, and are independent of the *relative rates* of the individual flows in the aggregate traffic. In contrast, a bandwidth allocation policy is concerned with the relative rates of individual flows inside aggregate

[1] For an exact definition, see §2.1.

[2] The enforcement happens as long as the the routers do not actively intervene to change this allocation.

[3] It is typical for the rates of competing TCP flows to differ by 25% to 50% of the fair share.

traffic. Thus, congestion control can be decoupled from the bandwidth allocation policy by using two controllers, one for controlling the aggregate traffic and another for controlling the relative throughputs of the flows inside the aggregate. More precisely, the congestion controller is concerned with matching the input aggregate traffic at a link to its capacity and draining any persistent queue. The allocation controller shuffles the bandwidth inside the aggregate traffic, changing the relative rates of the flows without changing the rate of the aggregate so that the flows converge to the desired bandwidth allocation.

We show that this decoupling simplifies protocol design, improves network efficiency, and enables the protocol to better match the desired bandwidth allocation, all of which can be achieved while maintaining the scalability of current congestion control. More importantly, the decoupling allows us to solve the poor performance problem facing TCP in high bandwidth-delay product networks.

## 1.2 Congestion Control in Large Bandwidth-Delay Product Networks

High bandwidth-delay networks are environments where the per-flow bandwidth can reach a few Gb/s, or the delay is so large that the flow has a few MB of outstanding traffic in the pipe. These environments are becoming more common in today's Internet [3, 28, 2]. In particular, with the advent of gigabit Ethernet, many paths can potentially provide a very large end-to-end throughput. Further, with the fiber capacity increasing exponentially with time [21, 56], these environments are expected to become more common. The huge per-flow bandwidth is unlikely to eliminate network congestion because demands for bandwidth are doubling every year [21]. In today's Internet, many organizations generate a few Gb/s of data that they want to transfer over the Internet from the collecting site to the processing site (e.g., astronomy researchers, physicists, backup systems, and shared storage). However, current TCP-based congestion control does not allow a few Gb/s end-to-end steady state throughput [5, 28, 32]. There are two major reasons why TCP cannot maintain a very large per-flow throughput. First, the AIMD law in TCP increases by one packet every round trip time (RTT). Although this increase is reasonable in current networks, it becomes too slow in environments with very large spare bandwidth; TCP might take thousands of RTTs to ramp up to full utilization when a large amount of bandwidth becomes suddenly available after a period of congestion. Second, the TCP throughput is inversely proportional to the square root of the packet loss rate. A very large end-to-end TCP throughput requires an exceptionally small packet error rate

(PER) that is challenging even to low bit error rate fiber link technology.[4]

As a remedy, we propose the eXplicit Control Protocol (XCP), a congestion control protocol that embodies the principle of decoupling congestion control from bandwidth allocation and uses this decoupling to achieve good performance in high bandwidth-delay networks. XCP provides a joint design of the end systems and the router and is characterized by the existence of separate congestion/efficiency controller and fairness controller at the router. The separation of the controllers allows XCP to use a fast controller to control congestion, which increases the rate proportionally to the amount of unused bandwidth along the path and decreases it proportionally to the degree of congestion. This allows XCP to quickly increase its rate when a large amount of spare bandwidth becomes suddenly available. To control fairness, XCP uses an AIMD control law, which has been shown to achieve fair bandwidth allocation [20].

## 1.3 Contributions & Organization

This dissertation contributes a new architecture for Internet congestion control that is based on decoupling congestion control from the bandwidth allocation policy. It develops XCP, an explicit congestion control protocol that performs efficiently in large bandwidth-delay product networks and traditional lower bandwidth environments. The dissertation studies the performance of XCP using extensive simulations and demonstrates some of its stability properties using control theoretic analysis. Further, it develops a simple approach for quality of service that uses the XCP framework to deliver proportional and guaranteed bandwidth allocations. A complete description of our contributions is in §9.

The rest of this dissertation is organized as follows. In §2, we provide the background necessary to understand the objectives of a congestion control protocol and the constraints that involve its design. We also describe previous work in the area. In §3, we describe in more details the problems facing TCP congestion control as the per-flow bandwidth-delay product increases. §4 forms the core of this thesis, where we present XCP, a congestion control protocol that decouples congestion control from bandwidth allocation. Using extensive simulations, we show in §5 that XCP outperforms TCP both in traditional environments and high bandwidth-delay product networks. In §6,

[4] The typical bit error rate (BER) over fiber is $10^{-10}$ to $10^{-12}$ [69, 10]. Assuming a packet size of 1000 bytes and a BER = $10^{-12}$, the packet error rate (PER) is around $10^{-8}$. Substituting this value in the TCP throughput equation and assuming RTT = 100ms, one finds that the throughput is slightly less than 1 Gb/s. If one takes into account the existence of multiple links along the path and other sources of errors at routers or switches, the maximum throughput would be substantially smaller.

we show that the XCP framework is flexible enough to provide both proportional and guaranteed bandwidth allocations. Further, XCP provides all of that without any per-flow state at the routers. In §7 we analyse the robustness of XCP against users' misbehavior in, and in §8 we provides a gradual deployment path. Finally, we conclude in §9 with a discussion of the advantages and limitations of the proposed architecture along with directions for future work. Additionally, this dissertation contains two appendixes. Appendix A explains the details of our *ns* [59] implementation, whereas appendix B uses tools from Control Theory to demonstrate the stability of XCP.

## 1.4 Summary

This thesis addresses the problem of providing efficient congestion control in high bandwidth-delay product environments while maintaining the desirable properties of current congestion control. To solve this problem, we introduce an architecture that decouples congestion control from the bandwidth allocation policy. We use this architecture to develop a congestion control protocol called XCP (eXplicit Control Protocol). In the rest of this dissertation, we show that XCP outperforms TCP both in traditional environments and in high bandwidth-delay product networks. Further, XCP maintains TCP's scalability and robustness, and can provide a variety of qualities of service.

# Chapter 2

# Background & Related Work

The purpose of this chapter is to provide the reader with a general understanding of Internet congestion control and its relation to bandwidth allocation. We start by providing a few definitions. Next, we describe previous work in congestion control and its relevance to this thesis. Then, we talk about the difficulties and the design choices made by various congestion control protocols and queuing disciplines, which differ in the amount of state they store in the network and the precision or expressiveness of the congestion feedback.

## 2.1 Definitions

Below are a few definitions that we use throughout this document.

(a) **Flow**: A network flow is a sequence of packets that share the sender and destination IP addresses, and the sender and destination port numbers. To a large extent, packets in a flow follow the same path in the network [65].

(b) **Round Trip Time (RTT):** The round trip time for a flow is the total time taken by the network both to deliver a packet from this flow to its receiver, and to deliver its acknowledgment to the sender.

(c) **Propagation Delay:** The propagation delay is the round trip delay after subtracting the queuing time.

(d) **Congestion Window :** The congestion window is the maximum number of packets a flow can send in an RTT. When the application has data to send, the congestion window also refers to the number of outstanding packets for which the sender has not received an acknowledgment, and the flow's rate (i.e., throughput) is the ratio of its congestion window to its RTT.

(e) **Window vs. Rate Based Congestion Control Protocols:** Congestion control protocols govern the sending rate and the burstiness of the sources by deciding the maximum amount of traffic a source can send in an RTT. In a rate-based protocol the source uses a timer to pace the traffic it is allowed to send. In contrast, a window-based congestion control protocol limits the sending rate of a flow by regulating the size of its congestion window. The congestion window is updated upon the arrival of an acknowledgment and constitutes an upper bound on the burstiness of the source. The advantage of a rate-based protocol is its smoothness caused by the spreading of traffic over the RTT. The advantage of the window-based algorithm stems from its robustness against light congestion. In particular, when congestion begins, packets start accumulating at the router creating a queue. The queuing time increases the RTT, which decreases the sending rate of the flow without the need for the congestion control protocol to exercise any direct control over the rate of the source. Additionally, in window-based protocols, new packets are sent upon the arrival of an acknowledgment (self-clocking), a characteristic that has been shown to increase robustness against high drop rate [14].

(f) **Max-Min Fairness:** Max-min fairness maximizes the minimum throughput of the flows sharing a single bottleneck. More precisely, when multiple sources with infinite demands share a bottleneck link, they all obtain equal throughputs. If a source's demands are less than its fair share of the bandwidth, the source gets the minimum of its demands and its fair share. The extra bandwidth is divided equally among the other sources. Many protocols such as TCP try to approximate max-min fairness. We refer to these approximations using the general term of "fairness".

(g) **Weighted Fairness:** Weighted fairness allocates the bandwidth of the bottleneck to the sources proportionally to their weights.

## 2.2 Internet Congestion Control

The congestion control problem is to control the traffic rate in the network to achieve high link utilization, small queue size, and few packet drops.

The control of Internet congestion is the result of the cooperative work of both the routers and the end systems (i.e., senders and receivers). Traditionally, the role of the routers is limited to dropping packets as a sign of congestion. The senders run a congestion control protocol that interprets a packet loss as a sign of congestion to which they react by decreasing the sending rate. This model still dominates most of the Internet, where most of the routers are drop-tail routers (drop packets upon buffer overflow) and most senders run TCP as the congestion control protocol. However, the research community has moved toward giving the router a more active role in anticipating congestion and signaling it to the senders. Congestion controllers located at the routers, such as RED [35], REM [11], AVQ [50], ARED [33], etc., are referred to as Active Queue Management schemes (AQMs).

We note that the majority of previous work in Internet congestion control concerns itself with either the design of the controllers at the routers, i.e., an AQM, or the design of the congestion control protocol at the end systems (i.e., sender and receiver). In contrast, this thesis presents a joint design of the end systems and the routers.

### 2.2.1 End-System Congestion Control Protocols

Internet congestion control protocols use a closed-loop control; they probe the network to discover whether there is congestion. The protocols use packet drops as an implicit signal of congestion. They can discover drops by sequentially numbering the packets and having the receiver acknowledge the packets it receives. As long as there is no drop, the sender increases its sending rate. When the sender discovers a drop, it interprets it as a signal of congestion and decreases the sending rate. The protocols differ from one another by their increase-decrease rules. The increase-decrease rules are designed such that the average sending rate of a flow is around its fair share of the bandwidth. Of these protocols, TCP stands out as the de facto congestion control protocol of the Internet.

#### Transmission Control Protocol (TCP)

TCP was designed to provide reliable transmission of data [66]. However, it has evolved to provide congestion control too [39]. Below, we focus on just the congestion control aspects of TCP.

TCP is a window-based congestion control protocol. Thus, it controls the window of outstanding packets in the network. Like most other congestion control protocols, TCP relies on drops to detect congestion. It has two mechanisms to discover the occurrence of a drop. First, when a packet is sent, the TCP sender initializes a timer. If the timer expires before the packet is acknowledged, TCP considers the packet to have been dropped. Second, when the TCP receiver receives an out-

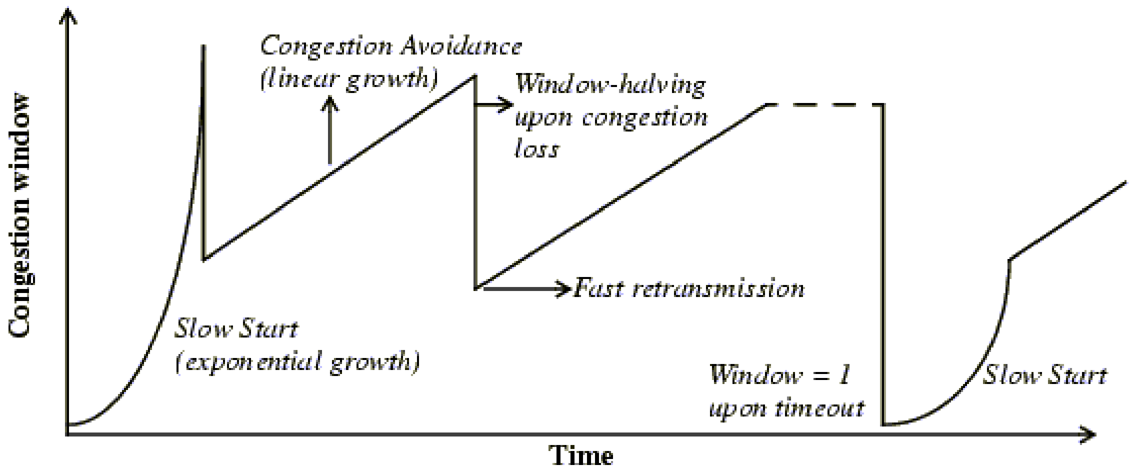



Figure 2-1: TCP

of-order packet, it sends an acknowledgment (Ack) for the most recent in-order packet it received. For example, assume the receiver receives packets 1 to 5, and packet 6 gets dropped. When the receiver receives packet 7, it sends a dupack for packet 5. A TCP sender considers the arrivals of three duplicate acknowledgment (dupacks) as a sign of a packet drop.

A TCP connection goes through two phases: slow-start and the AIMD phase. Figure 2-1 shows a typical trajectory of the TCP congestion window.

(a) **Slow-Start:** TCP goes to slow-start mode at the beginning of a connection. During slow-start, the sender increases its sending rate exponentially. In particular, at the beginning of slow-start the congestion window is set to one packet. Thus, the sender sends a packet and waits until the receiver acknowledges it. Once the Ack reaches the sender, the sender increases its congestion window by 1, sends two packets, and waits for the corresponding Acks. For each arriving Ack the sender can send two packets, which results in an exponential increase in the congestion window. TCP exits slow-start when a packet is dropped. The drop causes the sender to halve its congestion window and enter the AIMD phase.

(b) **Additive Increase Multiplicative Decrease (AIMD) :** In this mode, as long as there is no drop, a TCP sender increases its congestion window by one packet per RTT. When a packet is dropped, TCP halves the current congestion window. As a result, the throughput exhibits a sequence of additive increase followed by a multiplicative decrease event. This behavior is usually referred to as the "TCP sawtooth" (see Figure 2-1). The objective of AIMD is to allow TCP to operate around the correct sending rate. Further, it causes TCP flows that are competing for the same bottleneck to approach a fair bandwidth allocation [20].[1]

Finally, when congestion is severe, too many packets might get dropped, which causes the sender to experience timeout while waiting for the acknowledgment of its packets. A timeout causes TCP to go again into slow-start.

### 2.2.2 Router Congestion Controllers

The involvement of the routers in controlling congestion varies dramatically. Traditionally, the role of routers in controlling the traffic is limited to dropping packets when the buffer overflows. This scheme is called *drop-tail*, which is the common queuing policy in the current Internet.

There are many well-known problems with drop-tail such as being biased against bursty flows, and maintaining a persistent large queue size. To mitigate these problems, researchers have proposed that the routers take a more active role in the control protocol and send an early signal to the senders about anticipated congestion. Such queuing policies are called *Active Queue Management (AQM)*.

*Random Early Detection (RED)* [35] drops packets proportionally to the average queue length. More precisely, RED defines a running average of the queue length:

$$q_{avg} = w \times q_{avg} + (1 - w) \times q,$$

where $0 < w < 1$, and $q$ is the queue length seen by an arriving packet. Packets are dropped with probability $p$ computed as follows:

$$\begin{aligned} q_{avg} < min \quad &\Rightarrow \quad p = 0, \\ min \leq q_{avg} < max \quad &\Rightarrow \quad p = \frac{q_{avg} - min}{max - min} \times p_{max}, \\ q_{avg} \geq max \quad &\Rightarrow \quad p = 1, \end{aligned}$$

where *min* and *max* are constant parameters, and $p_{max}$ is the maximum drop probability.

One objective of RED is to prevent bursty drops. By starting to drop early, RED spreads the drops over a longer interval and provides smoother behavior. Further, it sends an early congestion signal to the flow to slow down before the queue overflows, which helps in maintaining a smaller

[1] Actually, in TCP the average congestion window - not the rate - approaches fair allocation.

average queue length.

RED has inspired a large number of AQM proposals that differ in the way they compute the dropping probability [35, 11, 50]. We describe REM and AVQ because they are the most famous schemes and the ones against which we compare XCP.

*Random Early Mark (REM)* [11] makes the dropping probability a function of the mismatch in traffic rate and the mismatch in the queue size. In particular, it defines a quantity called price, $pr$, which is updated upon the arrival of a packet:

$$pr(t+1) = pr(t) + \gamma \cdot (\alpha \cdot (q(t+1) - q_{target}) + x(t+1) - C),$$

where $q$ is the queue size, $q_{target}$ is a constant that specifies a desirable target queue length, $x$ is the traffic rate, $C$ is the capacity of the link, and $\gamma > 0$ and $\alpha > 0$ are constant parameters. The drop probability is chosen as $1 - \phi^{-pr}$, where $\phi$ is a constant. Thus, while RED looks only at the queue size, REM takes into account the mismatch between the input traffic and the capacity of the link. Further, REM addresses a particular problem with RED, which is the coupling of the drop probability and the queue size. In particular, the TCP throughput is inversely proportional to the square root of the drop probability [60]. Hence, as the number of flows increases the drop probability should increase if one to maintain a particular aggregate throughput. However, since RED makes the drop probability proportional to the average queue, a larger drop rate will increase the queue size and consequently the queuing delay. By introducing the price variable, REM decouples the drop probability form the queue size, allowing the system to stabilize around a target queue that is independent of the number of flows.

*Adaptive Virtual Queue (AVQ)* [50] maintains a virtual queue whose capacity is less than the actual capacity of the link. The capacity of the virtual queue is updated according to $\dot{\tilde{C}} = \alpha \cdot (\gamma \cdot C - \lambda)$, where $\tilde{C}$ is the virtual capacity, $C$ is the actual capacity of the link, $\lambda$ is the input traffic rate, $0 < \alpha < 1$ and $0 < \gamma < 1$ are constant parameters. The intuition underlying AVQ is that since the virtual capacity is always smaller than the real capacity, the arrival rate into the real queue is smaller than the departure rate. Consequently, any persistent real queue is bound to drain after some time.

There is a large amount of research that attempts to characterize the various AQMs. Yet, the community still has concerns about the safety of deploying AQM schemes in the Internet [54].

One important problem with AQM schemes is the existence of many tunable parameters whose values significantly affect the performance. Good operational values for these parameters depend on the number of flows traversing the bottleneck, the capacity of the bottleneck, and the round trip delays. Some of these parameters are unknown to the operator, might vary with time, and are hard to estimate.

## 2.3 Expressiveness & State

Two important characteristics of congestion control protocols are the expressiveness of the congestion feedback and the amount of state maintained at each of the routers. *Expressiveness* refers to the amount of information about the state of the network that is returned to the senders. Most congestion control protocols have a very low level of expressiveness [39, 13, 34, 72]. In particular, TCP uses implicit and binary congestion feedback. The feedback is implicit because the sender interprets a packet loss as a sign of congestion. The feedback is binary as well because the TCP sender identifies only two cases: a loss indicating congestion; no loss indicating no congestion. Given that the congestion information is limited to whether a drop occurred or not, the sender has to oscillate between overestimating the available bandwidth and underestimating it. To some extent this coarse reaction and the corresponding oscillations seem to be avoidable. Routers have information about the degree of congestion such as how big the current queue is or by how much the input traffic exceeds the link capacity. By restricting ourselves to this very coarse congestion feedback that relies on a packet drop, it seems that we are wasting information.[2]

So why not use this congestion information possessed by the routers to enhance the performance of our congestion control protocols? One of the earliest examinations of congestion control in a control theoretic framework demonstrates the power of expressive control to eliminate oscillations even in the case of networks with long delays [70]. Other proposals have considered protocols that use more expressive feedback than packet drops. The simplest of these is called Explicit Congestion Notification (ECN) [67]; it replaces a drop by a one-bit congestion mark in the IP header. In contrast, the authors [40, 18] have attempted more drastic change. They have sent precise information about the exact rate that the sender should use. The problem these protocols have faced is that to decide the fair rate of a sender, the router needs to know the number of senders traversing the link and which of them can increase their sending rates (and which are prevented from doing so because

[2] Note that the argument above does not mean to ignore drops as a sign of congestion but rather to strengthen and fine-tune this signal by using more information.

they are blocked at a different bottleneck). One approach for solving this problem is to count the flows at the router and maintain some information about the sending rates and the demands of these flows. However, this leads to installing per-flow state in each router, which limits the scalability of the system. To understand why this is the case, let us define the term “per-flow state”.

When we talk about the *state* maintained for congestion control, we focus usually on the state at the routers. This state is expensive because the routers have limited resources and need to operate at a very high speed. As a result, the more state they keep, the more the constraints on their performance. For example, drop-tail, RED [35], and most of Internet queuing disciplines [33, 11, 50, 31, 75, 64] do not require per-flow state at routers. On the other hand, ATM congestion control protocols [40, 18, 41] usually make the switch maintain an entry for every flow traversing it. The common wisdom in the Internet is to avoid per-flow state at routers because maintaining such a state in an environment as dynamic as the Internet would limit the speed of the routers. In particular, the number of flows traversing a backbone router could be tens of thousands. Classifying packets from these flows and updating the per-flow information to reflect which flows are active at any point in time is too much work for the routers.

From the above discussion we see that though the expressiveness of the protocol and the state it maintains at routers are different issues, the use of more expressiveness usually leads to per-flow state at routers, which limits the scalability of the congestion control protocol. Below we briefly discuss a few protocols that differ in their expressiveness and the state they maintain.

### 2.3.1 ECN

Explicit Congestion Notification (ECN) attempts to prevent packet drops through the use of a special field in the IP header to signal congestion to the sender. Thus, ECN moves from an implicit form of congestion feedback to an explicit feedback. Yet, ECN maintains the binary nature of the congestion feedback where the sources know only whether there is congestion or not. As a result, ECN does not change the dynamics or characteristics of the congestion control protocol (i.e., TCP).

ECN was originally designed to work in conjunction with RED queues to send an early signal to the sources about anticipated congestion. The hope was that the early reduction in the sending rate would prevent a queue overflow and the ensuing packet drops. Recently, ECN has been used with congestion controllers other than RED [62, 11].

XCP generalizes ECN because it replaces the one-bit congestion feedback with a field that reflects the degree of congestion along the path. This replacement allows a faster and more accurate response from the senders.

### 2.3.2 Packet Pair & Fair Queuing

Packet pair [48] is a rate based congestion control protocol that addresses both router and end-system design. Routers queue packets from different flows separately and drain these per-flow queues by simulating round-robin at the byte level (see [25] for details), a disciplines usually referred to as "fair queuing". A sender continuously estimates the speed of its bottleneck by transmitting its data as pairs of back-to-back packets. When a pair of back-to-back packets traverses the bottleneck link, the round-robin forwarding paces the packets according to sender's fair share of the capacity. The separation between the data packets is reflected in the separation of the Acks. By measuring the separation between pairs of returning Acks, the sender discovers the departure rate from its queue and adjusts its rate to avoid both overflow and underflow of its buffer.

In comparison with the combination of TCP and AQM or drop-tail, this scheme is less scalable because it requires per-flow queuing at routers. On the other hand, packet pair is fairer and provides good protection against misbehaving flows.

Packet pair is particularly interesting because similarly to XCP it decouples the concept of congestion control from fairness. Fairness is established using fair queuing at routers. Congestion control is delegated to the sources which measure the speed of their bottleneck using packet pairs. Nonetheless, XCP is substantially different from packet pair because it uses explicit feedback and does not require per-flow state in the network.

### 2.3.3 Core Stateless Fair Queuing (CSFQ)

Core Stateless Fair Queuing attempts to solve the contradiction between providing better fairness and maintaining the high scalability of stateless networks. Estimating the fair bandwidth share of a flow requires knowing the number of flows traversing a link and identifying which of them can use its fair share and which of them is limited to a lower throughput by another link along its path. This information necessarily implies some form of per-flow state. The contribution of CSFQ is in the recognition that a flow's state can be provided in its packets and need not be kept at the router. As such, the router need not classify an arriving packet to decide which flow it comes from and how much bandwidth that flow is currently using. All of that state can be kept in the packet.

Since legacy TCP senders do not provide a flow's state in the packets, CSFQ works in a network cloud where edge routers estimate the flows' rates and insert the estimates in the packets' header.

Core routers estimate the fair rate and drop packets preferentially on the basis of the differences between the estimated rate and the fair rate.

XCP builds on CSFQ's idea of pushing the per-flow state to the edges of the network. However, the state used in XCP is different from that in CSFQ. In addition to the flow's current sending rate, the XCP state contains the flow's RTT. This information allows the routers to scale the control according to the feedback delay in the system, which is essential for the stability of any control system [62, 52].

### 2.3.4 ABR Congestion Control

The ATM Forum has adopted an explicit end-to-end rate-based flow-control protocol (EERC) for controlling Available Bit Rate (ABR) traffic [49]. In EERC, each source periodically sends special resource management (RM) cells. These cells include a field called the explicit rate field (ER), which the source initializes to the desired sending rate. Each switch along the path computes the source's max-min fair rate. If the max-min fair rate is smaller than the value in the ER field, the switch sets ER to the max-min fair rate. The receiver relays the RM cells back to the sender, which then adjusts its rate to that indicated in the RM cell. Note that EERC does not specify how the max-min fair rate is computed. A few algorithms have been proposed for doing this computation [8, 18, 40, 41].

XCP resembles ABR congestion control in several ways. First, they both send explicit and precise feedback. Second, in both cases, the framework is independent from the exact control law used so that the control law can be changed while the framework is maintained. Yet, the differences between the two approaches are crucial. First, most ABR flow control protocols maintain per-flow state at the switches [8, 18, 40, 41], whereas XCP does not keep any per-flow state in routers. Second, in ABR control protocols the switch computes the fair bandwidth allocation of each flow and tries to jump to that allocation in one step. In contrast, in XCP the router makes gradual rate adjustments whose size is chosen to maintain stability. Third, XCP has self-clocking, a characteristic that improves stability [15], and is not provided by ABR congestion control. Finally, XCP is built on top of TCP. XCP's window adjustments have finer granularity, but when a drop occurs it reverts to a TCP-like behavior. This allows XCP to enjoy TCP's well-tested robustness to congestion collapse.[3]

[3] Informally, congestion collapse is a long period of severe congestion.

## 2.4 The Challenges for Internet Congestion Control

The Internet literature is rich with proposals for congestion control. Despite the differences in their design, all of these proposals attempt to maximize the link utilization while minimizing the queue size and number of drops. In their effort to achieve the above objective, Internet congestion control protocols try to build on a few principles that have proven useful in practice.

1. **A distributed protocol:** It is hard to imagine controlling congestion in the Internet by sending the state of every link and sender to a centralized computer and computing the optimal sending rate of each flow. The size and dynamics of the Internet make such an approach impractical. Thus, all congestion control protocols must operate in a distributed manner, where each sender and router makes decisions by looking only at local information and without consulting distant network entities.

2. **Router simplicity:** Internet routers are complex and expensive devices that run at very high speeds. Today's high-speed routers can spend only a few nanoseconds on a packet before the packet is due to depart on the link. Hence, protocols that require the routers to maintain per-flow queues and classify every packet are considered too complex. Such protocols require more memory, making the routers too expensive. Furthermore, they perform a few memory accesses per packet, and this usually takes too long.

3. **Stability & robustness:** Robustness is another important principle in designing congestion control protocols. The Internet is a heterogeneous environment; different Internet paths contain links whose bandwidth, latency, error rate, and technology differ drastically. Additionally, an Internet flow may traverse a few networks operated by different entities. Thus, a congestion control protocol for the Internet should have minimal assumptions about the environment and be robust against errors, packet losses, and traffic variations.

4. **Deployment:** Deployment is an important problem that any practical congestion control protocol needs to address. For the protocol to perform correctly, the sender, the receiver, and the nodes along the path have to comply with the assumptions of the protocol. Updating all of these components at once is a difficult task. Thus, deployable proposals must include a description of a gradual deployment path that allows for the new protocol to coexist with legacy systems and for only a few network entities to be updated at any one time.

## 2.5 Summary

Congestion control is the problem of deciding the sources' rates to achieve high utilization without causing congestion. Currently, most hosts on the Internet run TCP as their congestion control protocol. Most routers are drop-tail; they use FIFO queues and drop packets when the queue overflows. This approach to congestion control is both robust and scalable; however, it shows limitations in controlling utilization and fairness. More stable and tighter control protocols are desirable but they should not jeopardize the scalability or the robustness of congestion control. Further, to be successful, a new congestion control protocol should maintain a simple behavior at the routers and provide a gradual deployment path.

# Chapter 3

# The Problem

For the Internet to continue to thrive, its congestion control mechanism must remain effective as the network evolves. Technology trends indicate that the future Internet will have a large number of very high-bandwidth links. Less ubiquitous but still commonplace will be satellite and multi-hop wireless links with high latency. These trends are problematic because TCP reacts adversely to an increase in the per-flow bandwidth-delay product.

## 3.1 TCP's Performance in High Bandwidth-Delay Product Networks

We use the term "high bandwidth-delay networks" to refer to environments where the per-flow bandwidth can reach a few Gb/s, or the delay is so large that the flow has a few MB of outstanding traffic in the pipe. Currently these environments are limited to a few research organizations connected to the Internet via high performance research networks [3, 2]. However, with the proliferation of gigabit Ethernet [7], more users will traverse high capacity end-to-end paths. Further, with fiber capacity and bandwidth demands both increasing exponentially with time [21, 56], achieving a large per-flow throughput will soon become a pressing issue. Indeed, even in today's Internet many organizations generate a few Gb/s of data that they want to transfer over the Internet from the generating site to the processing site (e.g., astronomy data, distributed storage, backup systems, etc.), yet they cannot because current TCP-based congestion control cannot achieve or maintain a very high end-to-end throughput, particularly over a wide-area network (WAN) [73, 28].

The problem facing TCP as the per-flow bandwidth-delay increases is multi-fold. Parts of the problem are caused by an inefficient implementation of the TCP stack at the host, which prevents the sender or receiver machine from sustaining a high-speed data stream. Recent papers have attempted to solve this problem by modifying the TCP stack in the operating system. In particular, in [19], the authors discuss a few optimizations to the host's TCP implementation that would allow TCP to send at a peak rate of one Gb/s over a LAN. The objective of these modifications is to minimize the overhead involved in sending and receiving packets by the host machine. The modifications include interrupt coalescing, checksum off-loading, and data copy avoidance by page remapping, as well as the use of a jumbo packet size.

Although optimizing the TCP implementation at the host is necessary to attain a very high per-flow throughput, this optimization constitutes only the first step toward this goal. Achieving and maintaining a very large per-flow throughput over a WAN will necessarily require some changes to TCP's congestion control algorithm. The objective of this thesis is to redesign Internet congestion control to work efficiently in these environments. Below, we describe the main challenges to TCP's congestion control as the per-flow bandwidth-product increases.

**1. Ramping up quickly when there is spare bandwidth:** The poor performance of TCP in high bandwidth-delay environments is caused by both its slow-start and AIMD modes. In particular, AIMD limits TCP's ability to acquire spare bandwidth to one packet per RTT. In high bandwidth environments, it is common to allocate or deallocate a full pure optical path. As a result, a huge amount of spare bandwidth becomes available suddenly. However, a TCP, increasing by one packet per RTT, would take a long period to acquire this bandwidth.[1]

Similarly, slow-start adversely interacts with a high bandwidth-delay product. At first glance, it might seem that slow-start with its exponential increase should help TCP quickly acquire the spare bandwidth, and consequently should mitigate the problem caused by the slow adaptation of AIMD. Unfortunately, this is not the case. The exponential increase performed by a TCP in slow-start is too aggressive and causes an adverse effect. In particular, as the per-flow bandwidth-delay increases, there will be some flows at the bottleneck with very large congestion windows (cwnd). When a new TCP flow starts increasing exponentially, it causes many drops from these flows with very large cwnd. As a result, these flows release a large amount of spare bandwidth into the system. The other flows start grabbing the excess bandwidth by increasing their throughput by one packet per RTT. Given that the released throughput might be thousands of packets, it takes a while before the flows ramp up again. This process repeats with the arrival of a long TCP flow, causing the average link

[1] It is worth noting that the use of jumbo packets improves the performance but does not solve the problem. Jumbo packets are likely to get fragmented in a WAN environment. Also, there are limitations on the size of the packet that are imposed by the size of the fields in various protocols and the effectiveness of error correction at the data link layer. For example, the Ethernet CRC becomes ineffective for packets larger than 9000 bytes.

utilization to stay low.

Since TCP's increase rule is too slow for high bandwidth environments, it is natural to ask: Why would TCP not increase faster than one packet per RTT? Indeed, if TCP increased faster it would acquire the fast response necessary for high bandwidth-delay environments, but while doing so TCP might cause severe congestion in low bandwidth environments. In particular, TCP uses a packet loss as a congestion signal. Hence, its congestion feedback is too crude and does not carry information about the degree of congestion or the amount of spare bandwidth in the network. In the absence of this information, TCP adopts a conservative strategy and increases only by one packet per RTT. Note that one packet/RTT can be too much if the spare bandwidth is hovering around zero or if the capacity of the bottleneck link is just a few Kb/s. One packet/RTT might be too little if the spare bandwidth is a few Gb/s, as is likely in very high bandwidth-delay product environments. The binary feedback does not allow TCP to distinguish between these two cases. Thus, TCP does not have the information necessary to allow it to increase quickly without risking severely congesting the network.

**2. Sustaining a few Gb/s steady state throughput:** The interaction between TCP's congestion control and the link bit errors prevents a flow from maintaining a few Gb/s steady state throughput. As noted in [60], TCP's average throughput, $R$, is inversely proportional to the square root of the packet drop rate, $p$:

$$R \approx \frac{\alpha \cdot s}{RTT\sqrt{p}}, \tag{3.1}$$

where $s$ is the packet size and $\alpha$ is a constant equal to 1.2.[2] Thus, to maintain an average throughput on the order of a few Gb/s, the packet drop rate needs to be exceptionally small. As an example, for a standard TCP with a 1500-bytes packets to maintain a 1 Gb/s steady state throughput over a 100 ms round-trip path, the protocol needs a drop rate smaller than $10^{-8}$, and a bit error rate (BER) less than $10^{-12}$. This BER is challenging even to low BER fiber links [69, 10]. The problem becomes significantly harder once we take into account higher rates, multiple links along the path, and other sources of errors.

**3. Queue size and routers price:** Routers are complex and expensive devices. A large proportion of the router's cost is dedicated to providing a large and fast buffer (usually a combination of SRAM and DRAM). In particular, because TCP is oscillatory, it requires a large queue size to absorb

[2]To simplify the equation, we ignored the term caused by timeouts. Also, we note that the units in the equation should be matched (e.g., $R$ in Kb/s, $s$ in Kb, and RTT in seconds).

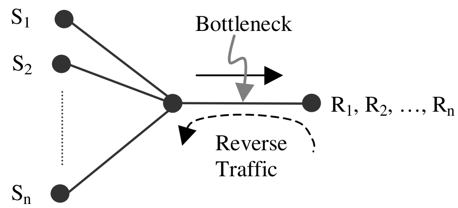



Figure 3-1: A single bottleneck topology.

its oscillations. The current recommendation is to make the buffer size at the routers equal to the bandwidth-delay product. However, as the link capacity becomes very large, it becomes very expensive to provide such a huge amount of memory in the router.[3]

**4. More oscillations and less stability:** Mathematical analysis of current congestion control algorithms reveals that, regardless of the queuing scheme, as the per-flow bandwidth-delay product increases, TCP becomes oscillatory and prone to instability. By casting the problem into a control theory framework, Low et al. [53] show that as capacity or delay increases, Random Early Discard (RED) [35], Random Early Marking (REM) [11], Proportional Integral Controller (PI) [38], and Virtual Queue (VQ) [36] all eventually become oscillatory and prone to instability. They further argue that it is unlikely that *any* Active Queue Management scheme (AQM) can maintain stability over very high-capacity or large-delay links. Furthermore, Katabi and Blake [44] show that Adaptive Virtual Queue (AVQ) [50] also becomes prone to instability when the link capacity is large enough (e.g., gigabit links).

**5. Increased network bandwidth is useless for short flows:** The majority of the flows in the Internet are short flows of a few tens of packets. The increase in link capacity does not improve the transfer delay of short flows because these flows cannot acquire the spare bandwidth faster than slow-start permits. Consequently, they will waste valuable RTTs ramping up even when bandwidth is available.

## 3.2 Simulation Results

In this section, we show simulation results that reveal TCP's inefficiency in high bandwidth-delay environments. Our simulations are run in *ns-2* [59], which provides a fairly detailed implementation of TCP. We simulate the topology in Figure 3-1. The simulations contain 50 long-lived FTP flows

[3] Even in today's Internet, router manufacturers are finding it hard to provide a bandwidth-delay product of buffer space and are pushing toward smaller buffers [6].

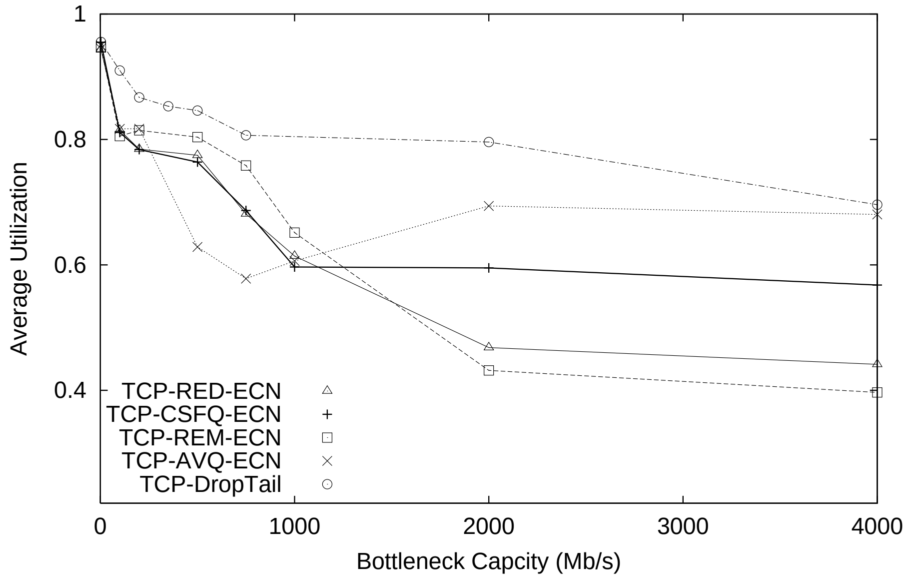



Figure 3-2: TCP's average utilization decreases with increased bottleneck bandwidth.

with the same RTT. The flows interarrival time is exponentially distributed with an average of 2 RTTs. The data packet size is 1000 bytes, the flows are TCP Reno, and the buffer size is always set to the delay-bandwidth product. All simulations run for at least 300 RTTs and contain reverse traffic created by 50 TCP flows. We repeat these simulations for various queuing mechanisms at the routers, such as drop-tail, RED, REM, CSFQ, and AVQ. The parameters of these AQMs are set according to their authors' recommendations [68, 11, 75, 50]. In particular, the RED parameters are $w = 0.002$, $p_{max} = 0.1$, $min_{thresh}$ is one-third of the buffer, $max_{thresh}$ is two-thirds of the buffer, and the gentle option is on. For REM, $\phi = 1.001$, $\gamma = 0.001$, the update interval is set to the transmission time of 10 packets, and *qref* is set to one-third of the buffer size. For AVQ, $\gamma = 0.98$ and $\alpha = 0.15$. For CSFQ, the averaging constants are set to twice as much as the maximum queuing delay as recommended in [75], and the other parameters are set to their default values. In Figure 3-2, we set the round trip propagation delay to 80ms and vary the bottleneck capacity, whereas in Figure 3-3 we set the bottleneck capacity to 155 Mb/s and vary the propagation delay.

The simulation results show that TCP performs poorly as the per-flow bandwidth-delay product increases. In particular, Figures 3-2 and 3-3 show the average bottleneck utilization as a function of increased bandwidth and as a function of increased delay, respectively. (Each point on the graph is the average utilization of a single run.) The figures reveal that, regardless of the queuing discipline, TCP wastes the bandwidth of the bottleneck as either the per-flow bandwidth or delay increases. Given that TCP's behavior depends on the simulation's topology and parameters, these

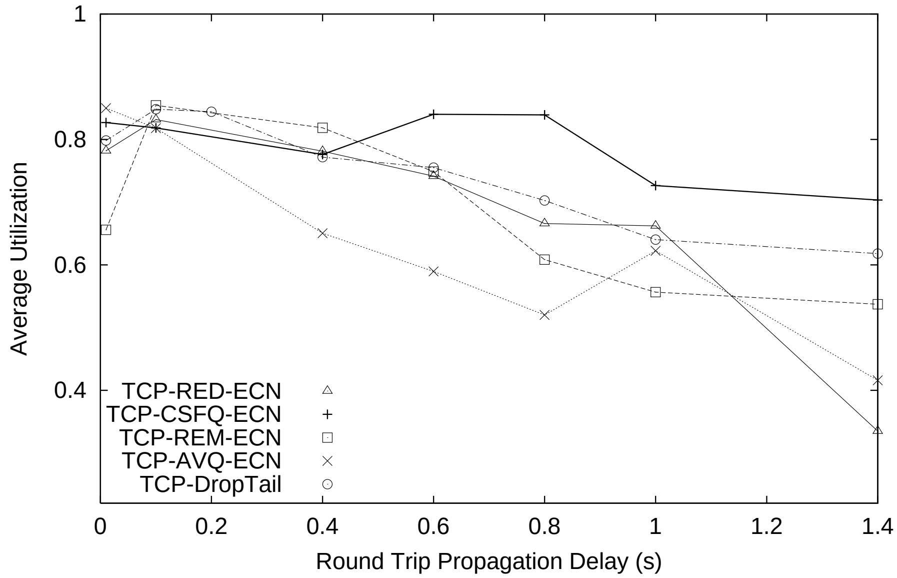



Figure 3-3: TCP's average utilization decreases with increased delay.

figures should not be taken as an exact prediction of TCP's utilization as the bandwidth or delay increases. Rather, they should be taken as an illustration of a clear trend of degraded performance as the per-flow bandwidth-delay product increases.

## 3.3 Summary

Traditionally, the trends in the Internet have pushed toward an increase in the per-flow bandwidth-delay product. This is expected to continue since the capacity of the links is increasing exponentially. Further, the integration of satellite and multi-hop wireless links in the Internet increases the delay along the paths containing such links.

Unfortunately, TCP was not designed to work with a very large per-flow bandwidth-delay product. The TCP congestion control mechanism does not allow it to achieve and maintain a few Gb/s steady-state throughput over the WAN. We can identify a few reasons for this poor performance. The first and most important reason is the lack of fast response; TCP increases by one packet every RTT, which is too slow for high bandwidth-delay environments. Second, TCP's throughput is inversely proportional to the packet drop rate. As a result, achieving a very high per-flow throughput requires links with exceptionally low bit error rate. Third, TCP's oscillations require a buffer of bandwidth-delay product at the router. As the bandwidth increases, this requirement will be extremely difficult to satisfy and will drastically increase the router's cost. Finally, TCP becomes more oscillatory and less stable as either delay or capacity increases.

Although the full impact of large bandwidth-delay products is yet to come, we can see the seeds of these problems in the current Internet. For example, TCP over satellite links has revealed network utilization issues and an undesirable bias against long RTT flows [9]. Also, currently many organizations would like to use Internet2 to attain a few Gb/s end-to-end throughput, but they are unable to achieve this large throughput using TCP [73, 5, 28].

# Chapter 4

# XCP: eXplicit Control Protocol

XCP is a congestion control protocol that embodies the principle of decoupling congestion control from bandwidth allocation and uses this decoupling to achieve good performance in high bandwidth-delay networks. Although our main objective is efficient congestion control in high bandwidth-delay networks, we also want XCP to excel in the traditional environments and to be as scalable and robust as current congestion control. Thus, we start this chapter by asking a general question of how to design congestion control protocols. We argue that decoupling congestion control (i.e., the efficiency question) from fairness (i.e., the bandwidth allocation policy) is a desirable architecture, independent of the networking environment. Then we show that this architecture is particularly important in high bandwidth-delay product networks because it provides the fast response necessary for these environments. Next, we develop the details of the protocol and analyze its characteristics.

## 4.1 Design Rationale

Our initial objective is to step back and rethink Internet congestion control without caring about backward compatibility or deployment. If we were to build a new congestion control architecture from scratch, what might it look like?

The first observation is that packet loss is a poor signal of congestion. While we do not believe a cost-effective network can always avoid loss, dropping packets should be a congestion signal of last resort. As an implicit signal, loss is bad because congestion is not the only source of loss, and because a definite decision that a packet was lost cannot be made quickly. As a binary signal, loss only signals whether there is congestion (a loss) or not (no loss). Thus senders must probe the network to the point of congestion before backing off. Moreover, since the feedback is imprecise, to prevent congestion the increase policy must be conservative and the decrease policy must be aggressive.

Tight congestion control requires explicit and precise congestion feedback. Congestion is not a binary variable, so congestion signalling should reflect the degree of congestion. We propose using precise congestion signalling, where the network explicitly tells the sender the state of congestion and how to react to it. This allows the senders to decrease their sending windows quickly when the bottleneck is highly congested, while performing small reductions when the sending rate is close to the bottleneck capacity. The resulting protocol is both more responsive and less oscillatory.

Second, the aggressiveness of the sources should be adjusted according to the delay in the feedback-loop. The dynamics of congestion control may be abstracted as a control loop with feedback delay. A fundamental characteristic of such a system is that it becomes unstable for some large feedback delay. To counter this destabilizing effect, the system must slow down as the feedback delay increases. In the context of congestion control, this means that as delay increases, the sources should change their sending rates more slowly. This issue has been raised by other researchers [53, 62], but the important question is how exactly feedback should depend on delay to establish stability. Using tools from control theory, we conjecture that congestion feedback based on rate-mismatch should be inversely proportional to delay, and feedback based on queue-mismatch should be inversely proportional to the square of delay.[1]

Robustness to congestion should be independent of unknown and quickly changing parameters, such as the number of flows. A fundamental principle from control theory states that a controller must react as quickly as the dynamics of the controlled signal; otherwise the controller will always lag behind the controlled system and will be ineffective. In the context of current proposals for congestion control, the controller is an Active Queue Management (AQM) scheme. The controlled signal is the aggregate traffic traversing the link. The controller seeks to match input traffic to link capacity. However, this objective might be unachievable when the input traffic consists of TCP flows, because the dynamics of a TCP aggregate depend on the number of flows ($N$). The aggregate rate increases by $N$ packets per RTT, or decreases proportionally to $1/N$ for each drop. Since the number of flows in the aggregate is not constant and changes over time, no AQM controller with constant parameters can be fast enough to operate with an arbitrary number of TCP flows. Thus, a third objective of our system is to make the dynamics of the aggregate traffic independent of the number of flows.

[1] See Appendix B for the derivation.

This leads to the need for decoupling *efficiency control* (i.e., control of utilization or congestion) from *fairness control*. Robustness to congestion requires the behavior of aggregate traffic to be independent of the number of flows in it. However, any fair bandwidth allocation intrinsically depends on the number of flows traversing the bottleneck. Thus, the rule for dividing bandwidth among individual flows in an aggregate should be independent from the control law that governs the dynamics of the aggregate.

Traditionally, efficiency and fairness are coupled since the same control law (such as AIMD in TCP) is used to obtain both fairness and efficiency simultaneously [8, 18, 40, 41, 39]. Conceptually, however, efficiency and fairness are independent. Efficiency involves only the aggregate traffic's behavior. When the input traffic rate equals the link capacity, no queue builds and utilization is optimal. Fairness, on the other hand, involves the relative throughput of flows sharing a link. A scheme is fair when the flows sharing a link have the same throughput irrespective of congestion.

In our approach, a router has both an efficiency controller (EC) and a fairness controller (FC). This separation simplifies the design and analysis of each controller by reducing the requirements imposed. It also permits modifying one of the controllers without redesigning or re-analyzing the other. Furthermore, the separation of the controllers provides a flexible framework for integrating differential bandwidth allocations. For example, allocating bandwidth to senders according to their priorities or the price they pay requires changing only the fairness controller and does not affect the efficiency or the congestion characteristics. Finally, the decoupling allows XCP to attain a very large end-to-end throughput and perform efficiently in high bandwidth-delay product networks. This is because XCP uses a fast controller to control congestion, which increases the traffic proportionally to the amount of spare bandwidth in the network and decreases it proportionally to the degree of congestion. To control fairness, XCP uses an AIMD law which has been shown to lead to fairness [20].

## 4.2 The XCP Protocol

XCP provides a joint design of end systems and routers. Like TCP, XCP is a *window-based* congestion control protocol intended for best effort traffic. However, its flexible architecture can easily support differentiated services as explained in §6. The description of XCP in this section assumes a pure XCP network. In §8, we show that XCP can coexist with TCP in the same Internet and be TCP-friendly. Further, our description assumes that Ack traffic is rarely the cause of congestion. If needed, it is fairly easy to extend the protocol to control Ack traffic in addition to data traffic.

First we give an overview of how control information flows in the network, then in §4.7 we explain feedback computation.

## 4.3 Framework

Senders maintain their congestion window, cwnd, and round trip time, rtt.[2] They communicate their throughput (i.e., cwnd/rtt) and their rtt to the routers via a congestion header in every packet. Routers monitor the input traffic rate to each of their output queues. Based on the difference between the link bandwidth and its input traffic rate, the router tells the flows sharing that link to send faster or slower. It does this by annotating the congestion header of data packets. The value of feedback is divided between flows based on their throughput and RTT values so that the system converges to fairness. A more congested router later in the path can further reduce the feedback in the congestion header by overwriting it. Ultimately, the packet will contain the feedback from the bottleneck along the path. When the feedback reaches the receiver, it is returned to the sender in an acknowledgment packet, and the sender updates its rate accordingly.

## 4.4 The Congestion Header

Each XCP packet carries a congestion header (Figure 1), which is used to communicate a flow's state to routers and feedback from the routers on to the receivers. The field $H\_throughput$ is the sender's current throughput, whereas $H\_rtt$ is the sender's current RTT estimate. These are filled in by the sender and never modified in transit. The remaining field, $H\_feedback$, takes positive or negative values and is initialized by the sender. Routers along the path modify this field to directly control the throughputs of the sources.

## 4.5 The XCP Sender

As with TCP, an XCP sender maintains a congestion window of the outstanding packets, cwnd, and an estimate of the round trip time, rtt. On packet departure, the sender attaches a congestion header to the packet and sets the $H\_throughput$ field to $cwnd/rtt$ and $H\_rtt$ to its current rtt. In the first

[2] In this document, the notation RTT refers to the round trip time, rtt refers to the variable maintained by the source's software, and $H\_rtt$ refers to a field in the congestion header.

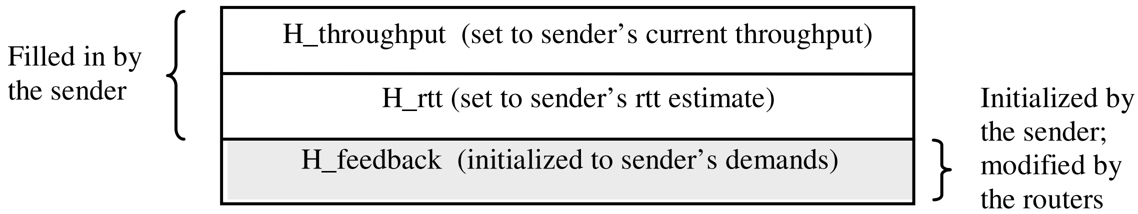



Figure 4-1: Congestion header.

packet of a flow, $H\_rtt$ is set to $-1$ to indicate to the routers that the source does not yet have a valid estimate of the RTT.

The sender initializes the $H\_feedback$ field to request its desired throughput increase. For most best effort applications the desired rate is assumed to be infinite and the H_feedback field is initialized to the maximum possible value. Yet, when the application has a particular desired rate, $r$, the sender sets $H\_feedback$ to the desired increase in the rate ($r - cwnd/rtt$) divided by the number of packets in the current congestion window. If bandwidth is available, this initialization allows the sender to reach the desired rate after one RTT.

Whenever a new acknowledgment arrives, positive feedback increases the senders cwnd and negative feedback reduces it:

$$cwnd = \max(cwnd + H\_feedback \times rtt, s), \tag{4.1}$$

where $s$ is the packet size, and rtt is the most recent estimate of the round trip time. Note that although XCP is a window-based protocol, the feedback returned by the router is an increment in the throughput, which the sender translates into an increment in cwnd.

Additionally, when the application does not send enough traffic to fill the congestion window, the XCP sender quickly ages its cwnd variable to reflect the used window. In particular, for every RTT in which the sender does not send enough to fill the congestion window, cwnd is reduced by $0.5 \times (cwnd - K)$, where $K$ is the number of outstanding packets. Also, as long as the sender does not use all of its cwnd, it initializes H_feedback to zero.

Finally, in addition to direct feedback, XCP still needs to respond to packet losses. At this stage of the design and until the community builds more experience with XCP, we choose to be conservative and react in a similar manner to TCP (i.e., halve cwnd). However, we believe that a better approach would use the TCP throughput equation to estimate a TCP fair rate from recent drops [60]. When drops occur the XCP sender sets its rate to the minimum of the rate derived from the explicit feedback and that derived from the TCP throughput equation. This prevents drastic rate reduction when a packet is lost because of errors at the data link layer. We have not experimented with this approach and we leave it for future work.

## 4.6 The XCP Receiver

An XCP receiver is similar to a TCP receiver except that when acknowledging a packet, it copies the feedback from the data packet to its acknowledgment.[3]

## 4.7 The XCP Router: The Control Laws

The job of an XCP router is to compute a feedback that causes the system to converge to optimal efficiency and max-min fairness. XCP does not drop packets. It operates on top of a dropping policy such as drop-tail, RED, or AVQ. The objective of XCP is to prevent, as much as possible, the queue from building up to the point at which a packet has to be dropped.

To compute the feedback, an XCP router uses an *efficiency controller* and a *fairness controller*. Both of these compute estimates over the average RTT of the flows traversing the link, which smooths the burstiness of a window-based control protocol. Estimating parameters over intervals longer than the average RTT leads to sluggish response, while estimating parameters over shorter intervals leads to erroneous estimates. Also, the XCP controllers make a single control decision every average RTT (the control interval). This is motivated by the need to observe the results of previous control decisions before attempting a new control. For example, if the router tells the sources to increase their congestion windows, it should wait to see how much spare bandwidth remains before telling them to increase again.

The router maintains a per-link estimation-control timer that is set to the most recent estimate of the average RTT of the flows traversing the link. In the remainder of this chapter, we refer to the router's current estimate of the average RTT as $d$ to emphasize this is the feedback delay. The average RTT of the flows is computed every estimation-control interval using the information in the congestion header. To compute the average RTT of the flows the router needs to count the RTT of each flow only once and divide by the number of flows. This however is not simple since the router does not have per-flow state, and consequently cannot distinguish packets of different flows. If the

[3] Note that an XCP receiver may use delayed Acks, in which case the receiver accumulates the feedback and sends the sum of all pending feedback since the last Ack.

router computes the average RTT over the packets then the RTT of each flow would be multiplied by the number of packets this flow sends in the estimation interval $d$. Thus, one way to estimate the average RTT of the flows is to take the average RTT over the packets normalized by the number of packets a flow sends in a control interval. The number is $\frac{r_i}{s_i} \cdot d$, where $r_i$ is the flow's throughput in bytes/sec, and $s_i$ is the packet size in bytes. Thus, the average RTT is:

$$\overline{RTT} = \frac{\sum rtt_i \cdot \frac{s_i}{r_i}}{\sum \frac{s_i}{r_i}} \tag{4.2}$$

where both sums are over the packets observed by the router in a control interval. (Note that $d$ is constant for the duration of the estimation interval, and hence it cancels out.)

The router also estimates the input traffic rate at each link by summing up the traffic arriving during an estimation interval and dividing the sum by $d$. Upon timeout the router updates its estimates and its control decisions.

### 4.7.1 The Efficiency Controller (EC)

The efficiency controller's goal is to maximize link utilization while minimizing drop rate and persistent queues. It looks only at aggregate traffic and need not care about fairness issues, such as which flow a packet belongs to.

The EC computes a desired increase or decrease in the aggregate traffic rate. This aggregate feedback, $\phi$, is computed each control interval:

$$\phi = \alpha \cdot S - \beta \cdot \frac{Q}{d}, \tag{4.3}$$

$\alpha$ and $\beta$ are constant parameters, whose values are set based on our stability analysis (§4.8) to $0.4$ and $0.226$, respectively. The term $d$ is the average RTT, and $S$ is the spare bandwidth defined as the difference between the input traffic rate and link capacity. Note that $S$ can be negative. Finally, $Q$ is the persistent queue size (i.e., the queue that does not drain in a round trip propagation delay), as opposed to a transient queue that results from the bursty nature of all window-based protocols. We compute $Q$ by taking the minimum queue seen by an arriving packet during the last propagation delay, which we estimate by subtracting the local queuing delay from the average RTT.

Equation 9.3.1 makes the feedback proportional to the spare bandwidth because, when $S \geq 0$, the link is underutilized and we want to send positive feedback, while when $S < 0$, the link is congested and we want to send negative feedback. However this alone is insufficient because it would mean we give no feedback when the input traffic matches the capacity, and so the queue does not drain. To drain the persistent queue, we make the aggregate feedback proportional to the persistent queue too. Finally, since the feedback is in bytes/s, to match the unit the queue, $Q$, is divided by the control interval $d$. (Indeed, $Q$ should be scaled down by $d$ to ensure that the system is stable for any feedback delay, as shown in Appendix B.)

To achieve efficiency, we allocate the aggregate feedback to single packets as $H\_feedback$. Since the EC deals only with the aggregate behavior, it does not care which packets get the feedback and by how much each individual flow changes its rate. All the EC requires is that the total traffic changes by $\phi$ over this control interval. How exactly we divide the feedback among the packets (and hence the flows) affects only fairness, and so is the job of the fairness controller.

### 4.7.2 The Fairness Controller (FC)

The job of the fairness controller (FC) is to apportion the feedback to individual packets to achieve fairness. The FC relies on the same principle TCP uses to converge to fairness, namely *Additive-Increase Multiplicative-Decrease (AIMD)*. Thus, we want to compute the per-packet feedback according to the policy:

*If $\phi > 0$, allocate it equally to all flows.*

*If $\phi < 0$, allocate it to flows proportionally to their current throughputs.*

This ensures continuous convergence to fairness as long as the aggregate feedback $\phi$ is not zero. To prevent convergence stalling when efficiency is around optimal ($\phi \approx 0$), we introduce the concept of *bandwidth shuffling*. This is the simultaneous allocation and deallocation of bandwidth such that the total traffic rate (and consequently the efficiency) does not change, yet the throughput of each individual flow changes gradually to approach the flow's fair share. The shuffled traffic is computed as follows:

$$h = \max(0, \gamma \cdot y - |\phi|), \tag{4.4}$$

where $y$ is the input traffic rate and $\gamma$ is a constant set to 0.1. This equation ensures that, every average RTT, at least 10% of the traffic is redistributed according to AIMD.[4] The choice of 10% is a tradeoff between the time to converge to fairness and the disturbance the shuffling imposes on a system that is near optimal efficiency.

[4] Note that $\phi$ is always distributed according to AIMD. Thus, the total amount redistributed according to AIMD every $d$ is $h + |\phi|$, which is at least 10% of the traffic.

Next, we compute the per-packet feedback that allows the FC to enforce the above policies. Since the increase law is additive whereas the decrease is multiplicative, it is convenient to compute the feedback assigned to packet $i$ as the combination of a positive feedback $p_i$ and a negative feedback $n_i$.

$$H\_feedback_i = p_i - n_i. \tag{4.5}$$

First, we compute the case when the aggregate feedback is positive ($\phi > 0$). In this case, we want to increase the throughput of all flows by the same amount. Thus, we want the change in the throughput of any flow $i$ to be proportional to the same constant, (i.e., $\Delta r_i \propto constant$). The next step is to translate this desired change of throughput to per-packet feedback that will be reported in the congestion header. The total change in the throughput of a flow is the sum of the per-packet feedback it receives. Thus, we obtain the per-packet feedback by dividing the change in throughput by the expected number of packets from flow $i$ that the router sees in a control interval $d$. This number is proportional to the flow's throughput (in bytes per second) divided by its packet size (in bytes), $\frac{r_i}{s_i}$. Since $d$ is constant for the duration of the control interval, the per-packet positive feedback is inversely proportional to its throughput divided by its packet size, (i.e., $p_i \propto \frac{1}{r_i/s_i}$). Thus, positive feedback $p_i$ is given by:

$$p_i = \xi_p \frac{s_i}{r_i}, \tag{4.6}$$

where $\xi_p$ is a constant.

The total increase in the aggregate traffic rate is $h + \max(\phi, 0)$, where $max(\phi, 0)$ ensures that we are computing the positive feedback. This is equal to the sum of the increase in the rates of all flows in the aggregate, which is the sum of the positive feedback a flow has received, and so:

$$h + \max(\phi, 0) = \sum^{L} p_i, \tag{4.7}$$

where $L$ is the number of packets seen by the router during the control interval $d$. From this, $\xi_p$ can be derived as:

$$\xi_p = \frac{h + \max(\phi, 0)}{\sum \frac{s_i}{r_i}}. \tag{4.8}$$

Similarly, we compute the per-packet negative feedback given when the aggregate feedback is negative ($\phi < 0$). In this case, we want the decrease in the throughput of flow $i$ to be proportional to its current throughput (i.e., $\Delta r_i \propto r_i$). Again, the desired per-packet feedback is the desired change in throughput divided by the expected number of packets from this flow that the router sees in an interval $d$. As shown above, this latter number is proportional to $\frac{r_i}{s_i}$. Thus, we finally find that the per-packet negative feedback should be proportional to the packet size (i.e., $n_i \propto s_i$). Thus negative feedback $n_i$ is given by:

$$n_i = \xi_n \cdot s_i \tag{4.9}$$

where $\xi_n$ is a constant.

As with the increase case, the total decrease in the aggregate traffic rate is the sum of the decrease in the rates of all flows in the aggregate:

$$h + \max(-\phi, 0) = \sum^{L} n_i. \tag{4.10}$$

As so, $\xi_n$ can be derived as:

$$\xi_n = \frac{h + \max(-\phi, 0)}{\sum s_i}, \tag{4.11}$$

where the sum is over all packets in a control interval (average RTT).

The per-packet feedback derived above allows the FC to allocate bandwidth according to the AIMD policy. All of the values that appear in the above equations are easily obtained at the router. In particular, the throughput, $r_i$, is in the congestion header, the packet size, $s_i$, is in the IP header, and the aggregate traffic rate, $y$, and the average RTT, $d$, are measured by the router.

Finally, the FC tracks the total amounts of positive and negative allocations since the beginning of the current control interval. It stops giving positive feedback when the sum of the positive feedback allocated since the beginning of the interval reaches the target of $h + max(\phi, 0)$, and stops allocating negative feedback when the sum of the negative feedback allocated since the beginning of the interval reaches the target of $h + max(-\phi, 0)$. This helps bounding the allocation error in the aggregate traffic and tightly adheres with the decision of the EC.

### 4.7.3 Notes on the Efficiency and Fairness Controllers

This section summarizes the important points about the design of the efficiency controller and the fairness controller.

- As mentioned earlier, the efficiency and fairness controllers are decoupled. Specifically, the efficiency controller uses a Multiplicative-Increase Multiplicative-Decrease law (MIMD), which increases the traffic rate proportionally to the spare bandwidth in the system (instead of increasing by one packet/RTT/flow as TCP does). This allows XCP to quickly acquire the positive spare bandwidth even over high capacity links. The fairness controller, on the other hand, uses an Additive-Increase Multiplicative-Decrease law (AIMD), which converges to fairness [20]. Thus, the decoupling allows each controller to use a suitable control law.

The particular control laws used by the efficiency controller (MIMD) and the fairness controller (AIMD) are not the only possible choices. For example, in [45] we describe a fairness controller that uses a binomial law similar to those described in [13]. We chose the control laws above because our analysis and simulation demonstrate their good performance.

- We note that the efficiency controller satisfies the requirements in §4.1. The dynamics of the aggregate traffic are specified by the aggregate feedback and stay independent of the number of flows traversing the link. Additionally, in contrast to TCP where the increase/decrease rules are indifferent to the degree of congestion in the network, the aggregate feedback sent by the EC is proportional to the degree of under- or over-utilization. Furthermore, since the aggregate feedback is given over an average RTT, XCP becomes less aggressive as the round trip delay increases. Scaling down the gain with increased delay is a well-known technique that helps stabilizing feedback control loops [62].[5]

- Although the fairness controller uses AIMD, it converges to fairness faster than TCP. As shown in [20], TCP converges to fairness because of the sequence of additive increases and multiplicative decreases. However, in TCP, multiplicative-decrease is tied to the occurrence of a drop, which should be a rare event. In contrast, with XCP multiplicative-decrease is decoupled from drops and is performed every average RTT.

- XCP is fairly robust to estimation errors. For example, we estimate the value of $\xi_p$ every $d$ and use it as a prediction of $\xi_p$ during the following control interval (i.e., the following $d$). If we underestimate $\xi_p$, we will fail to allocate all of the positive feedback in the current control interval. Nonetheless, the bandwidth we fail to allocate will appear in our next estimation of the input traffic as a spare bandwidth, which will be allocated (or partially allocated) in the following control interval. Thus, in every control interval, a portion of the spare bandwidth is allocated until none is left. Since our underestimation of $\xi_p$ causes reduced allocation, the convergence to efficiency is slower than if our prediction of $\xi_p$ had been correct. Yet the error does not stop XCP from reaching full utilization. Similarly, if we overestimate $\xi_p$ then we will

[5]The relation between XCP's dynamics and feedback delay is hard to fully grasp from Equation 9.3.1. We refer the reader to Equation B.2, which shows that the change in throughput based on rate-mismatch is inversely proportional to delay, and the change based on queue-mismatch is inversely proportional to the square of delay.

allocate more feedback to flows at the beginning of a control interval and run out of aggregate feedback quickly. This uneven spread of feedback over the allocation interval does not affect convergence to utilization but it slows down convergence to fairness. A similar argument can be made about most estimation errors; they mainly affect the convergence time rather than the correctness of the controllers.

There is however one type of error that may prevent convergence to complete efficiency, which is the unbalanced allocation and deallocation of the shuffled traffic. For example, if by the end of a control interval we deallocate all of the shuffled traffic but fail to allocate it, then the shuffling might prevent us from reaching full link utilization. Yet note that the shuffled traffic is only $10\%$ of the input traffic. This limits the impact of the worst case under-utilization that might happen because of a completely unbalanced shuffling. Furthermore, shuffling exists only when $|\phi| < 0.1y$.

- XCP's parameters (i.e., $\alpha$ and $\beta$) are constant whose values are independent of the number of sources, the delay, and the capacity of the bottleneck. This is a significant improvement over previous approaches where specific values for the parameters work only in specific environments (e.g, RED), or the parameters have to be chosen differently depending on the number of sources, the capacity, and the delay (e.g., AVQ). In §4.8, we show how these constant values are chosen.

- Finally, implementing the efficiency and fairness controllers is fairly simple and requires only a few lines of code as shown in Appendix A. We note that an XCP router performs only a few additions and 3 multiplications per packet, making it an attractive choice even as a backbone router. A pseudo code of our implementation is provided in Appendix A.

## 4.8 Stability Analysis

We use a fluid model of the traffic to analyze the stability of XCP. Our analysis considers a single link traversed by multiple XCP flows. For the sake of simplicity and tractability, similarly to previous work [50, 38, 53, 57], our analysis assumes flows are long-lived and have a common, finite, and positive round trip delay, and neglects boundary conditions (i.e., queues are bounded, rates cannot be negative). In §5, we demonstrate through extensive simulations with larger topologies, different RTTs, and boundary conditions, that our results still hold for these cases.

The main result can be stated as follows.

**Theorem 1** *Suppose the round trip delay for all sources is d. If the parameters $\alpha$ and $\beta$ satisfy:*

$$0 < \alpha < \frac{\pi}{4\sqrt{2}} \quad and \quad \beta = \alpha^2\sqrt{2},$$

*then the system is stable independently of delay, capacity, and number of sources.*

The details of the proof are given in Appendix B. The idea underlying the stability proof is the following. Given the assumptions above, our system is a linear feedback system with delay. The stability of such systems may be studied by plotting their open-loop transfer function in a Nyquist plot. We prove that by choosing $\alpha$ and $\beta$ as stated above, the system satisfies the Nyquist stability criterion with gain margin greater than one and positive phase margin, independently of delay, capacity, and number of sources.[6]

Theorem 1 provides a range for $\alpha$. The exact value of $\alpha$ is a trade-off between fast response and robust stability. A large $\alpha$ allows XCP to converge quickly to optimal utilization but a small $\alpha$ increases the protocol's robustness against estimation errors.

## 4.9 Summary

We have presented XCP, a new congestion control protocol designed around the principle of decoupling the mechanism that controls congestion from the bandwidth allocation policy. We have analyzed the design principles underlying XCP and shown that the decoupling is a desirable architecture for any networking environment, and is particularly useful in high bandwidth-delay product networks. XCP is a window-based protocol in which the routers tell the senders explicitly how to adjust their rates to eliminate congestion and achieve good utilization. An XCP router has a congestion controller and a fairness controller. The congestion controller controls the aggregate traffic matching it to the capacity of the link. It increases proportionally to the spare bandwidth which allows it the fast response necessary for sustaining a large per-flow bandwidth-delay product. The fairness controller, on the other hand, controls the relative throughput of the flows in the aggregate without affecting the aggregate rate. It uses an AIMD control law to converge to fair bandwidth

[6]The gain margin determines by how much we can multiply the signal (i.e., total traffic) while maintaining stability. The phase margin is the difference between the maximum phase shift and 180 degrees and addresses the stability with respect to delays.

allocation. Using tools from control theory, we have modeled XCP and shown the model is stable independent of delay, capacity, and number of sources.

# Chapter 5

# Performance

In this chapter, we use extensive simulations to explore the characteristics of XCP and compare it with TCP. We focus on a pure XCP network where all flows use XCP for congestion control. In §8, we explore the coexistence of XCP with other types of traffic, particularly TCP. Our simulations demonstrate that XCP outperforms TCP both in conventional and high bandwidth-delay environments. They also reveal that XCP has the characteristic of almost never dropping packets. Additionally, the simulations show that in contrast to TCP, the new protocol dampens oscillations and smoothly converges to high utilization, small queue size, and fair bandwidth allocation. Further, they show that the protocol is robust to highly varying traffic demands and high variance in the flows' round trip times. Finally, by complying with the conditions in Theorem 1, we were able to a priori choose constant values for $\alpha$ and $\beta$ that worked for every single scenario that we simulated.

The simulations in this chapter cover a wide variety of scenarios. In particular, we simulate capacities in [1.5 Mb/s, 4 Gb/s], propagation delays in [10 ms, 1.4 sec], number of long-lived sources in [1, 1000], and arrival rates of Web-like flows in [10/sec, 1000/sec]. We also simulate 2-way traffic (with the resulting Ack compression[1] [80]) and dynamic environments with arrivals and departures of short Web-like flows. In all of these simulations, we set $\alpha = 0.4$ and $\beta = 0.226$ showing the applicability of the results in Theorem 1.

[1] Ack compression refers to the phenomenon in which multiple Acks arrive at the sender back-to-back, which causes the TCP sender to transmit a burst of back-to-back packets.

## 5.1 Simulation Setup

Our simulations use the packet-level simulator *ns-2* [59], which we have extended with an XCP module.[2] We compare XCP with TCP Reno[3] over the following queuing disciplines:

- **Random Early Discard** (RED [35]). Our experiments use the "gentle" mode and set the parameters according to the authors' recommendations in [68]. The minimum and the maximum thresholds are set to one-third and two-thirds the buffer size, respectively.

- **Random Early Marking** (REM [11]). Our experiments set REM parameters according to the authors' recommendation provided with their code. In particular, $\phi = 1.001$, $\gamma = 0.001$, the update interval is set to the transmission time of 10 packets, and *qref* is set to one-third of the buffer size.

- **Adaptive Virtual Queue** (AVQ [50]). As recommended by the authors, our experiments use $\gamma = 0.98$ and compute $\alpha$ based on the equation in [50]. Yet, as shown in [44], the equation for setting $\alpha$ does not admit a solution for high capacities. In these cases, we use $\alpha = 0.15$ as used in [50].

- **Core Stateless Fair Queuing** (CSFQ [75]). In contrast to the above AQMs, whose goal is to achieve high utilization and small queue size, CSFQ aims for providing high fairness in a network cloud with no per-flow state in core routers. We compare CSFQ with XCP to show that XCP can be used within the CSFQ framework to improve its fairness and efficiency. Again, the parameters are set to the values chosen by the authors in their *ns* implementation except for the averaging constants which are set to 100 ms as recommended in [75].

The simulator code for these AQM schemes is provided by their authors. Further, to allow these schemes to exhibit their best performance, we simulate them with ECN enabled.

In all of our simulations, the XCP parameters are set to $\alpha = 0.4$ and $\beta = 0.226$. We experimented with XCP with both drop-tail and RED dropping policies. There was no difference between the two cases because XCP almost never dropped packets.

Most of our simulations use the topology in Figure 5-1. The bottleneck capacity, the round trip delay, and the number of flows vary according to the objective of the experiment. The buffer size

[2] The code is available at www.ana.lcs.mit.edu/dina/XCP.
[3] Reno is a particular variant of TCP that is widely used in today's Internet [61].

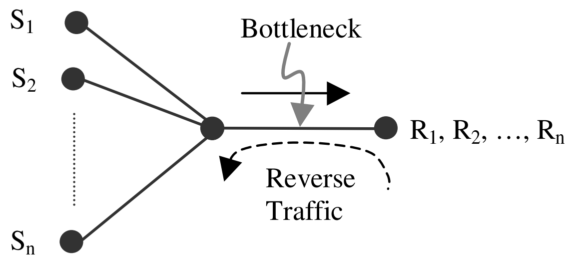



Figure 5-1: A single bottleneck topology.

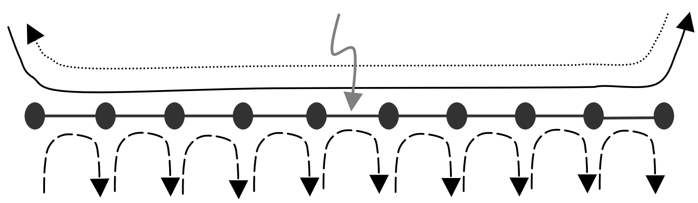



Figure 5-2: A parking lot topology. Arrows represent traffic directions.

is always set to the delay-bandwidth product. The data packet size is 1000 bytes. Simulations over the topology in Figure 5-2 are used to show that our results generalize to larger and more complex topologies. When unspecified, it should be assumed that the simulation topology is that in Figure 5-1, the flows RTTs are equivalent, and the sources are long-lived FTP flows. The simulation running times vary depending on the propagation delay but are always larger than 300 RTTs.

## 5.2 Comparison with TCP and AQM Schemes

In this section, we compare XCP's performance with that of TCP with various queuing schemes. We show that, unlike TCP, XCP provides good performance in environments with large per-flow bandwidth-delay product. Further, even in traditional environments, XCP is more efficient and fairer than TCP.

### 5.2.1 Impact of High Capacity

Since our main objective in designing XCP is to solve the problems TCP faces in large bandwidth-delay networks, we start our XCP evaluation by examining the behavior of both TCP and XCP as the per-flow bandwidth increases. In this experiment, 50 long-lived FTP flows share a bottleneck. The round trip propagation delay is 80 ms. Additionally, there are 50 flows traversing the reverse path and used merely to create a 2-way traffic environment with the potential for Ack compression.

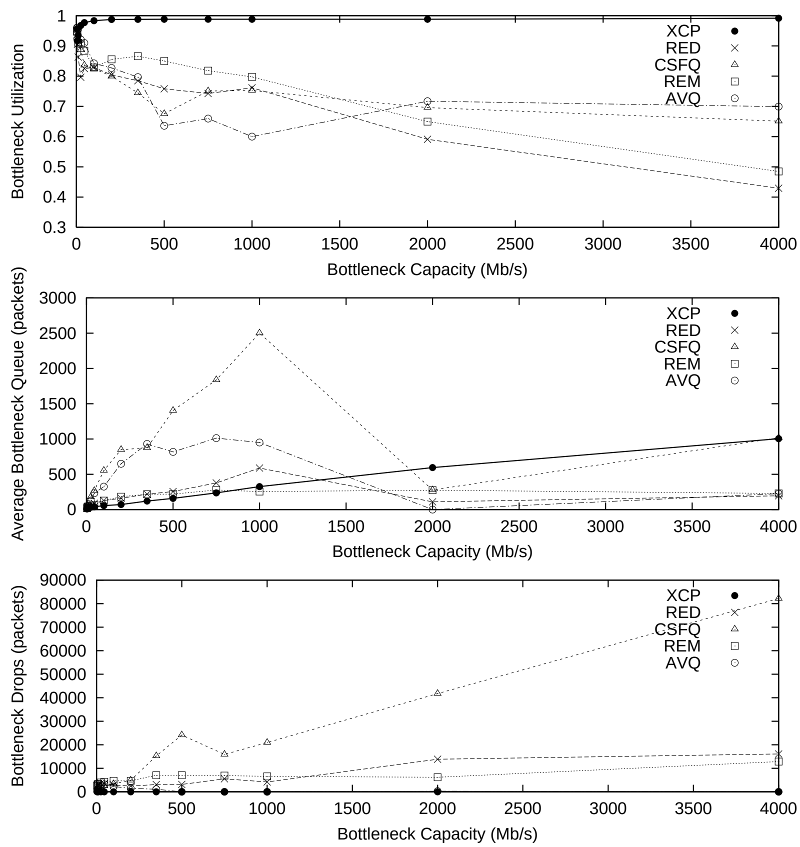



Figure 5-3: XCP significantly outperforms TCP in high bandwidth environments. The graphs compare the efficiency of XCP with that of TCP over RED, CSFQ, REM, and AVQ, as a function of capacity.

Since XCP is based on a fluid model and estimates some parameters, the existence of reverse traffic, with the resulting burstiness, tends to stress the protocol.

Figure 5-3 shows that an increase in link capacity (with the resulting increase of per-flow bandwidth) causes a significant degradation in TCP's performance, irrespective of the queuing scheme. (The trend in Figure 5-3 also applies to drop-tail, which we did not include in the figure to maintain readability. See Figure 3-2.) In contrast, XCP's utilization is always near optimal, independent of the link capacity. Furthermore, XCP never drops any packet, whereas TCP, with all AQMs but AVQ, drops thousands of packets despite its use of ECN. Although the XCP queue increases with the capacity, the queuing delay does not increase because the larger capacity causes the queue to drain faster.

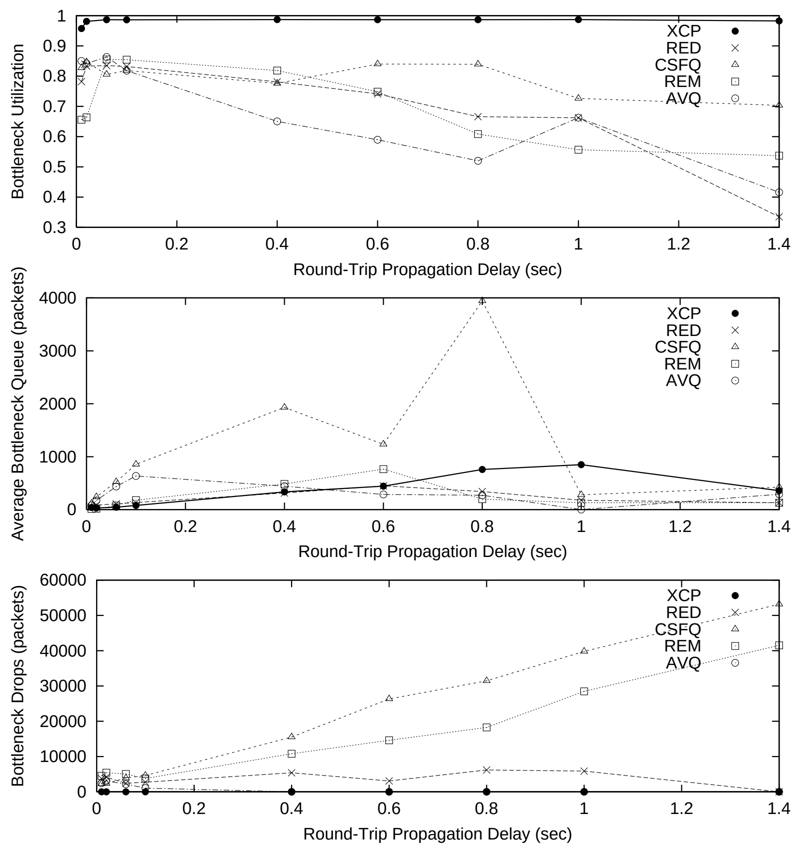



Figure 5-4: XCP significantly outperforms TCP in high delay environments. The graphs compare bottleneck utilization, average queue, and number of drops as round trip delay increases when flows are XCPs and when they are TCPs over RED, CSFQ, REM, and AVQ.

### 5.2.2 Impact of Feedback Delay

We set the bottleneck capacity to 150 Mb/s and study the impact of increased delay on the performance of congestion control. All other parameters have the same values used in the previous section.

Figure 5-4 shows that as the propagation delay increases, TCP's utilization degrades considerably regardless of the queuing scheme. (The trend in Figure 5-4 also applies to drop-tail, which we did not include in the figure to maintain readability. See Figure 3-3.) In contrast, XCP maintains high utilization independently of delay. The adverse impact of large delay on TCP's performance has been noted over satellite links. The bursty nature of TCP has been suggested as a potential explanation and packet pacing has been proposed as a solution [9]; however, this experiment shows that burstiness is a minor factor. In particular, XCP is a bursty window-based protocol but it copes with delay much better than TCP. It does so by adjusting its aggressiveness according to round trip

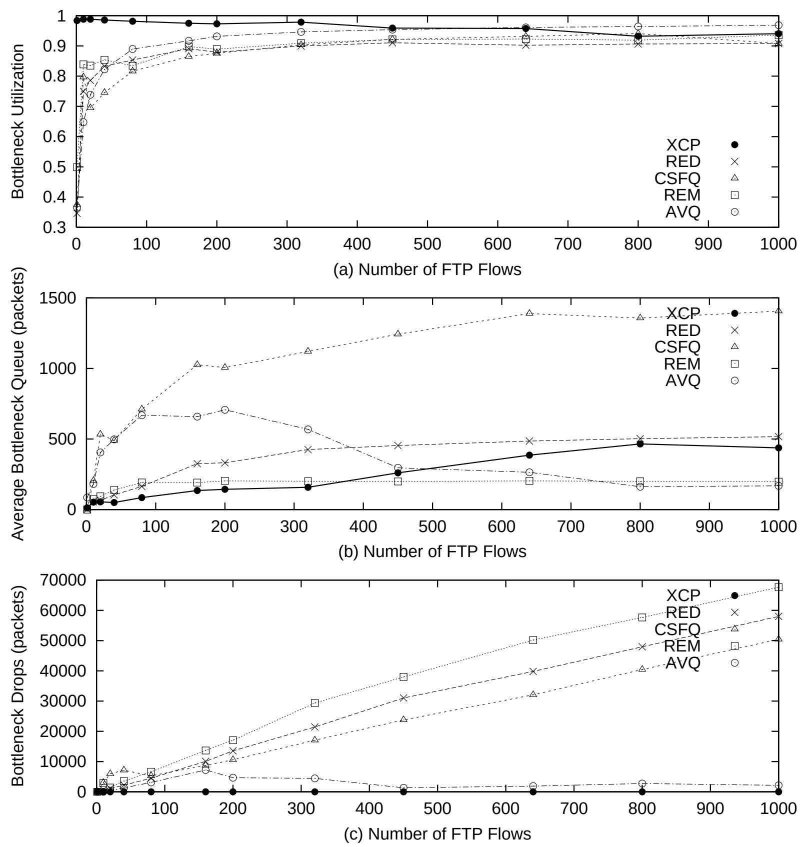



Figure 5-5: XCP is efficient with any number of flows. The graphs compare the efficiency of XCP and TCP with various queuing schemes as a function of the number of flows.

delay.

### 5.2.3 Impact of Number of Flows

We set the bottleneck capacity to 150 Mb/s and round trip propagation delay at 80 ms and repeat the same experiment with a varying number of FTP sources. Other parameters have the same values used in the previous experiment.

Figure 5-5 shows that, overall, XCP exhibits good utilization, reasonable queue size, and no packet losses. The increase in XCP queue as the number of flows increases is a side effect of its high fairness (see Figure 5-7). When the number of flows is larger than 500, the fair congestion window is between two and three packets. In particular, the fair congestion window is a real number but the effective (i.e., used) congestion window is an integer number of packets. Thus, as the fair window size decreases, the effect of the rounding error increases causing a disturbance. Consequently, the queue increases to absorb this disturbance. The various AQM schemes are not as affected by

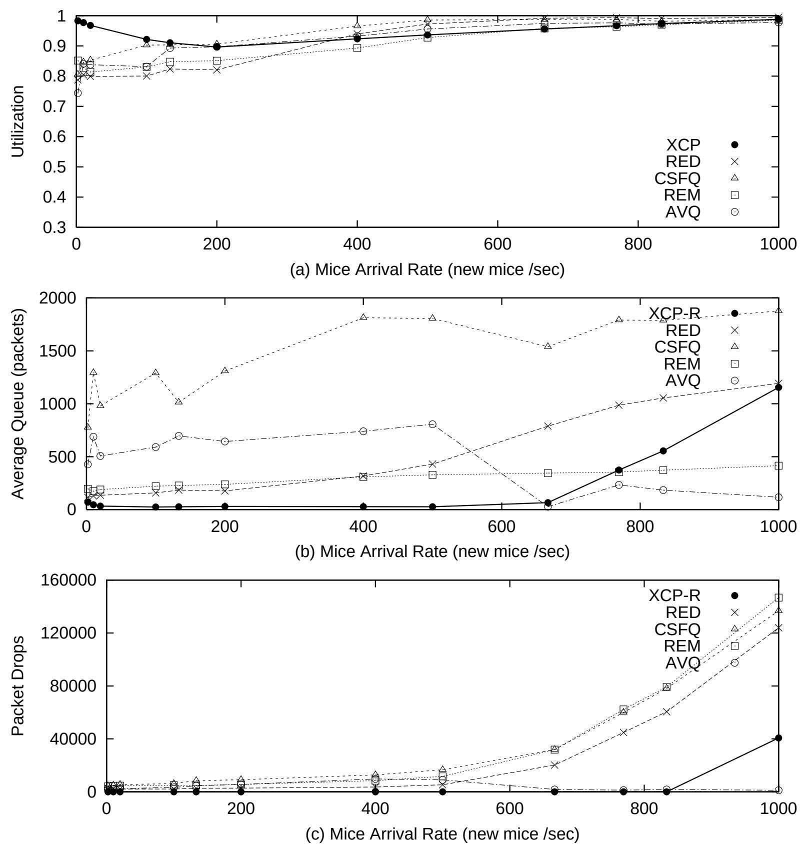



Figure 5-6: XCP is robust and efficient in environments with arrivals and departures of short web-like flows. The graphs compare the efficiency of XCP to that of TCP over various queuing schemes as a function of the arrival rate of web-like flows.

rounding errors because they are not as fair as XCP (see Figure 5-7). Also, it is worth noting that as the number of sources increases to the point that the per-flow cwnd is a few packets (i.e., 2 or 3 packets), AVQ achieves 0.03 more utilization than XCP and almost half the average queue size. On the other hand, in these simulations AVQ (with ECN) drops thousands of packets, whereas XCP drops no packets at all.

### 5.2.4 Impact of Short Web-Like Traffic

Since most of flows in the Internet are short web-like flows, it is important to investigate the impact of such dynamic flows on congestion control. In this experiment, we have 50 long-lived FTP flows traversing the bottleneck link. Also, there are 50 flows traversing the reverse path whose presence emulates a 2-way traffic environment with the resulting Ack compression. The bottleneck bandwidth is 150 Mb/s and the round trip propagation delay is 80 ms. Short flows arrive according to a Poisson process. Their transfer size is derived from a Pareto distribution with an average of 30 packets (*ns* implementation with `shape_` $= 1.35$), which complies with real web traffic [23].

Figure 5-6 plots bottleneck utilization, average queue size, and total number of drops, all as functions of the arrival rate of short flows. The figure demonstrates XCP's robustness in dynamic environments with a large number of flow arrivals and departures. XCP continues to achieve high utilization, small queue size, and zero drops even as the arrival rate of short flows becomes significantly high. At arrival rates higher than 800 flows/s (more than 10 new flows every RTT), XCP starts dropping packets. This behavior is not caused by the environment being highly dynamic, it happens because at such high arrival rates the number of simultaneously active flows is a few thousand. Thus, there is no space in the pipe to maintain a minimum of one packet from each flow and XCP drops become inevitable. In this case, XCP's behavior approaches the underlying dropping policy, which is RED for Figure 5-6. The above argument might seem contradictory since AVQ can maintain a low drop rate for the same number of flows. But actually, for XCP, drops are inevitable because our current *ns* implementation does not decrease the congestion window below one packet per-flow regardless of the how negative the feedback is. Thus, to reduce the sending rate to less than 1 packet/RTT, the router has to drop packets to activate the exponential back-off.[4] In the AVQ simulation, we have the ECN option on. Thus, the router can activate the exponential back-off by marking the packets instead of dropping them. The exponential back-off causes some flows to stay silent for most of the simulation time leaving a smaller number of simultaneously active flows. We note that we did not call the exponential back-off function when an XCP sender with cwnd = 1 receives a negative feedback only to simplify the ns code, however, we believe that in this case either the sender should back-off or decrease the packet size appropriately. It is also worth noting, that in comparison with RED, CSFQ, and REM which are simulated with ECN on, AVQ does a better job at higher arrival rates.

### 5.2.5 Fairness

Next, we shift our attention to fairness and show that XCP is significantly fairer than TCP, regardless of the queuing scheme. We have 30 long-lived FTP flows sharing a single 30 Mb/s bottleneck. We conduct two sets of simulations. In the first set, all flows have a common round-trip propagation delay of 40 ms. In the second set of simulations, the flows have different RTTs in the range [40 ms, 330 ms] ($RTT_{i+1} = RTT_i + 10ms$).

[4] The "exponential back-off" happens in both XCP and TCP when a sender with cwnd of 1 packet receives a drop or an ECN mark. In this case, the sender keeps cwnd = 1 and stops sending for sometime. If the sender receives more congestion signals then the silence period increases exponentially.

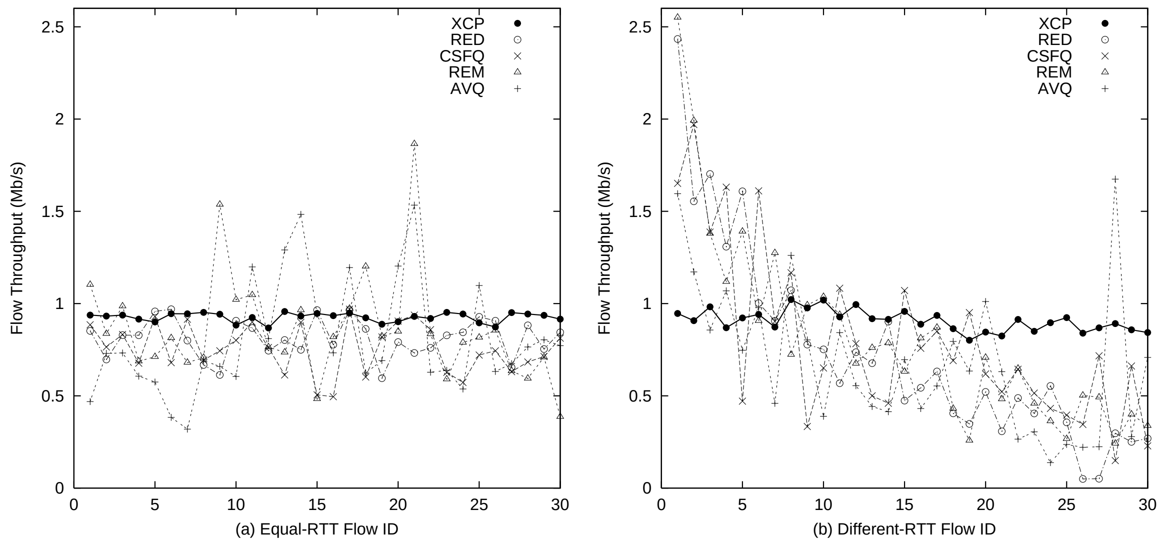



Figure 5-7: XCP is fair to both equal and different RTT flows. The graphs compare XCP's Fairness to that of TCP over RED, CSFQ, REM, and AVQ. Graph (b) also shows XCP's robustness to environments with different RTTs.

Figures 5-7-a and 5-7-b demonstrate that, in contrast to other approaches, XCP provides a fair bandwidth allocation and does not have any bias against long RTT flows. Furthermore, Figure 5-7-b demonstrates XCP robustness to high variance in the RTT distribution. Thus, although XCP computes an estimate of the average RTT of the system, it still operates correctly in this environment where different flows have substantially different RTTs. This issue is further investigated in §5.4.

### 5.2.6 A More Complex Topology

For the sake of clarity, the above experiments used a simple topology with a single bottleneck. In this section, we show that XCP maintains a good performance in more complex scenarios with larger topologies and multiple bottlenecks. This experiment uses the 9-link topology in Figure 5-2, although results are very similar for topologies with more links. Link 5 has the lowest capacity, namely 50 Mb/s, whereas the other links are 100 Mb/s. All links have 20 ms one-way propagation delay. Fifty flows, represented by the solid arrow, traverse all links in the forward direction. Fifty cross flows, illustrated by the small dashed arrows, traverse each individual link in the forward direction. Fifty flows also traverse all links along the reverse path.

Figure 5-8 illustrates the average utilization, queue size, and number of drops at every link. In general, all schemes maintain a reasonably high utilization at all links (note the y-scale). However, the trade-off between optimal utilization and small queue size is handled differently in XCP from

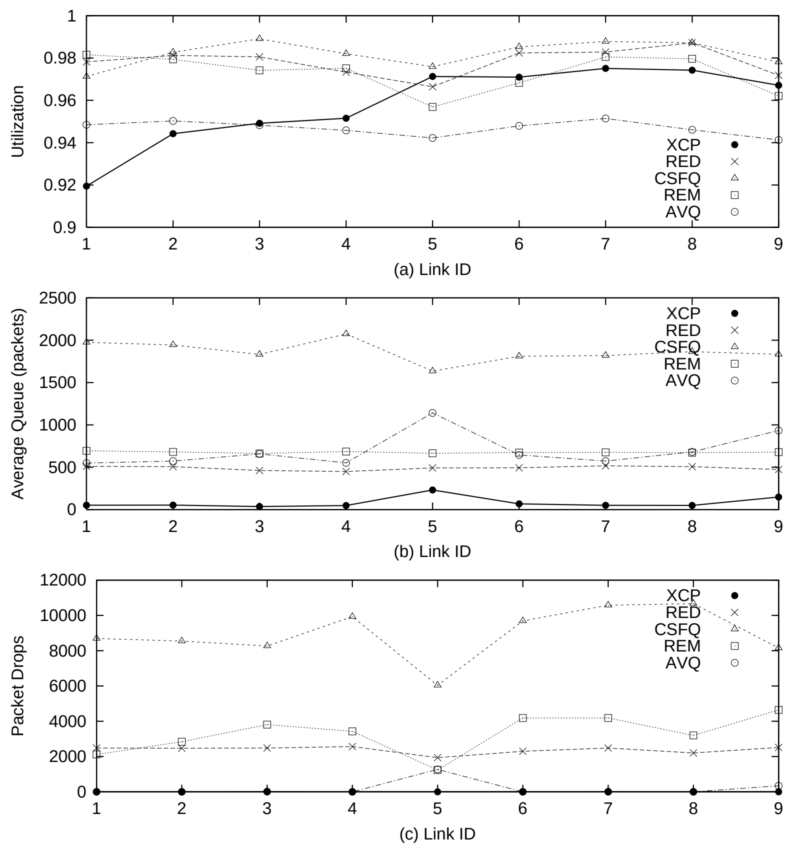



Figure 5-8: Simulation with multiple congested queues. Utilization, average queue size, and number of drops at nine consecutive links (topology in Figure 5-2). Link 5 has the lowest capacity along the path.

the various AQM schemes. XCP trades a few percent of utilization for a considerably smaller queue size. XCP's lower utilization in this experiment compared to previous ones is due to disturbance introduced by shuffling. In particular, at links 1, 2, 3, and 4 (i.e., the set of links preceding the lowest capacity link along the path), the fairness controller tries to shuffle bandwidth from the cross flows to the long-distance flows, which have lower throughput. Yet, these long-distance flows are throttled downstream at link 5, and so cannot benefit from this positive feedback. This effect is mitigated at links downstream from link 5 because they can observe the upstream throttling and correspondingly reduce the amount of negative feedback given (see implementation in Appendix A). In any event, as the total shuffled bandwidth is less than 10%, the utilization is always higher than 90%.

It is possible to modify XCP to maintain 100% utilization in the presence of multiple congested links. In particular, we could modify XCP so that it maintains the queue around a target value rather than draining all of it. This would cause the disturbance induced by shuffling to appear as a fluctuation in the queue rather than as a drop in utilization. However, we believe that maintaining

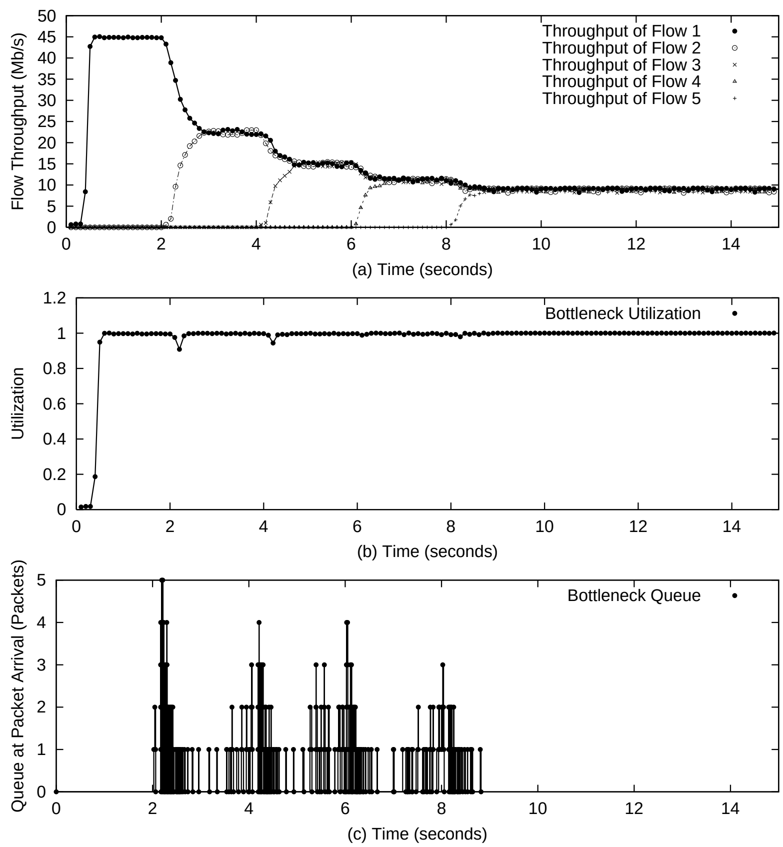



Figure 5-9: XCP's smooth convergence to high fairness, good utilization, and small queue size. Five XCP flows share a 45 Mb/s bottleneck. They start their transfers at times 0, 2, 4, 6, and 8 seconds.

a small queue size is more valuable than a few percent increase in utilization when flows traverse multiple congested links. In particular, it leaves a safety margin for bursty arrivals of new flows. In contrast, the large queues maintained at all links in the TCP simulations cause every packet to wait at all of the nine queues, which considerably increases end-to-end delay.

At the end of this section, it is worth noting that, in all of our simulations, the average drop rate of XCP was less than $10^{-6}$, which is on average three orders of magnitude smaller than the other schemes despite their use of ECN.

## 5.3 The Dynamics of XCP

While the simulations presented above focus on long term average behavior, this section shows the short term dynamics of XCP. In particular, we show that XCP's utilization, queue size, and throughput exhibit very limited oscillations. Therefore, the average behavior presented in the section above is highly representative of the general behavior of the protocol.

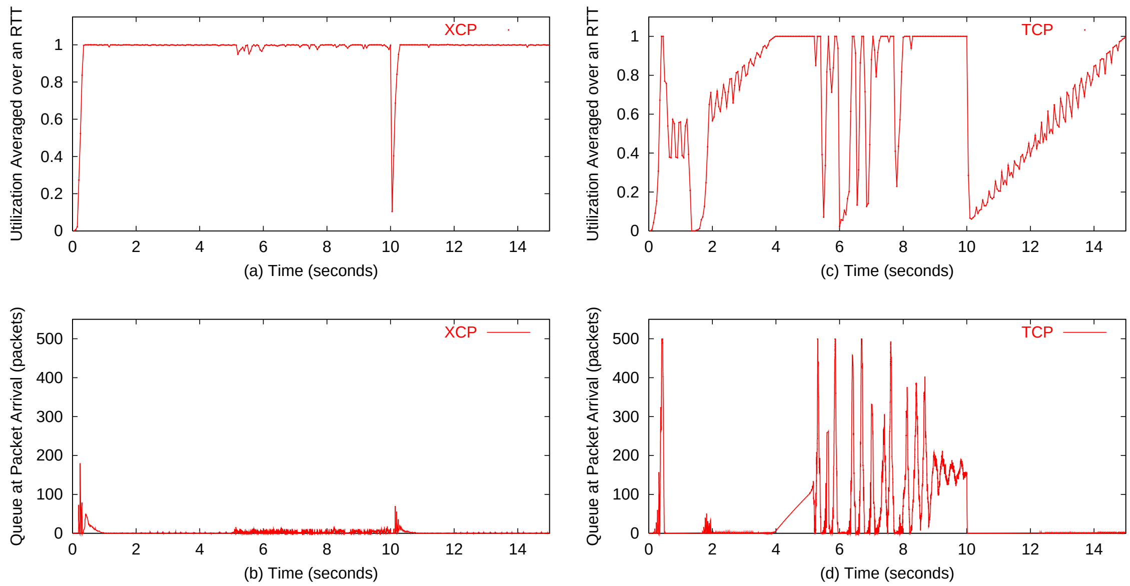



Figure 5-10: XCP is more robust against sudden increase or decrease in traffic demands than TCP. Ten FTP flows share a bottleneck. At time $t = 4$ seconds, we start 100 additional flows. At $t = 8$ seconds, these 100 flows are suddenly stopped and the original 10 flows are left to stabilize again.

### 5.3.1 Convergence Dynamics

We show that XCP dampens oscillations and converges smoothly and quickly to high utilization small queues and fair bandwidth allocation. In this experiment, 5 long-lived flows share a 45 Mb/s bottleneck and have a common RTT of 40 ms. The flows start their transfers two seconds apart at 0, 2, 4, 6, and 8 seconds.

Figure 5-9-a shows that whenever a new flow starts, the fairness controller reallocates bandwidth to maintain max-min fairness. Figure 5-9-b shows that decoupling utilization and fairness control ensures that this reallocation is achieved without disturbing the utilization. Finally, Figure 5-9-c shows the instantaneous queue, which effectively absorbs the new traffic and then drains afterwards.

### 5.3.2 Robustness to Sudden Increase or Decrease in Traffic Demands

In this experiment, we examine performance as traffic demands and dynamics vary considerably. We start the simulation with 10 long-lived FTP flows sharing a 100 Mb/s bottleneck with a round trip propagation delay of 40 ms. At $t = 4$ seconds, we start 100 new flows and let them stabilize. At $t = 8$ seconds, we stop these 100 flows, leaving the original 10 flows in the system.

Figure 5-10 shows that XCP adapts quickly to a sudden increase or decrease in traffic. It shows the utilization and queue, both for the case when the flows are XCP, and for when they are TCPs

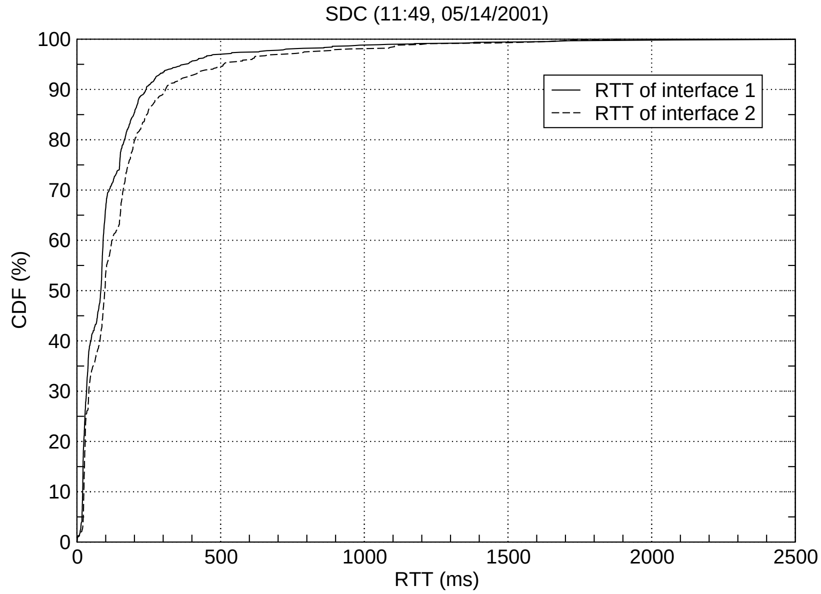



Figure 5-11: The RTT distribution at an OC3 access link at the Supercomputer center in San Diego, (a reproduction of Figure 13-a-top in [42]).

traversing RED queues. XCP absorbs the new burst of flows without dropping any packets, while maintaining high utilization. TCP, on the other hand, is highly disturbed by the sudden increase in the traffic and takes a long time to restabilize. When the flows are suddenly stopped at $t = 10$ seconds, XCP quickly reallocates the spare bandwidth and continues to have high utilization. In contrast, TCP takes long time to acquire the spare bandwidth and wastes the network resources.

## 5.4 Robustness to Large RTT Variance

Our model and analysis of XCP assume that flows have the same RTT, yet in reality flows differ widely in their RTTs. Thus, our implementation uses the average RTT as an estimate of the feedback delay in the system. In particular, XCP sets the estimation control interval to $d = min(5ms, \overline{RTT})$, where $\overline{RTT}$ is the average RTT of the flows traversing the link. In this section, we examine the robustness of using the average RTT as our control and estimation interval when the flows traversing the bottleneck have very different RTTs.

First, we simulate an environment where the RTT distribution is the same as the one reported in recent measurements [42]. Figure 5-11[5] shows the RTT distribution on both interfaces of an OC3 access link at the San Diego Supercomputer center which carries commodity Internet traffic, a distribution that is fairly representative of major high bandwidth links [42]. We use this CDF

[5]The figure is a reproduction of Figure 13-a-top in [42].

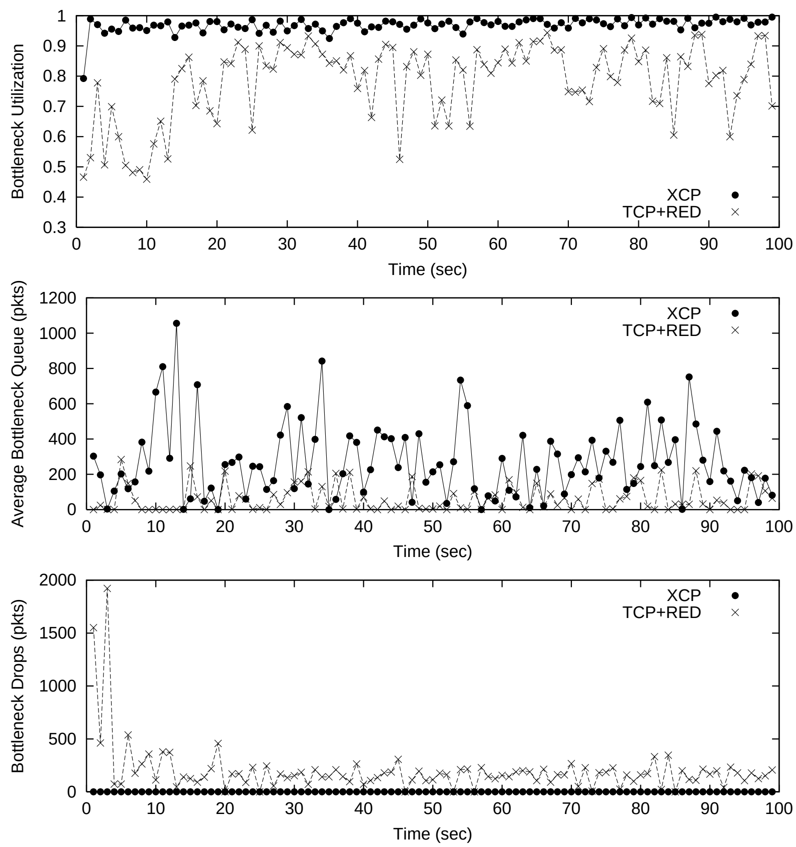



Figure 5-12: The performance of XCP and TCP+RED+ECN when the RTT is distributed according to Figure 5-11 (values are logged every second).

to generate the RTTs of the flows in our simulation, which we run on the topology in Figure 5-1. There are 100 long-lived flows along the forward direction. The bottleneck capacity is 100 Mb/s, the queues have enough space to buffer packets for 200 ms, and the reverse traffic is generated by 50 long-lived flows (all other parameters comply with the description in §5.1). Figure 5-12 shows the performance of XCP given the RTT distribution in Figure 5-11. (The values are logged every second.) The XCP performance is very satisfactory; the utilization is close to optimal and the drop rate is virtually zero. In comparison with TCP over RED+ECN, XCP shows a larger average queue. XCP uses this queue to absorb the oscillations resulting from the variance in the RTT, and to maintain optimal utilization and zero drop rate, which TCP fails to achieve.[6]

Next, we want to stress the design further by increasing the RTT variance. In particular, although the RTTs in Figure 5-11 span a wide range, most flows have an RTT $< 200$ ms. To increase the variance, we use a uniform distribution of RTTs over the range [5 ms, $\tau$]. We repeat the above

[6] We have run the same simulation with TCP over drop-tail. We have also repeated the simulations with short Web-like flows which arrive at 100 flow/s. The results are similar in nature to those reported in Figure 5-12.

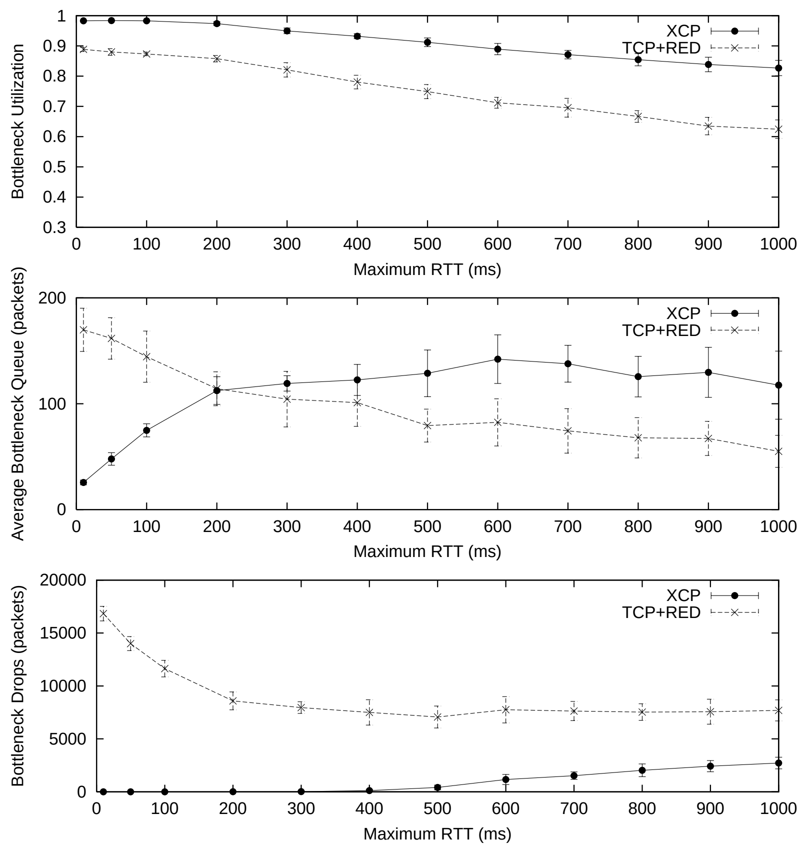



Figure 5-13: The performance of XCP and TCP+RED when the RTT is uniformly distributed over [5 ms, $\tau$], where $\tau$ is the value on the x-axis.

experiment with various values of $\tau$ and plot the average utilization, average queue size, and total number of drops all as functions of $\tau$ for both XCP and the combination of TCP Reno with RED queues. The TCP plot is used as a reference. As seen in Figure 5-13, for $\tau < 200$ ms XCP performance is nearly optimal. As $\tau$ increases, the performance degrades smoothly but for RTTs $\in$ [5 ms, 1000 ms], the XCP performance stays acceptable. In comparison, TCP over RED also seems to suffer from increased RTT variance. The impact in the case of TCP is more complex since TCP suffers from the increase in the average delay (see Figure 5-4) as well as the increases in the variance.

## 5.5 Impact of Error-Based Drops

Packets may be dropped because of errors at the physical or data-link layer. In this section, we use simulation to explore the impact of such non-congestion drops on XCP. Before presenting our simulation results, we emphasize that the impact of a certain packet error rate on XCP's performance

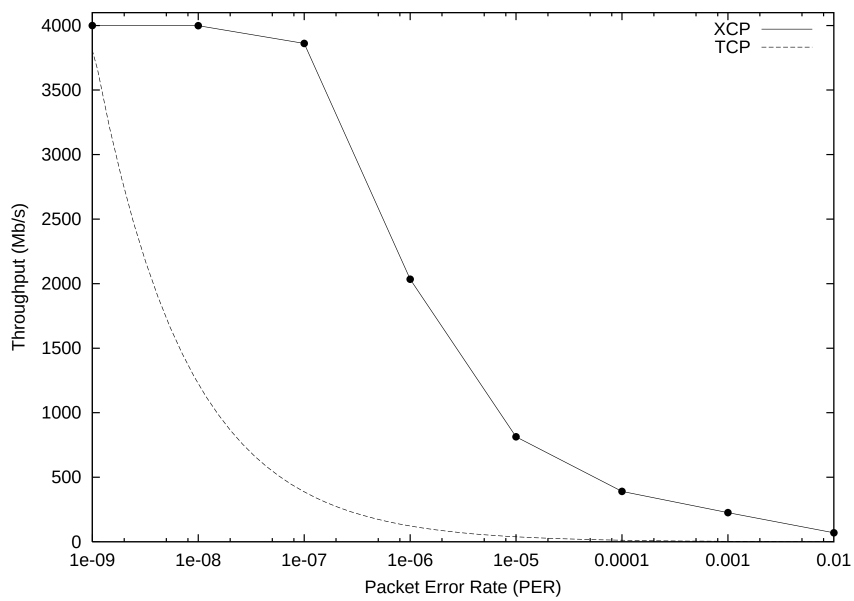



Figure 5-14: The throughput of a single XCP flow as a function of packet error rate. Capacity is 4 Gb/s, RTT = 80 ms. For a reference, the figure also shows the steady state throughput of a TCP flow as a function of the packet error rate.

depends on the way XCP reacts to drops. Our current implementation reacts to a drop by halving cwnd. We believe this reaction is sub-optimal, but until the community gains more experience with XCP, it is desirable to be as conservative as TCP is in reacting to drops. The simulations in this section show the impact of error-based drops on XCP's throughput given the current design.

In this experiment, we run a single flow over a 4 Gb/s link. The round trip delay is 80 ms, and the buffer size is set to a bandwidth-delay product. The duration of each simulation varies between 50 sec and 300 sec depending on the error rate. The link drops packets according to a Bernoulli random variable with a varying average. Figure 5-14 shows the throughput of a single XCP flow as a function of the packet error rate (PER). For comparison, the figure shows the steady state TCP throughput as estimated by Equation 3.1.[7] The figure shows that XCP's throughput is almost an order of magnitude larger than TCP's throughput for the same PER and RTT. (Note that the throughput cannot be larger than 4 Gb/s, which is the capacity of the link.)

[7]We compare against the TCP throughput as computed by equation 3.1 rather than the simulated throughput because, unless we run the simulations for very long time, the simulation results will be drastically affected by slow-start and do not reflect the steady state TCP throughput. This is particularly true for low drop rates. Given the high capacity of the link it takes a few days of real time to run such simulations. XCP on the other hand does not perform slow start, and thus does not suffer from this effect.

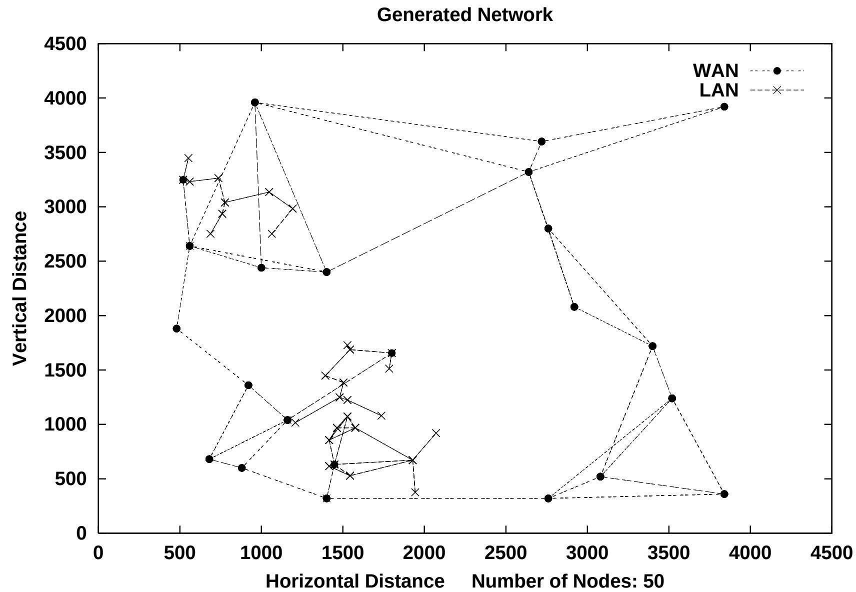



Figure 5-15: A large simulation with 50 nodes, 153 links, and 1000 flows.

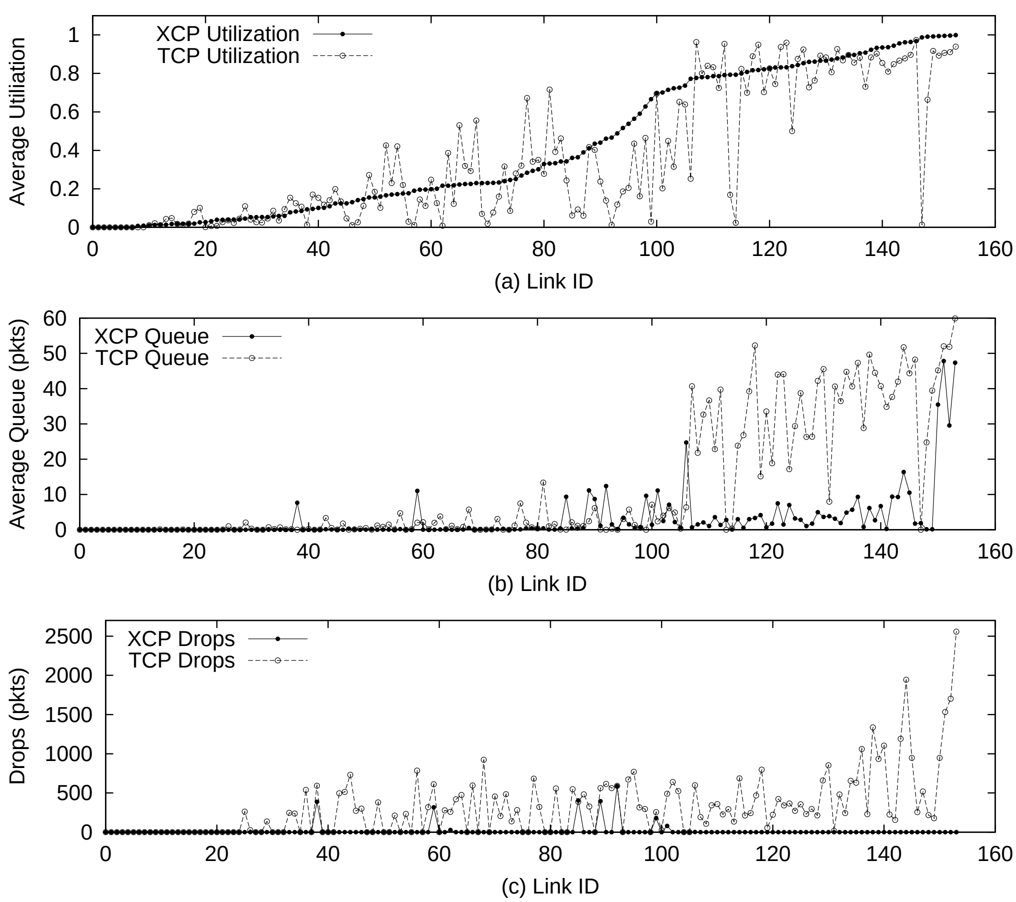



Figure 5-16: A comparison between the efficiency of XCP and TCP over RED.

## 5.6 Large Scale Simulations

The objective of this section is to increase our confidence that XCP is safe to run in the Internet, and that the performance it shows over small topologies is representative and scales to larger and more complex scenarios. Although the simulations in this section are in no way close in size or heterogeneity to that of the Internet, they are much larger than the ones traditionally used to evaluate congestion control protocols. We note that, since we have already established XCP's superior performance in large bandwidth-delay networks (see §5.2.1), in this section we only use links with moderate capacity and delay, showing that even in these traditional environments, XCP outperforms TCP.

Our simulation uses the topology in Figure 5-15, which has been generated using the *tiers* topology generator [26]. The topology consists of a WAN and three LANs, and contains 50 nodes and 153 simplex links. The links' capacities are either 100 Mb/s or 10 Mb/s, and their one-way propagation delays vary between 2 ms and 45 ms. Values for capacities and delays are chosen automatically by *tiers*. The simulation contains 1000 FTP flows, whose sources and destination are chosen randomly from the nodes in the graph. The buffer size is set to 150 packets at all queues. We run the same simulation first with XCP, then with TCP with RED queues and the ECN option on.

Our results show that XCP has a better performance than TCP. In particular, Figure 5-16-a illustrates the average utilization for all links in the graph. The x-axis is the link ID, and the y-axis is its average utilization. The IDs are ordered according to the increased XCP utilization. The lines connecting the points only help in tracking the various values. Some links have low XCP utilization because they are not bottlenecks and the flows traversing them are constrained somewhere else. Figure 5-16-a shows that in most cases XCP improves utilization. In particular, there are a few cases where XCP results in a good utilization while TCP shows a 0% utilization (e.g., link 147). These links usually have one or two flows that encountered a large number of drops in slow start. The drops cause an exponential back-off which ultimately stall the flows. In XCP, there are no drops and the flows efficiently use the links.

Figure 5-16-b plots the average queue size. It shows that XCP always maintains smaller queues than TCP. Finally, Figure 5-16-c, shows that while TCP dropped tens of thousands of packets despite the ECN option, XCP dropped only 5 packets.

## 5.7 Summary

We use extensive *ns* simulations to evaluate XCP and compare it with TCP. Our simulations show that XCP is particularly useful in high bandwidth-delay environments. It quickly acquires a large amount of spare bandwidth when the link is underutilized, and releases it in time of congestion. Even in traditional environments, XCP improves the utilization, reduces the queue size, and almost eliminates packet drops. Further, XCP is fairer than TCP and has no bias against large RTT flows.

# Chapter 6

# Quality of Service (QoS)

Internet quality of service (QoS) has been an extremely active area of research for many years. It addresses the problem of providing a controllable and predictable differentiation in the throughput, delay, or loss rate of different flows. The differentiation may be in absolute terms (e.g., flow $i$ obtains a 1 Mb/s throughput) or in relative terms (e.g., flow $i$ obtains more throughput than flow $j$).

The objective of this chapter is two-fold. First, we show that the XCP framework can accommodate QoS, and that previous QoS proposals continue to be applicable. Second, we develop in detail a small set of simple services that can support almost all application requirements. In particular, since XCP maintains small queues and very low drop rates, QoS mechanisms that provide delay or loss differentiation become less important than in the context of a TCP-based Internet. Thus, we focus on bandwidth differentiation and develop mechanisms that provide both relative and absolute bandwidth allocations.

In this chapter, we assume the users comply with the rules of the protocol and do not address misbehaving, which we discuss in §7. The rest of this chapter is organized as follows. First, we look at the various QoS models and the services they provide. Second, we explore the QoS that can be provided within the XCP framework. Then, we explain our relative and absolute bandwidth differentiation schemes.

## 6.1 QoS Models

Internet QoS may take one of two forms: 1) relative service differentiation; 2) absolute service differentiation. Relative service differentiation is motivated by the fact that users differ in the value they attach to using the network. Some users are willing to pay more than others for obtaining better service in time of congestion. Thus, the relative service differentiation model does not specify the bandwidth, delay, or loss rate a flow perceives in absolute terms. It only specifies how much better the service would be compared to that received by a lower service flow.

The relative differentiation can be coarse where there are few classes from which the user chooses [27], or it can be fine-grained where there is an infinite spectrum of service differentiation [46, 24, 75, 25]. For example, in [27], Dovrolis et al propose a scheduler that provides proportional delay differentiation between a small number of service classes. The network operator controls the ratios of the average delays in these classes by adjusting the associated weights. The user chooses the class that fits its application requirements, knowing that flows in higher classes experience lower delays. In contrast to the previous example where there are a few service classes, Crowcroft and Oechslin propose a scheme for an infinite spectrum of relative bandwidth differentiation [24]. Their protocol, called MulTCP, is a modified TCP that allocates bandwidth to flows proportionally to their weights. MulTCP achieves this differentiation by simulating a number of TCPs equal to the flow's weight.

The second QoS model, called absolute service, is motivated by application needs. Some applications such as video streaming, IP-telephony, and games have specific throughput or delay requirements and do not adapt well to variations in these parameters. To accommodate these applications, the absolute service model provides the users with some guarantees regarding their bandwidth, delay, or drop rate. Unlike relative services, the absolute service model specifies the quality of the service in absolute terms. For example, a flow might be guaranteed a bandwidth of one Mb/s or a zero drop rate.

There are three different approaches for providing absolute QoS. The first is *guaranteed service*, which is defined in the context of the IntServ IETF group [4]. It provides per-flow bandwidth and delay guarantees at the expense of high router complexity. In particular, current solutions for providing guaranteed service require each router to process a per-flow signaling message and to maintain per-flow state on the control path, as well as to perform per-flow classification, scheduling and buffer management on the data path. Performing per-flow maintenance at the routers is problematic because the size of this information increases linearly with the number of flows. Even more important, the information is dynamic, changing very quickly. As a result, tracking the state of the active flows is difficult and not robust.

The second approach is *premium service*, which is defined in the context of the DiffServ IETF group [1]. It provides per-flow bandwidth guarantees and per-aggregate delay guarantees. In con-

| Relative Service | | Absolute Service | | |
|---|---|---|---|---|
| Few Classes | Infinite Differentiation | IntServ | DiffServ | SCORE |

Table 6.1: QoS Models

trast to guaranteed service, to provide a premium service in DiffServ, only edge routers need to maintain per-flow state, while maintaining simple behavior at core routers. Pushing the complexity to the edge routers improves the scalability of the service.

The third approach, called SCORE [74], bridges the gap between a stateful and stateless architecture. In SCORE, only edge routers maintain per-flow state, which they transfer to core routers by annotating the packets of a flow. The core routers do not keep any per-flow state. They use the state in the packet to make their decisions. A SCORE network emulates a Jitter Virtual Clock (JVC) and provides guarantees on the per-packet delay. Thus, SCORE provides a service similar to IntServ but with lower complexity. However, SCORE is still more complex than DiffServ since core routers have to do per-packet scheduling and random access to packets in the queue.

Independent of the approach, absolute QoS models require either a substantially provisioned network or admission control. Otherwise, the resources might become over-committed, which prevents the delivery of the service. Table 6.1 summarizes the various QoS models.

## 6.2 Providing QoS Within the XCP Framework

Since there is a huge number of QoS proposals, it is very difficult to discuss each one of them and its applicability to an XCP environment. Thus, we limit the discussion in this chapter to the QoS models themselves. First, we look at the relative service model. We note that delay and loss differentiation are not as important in the XCP framework because XCP exhibits a small queue and a near zero drop rate as shown in §5. Thus, we focus on relative bandwidth differentiation and demonstrate in §6.3 that XCP can effectively provide a wide range of continuous bandwidth differentiation.

Shifting attention to the absolute QoS model, we note that mechanisms that provide absolute bandwidth or delay guarantees usually do not need congestion control. In this case, each flow has a service profile that specifies its rate and burstiness. Adherence to the service profiles prevents congestion. Thus, in this case, the absolute QoS mechanism controls the guaranteed traffic. Still, XCP can play an important role by controlling the best-effort traffic to efficiently use the spare bandwidth left by the guaranteed traffic. In §6.4, we present a DiffServ approach for providing bandwidth guarantees in an XCP network.

## 6.3 Relative Bandwidth Allocation in XCP

Since XCP naturally exhibits small queue size and near-zero drop rate, providing relative delay or loss differentiation is less important in an XCP network than it is in traditional TCP-like environments. Therefore, in this section we focus on relative bandwidth allocation.

By decoupling efficiency and fairness, XCP provides a flexible framework for designing a variety of bandwidth allocation policies. In particular, the max-min fairness controller, described in §4.2, may be replaced by a controller that causes the flows' throughputs to converge to a different bandwidth allocation (e.g., proportional fairness, priority, etc.). To do so, the designer needs to replace the AIMD policy used by the FC with a policy that allocates the aggregate feedback to the individual flows so that they converge to the desired rates.

We describe a simple scheme that provides relative bandwidth allocation. Each flow has a weight. As long as the flow does not face any congestion, it can increase its rate to satisfy its demands. In time of congestion, the network allocates bandwidth so that the throughputs of the flows bottlenecked at the same link are proportional to their weights, (i.e., $\frac{throughput_i}{weight_i} = \frac{throughput_j}{weight_j}$).[1] We provide this relative bandwidth differentiation by replacing the AIMD policy with:

*If $\phi > 0$, allocate it to flows proportionally to their weights;*

*If $\phi < 0$, allocate it to flows proportionally to their current throughputs.*

The algorithm above allows each flow to simulate multiple flows, where the number of simulated flows is proportional to the weight. This causes the flows to achieve the desired relative bandwidth allocation.

We can implement the above algorithm by modifying the congestion header. In particular, the sender replaces the $H\_throughput$ field with the current throughput divided by the weight of the flow (i.e, throughput/weight). Since the fairness controller uses only the H_throughput field to define the fair share of a flow, this minor modification of the meaning of H_throughput is enough to produce a service that complies with the above model.

[1] Of courese, flows that do not have enough demands can send at lower throughputs than the ones indicated by the equality. One can think of these flows as being bottlenecked at the source rather than the link.

### 6.3.1 Evaluation

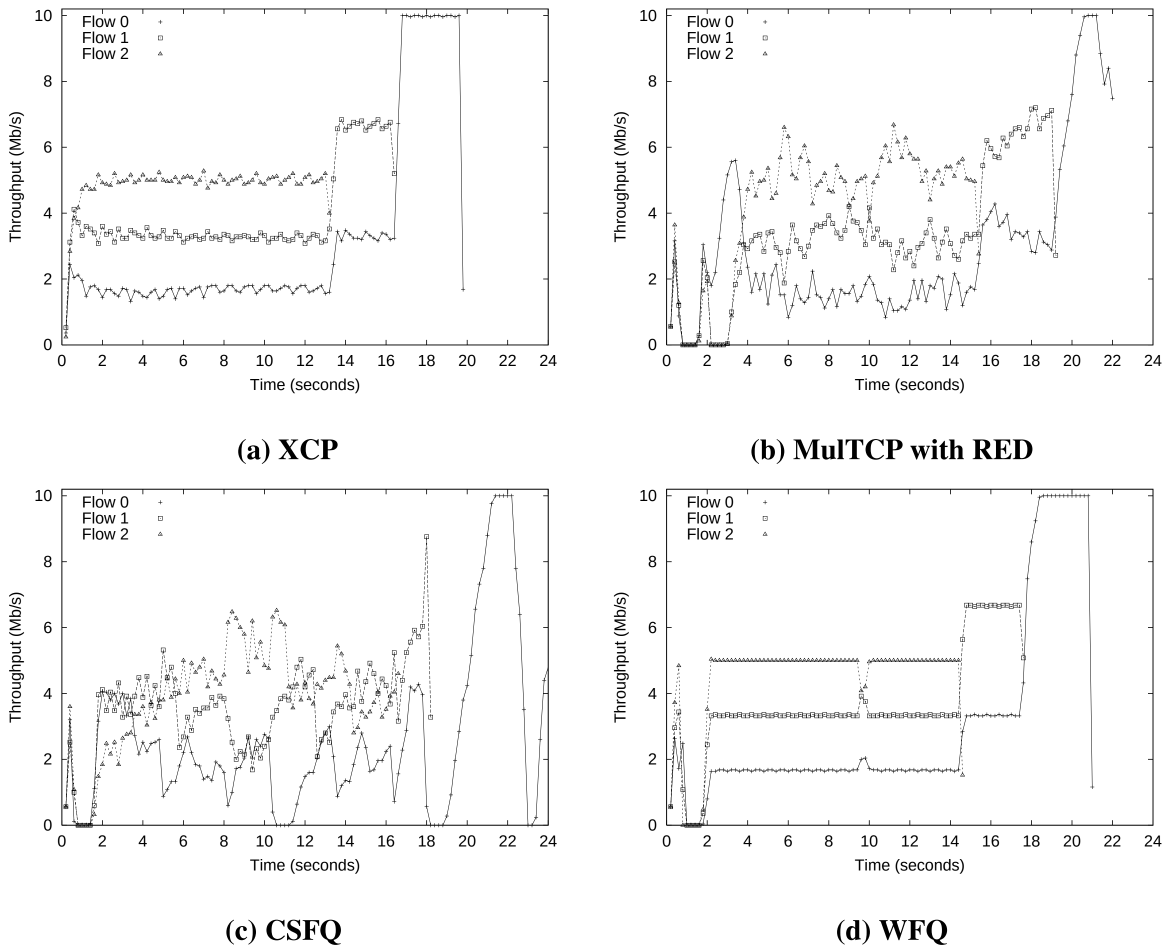
In this section, we show that, in contrast to other protocols, XCP can provide relative bandwidth allocation over very short time scales (a few RTTs). Further, the relative differentiation of sources' rates in XCP can reach more than three orders of magnitude, whereas other schemes fail in providing accurate differentiation when the weight ratios become very large.

**Experiment 1:** In the first experiment, three XCP sources share a 10 Mb/s bottleneck. The corresponding weights are $w_1 = 5$, $w_2 = 10$, and $w_3 = 15$. Each source wants to transfer a file of 10 MB, and they all start together at $t = 0$. The simulations are run over the topology in Figure 5-1 where the round trip delay is set to 80 ms and the queue size is set to the delay bandwidth product. The results are illustrated in Figure 6-1-a. At the beginning, when all flows are active, their throughputs are 5 Mb/s, $3\frac{1}{3}$ Mb/s, and $1\frac{2}{3}$ Mb/s, which are proportional to their corresponding weights. After Flow 1 finishes its transfer, the remaining flows acquire the freed bandwidth such that their throughputs continue to be proportional to their weights. Note the high responsiveness of the system. In particular, when Flow 1 finishes its transfer, freeing half of the link capacity, the other flows' sending rates adapt in a few RTTs (with no overshoot or oscillations).

We compare our scheme against MulTCP [24], CSFQ [75], and Weighted Fair Queuing (WFQ) [25]. We have implemented MulTCP in ns. The MulTCP simulations in this chapter use RED queues with $w = 0.002$, $p_{max} = 0.1$, $min_{thresh}$ is one-third of the buffer, $max_{thresh}$ is two-thirds of the buffer, and the gentle option is on. (We tried MulTCP with drop-tail but the performance was much worse because of drop synchronization.) The CSFQ code is provided by its designers. The averaging constants are set to twice as much as the maximum queuing delay as recommended in [75], and the queue threshold is set to $0.4$ the buffer size.[2] We extended the ns fair queuing (FQ) module to provide weighted fairness. One problem that we faced is that the FQ code allocates to each flow a separate buffer and does not allow different flows to share buffer space. Thus, we had to set the buffer space for each flow to a delay bandwidth product, which means that the total buffer size used by WFQ is three times as much as that used by the other schemes including XCP.

We repeat the experiment above with each of these schemes. The results are illustrated in Figures 6-1-b, 6-1-c, and 6-1-d. MulTCP and CSFQ are less accurate than XCP in allocating the bandwidth to the flows proportionally to their weights. Furthermore, the throughputs of the flows in 6-1-b and 6-1-c oscillate, and their transfers take longer, indicating that both MulTCP and CSFQ

[2] We could not find any clear recommendations for setting this parameter, so we set it to the value that seemed to achieve the best performance.




**(a) XCP**

**(b) MulTCP with RED**

**(c) CSFQ**

**(d) WFQ**

Figure 6-1: Comparison between XCP's relative bandwidth allocation and MulTCP [24], CSFQ [75], and Weighted Fair Queuing (WFQ) [25]. Three flows each transferring a 10 MB file over a shared 10 Mb/s bottleneck. Flow 1's weight is 5, Flow 2's weight is 10, and Flow 3's weight is 15. Throughput is averaged over 200 ms (5 RTTs).

do not use the link bandwidth as efficiently as XCP does. On the other hand, the weighted fairness achieved by WFQ is more accurate than XCP. This is expected given that WFQ maintains per-flow queues at routers and does more complex per-packet processing.[3] Due to WFQ's start-up transient, XCP finishes the transfers one second earlier than WFQ, indicating a better bandwidth utilization.

**Experiment 2:** Next, we want to explore the range of relative bandwidth differentiation available with XCP. For example, can XCP provide three orders of magnitude differentiation between the throughput of various flows? To explore this point, we run an experiment in which two flows share a single bottleneck. We set the weight of the first flow to $w_1 = 1$, and vary the weight of the

[3]In addition to having per-flow queues, WFQ needs to order the packets according to their departure time.

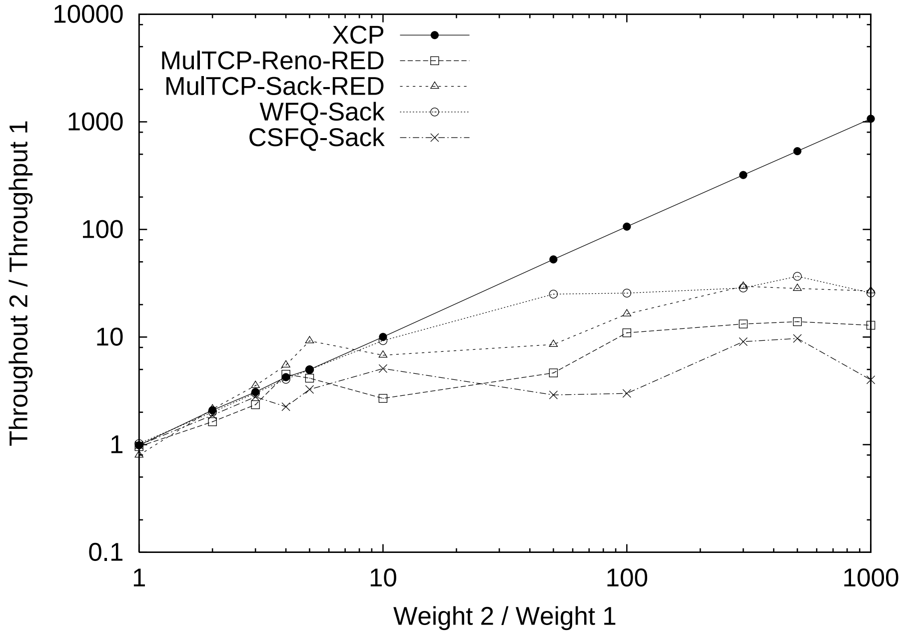



Figure 6-2: The range of relative bandwidth allocation provided with XCP, MulTCP, CSFQ, and WFQ. The figure shows the ratio of the throughputs of two flows sharing a bottleneck as a function of the ratio of their weights. Each point is the average of 10 runs.

second flow $w_2 \in [1, 1000]$. If XCP can support relative allocations in this range then the ratio of throughput$_2$ to throughput$_1$ should always follow $w_2/w_1$.

The simulations are run over the topology in Figure 5-1. The round trip propagation delay is 80 ms and the queue size is a delay bandwidth product. For this set of simulations, we use a small packet size of 100 bytes and we vary the capacity of the bottleneck with $w_2$ so that the fair share of flow 1 is always 100 Kb/s. We do so to prevent the rounding of the congestion window from biasing the results of the experiment. For example, assume $w_2 = 7$, $w_1 = 1$ and the pipe is 10 packets. In this case, to achieve the relative differentiation, flow 1's congestion window should be 1.25 packets, and flow 2's congestion window should be 8.75 packets. However, the congestion window has to be an integer number of packets. To prevent these rounding errors from skewing our results, we change the capacity of the bottleneck with $w_2$ so that the fair congestion window of flow 1 is always 10 packets. We use a small packet size so that the capacity can be kept below 100 Mb/s even when $w_2/w_1 = 1000$.

Figure 6-2 shows that XCP can provide accurate relative bandwidth allocation for a range that spans three orders of magnitude. In contrast, MulTCP fails in providing relative allocation when the ratio of the weights is larger than 10. This failing happens whether MulTCP simulates Sack or Reno TCP. The range of differentiation that may be provided with CSFQ is relatively small and the differentiation becomes inaccurate for weight ratio larger than 4. Finally, in our simulations, WFQ provides an accurate bandwidth differentiation for weight ratios below 25. For larger ratios, the throughput of Flow 2 becomes too large. The TCP sender of Flow 2 cannot ramp up quickly after a drop to re-acquire the bandwidth. Drops substantially affect the rate of Flow 2 when there are multiple drops from the same window, which tend to happen because FQ uses a drop-tail policy. The router drains packets from Flow 1 whenever there are no packets in Flow 2's queue, which inflates the rate of Flow 1.

At the end of this section, we note that the simulations addressed only the case in which the weighted fair share of all flows is larger than a few packets per RTT. For smaller weighted shares, some low-rate flows might enter the exponential back-off mode, which makes it considerably difficult for any of the above schemes to provide accurate differentiation.

## 6.4 Providing Bandwidth Guarantees in XCP

We now describe an approach that allows a flow to ask for a particular minimum bandwidth. Our approach falls within the DiffServ realm; it requires no per-flow state in the routers, provides per-flow bandwidth guarantees, and does not provide worst case per-flow delay guarantees. However, our approach differs from other services defined under DiffServ in that it supplies a distributed admission control scheme with no per-flow state.

We could have provided a stronger service that achieves worst case per-flow delay guarantees but we chose not to do so for the following reasons. First, worst case delays show up only in constructed and unrealistic environments [16, 17]. Common delays in our approach are very close to the round trip propagation delay (as shown in §6.4.6). Second, providing delay bounds increases the computational complexity of packet scheduling from $O(1)$ to $O(log\ n)$, where $n$ is the number of flows in IntServ and the number of queued packets in SCORE. The increase in complexity is fundamental and independent of the particular scheduling algorithm [79]. This higher complexity might burden high-speed routers and prevent them from forwarding packets at line speed.

### 6.4.1 Service Framework

A flow may belong to either one of two classes: a guaranteed bandwidth class, or a best-effort class. A flow in the guaranteed bandwidth class has a profile that indicates the minimum throughput the flow would accept (bytes/sec). When a flow with guarantees starts transmission, it uses its first packet to probe for admission into the network. The network can either admit the flow or deny it admission. Once the flow is admitted, the network allocates to it a rate that is at least equal to its profile. To maintain this guarantee the source should send every RTT at least one packet, declaring the profile in the congestion header. Without this refreshment message, the reservation will expire. The network also requires a guaranteed bandwidth flow to pace its packets.

Best-effort traffic, on the other hand, has no guarantees. It fills up the capacity unused by the guaranteed service flows. In this best-effort class, flows can still ask for relative bandwidth allocation (as described in §6.3). We note that the best-effort flows can use any reserved bandwidth not in use by the guaranteed bandwidth flows; however, they are forced to release the bandwidth when the guaranteed flows start sending again.

We assume the existence of policing agents at the edge of the network to ensure that the flows do not lie about their profiles and do abide by the feedback they receive. These agents can also take care of billing the users of the guaranteed service class. Furthermore, the agents can send packets on behalf of the user to refresh the reservations and can pace the traffic if the user fails to do so.

### 6.4.2 Modifying the Congestion Header

To provide guaranteed bandwidth, the routers need two pieces of information. First, they need to allow the flows with profiles to ask for admission. This requires one bit in the congestion header, which we call the AC bit (i.e., the admission control bit). Second, the routers needs to know the profile of each active flow. To do so, we add to the congestion header a new field called H_profile.[4]

### 6.4.3 Behavior of the Sender

The sender declares its profile in the H_profile field in the congestion header and sets the the admission control bit in its first packet. The receiver relays the value of the admission control bit to the sender in the acknowledgment. Thus, when the sender receives the Ack for its first packet, it knows whether the network can support the flow with the specified profile.[5]

If the flow is not admitted, the sender has two options. The sender can either give up and try later, or revert to best-effort. If the flow is admitted, the sender can send at a rate as large as its

[4] We note that since guaranteed flows receive no congestion feedback they need not use the the fields H_rtt and H_feedback. Thus, it is possible for these flows to use a smaller header that does not provide these fields.

[5] It is fairly easy to modify the scheme to allow the router to decrease the profile field in order to inform the sender of the maximum profile the router can guarantee.

profile without being controlled by the network. The sender can exceed the profile; however, it must set the H_profile field to zero in all out-of-profile packets. Further, the H_throughput field in the in-profile packets should be set to the throughput of the in-profile packets, whereas the H_throughput field in the out-of-profile packets should be set to the out-of-profile throughput. In other words, the sender can send more than its profile by simulating two flows: 1) an in-profile flow sending at the profile rate; 2) an out-of-profile flow that adjusts its sending rate to the feedback it receives. For simplicity, the rest of this chapter assumes that a guaranteed bandwidth flow never sends beyond its reserved bandwidth.[6]

### 6.4.4 Control Plane: Distributed Admission Control With No Per-Flow State

The first packet of each guaranteed bandwidth flow is used to ask for admission to the network. The sender sets the AC bit in the first packet. Each router along the path maintains an estimate of the total reserved bandwidth, $R_g$. The router admits the new flow if the sum of $R_g$ and the flow's profile does not exceed the capacity allocated to guaranteed traffic. Otherwise the router denies admission to the flow by resetting the AC bit in the packet. Once the AC bit is reset the flow is denied admission and downstream routers do not consider it for admission control. Eventually the packet reaches the receiver, which informs the sender whether the flow has been admitted or not.

Next we describe how the routers estimate the total reserved bandwidth $R_g$. The naive way to estimate the total reservations is to start from $R_g = 0$, and whenever a flow is accepted we update $R_g = R_g + profile$. This naive approach is problematic for two reasons. First, flows that are admitted at an upstream router might be denied admission at a downstream router. Thus, the upstream router will reserve the bandwidth though the flow might never use it. Second, flows usually do not free the resources after they leave the network, which prevents reallocating the resources to new flows.

Our objective is to compute an estimate of the total reserved bandwidth given that each flow tells us its profile in its packets. We compute the estimate by looking at the profile field of all packets in an interval $T_e$. If each flow sends exactly one packet in $T_e$, then we can compute $R_g$ by summing the values of the profile field in these packets. However, a flow sends on average $throughput \times T_e$ packets in $T_e$ seconds, where the throughput is in packets/sec. Thus, we can compute the total reserved bandwidth by dividing the profile by the number of packets from a flow, then taking the

[6] If the sender exceeds the profile packets may get out of order as a result of the different treatment of in-profile and out-of-profile packets. Our simulation results, below, show that the size of the out-of-profile queue is negligible, which means that the impact of reordering is limited.

sum, i.e., $R_g = \sum p_i \times \frac{s_i}{r_i \times T_e}$, where $p_i$ is the flow's profile, $T_e$ is the estimation interval, $s_i$ is the packet size in bytes, $r_i$ is the flow's throughput in bytes/sec.

However, this estimate ignores the most recent reservations. In particular, there might be recently admitted flows whose senders have not yet learned of the admission. To count these reservations we keep a second variable $R_r$, which is updated whenever a reservation is made $R_r = R_r + profile$, and is reset at the beginning of the estimation interval $T_e$. Then, $R_g = R_r + \sum p_i \times \frac{s_i}{r_i \times T_e}$.

Finally, for additional robustness, we measure the actual total throughput of the guaranteed traffic every $T_e$, which we call $R_u$, and use it as a lower bound on $R_g$, i.e.,

$$R_g = \max(R_r + \sum p_i \times \frac{s_i}{r_i \times T_e}, R_u). \tag{6.1}$$

Below is a pseudo code for both admission control and $R_g$ estimation. The first block of code runs when the router receives a request for reserving bandwidth with the AC bit set. The variable $G$ refers to the maximum bandwidth that may be allocated to guaranteed traffic. The value of this parameter is chosen by the network administrator. Choosing a small $G$ limits the total bandwidth allocated to guaranteed traffic and may cause the network to turn away guaranteed flows that could have been supported. On the other hand, a large value for $G$ increases the probability of over-committing the bandwidth. In particular, $R_g$ is an estimate of the currently committed bandwidth. The simulations in section §6.4.6 indicate that the error in this estimate is less than $0.2 \times C$, where $C$ is the capacity of the link. Thus, to prevent over-committing the bandwidth, we recommend setting $G$ to less than 80% of the capacity of the link.

The second block of code estimates the currently used reservations and the total committed bandwidth. Finally, the third block of code estimates $R_g$. To smooth the estimate, we use a running average that is biased toward larger values. The bias prevents underestimating the committed bandwidth, and thus prevents over-committing the resources.

On request arrival do:

    if ($R_g$ + H_profile $\leq$ G)

        reset the AC bit

    else

        $R_g$ += H_profile

        $R_r$ += H_profile

---

On packet arrival:

    if (H_profile $>$ 0)

        $R_u$ += packet_size / $T_e$

        sum_profiles += H_profile $\times$ packet_size / (H_throughput $\times$ $T_e$)

---

On $T_e$ timeout:

    tmp = max(sum_profiles, $R_u$) + $R_r$

    if (tmp $\geq$ $R_g$)

        $R_g$ = tmp

    else

        $R_g$ = 0.4 $\times$ tmp + 0.6 $\times$ $R_g$

    sum_profiles = 0

---

### 6.4.5 Data Plane

The router keeps a separate queue for the guaranteed traffic, which we call the G-queue. Packets in the G-queue are given priority over best-effort traffic and are transmitted before any queued best-effort packets. The average size of the G-queue stays small because admission control ensures that the arrival rate to the G-queue is smaller than the departure rate from the G-queue, (the latter is the link's capacity). Also, the variations in the G-queue size stay small because the guaranteed service traffic is paced and has very little burstiness. (The delay variations are not bounded as in IntServ but our simulations indicate that they are negligible.)

We allocate a separate queue to the guaranteed service flows to minimize the queuing delay and jitters they might incur. In particular, when a new reservation is made, some bandwidth should be moved from the best-effort traffic to the guaranteed traffic. The new guaranteed service flow starts sending at a high rate causing a queue buildup at the router and potential packet drops. It takes some time for the best-effort traffic to back-off and drain the queue. By putting the guaranteed traffic in a separate queue, we isolate it from the resulting jitter and delay.

### 6.4.6 Evaluation

We use simulation to show that:

1. $R_g$ tracks the reservations,
2. throughput of a guaranteed flow is equal to its profile,
3. the network operates efficiently, and
4. the best-effort traffic experiences a good fairness and a high efficiency.

We start by describing the simulation environment. We implement a guaranteed flow as a window-based flow. This means that the flow sets cwnd always to the reservations times the RTT, which it estimates. We use this approach because it minimizes the modifications needed to extend the XCP sender to provide guarantees.

Simulations in this section use the topologies in Figure 5-1 and Figure 5-2. We start our evaluation by looking at simple scenarios with one bottleneck, then we examine more complex scenarios with multiple bottlenecks. The maximum reserved bandwidth $G$ is set to 80% of the link capacity.

#### A. $R_g$ Tracks the Reservations

First, we show that our estimate of the total committed bandwidth tracks the reservations. This tracking is important to ensure that guaranteed service flows are not admitted when they cannot be satisfied and are not turned away when they can be supported.

We first start with an experiment in which a single bottleneck of 10 Mb/s is shared by best-effort traffic and guaranteed service flows (Figure 5-1). The round trip delay is 80 ms. There are 10 best-effort flows that start at time $t = 0$. Guaranteed flows start arriving at $t = 5$ separated by 5 seconds. Each guaranteed service flow asks for $15\%$ of the capacity of the bottleneck. Since G is set to 80% of the link capacity, at any time there could be no more than 5 guaranteed flows in the network and a maximum of 7.5 Mb/s of guaranteed traffic. At time $t = 50$, all of the currently active guaranteed

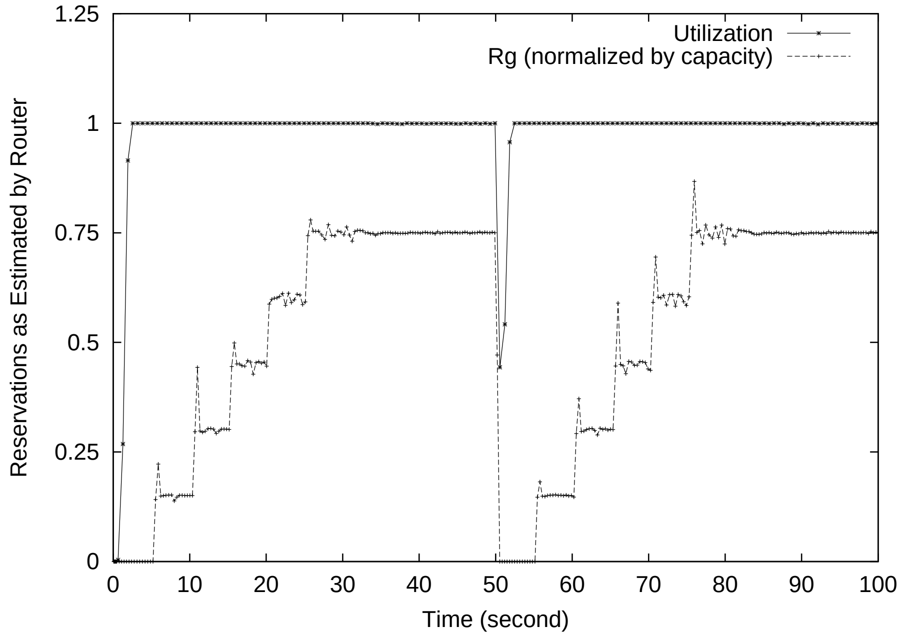


Figure 6-3: $R_g$ tracks the reservation made by the guaranteed bandwidth flows. The guaranteed bandwidth flows arrive separated by 5 sec. Each flow requests 15% of the bottleneck bandwidth. The rest of the bandwidth is filled up by best-effort flows. Since G = 0.8 capacity, we can accommodate at most 5 guaranteed bandwidth flows. At $t = 50$, active guaranteed flows are stopped, which frees up the reserved bandwidth. The cycle repeats with new arrivals of guaranteed flows.

flows are stopped, to let the sum of reservations, $R_g$, go to zero again. The release of reservations allows new guaranteed flows to be accepted into the system.

Figure 6-3 shows that the router's estimate of the reservations tracks the correct reservations with good accuracy. In particular, $R_g$ increases by 15% every 5 seconds with an arrival of a new guaranteed flow. The router accepts a maximum of 5 guaranteed flows. After that no more flows are accepted since the difference between the current reservations and $G$ does not satisfy the needs of any new guaranteed flow. At time $t = 50s$, all currently active guaranteed flows are terminated. Although none of the terminated flows sends a message to release the reserved bandwidth, the router quickly discovers the release of the reserved bandwidth and starts reallocating it to new requests. The figure also shows the utilization of the bottleneck link. It reveals that the bandwidth unused by the guaranteed flows is efficiently used by best-effort traffic.

Next, we examine the impact of on-off flows on the estimate of $R_g$. As mentioned above, an on-off flow can maintain its reservations by refreshing the reservations every RTT. The flow sends a small packet that contains only the congestion header. The H_profile field is set to the flow's reserved bandwidth, The H_rtt field is set to the flow's recent estimate of the RTT, and the H_throughput field

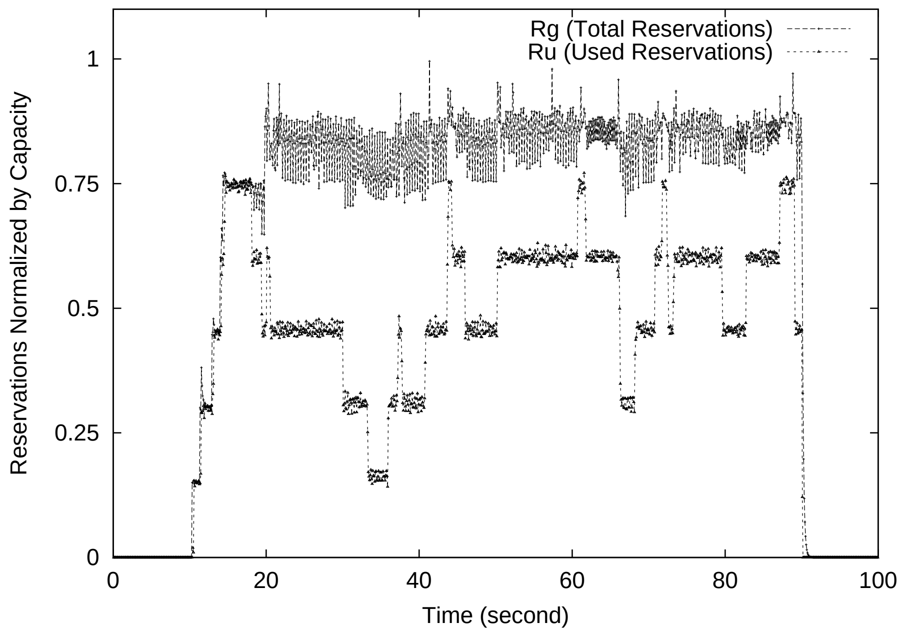
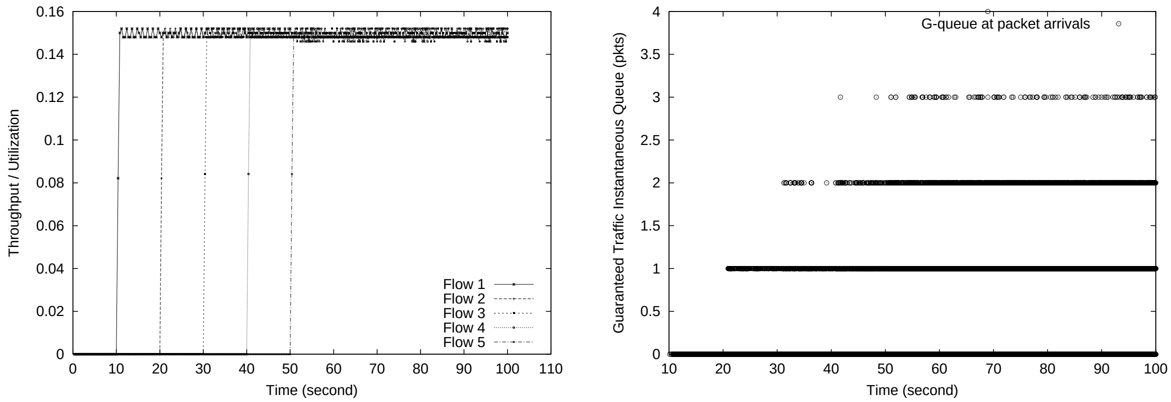



Figure 6-4: Admission control with on-off flows. The guaranteed service flows all arrive around $t = 10s$ and stop at $t = 90s$. The router accepts them as long as $R_g < G$. The figure shows that although the on-off flows do not use all of their reserved bandwidth ($R_u < R_g$), the router maintains the reservations.

is set to one packet per RTT. We run an experiment in which 5 guaranteed flows start together at $t = 10s$ and stop at $t = 90s$. The guaranteed traffic shares the link with best-effort flows that last for the whole simulation. Each guaranteed service flow asks for 15% of the link bandwidth. The guaranteed service flows are on-off flows with an idle period whose length is distributed according to a Pareto distribution with an average of 10 seconds. In this experiment, the maximum committed bandwidth, $G$, is set to 80% of the capacity of the link.

Figure 6-4 shows that our estimate of the total reservations tracks the committed bandwidth despite the fact that the on-off flows did not use all of their reserved bandwidth. In fact $R_g$ slightly overestimates the reservations. This overestimation is intentional and caused by the bias of our filter toward larger values of $R_g$. As mentioned above, we bias the $R_g$ estimate to prevent over-committing the bandwidth.

Finally, both Figures 6-3 and 6-4 indicate that our estimate of $R_g$ is within 10% of the actually committed bandwidth. To be conservative, we recommend setting $G < 0.8 \times C$, where $C$ is the link capacity. Although unlikely, it is still possible that the router might over-commit the bandwidth because of its underestimation of $R_g$ or because of a route change. Our protocol does not specify the reaction taken by the network in this case, which could vary depending on the policy of the




**(a) Throughput of Guaranteed Flows** **(b) Guaranteed traffic queue**

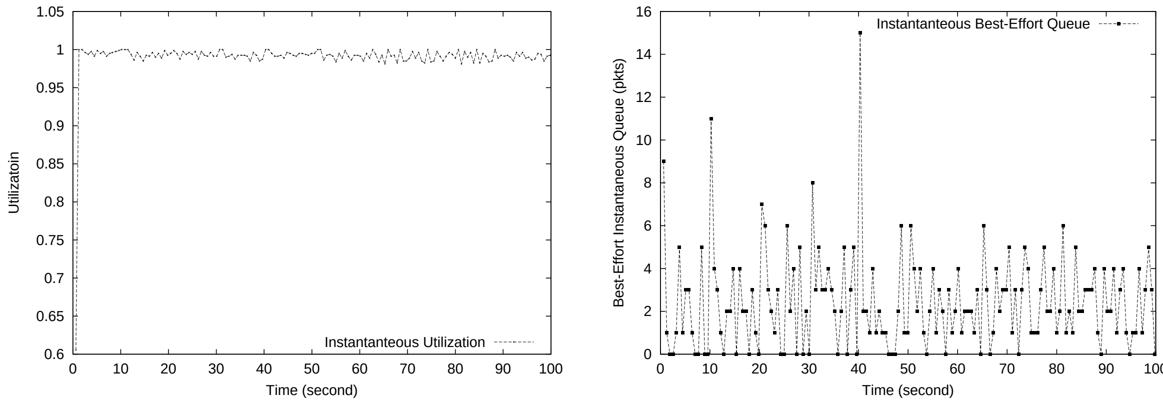
Figure 6-5: The service perceived by the guaranteed bandwidth flows. 6-5-a shows that the flows obtain their reserved throughput. 6-5-b shows that the delay perceived by guaranteed service flows is almost always the RTT.

administrator. Some administrators might choose to slow down all guaranteed service flows proportionally to their profiles. Other administrators may adapt an approach similar to that adapted by telephone networks and decide to deny service to some of the admitted flows.

**B. The Quality of the Guaranteed Service**

In this section, we examine the delay and throughput obtained by the guaranteed service traffic. We remind the reader that our scheme does not use any scheduling mechanism. To provide bandwidth and delay guarantees, it relies on the facts that the bandwidth of the guaranteed traffic is always below the capacity of the link and that the guaranteed service flows pace their packets. Thus, it is important to check that the scheme can provide the guaranteed service without scheduling.

In this experiment, guaranteed service flows start arriving at $t = 10s$ separated by 10 seconds. Each of them asks for 15% of the bandwidth of the link. In addition to the flows above, there are 10 best-effort flows that last for the duration of the simulation. Figure 6-5 shows the service perceived by the flows that obtained reservations. In particular, 6-5-a shows that the router made reservations for 5 flows. All of these flows obtained the throughput they were promised by the network. Furthermore, 6-5-b shows that the guaranteed service queue has been almost always empty, which means that the guaranteed service flows almost never faced any queuing delay. Although our scheme does not provide worst case delay guarantees, the simulation results show that the queuing delay experienced by guaranteed flows is negligible. In §6.4.6, we show that this behavior scales to larger




**(a) Bottleneck Utilization** **(b) Best-Effort Queue**

Figure 6-6: The service perceived by best-effort traffic. A 10 Mb/s bottleneck is shared by 10 best-effort flows and 5 guaranteed service flows. The guaranteed service flows are consuming 75% of the bandwidth. The rest is used by the best-effort traffic.

topologies and more complex scenarios.

### C. The Service Perceived by the Best-Effort Flows

In this section, we show that the best-effort flows fill up the bandwidth unused by the guaranteed service flows. Further, the existence of the guaranteed service flows does not cause unacceptable delay or drop rate to the best-effort traffic. In particular, Figure 6-6 shows the service perceived by the best-effort traffic in the experiment described in the previous section (i.e., §6.4.6-B). Figure 6-6-a shows that the bottleneck utilization is optimal, while 6-6-b shows that the best-effort queue size is just a few packets. Finally we note that there were no drops in this experiment.

### D. The Guaranteed Bandwidth Service in Complex Scenarios

Next we show that the good performance of the guaranteed bandwidth service scales to larger topologies and more complex scenarios. We simulate the topology in Figure 6-7, which has 50 nodes and 153 simplex links. Link capacities are either 10 Mb/s or 100 Mb/s, propagation delays vary between 2 ms and 45 ms. The simulation contains 500 best-effort flows and 100 guaranteed bandwidth flows. Each guaranteed bandwidth flow asks for 1.5 Mb/s. The source and destination of a flow are chosen randomly.

Figure 6-8 shows the *maximum* G-queue size for each link. This is the worst case queuing delay experienced by guaranteed bandwidth flows at that link. The figure shows that although our

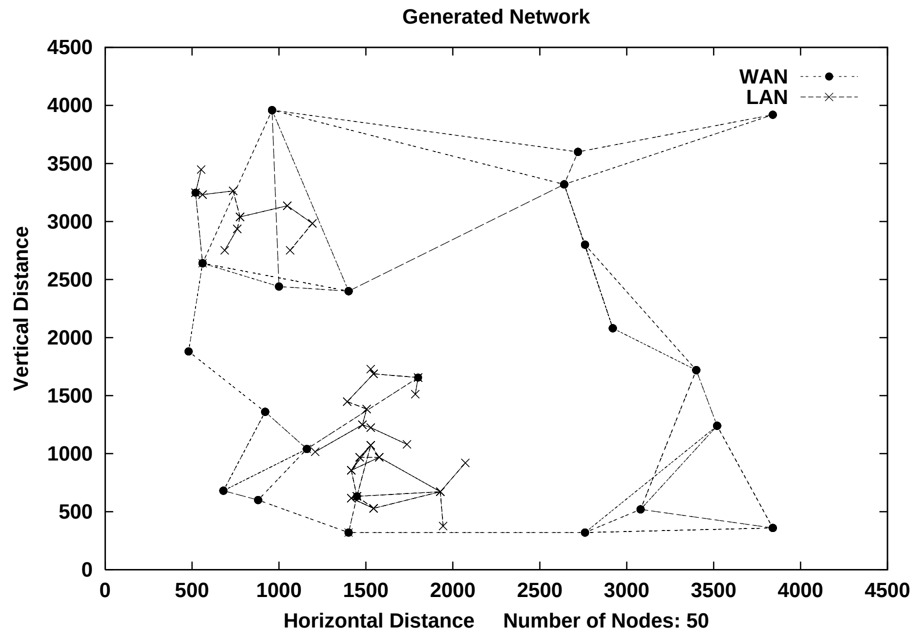



Figure 6-7: A large simulation topology with 153 links and 50 nodes.

service does not guarantee a small bound of queuing delay, in practice, the maximum queuing delay experienced by guaranteed bandwidth flows is insignificant.

Table 6.2 shows the throughput of the guaranteed bandwidth flows normalized by their profiles. Flows that did not obtain admission to the network did not send any traffic, and therefore do not show up in the table. As can be seen from the table, all flows obtained a throughput very close to their profile. The third column of the table shows the variance in the throughput. The low variance shows that the throughput was constant and smooth during the whole simulation time.

## 6.5 Concluding Remarks

In this chapter, we showed that the XCP framework is flexible enough to provide both relative bandwidth differentiation and absolute bandwidth guarantees.

The XCP framework can provide an infinite spectrum of relative bandwidth differentiation. This is done by a minor modification to the meaning of the fields in the congestion header and without adding any complexity to the routers. Our simulations show that XCP's relative bandwidth allocation is more accurate over short time scales than MulTCP and CSFQ and almost as accurate as WFQ. Further, XCP achieves a relative differentiation that spans more than three orders of magnitude, which we is difficult to obtain in other schemes.

| Flow ID | Avg. Throughput / Profile | Standard Deviation |
|---|---|---|
| 0 | 1.01162 | 0.000414976 |
| 1 | 1.02847 | 0.000629459 |
| 3 | 0.951795 | 0.00111228 |
| 8 | 1.00515 | 0.000763478 |
| 13 | 1.00099 | 0.00067902 |
| 18 | 1.01706 | 0.000618956 |
| 20 | 0.99539 | 0.000668748 |
| 21 | 1.05205 | 6.07975e-05 |
| 22 | 0.949202 | 0.000134563 |
| 25 | 0.972687 | 0.000337 |
| 29 | 1.00106 | 0.000498876 |
| 30 | 0.985789 | 0.000989847 |
| 38 | 1.00514 | 0.00059358 |
| 45 | 0.979103 | 0.000442315 |
| 50 | 1.00438 | 0.000390816 |
| 54 | 1.0063 | 0.00049031 |
| 55 | 0.984493 | 0.000326533 |
| 57 | 0.981836 | 0.000574069 |
| 59 | 1.00144 | 0.000647926 |
| 63 | 0.98392 | 0.000542434 |
| 64 | 0.990253 | 0.000619996 |
| 72 | 0.90828 | 0.000725442 |
| 80 | 0.991811 | 0.00063294 |
| 81 | 0.995918 | 0.000648337 |
| 84 | 0.908417 | 0.000711554 |
| 94 | 1.25362 | 0.0049869 |
| 95 | 1.00643 | 0.000728655 |

Table 6.2: The throughputs of the guaranteed bandwidth flows that got admission into the network in Figure 6-7.

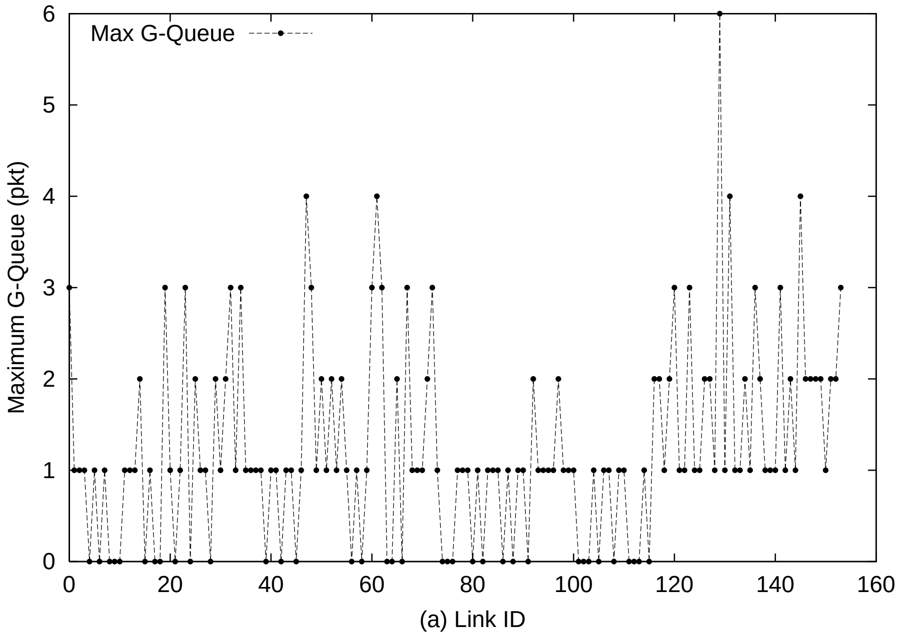



Figure 6-8: Guaranteed bandwidth flows experience negligible worst case queuing delays. The figure shows the maximum G-queue size at each of the links in Figure 6-7, which is the worst case queuing delay experienced by guaranteed bandwidth flows at that link.

The XCP guaranteed bandwidth service allows a sender to reserve bandwidth for the duration of the flow. Admission control is done in a distributed manner without any per-flow state at the router. The resulting scheme is work conserving, does not increase the complexity of the routers, and delivers the promised service. Further, the best-effort traffic enjoys a good service with a small queue size and almost no drops.

We have focused on mechanisms that provide relative or absolute bandwidth differentiation because we believe that providing delay or loss differentiation is less important in an XCP network than in a TCP network. We have shown in the previous chapter that XCP maintains small queues and almost never drops packets. Further, we have shown in this chapter that guaranteed flows usually experience no queuing delay and no drops. Thus, we believe that the vast majority of applications will be satisfied with the delay and loss rate achieved by XCP.[7]

[7] Still, mechanisms that provide delay bounds may be added if needed.

# Chapter 7

# Non-Compliant Flows

The Internet is a heterogeneous environment run by multiple administrative authorities and shared by users with different interests. It is natural for such an environment to contain misconfigured machines or malicious users who might not adhere to the rules of a congestion control protocol. Although current experience with the Internet shows that the number of such non-compliant entities is small [77, 55], it is important for any new congestion control protocol to explore the danger imposed by misbehaving.

In this chapter, we address the problem of misbehaving users. Misbehaving is an old problem, which is usually addressed by monitoring the flows (or a subset of the flows) inside the network [63, 76, 30]. Monitoring is a general solution that allows the administrator to identify and isolate misbehaving flows independently of the congestion control protocol. We start this chapter by showing that XCP facilitates monitoring and detection of misbehaving flows. Then, we explore various ways a user can misbehave and the impact of such behavior on the network in the absence of flow monitoring and protection.

## 7.1 Flow Monitoring

Similarly to TCP, in XCP security against misbehaving sources requires an additional mechanism that polices the flows and ensures that they obey the congestion control protocol. This monitoring may be done by policing agents located at the edges of the network or by statistical sampling at the routers.

1. **Policing Agents at the Edges of the Network:** The agents maintain per-flow state and monitor the rate of the flows to detect and isolate unresponsive sources. Because these agents are at the edges of the network they see fewer flows and can maintain per-flow state without endangering scalability.

   Unlike TCP, XCP facilitates the job of these policing agents because of its explicit feedback. Isolating the misbehaving source becomes faster and easier because the agent can use the explicit feedback to test a source. More precisely, in TCP isolating an unresponsive source requires the agent/router to monitor the average rate of a suspect source over a fairly long interval to decide whether the source is reacting according to AIMD. Also, since the source's RTT is unknown, the source's appropriate sending rate is unspecified, which complicates the task even further. In contrast, in XCP, isolating a suspect flow is easier. The router can send the flow a test feedback requiring it to decrease its congestion window to a particular value. If the flow does not react in a single RTT, then it is unresponsive. That the flow specifies its RTT in its packets makes the monitoring easier because the policing agent needs to monitor the flow only over a few RTTs. Further, since the flow cannot tell when an agent/router is monitoring its behavior, it always has to follow the explicit feedback.

2. **Statistical Sampling in the Network:** Statistical sampling is similar to edge policing except that flow monitoring is done independently at each router. To reduce the required per-flow state, a router samples the traffic to catch the high-rate flows. These are the ones worth monitoring, whereas small misbehaviors are not worth the effort of the router. Both [30] and [63] provide mechanisms for statistical sampling of traffic, using a manageable amount of state at routers. These mechanisms can capture the fast flows. Once the high-rate flows are detected they can be tested using the explicit feedback as described in the above paragraph.

## 7.2 Networks with No Flow Protection

Flow monitoring is relatively expensive because it requires the monitoring entities to maintain per-flow state. Current measurements of Internet traffic show that most flows do abide by the congestion control protocol, though no monitoring is in place. [77, 55]. Indeed, misbehaving requires either modifying the congestion control protocol in the kernel or running applications on top of UDP. Since most large bandwidth commercial applications use some form of congestion control, misbehaving is currently limited to a small subset of users who are willing to invest time and effort in cheating the network, and who do not have a moral prohibition against such misconduct.

Next, we examine the impact of misbehaving on XCP networks when no flow protection is available. Although flow monitoring solves all of the problems listed below, we are interested in examining the danger of misbehaving flows when the network does not provide any monitoring or flow protection.

## 7.3 Senders' Misconduct

What are the different ways an XCP source can misbehave? How do they affect the performance? And for each particular misconduct, does the source have an incentive to do it? These are the questions that we answer in this section.

The sender's misconduct could be intentional, aiming at obtaining more bandwidth or disrupting the service of the other users, or it could be unintentional, caused by bugs in its code. Independent of the reasons of misconduct, we can classify misbehaving XCP sources as liars and abusers. Liars lie about the information in the congestion header, reporting an incorrect throughput or round trip delay. We refer to flows which lie about their throughput as T-liars, and flows which lie about their round trip delay as D-liars. We call a lie that reports a value larger than the true one an upward lie, whereas we call a lie that reports a smaller value a downward lie. In contrast to liars, abusers report their correct throughput and delay, but they do not appropriately react to the feedback they receive. Some abusers may completely ignore the feedback they receive, in which case we call them unresponsive abusers. Less abusive are responsive abusers, which decrease in time of congestion and increase when there is spare bandwidth, yet they increase more and decrease less than they are told. We note that in practice a misbehaving user might be both a liar and an abuser. Our classification separates these issues to help disentangling the effects of various attacks. Table 7.1 shows some ways an XCP sender can misbehave, the impact of the attack on efficiency and fairness, and whether a strategic sender has an incentive to perform such an attack. The incentive is evaluated assuming the sender is selfish and interested in maximizing its own rate but is not particularly interested in harming the network. (This complies with the definition of a "strategic user" in Game Theory, but we do not claim that the "incentive" reflects the behavior of the strategic sender at the equilibrium of the game.) The rest of this section explains the results in Table 7.1.

| Misconduct | Direction | Impact on Efficiency | Impact on Fairness | Incentive |
|---|---|---|---|---|
| T-liar | Upward | No Impact | Makes sender looks like a fraction of a flow $\rightarrow$ gets less than fair share | No |
| | Downward | No Impact | sender simulates multiple flows $\rightarrow$ gets more than fair share | Yes |
| D-liar | Upward | Slows down convergence | Negligible impact (slows down convergence to fairness) | No |
| | Downward | Makes system faster; can cause instability if the true delay is much larger than the declared one | No impact | No |
| Abuser | IG $> 1$ | System is fairly robust against this attack but in the extreme it might get destabilized | Limited unfairness | Yes |
| | DG $< 1$ | System is fairly robust against this attack but in the extreme it might get destabilized | No significant impact | No |
| Unresponsive Abuser | $r_i \leq C$ | Almost no impact | Unfair | Yes |
| | $r_i > C$ | Destabilizes the system causing it to degenerate to the underlying dropping policy (i.e., RED) | Unfair | Yes (as long as the abuser does not mind the drops). |

Table 7.1: Various sender's misconducts, their impact on efficiency and fairness, and whether a strategic sender has an incentive to perform such a misconduct. IG and DG are the increase and decrease gains respectively, $r_i$ is the rate of unresponsive traffic, and $C$ is the capacity. To evaluate the incentive of the user, we assume that the sender is a strategic agent who misbehaves to increase its throughput.

### 7.3.1 The T-liar

We distinguish between two cases:

1. **Upward T-Liar:** In this case the sender declares a value larger than its throughput in the H_throughput field in the congestion header. This lie affects only fairness but does not affect the congestion state of the network, because the efficiency controller at the routers does not use the H_throughput field. It computes the total traffsic rate by measuring the incoming

traffic at the link. Thus, lying about the throughput has no impact on the congestion or the efficiency in the network. On the other hand, reporting a throughput larger than the true one causes the throughput to converge to a fraction of the fair share. In particular, although in XCP each flow increases by a constant amount, the increase is spread over all packets. Thus, by reporting more packets per second, the per-packet positive feedback given to this flow decreases. In contrast, the negative feedback increases since the fairness controller in times of congestion cuts down the rate of a sender proportionally to its throughput. Hence, given that the flow gets less than its fair share of the bandwidth, a strategic sender has no incentive to carry on such misconduct.

2. **Downward T-Liar:** In this case the sender declares a value smaller than its throughput in the H_throughput field in the congestion header. As a result, the sender would be simulating multiple flows, where the number of simulated flows is the ratio of the true throughput to the declared one. For example, if the sender advertises a throughput that is half its true throughput, then the router would consider the flow two separate flows. This happens because the router has no per-flow information and the only notion of a flow it has comes from the advertised throughput. Similarly to the upward T-liar, this lie does not affect the efficiency or the congestion state of the network because the efficiency controller relies on direct measurements of traffic and does not use the H_throughput field. As for fairness, by simulating multiple flows, the flow acquires more than its fair share. This increased throughput creates an incentive for the sender to misbehave.

   In general, simulating multiple flows is an attack to which no congestion control protocol is immune (e.g., XCP, TCP, etc.). Further, it is impossible to detect such attack without maintaining a per-flow state in the network.

#### Simulation Results

Below we show simulation results that support our argument above, namely that T-lying has impact only on fairness but not on utilization.

**Impact on Fairness:** In this simulation, a single T-liar shares a 20 Mb/s bottleneck with 19 compliant flows. The topology is that in Figure 5-1, the round trip propagation delay is 80 ms, and there are 20 flows along the reverse path. Other parameters comply with the description in §5.1. We call the ratio of the true throughput to declared throughput the gain of the T-liar. Figure 7-1 illustrates

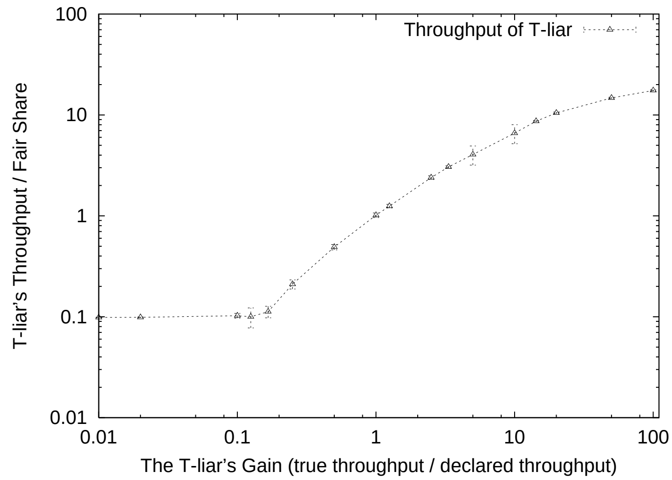



Figure 7-1: The impact of T-lying on fairness. The throughput of a single T-liar that is sharing a bottleneck with 19 compliant flows.

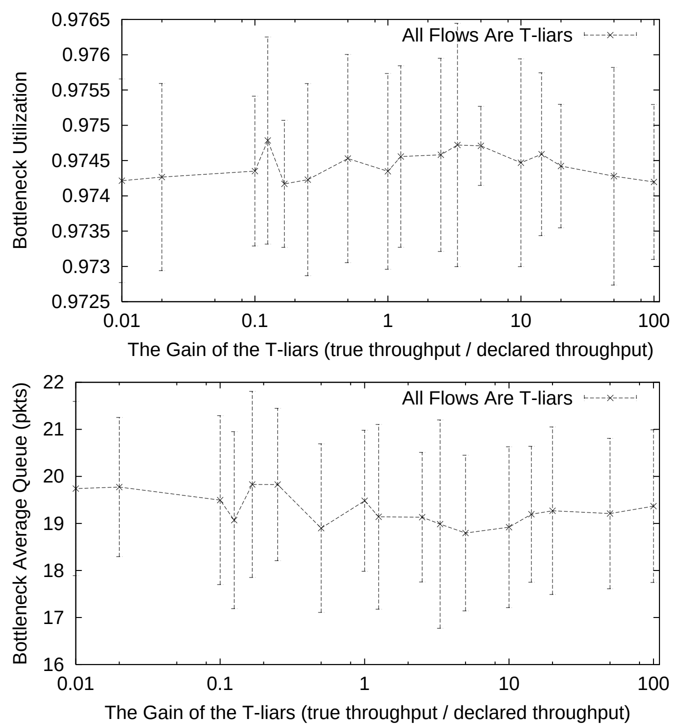



Figure 7-2: The impact of T-lying on efficiency. The utilization and queue size of a bottleneck traversed by 20 T-liars.

the ratio of the throughput of the T-Liar to its fair share as a function of its gain. The figure shows that for a wide range of gain (i.e., gain in $[0.2, 8]$), the T-liar achieves a throughput equal to its fair share multiplied by its gain. This is expected as the T-liar simulates a number of flows equal to its gain. As the gain increases beyond this range, the throughput of the T-liar starts leveling off. This is attributed to the fact that other flows continue to have at least a window of one packet. Similarly, when gain $< 0.1$, the T-liar reaches its minimum throughput which corresponds to one packet per RTT. Note that the exact gain values at which the throughput of the T-liar stabilizes in Figure 7-1 depend on the pipe size, but the trend is general.

**Impact on Efficiency:** Next, we look at the impact of T-lying on the efficiency of the network and its congestion state. In this simulation, all flows are T-Lying. The gain they are using is distributed according to a normal distribution with a standard deviation equal to the average (negative values are ignored). The average gain varies from one experiment to another and takes values in $[0.01, 100]$. We plot the average utilization, and the average queue size as functions of the T-liar gain. As seen in Figure 7-2, T-lying has no impact on the efficiency of the network. We also note that there were no drops in these sets of simulations.

### 7.3.2 The D-liar

We distinguish between the following two cases.

1. **Upward D-Liar:** By advertising a delay larger than its RTT, the flow increases the average delay computed by the router, and consequently increases the control interval. Both the EC and FC allocate bandwidth every control interval. Thus, an increase in this interval makes the controllers more sluggish and slows down convergence to fairness and optimal utilization. However, increasing the control interval does not prevent such convergence. A strategic sender has no incentive to intentionally advertise a larger delay.

2. **Downward D-Liar:** Advertising an RTT smaller than the true one has the opposite effect to the attack above. Namely, the router's estimated average delay becomes smaller, decreasing the control interval. As a result, the EC becomes more aggressive and attempts to change the traffic rate too quickly. If the control interval becomes too short in comparison with the RTT of the flows, the EC might get out of phase with the sources, which causes oscillations and decreases the utilization. As for fairness, advertising a smaller delay has no impact on the average bandwidth allocated to the flow, though it may increase the short term variance. Thus, a strategic sender has no incentive to intentionally advertise a smaller delay, but a vandal may use this attack to disturb the network performance.

To make XCP more robust against D-lying, the network operator can specify upper and lower bounds on the value of the control interval. These are bounds on the control interval not the RTT,

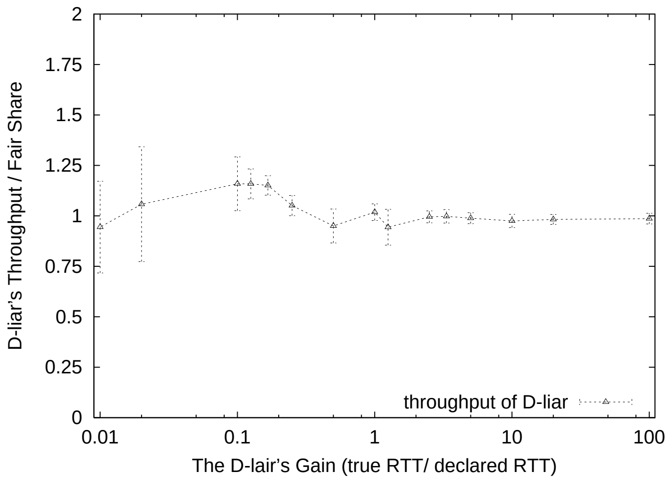



Figure 7-3: The impact of D-lying on fairness.The throughput of a single D-liar that is sharing a bottleneck with 19 compliant flows.

which could be outside the range specified by the bounds. Our *ns* implementation does not allow the control interval to become smaller than 5 ms (which is about the RTT on a LAN). Also, it does not allow the control interval to increase beyond 1 sec (which is about the maximum RTT in today's Internet). Our bounds are loose and have been chosen to allow both long and short-RTT simulations. In practice, the operator can choose tighter bounds. The propagation delay on the link is a natural bound on the minimum control interval. Further, the geographic location of the link usually holds information about the expected RTT distribution [42]. For example, transatlantic links may be configured with a minimum control interval of 30 ms, whereas access links may be configured with a maximum control interval of 200 ms.

### Simulation Results

We examine via simulation the impact of D-Lying on both efficiency and fairness. Our results indicate that D-lying has almost no impact on fairness. Further, as long as the number of D-liars is limited, D-lying does not cause any major inefficiency. However, the utilization considerably degrades when most flows declare an RTT much smaller than the true one.

**Impact on Fairness:** In this simulation, a single D-liar shares a 20 Mb/s bottleneck with 19 compliant flows. There are 20 flows along the reverse path. Other parameters comply with the description in §5.1. We call the ratio of the true RTT to declared RTT the gain of the D-liar. Figure 7-3 illustrates the ratio of the throughput of the D-Liar to its fair share as a function of its gain. The figure shows that the D-liar's throughput is the same as its fair share as long as its gain $\in [0.2, 100]$. For

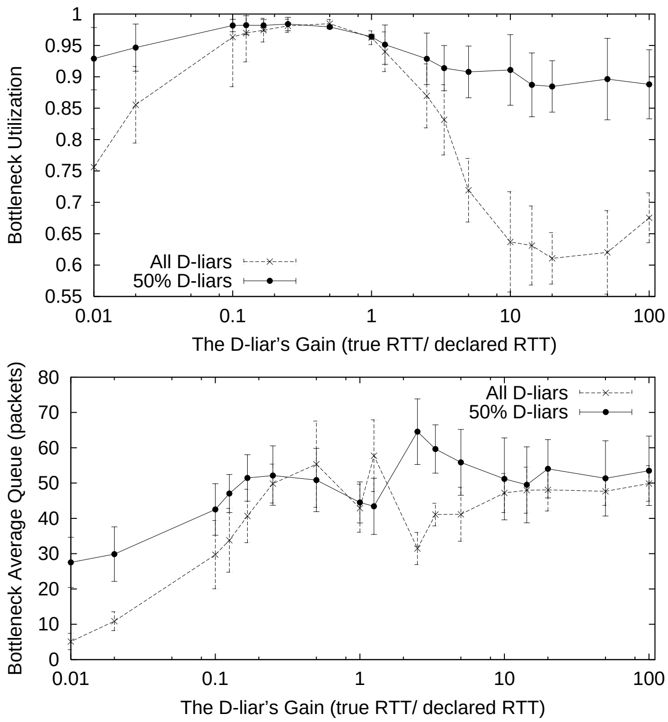



Figure 7-4: The impact of D-lying on efficiency. The utilization and queue size of a bottleneck shared by 20 flows. The graphs are for the cases when all flows are D-liars, 75% of the flows are D-liars, and 50% of the flows are D-liars.

gains $< 0.2$ (very large declared RTT), the variation in the throughput increases but the average remains close to the fair share (i.e., within 20%). We note that the light decrease in throughput for $gain < 0.1$ is caused by our maximum control interval of 1 second. When a flow declares an RTT larger than 1 sec, bandwidth allocation to this flow becomes slower than the other flows. As a result, the flow takes more time to ramp up, which appears as a slight decrease in the average throughput.

**Impact on Efficiency:** Next we simulate a bottleneck of 20 Mb/s shared by 20 flows where: 1)all flows are lying about their RTTs; 2)half the flows are lying about their RTTs. We call the ratio of the true RTT to the declared one the gain of the D-liar. For this experiment, the gain of the various flows is chosen from a normal distribution with a standard deviation equal to the average (negative values for the gain are ignored). The average of the distribution varies from one run to another and takes values in $[0.01, 100]$. Figure 7-4 shows the average utilization and the average queue size as functions of the gain for both when all flows are D-liars and when only 50% of the flows are D-liars.

Based on these figures, declaring an RTT larger than the true one decreases the efficiency. However, close inspection of the behavior of the flows reveals that the decrease in efficiency is caused by a long ramping up time at the beginning of the simulation. The link still converges to optimal utilization. Thus, the inefficiency is transient. On the other hand, drastically underestimating the RTT or declaring a large D-liar gain by most users causes a substantial degradation in utilization. This degradation happens because of oscillations in traffic rate, and thus is different in nature from the transit inefficiency at low gains. However, the figures also indicate that when most of the flows do not lie about their RTT, or when the lie is not substantial, the system maintains an acceptable level of efficiency. We note that it is possible to increase the robustness against a downward D-lie by using a smaller $\alpha$ or by increasing the minimum control interval. In both cases the operator would be accepting a more sluggish behavior in order to increase stability.

### 7.3.3 Responsive Abusers

In this case the sender reports its state correctly to the routers; but does not adhere to the feedback it receives. Whenever the feedback is positive, the abuser increases its rate by $IG \times feedback$, where $IG > 1$ is the increase gain. When the feedback is negative, the abuser decreases its rate by $DG \times feedback$, where $DG < 1$ is the decrease gain. By doing so, the abuser is trying to achieve higher throughput by using more aggressive parameters than the competing flows. The impact of the attack on the stability and efficiency of the system depends on the total amount of abusive traffic. We use the term 'abuse gain' to refer to the ratio of the total abusive traffic to all traffic at the bottleneck. The larger the gain is the more the system oscillates. Our simulations, described below, indicate that despite potential oscillations the system's performance remains acceptable for a fairly wide range of abuse gains.[1] As for fairness, using more aggressive parameters increases the abuser's throughput by a limited amount. Thus, there is some incentive for a selfish abuser to conduct such an attack.

#### Simulation Results

A responsive abuser uses an aggressive increase policy or a lenient decrease policy. In particular, when the legitimate behavior is to increase the rate by $\Delta r$, the abuser increases by $IG * \Delta r$, where $IG > 1$. When the legitimate behavior is to decrease the congestion window by $\Delta r$, the abuser decreases by $DG * \Delta r$, where $DG < 1$. We examine the impact of such malicious behavior on both XCP and TCP with RED queues. (A responsive TCP abuser increases its cwnd by $1 \times IG$ packets every RTT, and decreases it by $0.5 \times DG$ for each drop.) The simulations in this section use the topology in Figure 5-1. The bottleneck capacity is 20 Mb/s, the round trip propagation delay is 80

[1] In theory, if the abuse gain exceeds the gain margin, the abusive traffic might destabilize the router and cause oscillations. The gain margin decreases with an increase in $\alpha$. For $\alpha = 0.4$, the gain margin is around $2.2$.

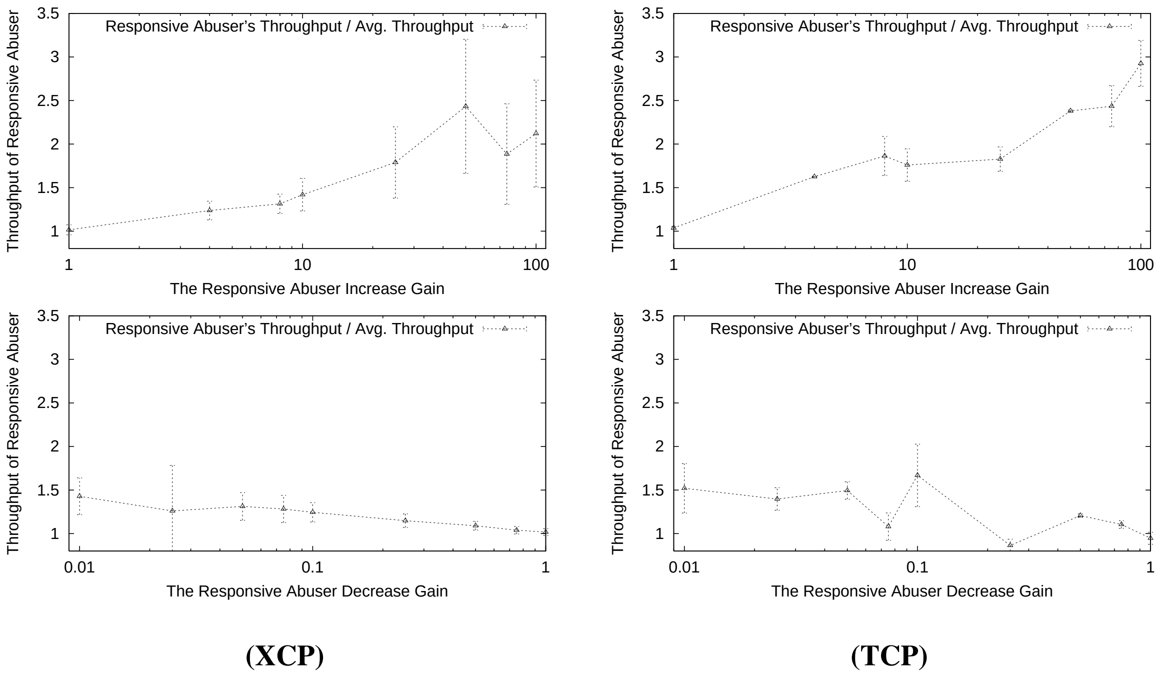
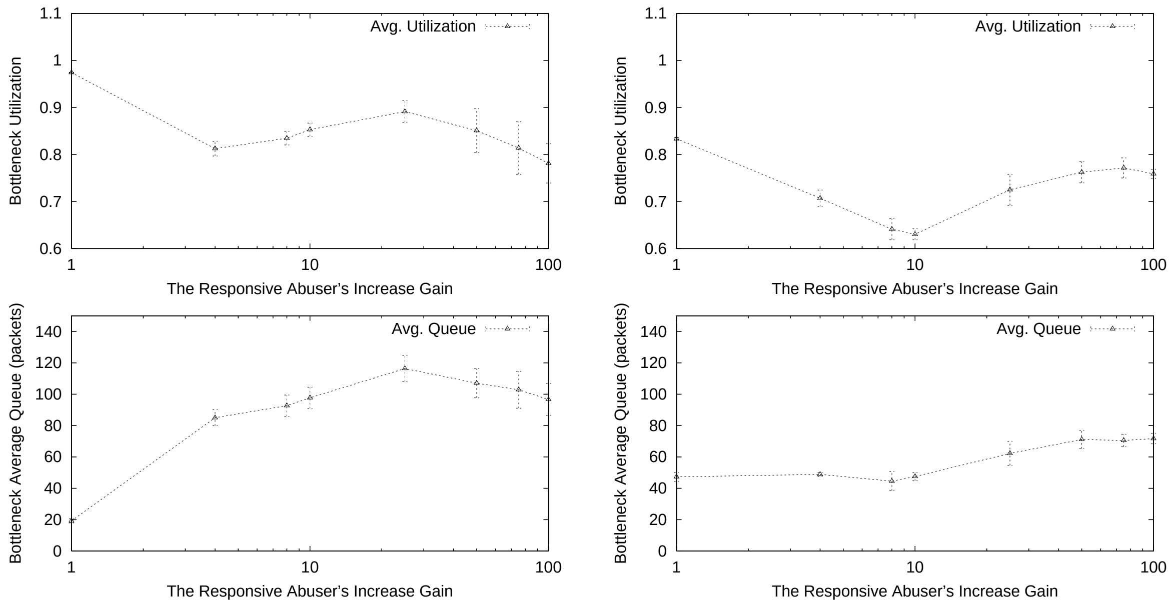



Figure 7-5: The impact of a responsive abuser on fairness. The graphs on the right use TCP, whereas the graphs on the left use XCP.

ms, and there are 20 flows along the reverse path. There are 20 flows along the forward path some of which may be abusive. Other parameters comply with the description in §5.1.

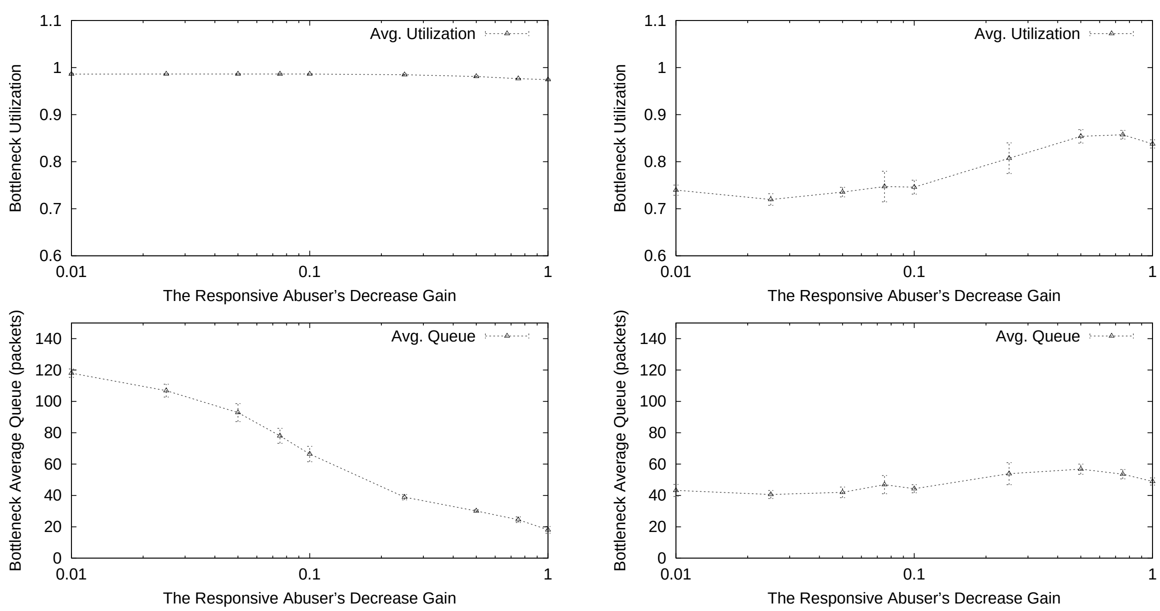
**Impact on Fairness:** In this experiment, one abusive flow is sharing the link with 19 compliant flows. Figure 7-5, plots the throughput of the abusive flow relative to its fair share as a function of the flow's gain for both XCP and TCP. The figure shows that when the abuser uses an aggressive increase or a lenient decrease parameter its throughput becomes larger than its fair share. The throughput increase is moderate for both XCP and TCP with RED. This is because the more the throughput of an XCP flow the larger the negative feedback it receives. Similarly, RED is designed to drop more from large bandwidth flows.

**Impact on Efficiency:** Next, we consider the impact of the responsive abuser on efficiency. In this experiment, all users are responsive abusers. Figure 7-6 shows the impact of abusers who increase aggressively on the average utilization and queue size, whereas Figure 7-7 shows the impact of abusers who decrease leniently on the average utilization and average queue size. The figures indicate that, both XCP and TCP are reasonably robust against responsive attack. In particular, the efficiency stays acceptable for an increase gain as large as 100 and a decrease gain as small as 0.01.




**(XCP)** **(TCP)**

Figure 7-6: A comparison between the efficiency of XCP and TCP, when all users increase aggressively by multiplying their fair positive increase by a gain $> 1$.




**(XCP)** **(TCP)**

Figure 7-7: A comparison between the efficiency of XCP and TCP, when all users decrease leniently by multiplying their fair negative decrease by a gain $< 1$.

### 7.3.4 Unresponsive Abuser

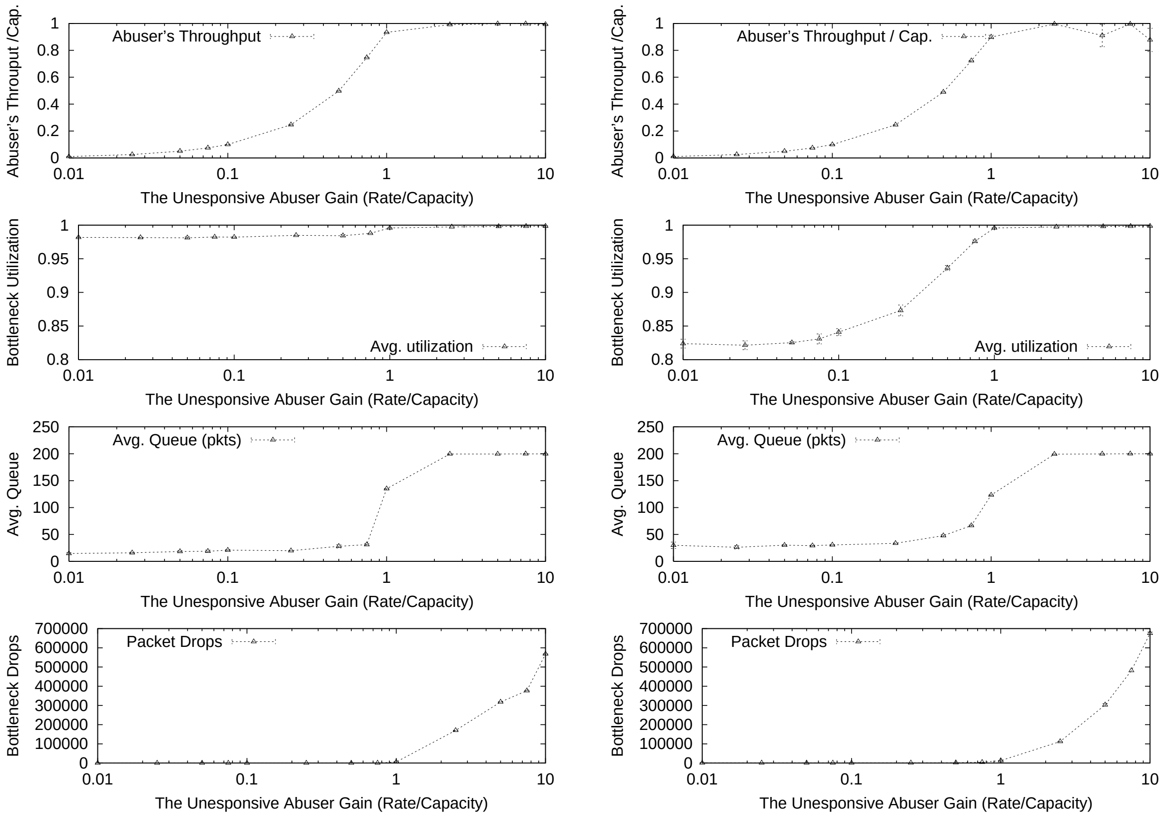
This abuser declares its true throughput and RTT, but completely ignores the feedback from the network, and continues sending at a particular rate it wants. In this case, the abuser achieves a throughput that is the minimum of its sending rate and the bottleneck bandwidth. Thus, the bandwidth allocation is not fair. As long as the abuser's sending rate is below the bottleneck bandwidth, the EC still maintains good utilization, but the unresponsiveness of the abuser causes the convergence to become slower.

#### Simulation Results

In this experiment, 10 unresponsive abusers share a 20 Mb/s bottleneck with 10 compliant flows. The round trip propagation delay is 80 ms, and there are 20 flows along the reverse path. Other parameters comply with the description in §5.1. We vary the abusive traffic rate and repeat the same experiment for different abuse gains, where the abuse gain is the ratio of the abusive traffic rate to the capacity of the link. We run the same experiments first with XCP then with Reno TCP and RED queues. Figure 7-8-a shows that the abusers obtain the throughputs they want as long as the sum of their rates is lower than the capacity of the link both in XCP and TCP networks. Figures 7-8-b, 7-8-c, and 7-8-d show that as long as the unresponsive abusers are sending less than the capacity of the bottleneck, the responsive flows fill up the rest of the bandwidth and the system stays efficient (though unfair). In general there is little difference between the reaction of XCP and TCP to non-responsive abusers. The only difference is that XCP shows better utilization than TCP when the abusive traffic is less than the capacity of the link.

## 7.4 Receivers' Misconduct

Even when the sender abides by the rules of the congestion control protocol, the receiver can cheat the sender into misbehaving by changing the feedback sent by the routers. This has been first observed in [71], where the authors explain how a TCP receiver can increase the throughput of the transfer by acking fractions of a TCP segments. Similarly, in [29], the authors note that a receiver can increase the sending rate beyond the fair share by resetting the ECN marks. As a remedy, they propose that the sender inserts a random nonce in the IP header. The routers reset the nonce if there is congestion. The receiver should relay the nonce to the sender. If the relayed nonce does not match the one the sender inserted in the packet, then the packet has faced congestion (i.e., packet has been




**(XCP)** **(TCP)**

Figure 7-8: The impact of a non-responsive abuser in an XCP and a TCP network. The graphs on the right use TCP whereas the graphs on the left use XCP.

marked).

Flow monitoring by policing agents at network edges or statistical monitoring at routers can catch a flow with a misbehaving receiver. Further, similarly to an ECN nonce, one can design an alternative solution that does not require any additional work from the routers. The XCP sender can initialize the feedback field in each packet to random values derived from a distribution with an average that matches the sender's demands and has a reasonable probability of values lower than the recently returned feedback. If the receiver lies about the feedback it conveys to the sender and sends a higher value than the one received in the packet, there will be a positive probability that the receiver picks a value larger than the initial value chosen by the sender. As a result, the sender will detect the misbehaving receiver. Since this approach is probabilistic allowing the receiver to get away with cheating some times, to disincentivize the receiver from cheating the punishment should be relatively severe such as shutting down the connection (and possibly labeling the receiver as a destination to avoid in future interactions).

## 7.5 Concluding Remarks

In this chapter, we examined the impact of users' misbehavior on the fairness and efficiency of an XCP network. We found that lying about one's throughput causes unfairness but does not affect efficiency, whereas lying about the RTT has little impact on fairness but may degrade the efficiency. In our simulations, the degradation is negligible when the declared RTT is within one order of magnitude of the true RTT, or when a the number of liars is small (i.e., less than 50% of the flows). But the degradation becomes significant when most of the flows significantly lie about their RTTs. We also studied the impact of ignoring the feedback or partially reacting to it. We found that when sharing the link with a completely unresponsive flow (i.e., a cbr), the XCP flows adapt to fill up the bandwidth unused by the unresponsive flow. On the other hand, if the abusive flow is partially reacting to the feedback but is too aggressive in grabbing the bandwidth and too lenient in releasing it, then it can obtain more bandwidth than other flows. All of these types of users' misconduct can be throttled by monitoring agents located at the edges of the network or via statistical sampling of the traffic.

There are other types of errors and malicious behavior that are not discussed in this chapter. In particular, we did not discuss the impact of malicious or erroneous routers. However, since most of the bugs at the routers result in an erroneous reading of the values in the congestion header or an erroneous computation of the feedback, the effect of these errors is similar to those created by a misbehaving sender who inserts incorrect values in the congestion header or does not react appropriately to congestion feedback. Also, we did not discuss the impact of a route change on XCP. Nonetheless, we note that a route change in XCP is more benign than in a TCP network. In particular, when a flow is moved from one route to another, the current throughput might be completely unsuitable to the new path, but the fact that XCP adapts quickly to the available bandwidth or the congestion state on the new path reduces the damaging effects of a route change.

# Chapter 8

# Gradual Deployment

Deploying a new congestion control protocol in the Internet is not an easy task. It is quite obvious that one cannot turn the Internet off, update all hosts and routers, then turn the Internet on again. A gradual deployment path is necessary for any practical protocol. Since XCP provides a joint design of the end systems and the routers, its deployment requires modifying the routers and the congestion control protocol in the operating system of the host machines. This chapter discusses two independent incremental deployment paths that allow updating a cloud of routers or a few machines at a time.

## 8.1 XCP-based Core Stateless Fair Queuing

XCP can be deployed in a cloud-based approach similar to that proposed by Core Stateless Fair Queuing (CSFQ) [75]. An XCP cloud could be a network domain or an autonomous system. This approach allows each ISP to update the routers in its network (or a cloud in its network) without coordinating with other ISPs, and independently from whether the end hosts have been updated.

To use XCP in this way, the edge routers surrounding the cloud would act on behalf of the senders and receivers. In other words, we map TCP or UDP flows across a network cloud onto XCP flows between the ingress and egress border routes. Each XCP flow is associated with a queue at the ingress router. Arriving TCP or UDP packets enter the relevant queue, and the corresponding XCP flow across the core determines when they can leave. For this purpose, $H\_rtt$ is the measured propagation delay between ingress and egress routers, and $H\_throughput$ is the speed at which the ingress router forwards packets from the flow's queue.

Maintaining an XCP core can be simplified further. First, there is no need to attach a congestion header to the packets, as feedback can be collected using a small control packet exchanged between border routers every RTT. Second, multiple TCP micro flows that share the same pair of ingress and egress routers can be mapped to a single XCP flow. The differential bandwidth scheme, described in §6.3, allows each XCP macro-flow to obtain a throughput proportional to the number of micro-flows in it. The router will forward packets from the queue according to the XCP macro-flow's rate. TCP will naturally cause the micro-flows to converge to share the XCP macro-flow fairly, although care should be taken not to mix responsive (e.g., TCP) and unresponsive (e.g., UDP) flows in the same macro-flow.

We note that an ISP has an incentive to update its network cloud to provide XCP even when end systems and others ISPs are not XCP enabled. Using XCP inside a cloud has several benefits. It would force unresponsive or UDP flows to use a fair share without needing per-flow state in the network core. It would improve the efficiency of the network cloud because an XCP core allows higher utilization, smaller queue sizes, and minimal packet drops. Further, it would allow an ISP to provide differential bandwidth allocation internally in their network. CSFQ shares these objectives, but our simulations indicate that XCP provides more fairness, higher utilization, lower delay, and smaller drop rate.

## 8.2 A TCP-Friendly XCP

In this section, we describe a mechanism that allows end-to-end XCP to compete fairly with TCP in the same network. This design can be used to allow XCP to exist in a multi-protocol network, or as a mechanism for incremental deployment.

To start an XCP connection, the sender must check whether the receiver and the routers along the path are XCP-enabled. If they are not, the sender reverts to TCP or another conventional protocol. These checks can be done using simple TCP or IP options.

We then extend the design of an XCP router to handle a mixture of XCP and TCP flows while ensuring that XCP flows are TCP-friendly.[1] The router distinguishes XCP traffic from non-XCP traffic and queues it separately. TCP (and other non-XCP) packets are queued in a conventional RED queue (the T-queue). XCP flows are queued in an XCP-enabled queue (the X-queue, described in §4.7). We need to use a separate queue for the XCP traffic to prevent TCP from starving the XCP flows. In particular, XCP reacts to congestion much earlier than TCP. While TCP reacts to

[1] A protocol is TCP-friendly if it fairly shares the bandwidth with competing TCP flows.

congestion after the queue has grown to the point of causing a drop, XCP starts decreasing its sending rate as soon as the input traffic rate exceeds the capacity. Thus, when the input traffic rate exceeds the capacity of the bottleneck, XCP starts releasing the bandwidth. TCP, which has not yet detected congestion, would be increasing its rate grabbing the bandwidth XCP is releasing. This continues until XCP releases all of its bandwidth to TCP. Then, the queue will buildup, causing a drop and TCP too will back off, but after it has starved the XCP flows.

To be fair, the router should process packets from the two queues such that the average throughput observed by XCP flows equals the average throughput observed by TCP flows, irrespective of the number of flows. This is done using weighted-fair queuing with two queues where the weights are dynamically updated and converge to the fair shares of XCP and TCP. The weight update mechanism uses the T-queue drop rate, $p$, to compute the average congestion window of the TCP flows. The computation uses a TFRC-like [34] approach, based on TCP's throughput equation:

$$\overline{cwnd}_T = \frac{s}{\sqrt{\frac{2p}{3}} + 12p\sqrt{\frac{3p}{8}} \times (1 + 32p^2)}, \tag{8.1}$$

where $\overline{cwnd}_T$ is the average TCP congestion window, and $s$ is the average packet size (both in bytes).

The average XCP congestion window is computed similarly to the average RTT (see Equation 4.2). In particular, the flow's congestion window is $cwnd_i = r_i \cdot rtt_i$, where $r_i$ is the flow's throughput and $rtt_i$ is its round trip time, both are in the header. As described in §4.7, to find the average cwnd of the flows, we take the average cwnd over the packets and normalize by the number of packets the router sees from a flow in a control interval $d$. This number is $d \cdot \frac{r_i}{s_i}$. Thus,

$$\overline{cwnd}_X = \frac{\sum rtt_i \cdot s_i}{\sum \frac{s_i}{r_i}}, \tag{8.2}$$

where both sums are over the packets observed by the router in a control interval.

When the estimation-control timer fires, the weights are updated as follows:

$$w_T = w_T + \kappa \frac{\overline{cwnd}_X - \overline{cwnd}_T}{\overline{cwnd}_X + \overline{cwnd}_T}, \tag{8.3}$$

$$w_X = w_X + \kappa \frac{\overline{cwnd}_T - \overline{cwnd}_X}{\overline{cwnd}_X + \overline{cwnd}_T}, \tag{8.4}$$

where $\kappa$ is a small constant in the range (0,1), and $w_T$ and $w_X$ are the T-queue and the X-queue

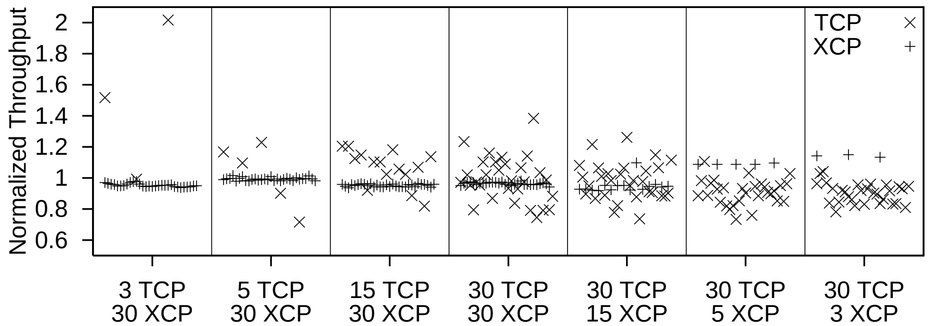



Figure 8-1: XCP is TCP-friendly.

weights. This updates the weights to decrease the difference between TCP's and XCP's average congestion windows. When the difference becomes zero, the weights stop changing and stabilize.

Finally, the aggregate feedback is modified to cause the XCP traffic to converge to its fair share of the link bandwidth:

$$\phi = \alpha \cdot S_X - \beta \cdot \frac{Q_X}{d}, \tag{8.5}$$

where $\alpha = 0.4$ and $\beta = 0.226$ are constant parameters, $d$ is the control interval, $Q_X$ is the size of the X-queue, and $S_X$ is XCP's fair share of the spare bandwidth computed as:

$$S_X = w_X \cdot c - y_X, \tag{8.6}$$

where $w_X$ is the XCP weight, $c$ is the capacity of the link, and $y_X$ is the total rate of the XCP traffic traversing the link.

**Simulation Results:** Figure 8-1 shows the throughputs of various combinations of competing TCP and XCP flows normalized by the fair share. The topology is that of Figure 5-1, the bottleneck capacity is 45 Mb/s, the round trip propagation delay is 40 ms, the buffer size is a delay-bandwidth product, and the data packet size is 1 KB. The RED parameters are $w = 0.002$, $p_{max} = 0.1$, $min_{thresh}$ is one-third of the buffer, and $max_{thresh}$ is two-thirds of the buffer. The simulations results demonstrate that XCP is as TCP-friendly as other protocols that are currently under consideration for deployment in the Internet [34].

## 8.3 Summary

This chapter shows that XCP can be deployed in the Internet incrementally. This gradual deployment can take two independent paths. First, the ISPs motivated by XCP's ability to maximize the utilization of their network resources might start deploying XCP in their network clouds. The border routers of these clouds provide the appropriate interface with the rest of the Internet. Second, the hosts may modify their network code to become XCP enabled. The sender runs XCP only if it detects that the whole path and the receiver are XCP enabled, otherwise it falls back to TCP.

# Chapter 9

# Conclusion & Future Work

In this chapter, we conclude the dissertation by summarizing our contributions, enumerating several remaining challenges, clarifying the limitations of our approach to congestion control, and proposing directions for future work.

## 9.1 Contributions

Below, we highlight the contributions of this thesis.

### 9.1.1 Good Performance in Networks with Large Bandwidth-Delay Product

The main reason why we re-examine congestion control now is to allow individual flows to obtain large end-to-end throughput (e.g., a few Gb/s). Current TCP-based congestion control cannot provide a large per-flow end-to-end bandwidth-delay product. This becomes a serious limitation as more users access the Internet using gigabit Ethernet or other high bandwidth link technologies. The limitation arises from two characteristics of TCP. First, TCP cannot quickly acquire large amounts of spare bandwidth because it increases by a constant amount of 1 packet/RTT. Second, TCP's throughput is inversely proportional to the packet drop rate. As a result, the packet error rate (PER) of the underlying link technology sets an upper bound on the end-to-end TCP throughput, which prevents the TCP from obtaining a very high throughput even over low PER fiber links [69].

In contrast, our simulation results in §5 show that XCP remains efficient as the bandwidth or the delay along the path increases. This happens because the XCP congestion controller increases the traffic rate proportionally to the spare bandwidth. The proposed adjustments allow the flows to ramp up quickly when the link is severely under-utilized, and perform minor adjustments when the traffic is near the capacity of the link. Even when a drop occurs due to a link error, XCP, with its fast dynamics, can re-acquire the bandwidth quickly and recover from this error.

Recently, a few researchers proposed various modifications to TCP to allow it to achieve a large end-to-end throughput [32, 47, 43]. At the time of writing this dissertation, these protocols are still works-in-progress and there is little experimental results to show their performance. Hence, in this document we do not provide a comparison between XCP and these protocols. However, we note that one major difference between these proposals and XCP is that the latter addresses the congestion control problem in general and attempts to provide fairer, more stable, and more efficient behavior both in large bandwidth-delay product networks, and in traditional environments.

### 9.1.2 XCP Outperforms Other Protocols With No Per-Flow State

Having no per-flow state is desirable since it reduces the complexity of the router and improves the scalability of the design. XCP and most queue management schemes adapt this simple design objective [75, 35, 50, 11]. Our simulation results in §5 show that XCP outperforms other congestion control protocols that similarly use no per-flow state, both in high bandwidth and traditional environments. In particular, XCP reduces the drop rate by three orders of magnitude, improves the utilization, and maintains on average a small queue size. Further, XCP exhibits no oscillation in steady state as seen in simulation and demonstrated using our model in §4.8. Additionally, while TCP is asymptotically fair, XCP is fair over a time scale of a few RTTs. This overall improved performance means that XCP contributes to the knowledge of the community by extending the space of achievable performance without per-flow state.

### 9.1.3 Integrating Control Theory with Internet Protocols

The design of XCP integrates concepts from control theory with the lessons the Internet community has learned from practice. Principles from control theory such as the Nyquist criterion are used to establish stability in steady state for any delay, capacity or number of flows. These are combined with Internet common practice such as self-clocking [39], which has been shown to increase robustness in dynamic environments [14]. Although this work is not the first to apply control theory to congestion control in the Internet, it manages to blend the principles of feedback control into the design of the XCP protocol while maintaining a simple distributed and practical implementation. Then, using large scale packet-level simulations, we have shown that the insight gained from the simple feedback control model has indeed resulted in an improved performance in both steady state

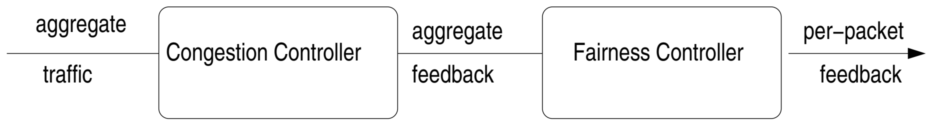



Figure 9-1: A black box model of the efficiency controller and the fairness controller.

and dynamic environments.

Early applications of control theory to congestion control required a repeated dumping of the state of all routers to a centralized server, which computes the control signal [22]. Later, Kelly et al have formulated the congestion control problem as a distributed optimization problem and proposed optimal distributed solution. This work has motivated a number of proposals including ours. In comparison to XCP's analysis, Kelly's model can incorporate complex topologies but it assumes no queues in the network. Recently, Doyle and Low have proposed a scalable distributed congestion control protocol whose design relies on control theory. In comparison with our protocol, their model is more general and takes into account heterogeneous RTTs and multiple control loops, but their protocol ignores the fairness requirement and focuses only on achieving good utilization.

### 9.1.4 New Concepts for Internet Bandwidth Management

In this dissertation we introduce a few new concepts and techniques that may be used by other congestion control protocols outside the XCP framework.

**Separate Congestion Controller and Fairness Controller at Routers:** We present an architecture where the congestion controller and the fairness controller at the router are separate. The two modules representing the controllers are designed as black boxes as shown in Figure 9-1. The congestion controller takes as input the aggregate traffic rate and the queue size. It generates an aggregate feedback that should be sent to the combination of the flows traversing the link. The fairness controller takes as input the aggregate feedback, and decides how to divide it between individual flows such that the system achieves a particular desired bandwidth allocation.

This modular architecture may be adapted by other congestion congestion control protocols without adapting the congestion control laws used by XCP. In particular, in XCP the congestion controller uses MIMD and the fairness controller uses AIMD. However, other protocols might want a different bandwidth allocation (e.g., proportional fairness) than the one provided by AIMD, so they might change the control law of the fairness controller, while maintaining the modular design.

**Bandwidth Shuffling:** We have introduced the concept of bandwidth shuffling which is the simultaneous allocation and deallocation of bandwidth such that the total traffic rate (and consequently the efficiency) does not change, yet the throughput of each individual flow changes gradually to approach the flow's fair share. This is a general concept that other explicit feedback protocols may adapt to improve fairness.

**Computing the Number of Flows Without per-Flow State:** We also provide a technique that computes an estimate of the number of flows without any per-flow state in the routers. It relies on a flow advertising its throughput in the packet header. Thus, it can work under CSFQ [75] and similar protocols.

Indeed, following the same style of equations and argument in §4.7, we can estimate the number of flows.[1] In particular, assume that the router estimates the number of flows over an interval $d$. Packet $i$ arrives with a label stating the rate of its flow $r_i$. The router would like to count the flows, however since it has no per-flow state it can only count the packets. Thus, we need a normalization factor that allow us to count the flows by counting their packets. This normalization factor is the expected number of packets flow $i$ sends in $d$ (i.e., $\frac{d \cdot r_i}{s_i}$). To see why this is the case, assume that every $d$, the router counts all packets from flow $i$ and divides the count by the average number of packets flow $i$ sends in $d$, the result on average should be 1. If the router sums this count over all packets from all flows, then the contribution of each flow to the sum is 1. Thus, the number of flows $N$ is:

$$N = \sum \frac{s_i}{d \times r_i}, \tag{9.1}$$

where $s_i$ is the packet size in bytes, $r_i$ is the flow's rate in bytes/sec and the sum is taken over all packets observed in the estimation interval $d$, which is measured in seconds.

### 9.1.5 A Simple Approach to Quality of Service

In §6, we presented a simple architecture that provides both guaranteed and proportional bandwidth allocation. We argued that these two allocation policies combined with XCP's natural tendency for small queue size and almost-zero drop rate satisfy most applications.

[1] Although we have not explicitly estimated the number of flows in previous chapters, the FC implicitly estimates the number of flows while computing the positive per-packet feedback. Besides the estimate is just an application of our general style of computing averages over flows using per-packet information. We have applied this same style in Equations 4.2, 6.1, and 8.2.

## 9.2 Limitations

As we look at the limitations of this work, we divide them into two categories. The first type of limitations is open problems that we do not address in this dissertation though they are potentially solvable within the XCP framework. These are discussed in §9.3. The second type of limitations is what we call fundamental limitations, which are intrinsic to our approach and constitute essential design trade-offs. These are discussed below.

### 9.2.1 Deployment

In this thesis, we have chosen to step back and reconsider the design of Internet congestion control with the objective of providing a fairer and more stable controller that remains efficient as the per-flow bandwidth-delay product increases. Since the control loop involves both the routers and the end systems, redesigning congestion control means allowing ourselves to change both. But then, because XCP modifies both the routers and the end systems, it is harder to deploy than a scheme that modifies solely the routers or the end systems. We mitigate the deployment hurdle by providing a gradual deployment path. However, we still recognize that the deployment problem is not only technical but also economical. We believe that as more organizations connect to the Internet using very high bandwidth access links, there will be a market for protocols like XCP. Whether the market really evolves in this direction is something to be discovered over the coming years.

### 9.2.2 Security Without Per-Flow State

All congestion control protocols that have no per-flow state cannot guarantee protection against malicious flows, which might try to deny compliant flows their fair share of the bandwidth. XCP being a stateless protocol cannot guarantee flow protection. Further, XCP, being a new and different architecture, exhibits new ways in which malicious users can cheat. Particularly, because XCP requires the users to declare their state in the header, flows can misbehave by lying about their state. In §7, we have studied via simulation these new types of attacks and shown that XCP is reasonably robust against them. Also, we have proposed two methods to detect and isolate misbehaving flows: policing agents at the edges of the network, and statistical sampling at the routers. We have also pointed out that XCP makes monitoring and detection of misbehaving flows easier because the monitoring agent can detect misbehavior by comparing the information in the congestion header against its measurements.

## 9.3 Outstanding Problems & Future Work

Below, we enumerate a few challenging issues that our dissertation does not solve. These are not fundamental limitations of XCP, rather they are outstanding problems that, we believe, are solvable within the XCP framework. We discuss potential solutions to these problems, but we do not explore these solutions in detail, and we leave them as directions for future work.

### 9.3.1 Multi-Access Links

In §4.7, we stated that job of the the efficiency controller is to allocate the spare bandwidth to the flows. In particular, the EC tries to cause the aggregate traffic to change its rate by the value of the aggregate feedback $\phi$:

$$\phi = \alpha \cdot S - \beta \cdot \frac{Q}{d},$$

where $\alpha$ and $\beta$ are constant parameters, the term $d$ is the average RTT, $Q$ is the persistent queue size, and $S$ is the spare bandwidth defined as the difference between the input traffic rate and the link capacity.

The above equation contains two implicit assumptions. First, the administrator knows the capacity of the link and can configure the router with this value. Although this is the case for most wired links it is not quite true for wireless links, where the capacity varies over time. The second assumption is that each link is accessed by a single router, and therefore the spare bandwidth is the difference between the capacity and the traffic emitted by the router. This assumption becomes incorrect over multi-access links such as Ethernet[2] and wireless links.

As directions for future work, we propose two potential solutions to this problem. The first and most intuitive approach is to estimate the the capacity of the link $\hat{c}(t)$, which is the speed at which the router drains packets. The equation above becomes:

$$\phi = \alpha \cdot (\hat{c}(t) - y) - \beta \cdot \frac{Q}{d},$$

where $y$ is the input traffic at the router. Further work is needed to study the robustness of this approach to typical variations in link capacity. The second potential solution is to use a different efficiency controller at the routers feeding multi-access link. There is no reason why the XCP controllers at different routers along the path should use the same control law. The efficiency controller

[2] Most Ethernet links are switched but the original design is for a multi-access medium.

at routers feeding multi-access links may be modified so that it does not compute the spare bandwidth (e.g., when the queue is empty, it increases proportionally to the capacity, and when the queue has some packets, it decreases proportionally to the queue size). This solution should be examined further to explore its performance and study its impact on the stability of the control loop.

### 9.3.2 Increasing Robustness to RTT Variance

The congestion control problem has a fundamental time scale, which is the RTT. Our analysis shows that when all flows at the bottleneck have the same RTT, setting the control interval $d$ to the RTT provides stability. However, in reality flows at a bottleneck have different RTTs. Since we need to pick a single value for the control interval, we chose to set $d$ to the average RTT of the flows (indeed, d = min(avg. RTT, 5 ms)). Our simulation results in §5 show that this choice is fairly robust for RTTs $\in$ [1 ms, 1000 ms], which is the range of RTTs in the current Internet. As the variance in the RTTs at a bottleneck exceeds the above range, setting $d$ to the average RTT might not work efficiently.

As future work, we propose addressing this issue by using a small and finite number of RTT groups. For example, one can group flows whose RTT $<$ 500 ms in one group, flows with 500 ms $\leq$ RTT $<$ 1000 ms in a second group, flows with 1000 ms $\leq$ RTT $<$ 1500 ms in a third group, and so on. Each of these groups will have a different efficiency and fairness controllers (i.e., each group will have separate parameters $d$, $\phi$, $\xi_p$, $\xi_n$,...). The fair share of each group can be computed based on the number of its flow, which is computed using equation 9.1 above. The job of the efficiency controller of each group is to make the traffic in the group converge to the group's fair share.

### 9.3.3 Specifying the Bit Format of the Congestion Header

XCP defines a new congestion header illustrated in Figure 4-1. This document does not specify the number of bits in each field in the header, or the way the values are represented (integers, floating points, etc.). The specification of these fields is left to the IETF or similar standardization organizations.

## 9.4 Final Remarks

In this dissertation, we take a step back and re-examine Internet congestion control, with the objective of enabling connections with large bandwidth-delay product and improving the overall performance. Motivated by CSFQ, we chose to convey control information between the end systems and the routers using a few bytes in the packet header. The most important consequence of this explicit control is that it permits a decoupling of *congestion control* from *fairness control*. In turn, this decoupling allows more efficient use of network resources and more flexible bandwidth allocation schemes.

Based on these ideas, we devised XCP, an explicit congestion control protocol and architecture that can control the dynamics of the aggregate traffic independently from the relative throughput of the individual flows in the aggregate. Controlling congestion is done using an analytically tractable method that matches the aggregate traffic rate to the link capacity, while preventing persistent queues from forming. The decoupling then permits XCP to reallocate bandwidth among individual flows without worrying about being too aggressive in dropping packets or too slow in utilizing spare bandwidth. We demonstrated a fairness mechanism based on *bandwidth shuffling* that converges much faster than TCP does, and showed how to use this to implement both max-min fairness and differential bandwidth allocation.

Our extensive simulations demonstrate that XCP maintains good utilization and fairness, has low queuing delay, and drops very few packets. We evaluated XCP in comparison with TCP over RED, REM, AVQ, and CSFQ queues, in both steady-state and dynamic environments with Web-like traffic and with impulse loads. We found no case where XCP performs significantly worse than TCP. In fact when the per-flow bandwidth-delay product becomes large, XCP's performance remains excellent whereas TCP suffers significantly.

We believe that XCP is viable and practical as a congestion control scheme. It operates the network with almost no drops, and substantially increases the efficiency in high bandwidth-delay product environments.

# Appendix A

# XCP Implementation

Implementing[1] an XCP router is fairly simple and is best described using the following pseudo code. There are four relevant blocks of code. The first block is executed at the arrival of a packet and involves updating the estimates maintained by the router.

---

```
On packet arrival do:
    input_traffic += pkt_size
    sum_inv_throughput += pkt_size / H_throughput
    if (H_rtt < MAX_INTERVAL) then
        sum_rtt_by_throughput += H_rtt × pkt_size / H_throughput
    else
        sum_rtt_by_throughput += MAX_INTERVAL × pkt_size / H_throughput
```

---

The second block of code is executed when the estimation-control timer expires. It involves updating our control variables, re-initializing the estimation variables, and re-scheduling the timer.

---

```
On estimation-control timeout do:
    avg_rtt = sum_rtt_by_throughput / sum_inv_throughput
    φ = α× (capacity - input_traffic / control_interval) - β× Queue / avg_rtt
    shuffled_traffic = 0.1× input_traffic / control_interval
```

[1]Our *ns* implementation is available at www.ana.lcs.mit.edu/dina/XCP.

```
    ξ_p = (max(φ,0) + shuffled_traffic) / sum_inv_throughput
    ξ_n = ((max(−φ,0) + shuffled_traffic) / input_traffic
    residue_pos_fbk = (max(φ,0)+ shuffled_traffic)
    residue_neg_fbk = (max(−φ,0)+ shuffled_traffic)
    input_traffic = 0
    sum_inv_throughput = 0
    sum_rtt_by_throughput = 0
    control_interval = max(avg_rtt, MIN_INTERVAL)
    timer.reschedule(control_interval)
```

The third block of code involves computing the per-packet feedback and is executed at packets' departure.

```
On packet departure do:
    pos_fbk = ξ_p × pkt_size / H_throughput
    neg_fbk = ξ_n × pkt_size
    feedback = pos_fbk - neg_fbk
    if (H_feedback ≥ feedback) then
        H_feedback = feedback
        residue_pos_fbk -= pos_fbk
        residue_neg_fbk -= neg_fbk
    else
        if (H_feedback ≥ 0)
            residue_pos_fbk -= H_feedback
            residue_neg_fbk -= (feedback - H_feedback)
        else
            residue_neg_fbk += H_feedback
            if (feedback ≥ 0) then
                residue_neg_fbk -= feedback
                residue_pos_fbk -= feedback
    if (residue_pos_fbk ≤ 0) then ξ_p = 0
```

if (residue_neg_fbk $\leq 0$) then $\xi_n = 0$

---

Finally, to improve the performance, we do not use the instantaneous queue in computing the aggregate feedback. Instead, we use an estimate of the persistent queue, which is computed as follows.

---

On packet departure do:

if (instantaneous_queue $<$ min_queue) then min_queue = instantaneous_queue

---

When the queue-computation timer expires do:

Queue = min_queue

min_queue = instantaneous_queue

Tq = max(MIN_INTERVAL, (avg_rtt - instantaneous_queue / capacity))/2

queue_timer.reschedule(Tq)

---

Note that the code executed when the estimation-control or the queue-estimation timer expires does not fall on the critical path. The per-packet code can be made substantially faster by replacing the throughput field in the congestion header by packet_size/throughput. This modification spares the router any division operation, **in which case, the router does only a few additions and** $3$ **multiplications per packet.**

# Appendix B

# Analysis of XCP Stability

## B.1 Model

Consider a single link of capacity $c$ traversed by $N$ XCP flows. Let $d$ be the common round trip delay of all users, and $r_i(t)$ be the sending rate of user $i$ at time $t$. The aggregate traffic rate is $y(t) = \sum r_i(t)$, and the shuffled traffic rate is $h(t) = 0.1 \cdot y(t)$. Every control interval, $d$, the router sends some aggregate feedback, which it computes according to Equation 9.3.1. The feedback reaches the sources after a round trip delay. It changes the sum of their rates (i.e.,$\sum r_i(t)$). Thus, the aggregate feedback sent per time unit is the sum of the derivatives of the rates:

$$\sum \frac{dr_i}{dt} = \frac{1}{d}\left(-\alpha \cdot (y(t-d) - c) - \beta \cdot \frac{q(t-d)}{d}\right).$$

Since the input traffic rate is $y(t) = \sum r_i(t)$, the derivative of the traffic rate $\dot{y}(t)$ is:

$$\dot{y}(t) = -\frac{\alpha}{d} \cdot (y(t-d) - c) - \frac{\beta}{d^2} \cdot q(t-d).$$

Ignoring the boundary conditions (i.e., queues are bounded and rates cannot be negative), the whole system can be expressed using the following delay differential equations.

$$\dot{q}(t) = y(t) - c \tag{B.1}$$

$$\dot{y}(t) = -\frac{\alpha}{d}(y(t-d) - c) - \frac{\beta}{d^2}q(t-d) \tag{B.2}$$

$$\dot{r}_i(t) = \frac{1}{N}\Big([\dot{y}(t-d)]^+ + h(t-d)\Big) - \frac{r_i(t-d)}{y(t-d)}\Big([-\dot{y}(t-d)]^+ + h(t-d)\Big) \tag{B.3}$$

The notation $[\dot{y}(t-d)]^+$ is equivalent to $\max(0, \dot{y}(t-d))$. Equation B.3 expresses the AIMD policy used by the FC; namely, the positive feedback allocated to flows is equal, while the negative feedback allocated to flows is proportional to their current throughputs.

## B.2 Stability

Let us change variable to $x(t) = y(t) - c$.

*Proposition:* The Linear system:

$$\dot{q}(t) = x(t)$$

$$\dot{x}(t) = -K_1 x(t-d) - K_2 q(t-d)$$

is stable for any constant delay $d > 0$ if

$$K_1 = \frac{\alpha}{d}, \quad and \quad K_2 = \frac{\beta}{d^2},$$

where $\alpha$ and $\beta$ are any constants satisfying:

$$0 < \alpha < \frac{\pi}{4\sqrt{2}}, \quad and \quad \beta = \alpha^2\sqrt{2}.$$

*Proof:*

The system can be expressed using a delayed feedback (see Figure B-1). The open loop transfer function is:

$$G(s) = \frac{K_1 \cdot s + K_2}{s^2} e^{-ds}$$

For very small $d > 0$, the closed-loop system is stable. The shape of its Nyquist plot, part of which is given in Figure B-2, does not encircle $-1$.

Next, we can prove that the phase margin remains positive independent of the delay. The magnitude and angle of the open-loop transfer function are:

$$|G| = \frac{\sqrt{K_1^2 \cdot w^2 + K_2^2}}{w^2},$$

$$\angle G = -\pi + \arctan \frac{wK_1}{K_2} - w \cdot d.$$

The break frequency of the zero occurs at: $w_z = \frac{K_2}{K_1}$.

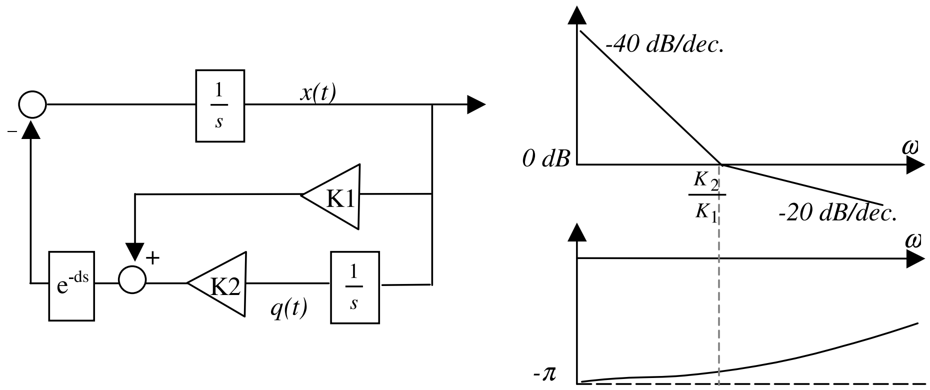



Figure B-1: The feedback loop and the Bode plot of its open loop transfer function.

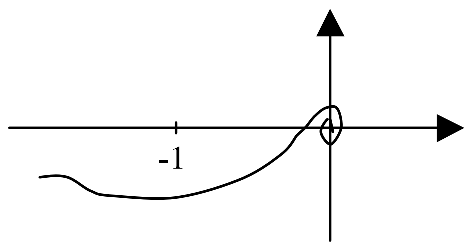



Figure B-2: The Nyquist plot of the open-loop transfer function with a very small delay.

To simplify the system, we decided to choose $\alpha$ and $\beta$ such that the break frequency of the zero $w_z$ is the same as the crossover frequency $w_c$ (frequency for which $|G(w_c)| = 1$). Substituting $w_c = w_z = \frac{K_2}{K_1}$ in $|G(w_c)| = 1$ leads to:

$$\beta = \alpha^2\sqrt{2}.$$

To maintain stability for any delay, we need to make sure that the phase margin is independent of delay and always remains positive. This means that we need:

$$\angle G(w_c) = -\pi + \frac{\pi}{4} - \frac{\beta}{\alpha} > -\pi \Rightarrow \frac{\beta}{\alpha} < \frac{\pi}{4}.$$

Substituting $\beta$ from above, we find that we need:

$$\alpha < \frac{\pi}{4\sqrt{2}},$$

in which case, the gain margin is larger than one and the phase margin is always positive (see the Bode plot in Figure B-1). This is true for any delay, capacity, and number of sources.

# Bibliography


[1] Differentiated services (diffserv). http://www.ietf.org/html.charters/diffserv-charter.html.

[2] Grid high-performance networking research group. http://www.csm.ornl.gov/ghpn/GHPNHome.html.

[3] high-performance research networks. http://abilene.internet2.edu/html/peernetworks.html.

[4] Integrated services (intserv). http://www.ietf.org/html.charters/intserv-charter.html.

[5] Internet2, giga-tcp. http://www.internet2.edu/ shalunov/gigatcp/.

[6] June 2002. Private Communication with Nick McKeown and Sally Floyd.

[7] 10 gigabit ethernet technology overview white paper - revision 2, draft a., 2002. The 10 Gigabit Ethernet Alliance - http://www.10gea.org/Tech-whitepapers.htm.

[8] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: A simple and effective flow control scheme. In *Proc. of ACM SIGCOMM*, 1996.

[9] M. Allman, D. Glover, and L. Sanchez. Enhancing tcp over satellite channels using standard mechanisms, Jan. 1999.

[10] G. Asthana and J. Denaro. Gigabit ethernet: An affordable solution. Technical Report ECE 4006C G4, Georgia Institute of Technology, Jan. 2002.

[11] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. Rem: Active queue management. *IEEE Network*, 2001.

[12] H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for internet hosts. In *Proc. of ACM SIGCOMM*, Sept. 1999.

[13] D. Bansal and H. Balakrishnan. Binomial congestion control algorithms. In *Proc. of IEEE INFOCOM '01*, Apr. 2001.

[14] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. *Proceedings of ACM SIGCOMM '01 (To appear)*, 2001.

[15] D. Bansal, H. Balakrishnan, and S. S. S. Floyd. Dynamic behavior of slowly-responsive congestion control algorithms. In *Proc. of ACM SIGCOMM*, 2001.

[16] J. C. R. Bennett, K. Benson, A. Charny, W. F. Courtney, and J. Y. L. Boudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. In *INFOCOM*, pages 1502–1509, 2001.

[17] J. L. Boudec and A. Charny. Packet scale rate guarantee for non-fifo nodes.

[18] A. Charny. An algorithm for rate allocation in a packet-switching network with feedback. Master's thesis, Massachusetts Institute of Technology, 1994.

[19] J. Chase, A. Gallatin, and K. Yocum. End-system optimizations for high-speed tcp. *IEEE Communications*, 2000.

[20] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. In *Computer Networks and ISDN Systems 17, page 1-14*, 1989.

[21] K. G. Coffman and A. M. Odlyzko. Growth of the internet. *In Optical Fiber Telecommunications IV B: Systems and Impairments*, pages 17–56, 2002.

[22] B. F. Controller. On rate-based congestion control in high speed networks: Design of an.

[23] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *IEEE/ACM Transactions on Networking, 5(6):835–846*, Dec. 1997.

[24] J. Crowcroft and P. Oechslin. Differentiated end-to-end internet services using a weighted proportional fair sharing tcp, 1998.

[25] A. Demers, S. Keshav, and S. S.

[26] M. Doar. Tiers topology generator. http://www.isi.edu/nsnam/ns/ns-topogen.html.

[27] C. Dovrolis, D. Stiliadis, and P. Ramanthan. Proportional differentiated service: Delay differentiation and packet scheduling. In *Proc. of ACM SIGCOMM'99*, Aug. 1999.

[28] P. Dykstra. Issues impacting gigabit networks: Why don't most users experience high data rates? http://sd.wareonearth.com/ phil/issues.html.

[29] D. Ely, N. Spring, D. Wetherall, S. Savage, and T. Anderson. Robust congestion signaling. In *ICNP*, 2001.

[30] C. Estan and G. Varghese. New directions in traffic measurement and accounting, 2002.

[31] W. Feng, D. Kandlur, D. Saha, and K. G. Shin. BLUE: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, 15, 1999.

[32] S. Floyd. Highspeed tcp for large congestion windows, 2002. Internet draft, work in progress.

[33] S. Floyd, R. Gummadi, and S. Shenker. Adaptive red: An algorithm for increasing the robustness of red, 2001.

[34] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM*, Aug. 2000.

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

[36] R. Gibbens and F. Kelly. Distributed connection acceptance control for a connectionless network. In *Proc. of the 16th Intl. Telegraffic Congress*, June 1999.

[37] T. H. Henderson, E. Sahouria, S. McCanne, and R. H. Katz. On improving the fairness of TCP congestion avoidance. *IEEE Globecomm conference, Sydney*, 1998.

[38] C. Hollot, V. Misra, D. Towsley, , and W. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proc. of IEEE INFOCOM*, Apr. 2001.

[39] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium*, 18, 4:314–329, Aug. 1988.

[40] R. Jain, S. Fahmy, S. Kalyanaraman, and R. Goyal. The erica switch algorithm for abr traffic management in atm networks: Part ii: Requirements and performance evaluation. In *The Ohio State University, Department of CIS*, Jan. 1997.

[41] R. Jain, S. Kalyanaraman, and R. Viswanathan. The osu scheme for congestion avoidance in atm networks: Lessons learnt and extensions. In *Performance Evaluation Journal, Vol. 31/1-2*, Dec. 1997.

[42] H. Jiang and C. Dovrolis. Passive estimation of tcp round-trip times, 2002.

[43] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, H. Newman, F. Paganini, S. Ravot, and S. Singh. Fast kernel: Background theory and experimental results. In *First International Workshop on Protocols for Fast Long-Distance Networks*, Feb. 2003.

[44] D. Katabi and C. Blake. A note on the stability requirements of adaptive virtual queue, 2002. MIT Technichal Memo, 2002.

[45] D. Katabi and M. Handley. Precise feedback for congestion control in the internet. Technical report, MIT, 2001.

[46] F. Kelly. Mathematical modeling of the internet. In *Proc. of Fourth International Congress on Industrial and Applied Mathematics*, 1999.

[47] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. In *Submitted for publication*, Dec. 2002.

[48] S. Keshav. Packet-pair flow control. Technical report, Murray Hill, New Jersey, 1994.

[49] S. Keshav. *An Engineering Approach to Computer Networks*. Addison Wesley, 1997.

[50] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue. In *Proc. of ACM SIGCOMM*, 2001.

[51] R. J. La, P. Ranjan, and E. H. Abed. Nonlinearity of tcp and instability with red. In *ITcom*, July 2002.

[52] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control: An analytical perspective. *IEEE Control Systems Magazine*, 2001.

[53] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle. Dynamics of tcp/aqm and a scalable control. In *Proc. of IEEE INFOCOM*, June 2002.

[54] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy red, June 1999.

[55] S. McCreary and K. C. Claffy. Trends in wide area ip traffic patterns. In *Tech. Rep.CAIDA*, Feb. 2000.

[56] N. McKeown. Do optics and routers belong together?, 2001.

[57] V. Misra, W. Gong, and D. Towsley. A fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proc. of ACM SIGCOMM'00*, Aug. 2000.

[58] R. Morris. Tcp behavior with many flows. In *in International Conference on Network Protocols*, Oct. 1997.

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

[60] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

[61] J. Padhye and S. Floyd. On inferring TCP behavior. In *Proceedings of ACM Sigcomm 2001*, pages 287–298.

[62] F. Paganini, J. C. Doyle, and S. H. Low. Scalable laws for stable network congestion control. In *IEEE CDC*, 2001.

[63] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping, 2001.

[64] R. Pan, B. Prabhakar, and K. Psounis. CHOKE, a stateless active queue management scheme for approximating fair bandwidth allocation. In *INFOCOM (2)*, pages 942–951, 2000.

[65] V. Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[66] J. Postel. Transmission control protocol, Sept. 1981.

[67] K. K. Ramakrishnan and S. Floyd. Proposal to add explicit congestion notification (ecn) to ip, Jan. 1999.

[68] Red parameters. http://www.icir.org/floyd/red.html#parameters.

[69] R.Hui, B. Zhu, R. Huang, C. Allen, and K. Demarest. 10 gb/s scm fiber system using optical ssb modulation. *IEEE Photonics Technology Letters*, 13, Aug. 2001.

[70] C. Rohrs, R. Berry, and S. O'Halek. Control engineer's look at atm congestion avoidance. *Comp Comm*, 19, Mar. 1996.

[71] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion contorl with a misbehaving receiver. *Computer Communication Review*, 29(5), 1999.

[72] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A transport protocol for real-time applications. *Internet-Draft ietf-avt-rtp-new-01.txt (work inprogress)*, 1998.

[73] S. Shalunov. Gigatcp - tcp on gigabit ethernet. *Internet2/NLANER Joint Techs Workshop, Boulder*, July 2002.

[74] I. Stoica. Stateless core: A scalable approach for quality of service in the internet, 2000.

[75] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proc. of ACM SIGCOMM*, Aug. 1998.

[76] I. Stoica, H. Zhang, and S. Shenker. Self-verifying csfq.

[77] K. Thompson, G. Miller, and M. Wilder. Wide-area internet traffic patterns and characteristics. In *IEEE vol. 11, no. 6*, Nov. 1997.

[78] A. Veres and M. Boda. The chaotic nature of TCP congestion control. In *INFOCOM (3)*, pages 1715–1723, 2000.

[79] J. Xu and R. J. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *Proc. of ACM SIGCOMM '02*, Aug. 2002.

[80] S. S. L. Zhang and D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. *ACM SIGCOMM 91*, 1991.