

In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

ACTIVE QUEUE MANAGEMENT POLICY BASED ON LIFETIME CLASSIFICATION

by

Xudong Wu ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** .

Department of Computing Science

Edmonton, Alberta
Fall 2003



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitons et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

ISBN: 0-612-88070-2

Our file *Notre référence*

ISBN: 0-612-88070-2

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author: Xudong Wu

Title of Thesis: Active Queue Management Policy Based on Lifetime Classification

Degree: Doctor of Philosophy

Year this Degree Granted: 2003

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.


The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: July 14 / 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Active Queue Management Policy Based on Lifetime Classification** submitted by Xudong Wu in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**



Dr. Ioanis Nikolaidis

Dr. ~~Ijil~~ Jana Trajkovic

Dr. Petr Musilek

Dr. Janelle Harms

Dr. Michael H. MacGregor

July 10/ 2003

Abstract

The unfairness problem in terms of throughput between competing TCP flows has been studied for at least a decade. The primary causes of this unfairness problem are the negative impact of mixing long- and short-lived flows and the difference in round trip times (RTTs) of different flows. Eventually, the unfairness may lead to starvation of particular TCP flows. To counteract the unfairness, we propose a *DiffServ*-like active queue management (AQM) scheme. It essentially involves classification of TCP flows into two classes, namely, long- and short-lived flows, and application of different control policies to each class. In this thesis, first, we consider applying the existing analytical throughput models to mixtures of long-lived TCP flows. This will help us understand the difficulty of the problem and the reason why the available models do not appear to be sufficient. We first describe our model-based policy, *FairShare*, that imposes fairness among long-lived flows. The policy is based on scheduled losses. With *FairShare*, max-min fairness can be achieved in single link environment. We also validate the performance of *FairShare* in larger networks and with dynamic background traffic. Subsequently, we illustrate how *FairShare* can achieve global max-min fairness. We use statistical studies of real Internet traffic to justify the complexity and feasibility of *FairShare*. We discuss general classification schemes based on lifetime and RTTs. In particular, we propose a scheme, *DAS*, to dynamically allocate bandwidth among classes based on measured demands. We thus advocate, that regardless whether the particular *FairShare* scheme is used or not, a separation of TCP flows in classes reflecting their RTT and lifetime membership is beneficial.

Acknowledgements

I owe this thesis to the help, support, guidance and encouragement of several people. My utmost gratitude goes to my adviser, Professor Ioanis Nikolaidis for his support and guidance all these years. It has been a privilege to have worked with such a researcher who maintains high standards for himself and insists the same from others around him. I would like to thank Professor Jenelle Harms who was once my co-adviser for her guidance and encouragement and support. I would also like to thank Professor Mike MacGregor for his encouragement and constructive advices, and Professor Pawel Gburzynski for his support. I am grateful to the rest of my dissertation committee - Professor Peter Musilek and Professor Ljiljana Trajkovic for their advice and support in improving the quality of the thesis. My graduate school experience has been vastly enriched by interactions with my colleagues in the research group over the past years - Yanxia Jia, Hongjun Zhang, Kui Wu, Qiang Ye, Weiguang Shi, Chong Wang, Lei Wang, Elvira Akhmetshina, Chong Zhang, Baochun Bai, Yuxi Li, Yuan Sha, Fulu Li, Junhui Shen, Kun Bai, Juhua Shi, and Ping Xian. I have always enjoyed and cherished their company. Many thanks to Edith Drummond who handles the graduate student affairs in the department. I would also like to thank Professor Armann Ingolfsson for his support and knowledge of stochastic process and optimization techniques I have learned from him.

Contents

1	Introduction and Background	1
1.1	A Brief Introduction to TCP	1
1.1.1	The Original TCP	1
1.1.2	The Presence of Congestion	2
1.1.3	The Reaction to Congestion	3
1.1.4	The Conservative Start	4
1.1.5	The Recovery from Congestion	5
1.1.6	TCP Variants	6
1.1.7	RTT Estimation in TCP	7
1.2	Definitions	9
1.2.1	Max-min Fairness	9
1.2.2	Goodput and Throughput	10
1.2.3	Loss (Wired vs. Wireless Environment)	11
1.3	The Basic Problems of TCP	11
1.3.1	Phase Effect and RED	12
1.3.2	Fairness Over Flows with Different RTTs	14
1.3.3	Long vs. Short TCP Flows	17
1.4	Thesis Objective and Outline	18
2	TCP Throughput Models	20
2.1	Simulation-based Models	20
2.2	Individual Flow Equilibrium Throughput Models	23
2.3	Other Approximation Models	28
3	Model-based Link Oriented Fairness	30
3.1	Introduction	30
3.2	The FairShare Scheme	31
3.2.1	Regulating TCP Flows via Scheduled Losses	32
3.2.2	The Algorithm	34
3.3	Simulation Study	37
3.4	Discussion	42
4	AQM and Global Fairness Objectives	44
4.1	Introduction	44
4.2	Distributed Global Fairness	45

4.2.1	Synchronized Algorithm	45
4.2.2	Distributed Algorithm and Convergence	49
4.2.3	FairShare Algorithm	54
4.3	Simulation Study	56
4.3.1	Experiment 1	56
4.3.2	Experiment 2	57
4.3.3	Experiment 3	59
4.3.4	Experiment 4	59
4.4	Conclusion	61
5	TCP Lifetime & RTT Classification	63
5.1	Introduction	63
5.2	Lifetime Classification	63
5.3	Identifying Long-Lived TCP Flows	66
5.4	RTT Classification	67
5.5	RTT Estimation	68
5.6	Classification Schemes	69
5.6.1	Classification Schemes Based on Simple Rules	69
5.6.2	Dynamic Bandwidth Allocation for Lifetime-Based TCP Classification	70
5.6.3	DAS-BV	72
5.6.4	DAS-ED	73
5.6.5	DAS-EL	75
6	Performance Investigation of Classification Schemes	80
6.1	Introduction	80
6.2	Evaluation of DAS	80
6.2.1	Simulation Setup	80
6.2.2	Experiment Results	82
6.3	Investigation of Classification Based on Simple Rules	88
6.3.1	Simulation Setup	88
6.3.2	Evaluation Results	89
6.4	Conclusions	96
7	Some Recent Results	98
7.1	Introduction	98
7.2	Evaluation of XCP	98
7.2.1	XCP Congestion Header and Efficient Control	99
7.2.2	XCP Fairness Control	100
7.2.3	XCP Performance Under Mixture of RTTs	103
7.2.4	XCP Performance Under Mixture of Lifetime and Dynamic Conditions	103
7.3	An Alternate Lifetime-Based Scheme	106
8	Summary and Future Work	109
8.1	Summary	109
8.2	Future Work	111

List of Figures

1.1	Illustration of TCP: (a) Tahoe and, (b) Reno.	8
1.2	RED Operation.	14
1.3	Unfairness due to RTT difference: (a) DropTail and (b) RED.	16
2.1	Experimental Topology.	21
2.2	Joint and marginal PDF of W_1 and W_2	26
3.1	Steady state congestion window behavior in, (a) TCP-Tahoe and, (b) TCP-Reno.	33
3.2	The <code>init_long_flow()</code> function.	35
3.3	The <code>tick()</code> function.	36
3.4	The <code>upon_packet_arrival()</code> function.	37
3.5	Throughput achieved by FairShare vs. that by DropTail and RED.	39
3.6	Throughput of the three competing flows under the FairShare scheme.	40
3.7	Throughput of flow 0 bottlenecked at a remote node.	41
4.1	The <code>find_global_optimal_rates</code> procedure.	46
4.2	Illustration of the <code>distributed_global_optimal_rates</code> procedure.	51
4.3	Topologies in, (a) scenario 1, (b) scenario 2.	57
4.4	Experiment 2, (a) topology and (b) goodput.	58
4.5	Experiment 3 (a) topology and (b) goodput.	60
4.6	Experiment 4 (a) topology and (b) goodput.	61
5.1	Pareto Distribution fit on Trace 1 for various thresholds.	65
5.2	The TCP Handshake Exchanges	67
5.3	RTT Spectrum: Y axis, Frequency, X axis, $\log_{10}(\text{RTT}/10^{-5})$	77
5.4	DAS-BV <code>upon_packet_arrival()</code>	78
5.5	DAS-ED <code>upon_packet_arrival()</code>	78
5.6	DAS-EL <code>upon_packet_arrival()</code>	79
6.1	The goodput of long and short-lived flows under the DAS schemes (RTT_Ratio=2).	85
6.2	The goodput of long and short-lived flows under the DAS schemes RTT_Ratio=10.	86
6.3	The goodput of long and short-lived flows for DropTail and RED.	87
6.4	Short-lived flow load on (a) fairness of long-lived flows, and (b) the response time of short-lived flows, with or without classification schemes, using DropTail.	89

6.5	Short-lived flow load on response time for fixed vs. dynamic (DAS) bandwidth allocation.	90
6.6	Short-lived flow load on fairness among long-lived flows for (a) lifetime-based, (b) RTT-based, and (c) combined, or no classification scheme (MIX-DT, MIX-RED).	91
6.7	Short-lived flow load on the average response time of short-lived flows for (a) lifetime-based, (b) RTT-based, and, (c) combined, or no classification scheme (MIX-DT, MIX-RED).	92
6.8	Fairness of long-lived flows for (a) lifetime classification, (b) RTT classification, and, (c) combined, or no classification scheme (MIX-DT, MIX-RED).	95
7.1	Fairness Under Different RTTs, (a) XCP, (b) FairShare.	104
7.2	Performance of XCP In a Dynamic Environment, (a), 100 PKT/s , (b), 200PKT/s, (c), 1000PKT/s.	105
7.3	Loss Probability vs. Maximum Burst Size	107

List of Tables

2.1	Experimental Configuration.	21
2.2	Throughput Ratio without Queries Time.	22
2.3	The Throughput Ratio with Queries Time.	23
2.4	Validation of Equilibrium Throughput: Individual Loss Probability.	25
2.5	Validation of Equilibrium Throughput: Common Loss Probability.	25
2.6	Validation of Equilibrium Throughput: Individual Loss Probability.	27
2.7	Validation of Equilibrium Throughput: Common Loss Probability.	28
4.1	The <code>find_global_optimal_rates</code> procedure.	48
4.2	The <code>distributed_global_optimal_rates</code> procedure.	50
4.3	The paths of flows in Experiment 2.	57
4.4	Per-flow bandwidth share in Experiment 2.	59
4.5	Measured vs. predicted bandwidth distribution.	62
5.1	Pareto distribution parameters (α, β) capturing the empirical distribution.	65
6.1	Short lived flow response times in seconds (<code>RTT_Ratio=2</code>).	83
6.2	Short lived flow response times in seconds (<code>RTT_Ratio=10</code>).	83

Chapter 1

Introduction and Background

Recent statistical studies of Internet traffic show that the Transmission Control Protocol (TCP) is still the dominant transport protocol used by many popular and important applications [1, 2, 3]. These applications are e-mail, news, remote login, file transfer, some streaming audio and video protocols and, most importantly, the World Wide Web. Moreover, the studies show that the TCP flows carry the majority of traffic in the Internet. It is reported that 95% of bytes and 90% of packets are controlled by TCP [1, 2, 3]. The predominance of TCP determines that efficient use of the network resources depends on the dynamics of TCP and its particular performance characteristics. To solve the problems brought by the increasing volume of new applications, it is an urgent and challenging task to study and improve the performance of TCP to provide QoS guarantees.

1.1 A Brief Introduction to TCP

1.1.1 The Original TCP

TCP was proposed to provide highly reliable host-to-host connections (in this thesis, we use connections and flows interchangeable) between hosts in the Internet [4]. The basic transmission unit of TCP is a packet (segment). Packets are obtained by segmenting a continuous stream of bytes. TCP is assumed to operate on a potentially unreliable packet-switching communication network. The data delivered in the underlying network can be corrupted, lost, duplicated, or delivered out of order. Thus, in order to achieve reliability, TCP assigns a sequence number to each byte transmitted and requires a positive cumulative acknowledgment (ACK) from the receiver's side. If the ACK is not received within a timeout

interval, the data is re-transmitted. On the other hand, the receiver side uses sequence numbers to correct the problems caused by out-of-order packets and to eliminate duplicates. Corrupted data are detected at the receiver by checking the checksum, appended at the end of each packet by the data link protocol. Because the IP and TCP checksums are not particularly strong, corrupted packets are discarded permanently at the data link layer. In the following, for the sake of simplicity and without loss of generality, we assume sequence numbers and ACKs are applied at the level of packets, rather than bytes. This approximation is generally accepted by numerous research groups and simulation platforms [5, 6] and causes no noteworthy inaccuracies. Flow control is another major function of TCP. A TCP sender maintains a window, which represents the allowed number of bytes that the sender may send before receiving further permission (acknowledgments convey such permission). The objective of the window is to match the transmitting rate at the sender side with the reception and processing rate at the receiver's side. TCP is a connection-oriented protocol. When two hosts wish to communicate, they must first establish a connection by initializing the relevant information on each side via handshake scheme. The connection is explicitly closed after the termination of data transfer.

1.1.2 The Presence of Congestion

The growth of the Internet in the mid 1980s caused severe problems of congestion. In the landmark paper by Van Jacobson [5], it is reported that 10% of packets arriving at Internet gateways were dropped. In October of 1986, the data throughput from Lawrence Berkley Laboratory (LBL) to UC Berkley dropped from 32Kbps to 40 bps. The problem was rooted in the flawed assumptions behind the interpretation of TCP's timeouts. Since the number of hosts hooked on the Internet was relatively small in the 70s, only the flow control mechanism was implemented in the original design of TCP. The original design was only concerned about matching the rates at the end hosts, which was addressed by flow control. The bandwidth in the intermediate links was assumed sufficient and end-points did not know about the available bandwidth. That is, once the end hosts set up the agreement on the rates via flow control, the intermediate links always delivered the transmitted data through at that rate. Because of the sufficiency of bandwidth in intermediate links, the packets were rarely dropped due to the overflow of buffers in gateways. Thus, when the sender detects any timeout, it interprets

that timeout (really, a missing ACK) as a signal of corrupted packets. Consequently, the sender re-transmits the supposed corrupted packet. After the rapid increase of hosts in the Internet, the traffic in the Internet experienced explosive growth. The Internet links were no longer sufficient in terms of bandwidth. Packets that cannot be delivered at the intermediate link right away are stored at the gateway buffer. Therefore, packets might be discarded if congestion persists. The congestion problem was beyond the consideration of the original design of TCP. In particular, with the original design, the TCP senders had no means to correctly acquire information regarding congestion along the path and what fraction of the available bandwidth should each flow use. The TCP senders could only interpret the cause of the increased number of timeouts as corrupted packets. Consequently, TCP re-transmitted all of the timed out packets again. These retransmissions aggravated the congestion rather than solve it. Such a problem could lead to the congestion collapse of the entire network.

1.1.3 The Reaction to Congestion

The essence of congestion control in TCP comes from re-defining the assumption about packet losses. A study [5] showed that the packets lost due to bit errors are very rare ($\ll 1\%$). On the other hand, the packet losses due to congestion might be over 10%. As a result, it is natural to modify the assumption on the cause of packets loss to reflect this fact. *The new assumption about packet losses is that all packet losses indicate the presence of congestion along the path.* The lost packets due to congestion overshadow the packets corrupted. The new assumption actually reflected the improvement of underlying infrastructure and problems caused by increased numbers of users in the Internet. The other important part of the re-designed TCP was the method of detecting lost packets. It could use a simple design, in which timeout could be the only mechanism of detecting packets loss or packet errors. However, since congestion occurs very often and accumulates very rapidly, a faster way of detecting lost packets was needed. In the new design [7, 8], the sequence numbers in ACKs are utilized to quickly detect lost packets. The scheme is called Fast Retransmission [7, 8]. The essential idea of the Fast Retransmission mechanism is to infer packet losses from out-of-order packets received at the receiver side. When packets are not received in the order of their sequence numbers, it might imply two things: First, the packets with the smaller sequence number might choose a longer path. Apparently, these packets may need longer

time to arrive at the receiver host. They may arrive at the receiver side even after the packets with larger sequence numbers that are sent later. A datagram packet-switched network allows this situation. Second, one or more packets could be dropped along the path. In either case, the sender will notice the occurrence of out-of-order by the sequence numbers of the ACKs. In fact, the senders cannot differentiate for sure between lost packets and just pure unlucky out-of-order packets. One thing that the sender knows is that if an unlucky packet causes an out-of-order event, the reordering will take place very soon. Otherwise, the sender will receive out-of-order packets persistently. Such knowledge is used in the design of Fast Retransmission. In the proposed mechanism [7, 8], the threshold number of duplicate ACKs of differentiating lost packets from out-of-order packets is 3, a somehow arbitrary value. It means that if 3 duplicate ACKs are received consecutively, then the sender should infer that a packet has been lost, with a high level of confidence. Another relevant claim is that the queue length at intermediate gateways increases exponentially when congestion occurs [5]. In order to respond to such rapid growth of congestion, the traffic should react by throttling itself very quickly. Specifically, the sender reduces its window size by a multiplicative factor, for example 50% [5].

1.1.4 The Conservative Start

In the original TCP design, the end hosts negotiate the desirable rate (advertised window) for both hosts, and start transmission at that speed, without considering the condition at the intermediate links. This (old version) design will lead to congestion at the intermediate links very easily when the total demands of all hosts exceed the link capacity. To avoid transiting into congestion state constantly, the re-designed TCP senders are more conservative. Using the Slow Start algorithm, TCP senders start transmission with the minimum possible rate and increase the rate gradually; thus, “pushing” the load to higher values until a loss occurs due to the flow attempting to use more bandwidth than available. Initially, the TCP sender sets the window value to one Maximum Segment Size (MSS). It increases the size of the window by one for every ACK it receives (exponential increase stage). Thus, the sending rate, that is, the window size over RTT, will quickly reach the available link bandwidth, that is the fraction of bandwidth apportioned at the moment. The real design of TCP is a little bit more complex. The window size controlling the sending rate is the minimum of

the advertised window negotiated by end hosts and the congestion window regulated by the congestion control algorithm. TCP has another way of increasing the congestion window that is called Linear Increase. That is, the congestion window size is increased by one for every round trip time (RTT). Linear increase is much slower than the exponential increase which doubles the window size every RTT. TCP assumes that the available bandwidth for this flow is close to the bandwidth it experienced recently. Thus, the Linear Increase algorithm produces a mild window increase when the current sending rate is close to the (assumed) available bandwidth.

1.1.5 The Recovery from Congestion

As the description in the above section points out, we know that the sender will reduce its congestion window by a factor, 50%, on congestion. The most conservative way of reduction is to reduce the congestion window to 1 and initiate the slow start algorithm (exponential increase or double the congestion window size for every RTT) from the beginning. The other, less conservative, way of reduction is to reduce the congestion window by 50% and start the congestion avoidance algorithm (Linear Increase, or increase window size by one MSS for every RTT) directly, which is called the Fast Recovery Algorithm [7]. In the first case, the recovery starts from a smaller window size but has a faster increase rate. While in the second case, it recovers from a larger congestion window but with milder increase. In most cases, the second case is faster, but it might be poor when the recovery starts from a severely congested state. To address such problem, Fast Recovery was proposed [7]. Fast Recovery and Fast Retransmission are usually combined together. The essential idea of Fast Recovery is injecting more ACKs in the transmission pipeline than those provided by linear increase. With these extra ACKs, the sender will increase its congestion window faster than with regular Linear Increase. Such a mechanism is particularly helpful in the case of a small congestion window. Specifically, a complicated but important mechanism called the “inflation of congestion window” is implemented. When congestion is detected at the sender via the Fast Retransmission mechanism (via 3-duplicate ACKs), the congestion window is reduced by half. This value is stored in a parameter called *ssthresh*. The lost packet is then retransmitted. Before the retransmitted packet is acknowledged, the congestion window is inflated by incorporating the outstanding packets cached at the receiver. At first, the

inflated congestion window value is $ssthresh+3$, and then it increases on every arrival of duplicate ACKs. The inflated congestion window, that increases exponentially, controls the sending rate. That is, in the process of retransmitting and confirming the lost packet, the congestion window increases exponentially. This inflation terminates when the ACK of the retransmitted packet returns to the sender; the congestion window size resumes to the value stored in $ssthresh$. So, by the end of the loss recovery, the links have conveyed some extra ACKs. Consequently, the congestion window will leap the first few rounds of small congestion window size very quickly, even if the congestion avoidance algorithm (Linear Increase) is active. The combination of linear increase and multiplicative decrease, are called the AIMD scheme and is known to lead to an efficient and fair operating point [10] when the delay of signaling is assumed to be zero. However, TCP's operation resembles AIMD without being exactly the same.

1.1.6 TCP Variants

The first TCP implemented with congestion control mechanism is 4.3 BSD Tahoe [7]. It includes slow start, congestion avoidance and fast retransmission. Specifically, the TCP increases its congestion window exponentially when the congestion window is smaller than $ssthresh$, while it increases linearly when the congestion window size is larger than $ssthresh$. In case of congestion, the congestion window is set to 1 immediately and set $ssthresh$ to the half of current congestion window size. TCP Tahoe [7] can be described by following equations (1.1, 1.2).

$$cwnd(t + \tau) = \begin{cases} cwnd(t) + 1, & \text{Slow Start Phase :} \\ & \text{if } cwnd(t) < ssthresh(t); \\ cwnd(t) + \frac{1}{cwnd(t)}, & \text{Congestion Avoidance Phase :} \\ & \text{if } cwnd(t) \geq ssthresh(t); \end{cases} \quad (1.1)$$

where $ssthresh(t)$ is the threshold value at which TCP changes from the slow start phase to the congestion avoidance phase. In (Equation 1.1), time evolves whenever an acknowledgment is received. In addition, when a packet loss is detected through a timeout, $cwnd(t)$ and $ssthresh(t)$ are updated as follows

$$cwnd(t + \tau) = 1; \quad ssth(t + \tau) = \frac{cwnd(t)}{2}. \quad (1.2)$$

The later improved version is called 4.3 BSD Reno (Equation 1.3). In TCP Reno, TCP responds to packet loss by reducing its congestion window by half and setting *ssthresh* to that value. Subsequently, the TCP sender starts the linear increase (congestion avoidance control) algorithm. The Slow Start algorithm is initiated in special cases in TCP Reno, such as at the very beginning of the TCP connection and after timeouts. Therefore, TCP Reno consists of congestion avoidance, fast retransmission, fast recovery and slow start algorithms. The detail of Tahoe and Reno are illustrated in Figure 1.1.

$$\begin{cases} cwnd(t + \tau) = cwnd(t) + \frac{1}{cwnd(t)}, & \text{if no congestion indication} \\ cwnd(t + \tau) = \frac{cwnd(t)}{2}; ssth(t + \tau) = \frac{cwnd(t)}{2}. & \text{if congestion indication} \end{cases} \quad (1.3)$$

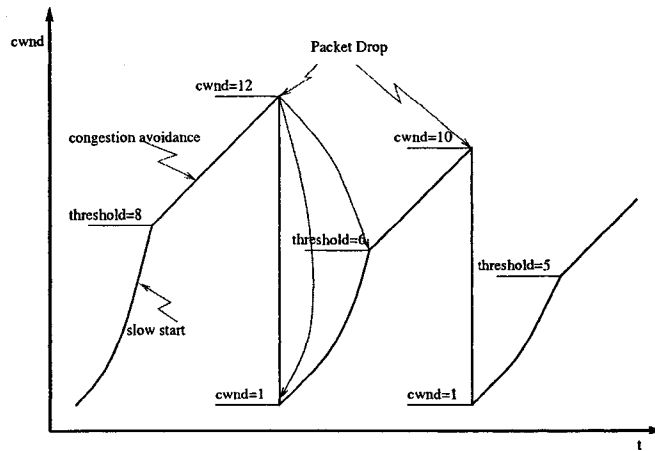
The underlying assumption in TCP Tahoe and TCP Reno is that, most of the time, only one packet could be lost at most per round trip time. Thus, only one packet is expected to be retransmitted when a fast retransmission is triggered. However, this assumption does not always hold. In TCP SACK [9], more precise information on packets received is sent back to the sender. With more accurate information on packets missing, the TCP SACK sender will arrange retransmissions of all lost packets immediately even if the number of lost packets is more than 1.

1.1.7 RTT Estimation in TCP

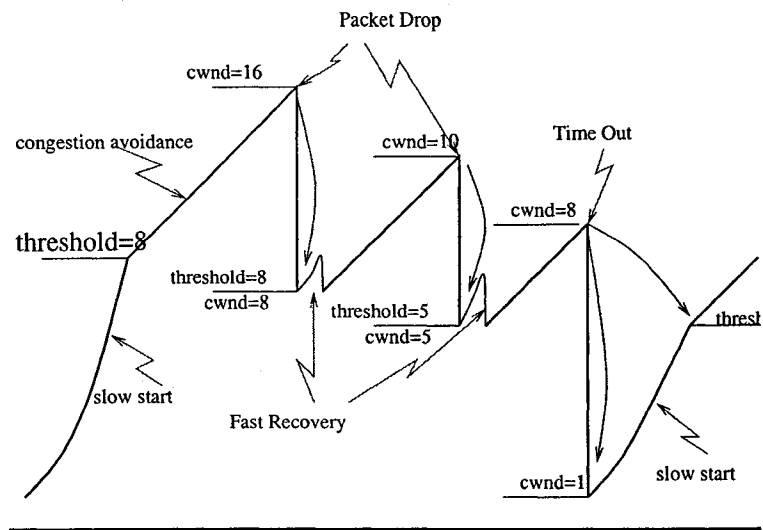
The sending rate of a TCP session is the ratio of its congestion window over its round trip time. TCP senders regulate their sending rate by adjusting the window size. RTT estimation is an essential part for the effectiveness of TCP control algorithms. In the original TCP [5], a low pass filter is used in estimating the average RTT, where R stands for the estimation of RTT and M stands for the latest measurement of RTT:

$$R \leftarrow (1 - \alpha) * R + \alpha * M. \quad (1.4)$$

Since the suggested value of alpha is 0.1, this algorithm filters out the instantaneous deviations and derives the long-term value of R . The timeout value (RTO) is set as $RTO \leftarrow 2 * R$ [4]. In order to account for the increased congestion behaviour, [5] suggested the following modification of the estimation method. Firstly, the alpha is modified to 1/8 since this value is close to the original value and more convenient to implement by a shift operation.



(a) Tahoe



(b) Reno

Figure 1.1: Illustration of TCP: (a) Tahoe and, (b) Reno.

Secondly, the modified TCP maintains two estimators: the average of RTT and the variance of RTT. Thirdly, the estimation of RTO, Retransmission TimeOut, accounts for the variance of RTT, namely

$$RTO \leftarrow E[RTT] + 4 * var(RTT). \quad (1.5)$$

1.2 Definitions

1.2.1 Max-min Fairness

Max-min allocation of bandwidth on a link can be described as an iterative process. Intuitively, the max-min fairness allocation means that users with small demands get all they want and users with larger demands will eventually split the leftover. The formal definition is: 1. Resources are allocated on the order of increasing demands 2. No user is allocated resources more than it demands 3. Large users with unsatisfied demands split the remaining resources For example, we have 10 resources and 4 users with demands of 1, 2, 4, 6, respectively. The allocation process is described as follows

	Flow 1	Flow 2	Flow 3	Flow 4	Total
Demand	(1)	(2)	(4)	(6)	(1+2+4+6 > 10)
Iteration 1	1	1	1	1	4 < 10
Iteration 2	1	2	2	2	7 < 10
Iteration 3	1	2	4	4	11 > 10
Final allocation	1	2	3.5	3.5	10=10

The MAXMIN fairness can be modified to accommodate the screwed demands of bandwidth. The demands for TCP flows with high bandwidth demands can be assigned a higher fair share. For example, the demand of flow 1 and 2 is 1:2. The fair share can also assigned to be 1:2. So, the bandwidth allocated is 1:2 when the both flows cannot be satisfied.

Apart from per-link fairness, the larger issue at stake is whether we can determine the fair allocation of bandwidth globally, for all the flows present in the network. Whether such an approach is technically feasible depends, apparently, on the type of fairness. For example, max-min fairness is difficult to achieve without global information. Approximations to global max-min fairness have been proposed, e.g., in [37], but the essence of such schemes is that they depend on the ability to determine the local fairness allocation at each link. Therefore, even though we make no claim for direct development of a globally fair allocation scheme, we cover the mechanisms necessary for local fairness, with the understanding that it

is the building block of a global fairness algorithm (in the sense used by Charny [47]) if such an objective is desired. An alternative would be the totally distributed implementation of a fairness scheme. Using end-to-end mechanisms [36] it was shown that it is possible to achieve $(p, 1)$ -proportional fairness, but max-min fairness has so far been unimplementable on an end-to-end basis. Moreover, even the implementable $(p, 1)$ -proportional fairness is unattractive because it requires a different end-to-end congestion control protocol, which is neither TCP nor any other legacy protocol. For this reason, and also because proportional fairness tends to victimize flows that span over a longer path, we consider max-min fairness only, and open the potential for mechanisms at the routers to assist in achieving the fair allocation of bandwidth among flows. This is where Active Queue Management (AQM) schemes become useful. Essentially, the thesis proposes AQM schemes that, when used throughout the network, provide globally fair allocation of bandwidth. This approach is distinctly different from other proposals where AQM schemes are studied on a single isolated link but never in a network-wide setting. Indeed, it is unknown whether applying the particular AQM on all nodes can lead the network to a specific operating point. The existence of such an operating point, as well any features that such a point might exhibit, are unknown. The inverse problem, i.e., desiring a particular operating point for the network and determining an AQM scheme in order to achieve it, is also rarely discussed. When a particular operating point is desired, this is caused by the reason that it exhibits a particular property, such as fairness. For example, an operator may have an incentive to drive the operating point of an entire network to be such that the flows of customers in its own network obtain the lion's share of available bandwidth. Clearly, the coexistence of multiple operators will be much better facilitated, if it is known that by using a particular AQM scheme fairness will be achieved.

1.2.2 Goodput and Throughput

In this thesis, the two terms, goodput and throughput, are used. The throughput is the sending rate, including both the successful transmission and retransmission:

$$\textit{Throughput} = \# \textit{Packets Sent} / \textit{Time}.$$

The goodput is the rate of packets received successfully.

$$\textit{Goodput} = \# \textit{Packets Received Successfully} / \textit{Time}.$$

Since the congestion is frequently encountered nowadays, we have to distinguish between the two definitions. In some of our experiments, we assume that the TCP sources have infinite data to send and are always regulated by the TCP congestion avoidance algorithm. In such environments, TCP flows' demands for bandwidth are neither their throughput nor their goodput. Their demands for bandwidth are infinite. Thus, such TCP flows are called "greedy" in the sense that they will attempt to consume whatever resources one provided to them.

1.2.3 Loss (Wired vs. Wireless Environment)

In our experiments, loss is only caused by congestion in the network. This assumption holds in the wired environment. However, in wireless environments, both unreliable transmission media and congestion can cause packets loss. Therefore, the extra packet losses caused by the physical layer in wireless environment will affect the overall behaviour of FairShare. For example, due to excessive packet loss in the wireless environment, TCP sources reduce its sending rate to a very small value. Thus, from the edge router point of view, the FairShare policy interprets such TCP sources as those with limited demands that often require no regulation. In short, FairShare policy does not interact with the wireless losses any different than how the end-to-end TCP flow does.

1.3 The Basic Problems of TCP

TCP is a transmission protocol that needs little or no support from the core of the network. This is considered a merit because TCP does not require particular complexity in the intermediate gateways. The only effort needed is deploying TCP at end hosts. As a result, TCP was widely deployed and dominates today's Internet. It nevertheless possesses some major inherent drawbacks affecting its performance in the environment of increased traffic volume. The first drawback of TCP is imprecision of congestion signalling. TCP has no explicit congestion signal defined. It infers congestion by packet loss: that is, it assumes all packet losses are caused by congestion. This assumption might not always hold, especially in wireless environments where the noise/signal ratio is high resulting in dropped packets due to bit errors. Incorrect interpretation of corrupted packets will lead to low utilization. The second drawback is the imprecision of the congestion signals. Packet losses occur when

buffers overflow. Only the TCP flows losing some packets recognize the congestion and respond by throttling their sending rate. The current mechanism does not guarantee that all TCP flows are notified when congestion occurs. As a result, some TCP flows will take advantage by consuming capacity released by other flows. The third drawback of TCP is the slow processing speed in detecting congestion. As the definition of Fast Retransmission, TCP infers packet loss by receiving 3 duplicate ACKs and speculates congestion. Again, the current mechanism reuses the information for the purpose of inferring congestion. The simplicity is compensated by slower inference time; TCP senders use 3 pieces of indirect information that could, in principle, be replaced by one single explicit piece of information. The fourth drawback is the imprecision of information about the available bandwidth. The congestion signal for TCP is a binary variable, that provides no information about the amount of available bandwidth for the TCP flows experiencing congestions. According to the equilibrium used in [5], the current mechanism assumes that the available bandwidth for a congested TCP flow converges to a particular value in the long term. TCP speculates that the available bandwidth of the near future is very close to the maximum bandwidth it received in the recent past. Thus, the congestion window increases cautiously (linearly) to the speculated size by $1/cwnd$ per RTT. This speculation might not be accurate when the system is extremely dynamic, e.g., when new connections are constantly being set up and terminated. The fifth drawback is the asymmetric response speed to the congestion. Each TCP is essentially a closed-loop control system. If TCP flows receive congestion signals, the congestion signals are transmitted at different speeds. That is, with the difference in their RTTs, TCP flows react to the congestion with different speed, even if they receive the congestion signals at the same time instant. This asymmetric behavior, combined with the imprecision, raises a serious fairness problem that will be addressed extensively in this thesis.

1.3.1 Phase Effect and RED

In [11], the Phase Effect was reported. Essentially, the traffic of a particular long TCP flow is periodic. That is, a TCP sender sends a bulk of packets with a period of RTT, although the sender adjusts the size of the bulk every RTT, and RTT is a random variable rather than a constant. The study [11] shows that when two TCP flows of different RTTs compete at a congested link, one TCP flow might lose packets consistently when congestion occurs and

end up with a relatively lower throughput. This phenomenon is called “Phase Effect” [11]. To solve the phase effect, [11] suggested to add randomization in the queue management policies. Random Early Detection (RED) was proposed as the solution of phase effect and a congestion control mechanism at gateways [12]. RED maintains two variables: the average queue size \bar{Q} and packet dropping probability p . The average queue size is obtained by a low pass filter, which is described by

$$\bar{Q} \leftarrow (1 - wq) * \bar{Q} + wq * Q'. \quad (1.6)$$

where \bar{Q} stands for the long term average queue size, Q' stands for instantaneous queue size, and wq is an operation parameter. This average queue size, \bar{Q} , is an indicator of congestion. Larger \bar{Q} , suggests severe congestion, and thus, needs more drastic rate throttling. Two preset threshold parameters: *minthresh* and *maxthresh*, are used to define the magnitude of congestion. With \bar{Q} smaller than *minthresh*, the congestion is considered non-existent. When \bar{Q} is larger than *maxthresh*, the congestion is severe and every packet is lost with a constant loss probability P_{max} (Figure 1.2). In the case where \bar{Q} is between the two thresholds, the congestion is considered medium. The realistic implementation of RED also accounts for the packets dropped in the recent past, which is described in the second equation in (Equation 1.7). Packet dropping is still interpreted as congestion signalling. According to the degree of congestion, RED controls the “strength” of signalling by adjusting the frequency of packet dropping. Technically, the regulation is implemented by adjusting the dropping probability associated with each arriving packet. Larger dropping probability suggests more frequent packet drops. Specifically, when the congestion is not present, the dropping probability is 0. When the congestion is medium, the dropping probability is proportional to the difference between the average Q size and minthresh. When the congestion is severe, the dropping probability is set to its maximum value.

$$\begin{cases} p' = P_{max} \times \frac{\bar{Q} - \text{minthresh}}{\text{maxthresh} - \text{minthresh}} \\ p = \frac{p'}{1 - \text{count} \times p'} \end{cases} \quad (1.7)$$

RED is a policy with useful features. Since congestion can develop very fast and is disruptive, detecting congestion earlier would be very appreciated for proactive operation. In addition, the randomization in selecting victims distributes packet losses across all flows.

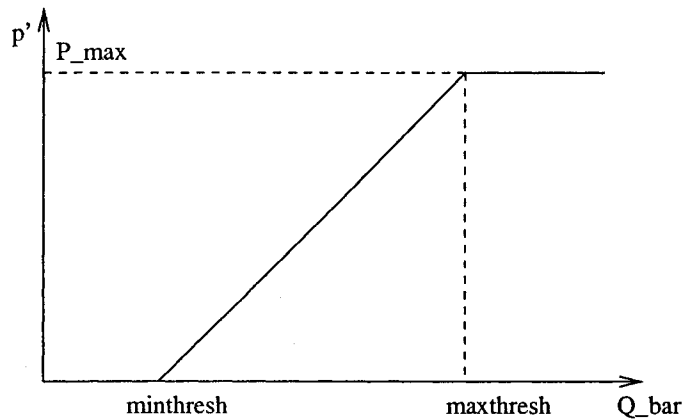


Figure 1.2: RED Operation.

In this way, TCP flows cannot easily escape from packet losses. Each packet has the same probability of being dropped regardless of the flows it belongs to. Lastly, the randomization also improves the fairness between flows. Flows with more packets stored in current queue are more likely to be victimized. It is natural to expect that the flows consuming more bandwidth are very likely the ones with more packets in the buffer.

1.3.2 Fairness Over Flows with Different RTTs

The fairness problem has been noticed and extensively studied for a long period of time [11, 12]. Specifically, TCP flows with relatively longer RTT received less bandwidth. The interaction between TCP flows of different RTTs is an extremely difficult problem that has not been solved yet analytically. We can provide only the qualitative argument. The regulation of TCP window is operated over the interval of its RTT. Thus, TCP flows with larger RTTs usually need longer time to respond and recover from congestion. However, router link scheduling policies like DropTail are work-conserving, that is, the link will never be idle when packets wait for service. As a result, the TCP flows with small RTTs could finish several rounds of transmission before the packets from TCP flows with longer RTTs arrive at the bottleneck. Consequently, the increase of congestion window of TCP flows with

smaller RTTs exceeds the one of TCP flows with larger RTTs.

Flow 1:	8	4	5	6	7	8	9	10
Flow 2:	8							4

We illustrate our argument with a simple example. In this example, we have two flows competing at a common bottleneck. The RTT of Flow 2 is 6 times of Flow 1. At the time instant 0, a congestion signal is noticed by both flows when their congestion window is 8. Both flows respond to the congestion via reducing their congestion window to 4. However, before the data of flow 2 arrives at the bottleneck, flow 1 has already made 10 successful transmissions and increased its congestion window to 10.

Drop Packet(s) from	Ratio of Next Congestion Window Size
Flow 1	5:4
Flow 1 & 2	5:2
Flow 2	11:2

The DropTail policy could possibly worsen the unfairness. There are 3 possible scenarios when at most 1 packet is dropped from each flow in case of congestion. Ideally, a packet from Flow 1, the one that consumes most of the bandwidth, should be dropped. However, if the bandwidth is already close to congestion when packets from Flow 2 arrive, those Flow 2 packets experiencing long journey are the vulnerable candidates of dropping. With the presence of accumulating queue due to congestion, there might be some packets accumulated in the buffer already. The Flow 2 packets will probably be placed at the end of buffer and be dropped. Consequently, the scenario 3 could happen very likely in case of incipient congestion. We do not claim that our analysis is general, but we do get the insight how DropTail can aggravate the unfairness. As RED introduces a randomization approach in selecting dropped packets, the next congestion window is easier to predict probabilistically. Specifically, since each packet is dropped with an identical probability and independently, for Flow 1, the probability of losing one packet is $\frac{10}{10+4}$ and the probability of not losing one packet is $\frac{4}{10+4}$. Thus, the expected window size of next time instant is the sum of the expected window size of both cases. The expected size of the next congestion window size of both flows are derived as follows:

	Expected size of next congestion window
Flow 1:	$11 * \frac{4}{14} + 5 * \frac{10}{14} = 6.71$
Flow 2:	$5 * \frac{10}{14} + 2 * \frac{4}{14} = 4.15$

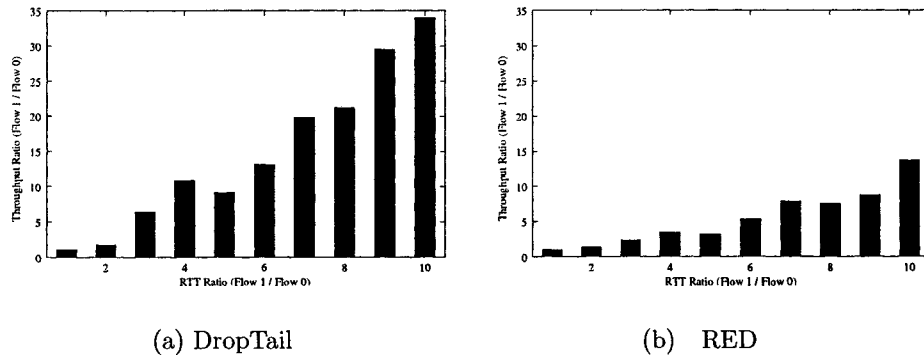


Figure 1.3: Unfairness due to RTT difference: (a) DropTail and (b) RED.

From the above example, we can see that although RED might not be able to make an optimum decision in terms of fairness, it can approach the optimum and avoid the most unfavorable scenario.

Figure 1.3 demonstrates the ratio of the throughput received by two flows, as a function of the ratio of their corresponding propagation delay RTTs. The two flows share the same bottleneck link with a buffer space of 24 packets and a link speed of 100 packets per second (fixed size packets) but the ratio of the RTTs of the two flows spans an order of magnitude. Specifically, the RTT of flow 1 is set to 100 msec (approximating the RTT of a flow within the North American continent), while that of flow 0 ranges from 100 msec to 1sec (representing the range from intra-continental to inter-continental traffic). As it can be seen, DropTail results in a ratio up to 34:1 between the throughput achieved by the two flows (the lesser throughput received by flow 0) while RED produces¹ a ratio up to 14:1. This is precisely what we wish to avoid. The ideal ratio in this example ought to be 1:1 regardless of the RTT of the two flows. From the figures (Figure 1.3) we can see that the fairness is improved by using RED policy at the gateways. However, we can see that the fairness is still far from ideal. This result is caused by inherent design limitations of RED. Firstly, like DropTail, RED is work-conserving. RED cannot leave the link idle while packets from flows are waiting in the queue. Thus, TCP flows with shorter RTT still have more opportunity of consuming more bandwidth; they consume all bandwidth if packets from long RTT flows have not arrived, and compete for bandwidth if they could. To achieve the same rate of TCP flows with shorter RTT, TCP flows with longer RTTs need a larger congestion window to balance the

¹RED parameters: $max_p = 0.02$, $min_{th} = 5$, $max_{th} = 15$, $w_q = 0.002$.

relatively longer RTTs. In RED, it is not easy for TCP flows with longer RTTs to reach a relatively larger congestion window. Secondly, the dropped packets are selected randomly. Although the principle is reasonable, the probability of making an incorrect decision is still high. Since the control decision will affect the TCP source behavior in the future, the price of making incorrect dropping decision is severe in terms of fairness. For the decision that has a long-term impact, the accuracy of decision is more valuable. Thirdly, the assumption, that the flows consuming more bandwidth store more packets in the buffer, holds for the long run only. However, the dropping decision is made over a snapshot of the system state, and the distribution of packets at a particular time instant. The flows consuming more bandwidth in the long run might not store more packets in the buffer at any time instant. Fourthly, the reaction to the congestion is proportional to the difference between the average queue size and minthresh. This design is acceptable when the queue is not so dynamic. However, if the average queue size deviates from the minthresh very quickly, RED might not be fast enough since its reaction is not considering the rate of change of the queue.

1.3.3 Long vs. Short TCP Flows

Statistical studies on real life traffic [13] also show a wide variety in terms of TCP flow lifetime distribution. Using a definition [13] of short-lived flows as those lasting shorter than 2 seconds and long-lived flows as those lasting longer than 15 minutes, 45% of flows are short-lived and a significant fraction of flows are long-lived (1.5%) that carry 50% - 60% of bytes. The measurement results in [13] are important in understanding the behavior of TCP flows. Essentially, TCP congestion control is designed for long-lived flows, where TCP sources could adjust their sending rate with the information inferred or received via feedback. With TCP flows lasting a very short amount of time, the senders might not benefit from the congestion control mechanism. It is reported that short-lived TCP flows are at a disadvantage when they compete against long-lived TCP flows [14, 15, 16]. Firstly, TCP flows increase its rate conservatively from the minimum possible value. Thus, short TCP flows are more likely to experience slow-start increase for their whole lifetime. Secondly, short TCP flows usually have small congestion windows. Thus, the packet loss might not be detected by 3-duplicated ACK due to the overly small congestion windows. Time out, which is much slower mechanism, might be used for packet loss detection. Thirdly, TCP flows use

their RTT estimation for time out value. A conservative large initial value for timeout is utilized before any information available for estimation, via feedback, is made available at the sender.

1.4 Thesis Objective and Outline

The objective of our research is to design an efficient and simple policy that could achieve fairness between competing TCP flows based on rigorously studied mechanisms. The extension of this is the ability to achieve global fairness as well. Our policy is expected to achieve fairness between TCP flows with different RTTs. One might debate if the absolute fairness is appropriate. We argue that we could extend our policy easily to the weighted fairness that could allow any allocation of weights. The point of our research is to regulate the bandwidth received by TCP flows regardless of their RTTs. Our policy is expected to also be efficient. The bandwidth is a precious resource in case of congestion. We argue that it is inappropriate to sacrifice high utilization for fairness allocation. Therefore, both high throughput and fairness are essential to be achieved. We also expect our policy to be simple and easy to be deployed. To this end, we avoid policies requiring modification of TCP headers. We argue that such policies need global cooperation and are unlikely to be deployed on Internet. Instead, we move the complexity to the periphery routers since core routers have significantly more flows and it is prohibitive expensive to keep states of thousands of flows on the fly. Our policy is based on a model of the behavior of TCP flows. We have noticed and verified models proposed recently. We indicate that models used for engineering TCP-based networks are not in presence. For example, they cannot predict the throughput of mixtures of TCP flows. The outline of this thesis is as follows. In Chapter 2, we validate the existing TCP throughput models through experiments. FairShare, the model-based policy which imposes fairness among TCP flows of different RTTs, is described in Chapter 3. We investigate the performance of FairShare in terms of global fairness in large and dynamic network in Chapter 4. In Chapter 5, we describe our statistical studies supporting classification schemes and several sub-schemes such as estimating RTT and identifying long-lived flows. We also describe DAS, a policy that can dynamically allocated bandwidth among classes and our study of classification schemes based on simple rules. The experimental evaluation of these classification schemes are presented in Chapter 6. We present and evaluate the recent

engineering efforts of fairness AQM and classification schemes in Chapter 7. Finally, Chapter 8 summarizes the contribution of this thesis and gives some points for the future work.

Chapter 2

TCP Throughput Models

The unfairness problem in TCP connections has been noticed and widely discussed soon after the congestion avoidance algorithm was implemented [18, 19, 20, 21, 24, 25]. Naturally, one way to approach the particular unfair behavior is to quantify it. The hope is that a rigorous qualification of the unfairness, will lead to better understanding of how severe, or under what parameter settings, the problem is most severe. Despite years of efforts, theoretical understanding of the existing Internet protocols, including TCP, is extremely challenging. It is difficult to select the most suitable framework of theory and making appropriate simplifications to make the problem mathematically tractable while retaining the essential features of the network system. Most of the theoretical studies of computer networks come from the legacy of studies on the telephone communication network. The prevailing belief in the early 1990s was that certain Markovian processes might be adequate to model the Internet traffic. However, the particular feedback stemming from the congestion avoidance algorithm was generally ignored. Although the congestion probing mechanism is fully accounted for in the studies of the individual TCP flow, the mechanism's impact on the equilibrium of *competing* TCP flows is not fully understood and has been oversimplified. In this document, we review the most representative models that were proposed to predict the throughput of combinations of TCP flows, or of individual TCP flows.

2.1 Simulation-based Models

A model based on extensive simulation experiments was proposed in [19]. In this model, TCP flows with different RTTs were studied. In the study, TCP flows possessed different

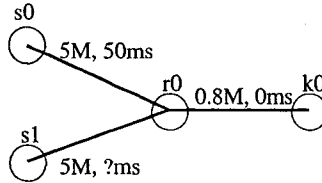


Figure 2.1: Experimental Topology.

RTTs by default. Based on simulations using ns-2 [6], [19] concluded that the ratio of TCP goodput between any two flows is bounded (Equation 2.1). The upper bound is the square of reciprocal of corresponding RTTs, while the lower bound is half of the upper bound. While the range of the ratio value is bounded, it is still wide enough to be useful in predicting the throughput within sufficient accuracy.

$$0.5 \times \left(\frac{RTT1}{RTT2}\right)^2 \leq \frac{Goodput2}{Goodput1} \leq \left(\frac{RTT1}{RTT2}\right)^2 \quad (2.1)$$

s0-r0	5 Mb, 50 ms
s1-r0	5 Mb, {50, 75, 100, 125, 150, 175, 200, 225, 250 ms}
r0-k0	0.8 Mb, 0 ms
r0-k0 Queue	DropTail, queue size=24
Packet Size	1Kb

Table 2.1: Experimental Configuration.

In [19], no evidence other than simulations conducted on ns-2 [6] were provided. Thus, in order to validate the inequality proposed, we have conducted simulations with the same simple topology (Figure 2.1) used in [19]. Two long-lived TCP Reno flows are initiated from s0 and s1, respectively. The sink is k0. By varying propagation delay of s1-r0, the two TCP flows capture flows with different RTT ratios. Simulations were run for 150 seconds of actual system activity. The trace of the first 30 seconds was removed in calculating goodput of TCP flows, in order to remove any transient effects. The simulation parameters are summarized

in Table 2.1. Each simulation was repeated 5 times to eliminate coincidence of individual experiment. In the table, P.Delay stands for Propagation Delay and Q.Delay stands for queueing delay.

RTT=P.Delay		
$0.5 * (\frac{RTT1}{RTT2})^2$	$\frac{Goodput2}{Goodput1}$	$(\frac{RTT1}{RTT2})^2$
0.50	1.00	1.00
0.22	1.63	0.44
0.12	0.71	0.25
0.08	0.28	0.16
0.06	0.24	0.11
0.04	0.19	0.08
0.03	0.22	0.06
0.02	0.36	0.05
0.02	0.13	0.04
RTT=P.Delay + Q.Delay		
$0.5 * (\frac{RTT1}{RTT2})^2$	$\frac{Throughput2}{Throughput1}$	$(\frac{RTT1}{RTT2})^2$
0.50	1.00	1.00
0.31	1.63	0.61
0.21	0.71	0.41
0.15	0.28	0.30
0.11	0.24	0.22
0.09	0.19	0.18
0.07	0.22	0.14
0.06	0.36	0.12
0.05	0.13	0.10

Table 2.2: Throughput Ratio without Queries Time.

The experiment results are summarized in Tables 2.2 and 2.3. We calculated the ratio between measured goodputs of the two TCP flows. Note that RTT is the sum of propagation and queueing delays. Since it was not clear in [19] whether the RTTs reported included the delay of queueing, in our experiments, RTTs are calculated both ways, that is, with or without average queueing delay included. As we can see in Table 2, most ratios of goodputs (bold font) exceed the bound suggested in [19]. In order to lessen the impact of the phase effect, we also introduce uniform random processing time for packets at the source. However, our experiment in (Table3) still gave us results that differ from those predicted by the model in [19].

RTT=P.Delay		
$0.5 * \left(\frac{RTT1}{RTT2}\right)^2$	$\frac{Throughput2}{Throughput1}$	$\left(\frac{RTT1}{RTT2}\right)^2$
0.50	1.03	1.00
0.22	0.66	0.44
0.12	0.48	0.25
0.08	0.30	0.16
0.06	0.38	0.11
0.04	0.18	0.08
0.03	0.22	0.06
0.02	0.26	0.05
0.02	0.15	0.04
RTT=P.Delay + Q.Delay		
$0.5 * \left(\frac{RTT1}{RTT2}\right)^2$	$\frac{Throughput2}{Throughput1}$	$\left(\frac{RTT1}{RTT2}\right)^2$
0.50	1.03	1.00
0.31	0.66	0.61
0.21	0.48	0.41
0.15	0.30	0.30
0.11	0.38	0.22
0.09	0.18	0.18
0.07	0.22	0.14
0.06	0.26	0.12
0.05	0.15	0.10

Table 2.3: The Throughput Ratio with Queries Time.

2.2 Individual Flow Equilibrium Throughput Models

A long-term equilibrium equation for TCP goodput was proposed in [20]. Unlike throughput, goodput only counts the packets successfully transmitted. In most scenarios, the packet-dropping rate is very small and below 1%. As a result, the difference between throughput and goodput is very small. In the proposed model [20], the goodput of TCP connections is formulated as a function of the loss probability it experiences. Similar equilibrium equations were proposed in other papers [21, 24, 27]. The given equation is widely cited and is the basis of some proposed engineering protocols [23, 28, 29]. This school of thought makes a core assumption: the loss process experienced by competing TCP flows is a stationary and independent stochastic process. For the sake of simplicity and without loss of generality, we

assume all packets are of constant unit size. The simplest model [20] suggested is:

$$Goodput = \frac{1}{RTT} \times \frac{\sqrt{3/2}}{\sqrt{p}}. \quad (2.2)$$

We also conducted simulations to validate the proposed equation in the same topology depicted in Figure 1 and with slightly different parameters. The bandwidth of s0-r0 is set to 8Mb instead of 5Mb. The s0-r0 queue size is 30 instead of 24. Table 2.4 summarized the measured goodput versus the predicted goodput by Equation 2.2. As in the previous section, the analytical predictions are made both with RTT equal to propagation delay and with RTT equal to the sum of propagation delay and average queueing delay. As we can see in Table 4, the predicted goodput by model [20] is not even close to the measured goodput. The most striking thing is that the sum of two predicted goodputs is larger than the capacity of the congested link. The assumption of the model [20] is that each packet is lost with a probability that is independent and follows the identical distribution between different packets, with the mean denoted by p . In the experiments, we use the post-experiment loss probabilities, that is, we got the average loss probability p from the trace of the whole simulation period. Since it is not clear in [20] how the loss probability p is calculated, we have two ways of calculating p . First, we measure the trace of individual flows and obtain the loss probability for each individual flow. Therefore, we have different loss probability p_i for each $flow_i$. The second way is calculating a common loss probability for all flows by using the trace of all flows. We argue the post-experiment loss probability is accurate. Based on the assumption in [20], each packet experiences an identical independent loss probability. As a result, the n packets can be formulated in independent experiments with identical “success” probability. The loss probability we measured is the mean of a sample of n independent identical experiments. When n is large, the sample mean \hat{p} is a good approximation of the real population p , namely:

$$\hat{p} = \frac{1}{n}(p_1 + p_2 + p_3 + \dots + p_n).$$

Since $p_1, p_2, p_3, \dots, p_n$ are independent

$$E[\hat{p}] = \frac{1}{n}(E[p_1] + E[p_2] + E[p_3] + \dots + E[p_n]).$$

Since $p_1, p_2, p_3, \dots, p_n$ are of identically distributed with the same mean p . Thus,

$$E[\hat{p}] = \frac{1}{n} \times np = p$$

Measured				Predicted			
				RTT=P.Delay		RTT=P.Delay+Q.Delay	
T1	T2	Loss1	Loss2	T1	T2	T1	T2
49.98	50.02	0.0048	0.0046	176.77	180.57	98.21	100.32
43.47	56.12	0.0080	0.0041	136.93	127.51	76.07	83.16
45.98	53.48	0.0074	0.0042	142.37	94.49	79.09	67.49
57.50	42.33	0.0049	0.0057	174.96	64.89	97.20	49.16
74.11	25.60	0.0029	0.0094	227.42	42.11	126.35	33.24
77.44	22.41	0.0027	0.0099	235.69	35.17	130.94	28.62
80.47	19.18	0.0025	0.0116	244.94	28.43	136.08	23.69
78.03	20.23	0.0028	0.0098	231.45	27.49	128.58	23.34
86.93	12.50	0.0025	0.0177	244.94	18.41	136.08	15.87

Table 2.4: Validation of Equilibrium Throughput: Individual Loss Probability.

Measured			Predicted			
			RTT=P.Delay		RTT=P.Delay+Q.Delay	
T1	T2	Loss	T1	T2	T1	T2
49.98	50.02	0.0047	176.77	180.57	98.21	100.32
43.47	56.12	0.0058	136.93	127.51	76.07	83.16
45.98	53.48	0.0057	142.37	94.49	79.09	67.49
57.50	42.33	0.0052	174.96	64.89	97.20	49.16
74.11	25.60	0.0046	227.42	42.11	126.35	33.24
77.44	22.41	0.0043	235.69	35.17	130.94	28.62
80.47	19.18	0.0043	244.94	28.43	136.08	23.69
78.03	20.23	0.0042	231.45	27.49	128.58	23.34
86.93	12.50	0.0044	244.94	18.41	136.08	15.87

Table 2.5: Validation of Equilibrium Throughput: Common Loss Probability.

By plugging loss probability we measured in model [20], we obtain the predicted goodput of each individual TCP flow (Table 2.4 and 2.5). As we can see, the discrepancy between the predicted [20] and measured goodput is significant whether we use pooled or individual loss probability. The failure of model in [20] is not a complete surprise. Actually, the goodput of two competing TCP flows should not be assumed independent to each other, like the model [20]. Assuming the congestion windows of two competing flows are W_1 and W_2 , the round trip time for flow 1 and 2 are RTT_1 and RTT_2 , respectively. We also assume that α fraction of resource is apportioned to flow 1, and $(1-\alpha)$ fraction of resources is apportioned to

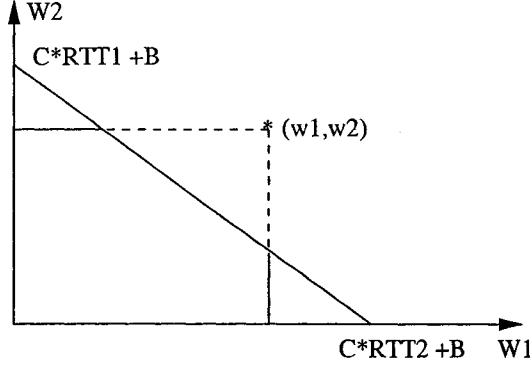


Figure 2.2: Joint and marginal PDF of W_1 and W_2 .

flow 2. C and B are link capacity and buffer size at the congested gateway. We have:

$$\begin{cases} W_1 \leq \alpha \times B + \alpha \times RTT_1 \times C. \\ W_2 \leq (1 - \alpha) \times B + (1 - \alpha) \times RTT_2 \times C. \end{cases} \quad (2.3)$$

Combing the two equations in (Equation 2.3), we have

$$\frac{W_1}{RTT_1 * C + B} + \frac{W_2}{RTT_2 * C + B} \leq 1$$

If we assume that W_1 and W_2 are two independent variables, we have

$$W_1 \times (RTT_2 * C + B) + W_2 \times (RTT_1 * C + B) \leq (RTT_1 * C + B) \times (RTT_2 * C + B)$$

The joint PDF of W_1 and W_2 is within a triangle in Figure 2.2.

For the sake of simplicity and without loss of generality, we assume the joint PDF is uniform. Thus, the joint PDF is the reciprocal of the triangle area, $\frac{2}{(RTT_1 * C + B)(RTT_2 * C + B)}$. The definition of independence between W_1 and W_2 requires for any w_1 and w_2 , the joint PDF $F_{W_1, W_2}(w_1, w_2) = F_{W_1}(w_1) \times F_{W_2}(w_2)$. That is, the joint PDF equals the product of marginal PDFs. To prove the dependence, we only need to give one example that the above equation does not hold. For a point $(w_1, w_2) = (\frac{3}{4} \times (RTT_1 * C + B), \frac{3}{4} \times (RTT_2 * C + B))$, the joint PDF is zero while the marginal PDFs are $\frac{1}{4} \times (RTT_2 * C + B)$ and $\frac{1}{4} \times (RTT_1 * C + B)$, respectively. Thus, W_1 and W_2 are not independent. To make our proof more general, we could relax the uniform assumption of joint PDF. Although we cannot compute the particular

marginal PDF easily, we know the two marginal PDFs at that point are greater than zero. Thus, our conclusion still holds. The intuitive interpretation of dependence between W_1 and W_2 is that the value of W_1 is affected by the given value of W_2 . That is, when W_2 is given a large number, W_1 is more likely to get a small value. Thus, the equation proposed in [20] which indicates that the W_1 has nothing to do with the other flows is inaccurate.

A more complicated model but with the same assumption on loss probability is given in [21]. The timeout behavior is included in the model. However, for the same reason this model is also not very accurate for predicting throughputs of combination of flows. We also validated the model (Equation 2.4) in [21] with experiments. The experiment parameters are the same as experiments above. Since the model in [21] includes the timeout, the model looks more complicated. However, due to small loss probability, the predicted throughputs of the two models are very close. From the experiment results (Tables 2.6 and 2.7), we find the discrepancy between the predicted goodput and measured goodput as well.

$$Goodput = \frac{1}{RTT \times \sqrt{\frac{2p}{3}} + RTO \times \min(1, 3 \times \sqrt{\frac{2p}{3}}) \times p \times (1 + 32p^2)} \quad (2.4)$$

Simulated				Predicted			
				RTT=P.Delay		RTT=P.Delay+Q.Delay	
T1	T2	Loss1	Loss2	T1	T2	T1	T2
49.98	50.02	0.0048	0.0046	173.04	176.91	97.04	99.18
43.47	56.12	0.0080	0.0041	132.16	125.20	74.58	82.17
45.98	53.48	0.0074	0.0042	137.78	92.74	77.66	66.59
57.50	42.33	0.0049	0.0057	171.19	63.26	96.02	48.22
74.11	25.60	0.0029	0.0094	224.50	40.39	125.44	32.17
77.44	22.41	0.0027	0.0099	232.87	33.66	130.06	27.62
80.47	19.18	0.0025	0.0116	242.22	27.01	135.24	22.70
78.03	20.23	0.0028	0.0098	228.57	26.33	127.69	22.50
86.93	12.50	0.0025	0.0177	242.22	17.04	135.24	14.84

Table 2.6: Validation of Equilibrium Throughput: Individual Loss Probability.

[21] was used in a comparison between the model (Equation 2.4) and empirical data. However, since the figures in [21] were plotted in log-scale, the agreement is not so close as suggested by visual inspection. Considering the way of the data were plotted in [21], the gap between the model and the empirical data could reach or even exceed 50%.

Measured			Predicted			
			RTT=P.Delay		RTT=P.Delay+Q.Delay	
T1	T2	Loss	T1	T2	T1	T2
49.98	50.02	0.0047	174.95	174.95	98.10	98.10
43.47	56.12	0.0058	156.72	104.48	88.06	68.75
45.98	53.48	0.0057	158.16	79.08	88.86	56.90
57.50	42.33	0.0052	165.95	66.38	93.14	50.57
74.11	25.60	0.0046	176.91	58.97	99.18	46.76
77.44	22.41	0.0027	183.22	52.35	102.65	42.76
80.47	19.18	0.0025	183.22	45.81	102.65	38.29
78.03	20.23	0.0028	185.47	41.21	103.90	35.09
86.93	12.50	0.0025	181.05	36.21	101.46	31.30

Table 2.7: Validation of Equilibrium Throughput: Common Loss Probability.

2.3 Other Approximation Models

Analytical models to capture the interaction of TCP with router policies have also been proposed in the past. Of particular interest is Flow-Proportional Queueing (FPQ) [38] with the objective of controlling TCP by varying the router’s queue length in proportion to the number of active TCP connections. A key observation made in [38] is that the used equations make sense if the TCP flows have similar round trip times. This is precisely where our work differs significantly in terms of assumptions and scope compared to [38], which attempts to solve the buffer dimensioning problem, not that of fairness. In particular, we are in agreement with the three points raised in the fourth section of [38]. To quote:

“Imposing a constant loss would cause three problems. First, if the number of connections is small, the loss rate may need to be reduced to allow window sizes large enough to keep the link busy. Second, the routers that are not bottlenecks should not impose any loss rate. Third, if typical paths through the network involve multiple bottlenecks, each such bottleneck should impose only its share of the total desired loss rate.”

As [38] suggests, all three issues can be dealt with, by assuming that each router maintains a count of the active connections sharing each of its links. This is precisely what we do for the long-lived flows. In particular the router mechanism is built with the intention of avoiding repeated consecutive losses of the same flow, hence, technically, avoiding timeouts.

We remind the reader that consecutive losses lead a TCP connection to timeout and a sequence of timeouts can lead to the adjustment of the timeout interval according to an exponential backoff scheme, hence severely degrading the throughput of the connection and jeopardizing the opportunity for a quick recovery (see [39] for a comprehensive analysis of TCP's latency in light of losses). The control of the loss instant (and hence the loss rate) comes at the expense of having to possess sufficient buffer space so that the losses do not occur from overflows. In terms of an ensemble of mechanisms meant to result in classes of "best-effort" service, we refer to an early example [41] which illustrates how the inclusion of traffic profiling devices, and specifically of a time-sliding window (TSW) for each flow, can be used to drive a tagging algorithm. In essence, [41] uses a similar apparatus to derive, implicitly, the demands of flows, but the demands are not used towards a specific objective (e.g. fair allocation) but, instead, guiding the marking of the RED with In/Out marking (RIO). Clearly, flows with different RTTs receive different performance. The situation was recognized in [41], and consequently prescribed constrained use of their proposal within a certain tight RTT regime (20 to 100 msec) and even then with occasional departure from the intended share between flows. That is, RIO still results in a probabilistic behavior with respect to the losses, so the occasional victimization of a flow (i.e. successive losses and/or timeouts) remains. The logic of the RIO marking scheme presented in [41] is similar to our approach (whenever the intended allocation is exceeded, as determined by the on-line measurements, packets are marked) but no guarantee exists in [41] that an action will be taken (it depends on eventual behavior of the RED queues of RIO). In addition, no identification exists as to whether a flow is constrained due to a bottleneck elsewhere in the network (hence, the complete link capacity is assumed to be always in demand by the flows), although one would expect that the latter point is easier to fix.

Chapter 3

Model-based Link Oriented Fairness

3.1 Introduction

In this chapter, we focus on the topic of fairness among long-lived TCP flows crossing a network link. We note that TCP connections can transmit from a few packets, e.g., short Web pages transfers, to extremely long sequences of packets, e.g., transfers of long archive files, resulting in a wide mixture of connection lifetimes. Coming up with a single metric that captures attributes of fairness across all TCP flow lifetimes does not make much sense. For example, flows with a few packets to deliver may not require long-run throughput fairness guarantees, preferring, instead, fair treatment with respect to the average delay performance for delivering their data. Long lived flows have certainly a legitimate interest in achieving long-run throughput fairness with respect to other long-lived flows. We will defer the study of the performance of short-lived flows for Chapter 5.

The proposed scheme is part of a class-based approach that groups the short- and long-lived TCP flows into two separate classes, and performs bandwidth allocation separately for each class. The separation into classes allows us to consider a Differentiated Services (*DiffServ*) environment, whereby each class receives different treatment at the routers. For the subset of long lived flows, a certain amount of state information will be maintained. Despite the problems that state information causes to scalability, we note that it is not expected that the state information we propose is particularly burdensome. Indeed, of the tens of thousands of flows that may be flowing through a core router in the Internet at any point in time, only a handful of them are long-lived flows, the rest being short by merit of the fact that they transfer mostly short Web pages. Hence, in absolute numbers, we

do not expect the size complexity of per-flow information of long-lived TCP flows to be significant, simply because the fraction of long-lived flows is small. The same observation is even more valid for routers close to the periphery of the network. We will present in Chapter 6 relevant results indicating that only a small fraction of flows qualify as long lived and therefore require additional state information. Specifically, we introduce `FairShare`, a scheme for fairness among long-lived flows. This can be seen as a replacement for `DropTail` or `RED`. Note that different traffic classes can (and should) use a different scheme to fulfill their own objectives. The key property of `FairShare` is that it explicitly calculates when a loss is to be inflicted on a flow in order to achieve a particular long-run window size, and hence, throughput. That is, the loss events are not left to the control of the transient dynamics of the aggregate traffic (as it happens in `DropTail`, `RED` and other schemes) but to the per-flow observed window size, thus requiring per-flow state.

The rest of this chapter is organized as follows. Section 3.2 presents previous related works and brings out the differences between that and our work. Section 3.3 introduces the `FairShare` scheme by pointing out its individual features. A subsection distills the essence of `FairShare` in the form of pseudocode for three functions, an initialization function, a per packet arrival function, and a periodic monitoring function. Section 3.4 presents simulation results that illustrate the effectiveness of the scheme, in delivering the desired fairness and coping with the varying nature of the flows. Section 3.5 discusses and clarifies issues raised in implementing the scheme.

3.2 The FairShare Scheme

The service considered in this chapter is (externally) *DiffServ* with class-based bandwidth reservation. Thus, given a link l with capacity C_l , the capacity is split between UDP traffic, long-lived TCP traffic, short-lived TCP traffic, and routing-related traffic, with capacity, correspondingly: C_l^{UDP} , $C_l^{\text{L-TCP}}$, $C_l^{\text{S-TCP}}$, and, C_l^{Route} , where $C_l = C_l^{\text{UDP}} + C_l^{\text{L-TCP}} + C_l^{\text{S-TCP}} + C_l^{\text{Route}}$. The rest of the chapter deals with how to administer the $C_l^{\text{L-TCP}}$ among long-lived flows. `FairShare` is the scheme that will perform this function. It is assumed that the class capacities are quasi-static, that is, they can change but with operator intervention and at time scales much larger than the average connection lifetime. We believe that this is a reasonable assumption, in light of the fact that operators may wish to, intentionally,

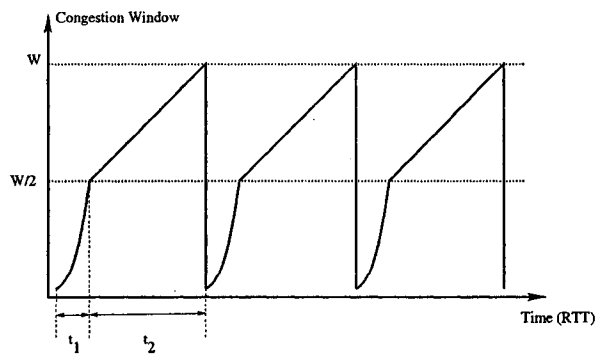
restrict the total bandwidth available to long-lived flows in order to provide short response times to short transfers, as is the case for most Web document transfers. First we discuss FairShare in the context of a fixed number of competing “greedy” TCP flows. Subsequently, we clarify how the continuously changing number of long-lived flows is accommodated, as well as how the non-“greedy” and bottlenecked behavior of the flows can be identified through periodic measurements. In the simulation study we will not consider a dynamic number of connections, because the scheme can be exercised by, equivalently, varying over time the demands of already admitted long flows.

3.2.1 Regulating TCP Flows via Scheduled Losses

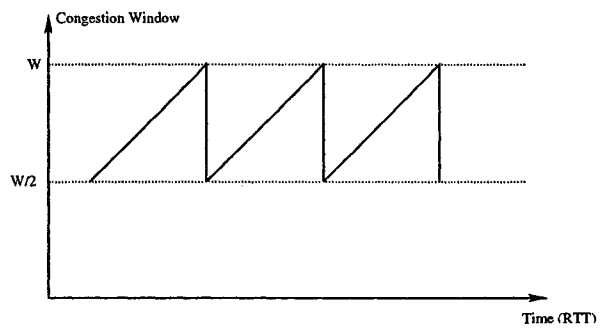
The first step in controlling the long-lived TCP flows is to determine the fair share of capacity at the bottleneck link for a particular TCP flow, i . Let us denote this share as share_i (in rate units). The share is the element of the allocation vector provided by the fairness algorithm¹, i.e., $\text{share}_i = \lambda_i$. Shares can be calculated on the basis of a demand vector. The demand vector is obtained by a measurement process that will be described later in this section. We will use demand_i to describe the measured demand of flow i . Given a calculated share value share_i and the RTT of flow i , rtt_i , the desirable long-run window size that would provide the share_i to flow i is simply $W' = \text{rtt}_i \cdot \text{share}_i$ (in bit units).

Figure 3.1.a depicts the evolution of the TCP-Tahoe congestion window. Let us define a TCP-Tahoe “epoch” as the time interval from the point that the TCP congestion window is 1 until the first packet loss occurs. Here, the y axis is the congestion window size of the TCP flow and the x axis represents time in RTT units. As we have seen in Figure 3.1.a, a single epoch of congestion window evolution consists of two periods, t_1 , the exponential growth period, and t_2 , the linear growth period. In steady state, the final window size at the end of the epoch, which coincides with the end of the linear growth, W , is twice as large as that of the exponential stage. Because the congestion window increases exponentially before it reaches $\frac{W}{2}$, we have $\frac{W}{2} = 2^y$, where y stands for the rounds that the TCP-Tahoe flow spends in exponential growth. Since the congestion window increases only by one per round, the rounds spent in linear growth is $\frac{W}{2}$ or 2^y . In short, the time of one epoch of TCP-Tahoe with RTT representing the time unit can be presented as $y + 2^y$. The total count of the packets

¹We tacitly assume in the rest that the fairness implemented is max-min fairness, but any fairness objective could be used in its place.



(a) TCP-Tahoe



(b) TCP-Reno

Figure 3.1: Steady state congestion window behavior in, (a) TCP-Tahoe and, (b) TCP-Reno.

communicated in one epoch is the sum of packets transferred in the exponential increase phase, $\int_0^y 2^x dx$, and those in the linear growth phase, $\int_y^T (2^y + x) dx$. Based on the congestion control window evolution algorithm, we can derive equation (3.1), where W' is the average congestion window. The numerator of the left side of equation (3.1) is the number of packets transferred in steady state during one epoch (including both exponential and linear increase phase). The denominator is the number of rounds within the complete TCP-Tahoe epoch.

$$W' = \frac{\int_0^y 2^x dx + \int_y^T (2^y + x) dx}{y + 2^y}. \quad (3.1)$$

If a packet loss is scheduled at the point where the window size is 2^{y+1} , it will effectively limit the long-term average flow to approximately W' . By setting, $W' = \text{rtt}_i \cdot \text{share}_i$, a loss scheduled to occur when the flow's window is 2^{y+1} effectively controls the long-run throughput of the flow to be consistent with the calculated fair share. Based on this observation, we claim that we can regulate the long-term flow rate by inflicting losses when the window of the flow reaches a specific value. Hence, we can use on-line observations of the window size to derive the loss instants with the intention of preserving a desirable long-term mean. Equation (3.1) can also be expressed as equation (3.2), but given the lack for an explicit solution for y to equation (3.2), a table lookup can be implemented to determine the target window size as soon as the average congestion window size objective is calculated.

$$\frac{1}{\ln 2} 2^y + \frac{3}{2} (2^y)^2 = W' (2^y + y) \quad (3.2)$$

For TCP-Reno, the calculation is straightforward because of the simplicity of the algorithm. We assume the ideal equilibrium state will be reached and the congestion window of TCP-Reno will construct a perfect "sawtooth" with a peak at $\frac{W}{2}$. Therefore, the time of one epoch in terms of RTT is $\frac{W}{2}$, and the packets communicated in one epoch is $\frac{(\frac{W}{2} + W) \frac{W}{2}}{2}$ (Figure 3.1.b). The average congestion window size is calculated by $W' = \frac{3W}{4}$.

3.2.2 The Algorithm

In this section, the pseudo-code of the FairShare algorithm is presented in the form of three basic functions. A packet is indicated by the variable p . In addition, $p.size$, $p.src$, and $p.dst$ stand, respectively, for the packet size, source and destination. The per-flow variables maintained for each long TCP flows are as follows:

init_long_flow(*p*):

1. $rtt_i \leftarrow \text{rttlookup}(p.src, p.dst);$
2. $count_i \leftarrow p.size;$
3. $demand_i \leftarrow C_i^{L-TCP};$
4. $share \leftarrow \text{maxmin}(demand, C_i^{L-TCP});$
5. $flag_i \leftarrow \text{FALSE};$
6. $dropevent_i \leftarrow 0;$
7. $\text{tick}();$

Figure 3.2: The **init_long_flow**() function.

$flag_i$ binary flag indicating whether a loss is due to be inflicted on the flow.

$dropevent_i$ timestamp of last loss inflicted on the flow.

$count_i$ bytes arriving from the flow within its last RTT.

rtt_i the RTT time of the flow.

$demand_i$ the flow demand (average of measured rate).

$share_i$ the flow share calculated (“allocated”) by the fairness scheme.

On deciding that the flow is a long TCP flow, **init_long_flow** (Figure 3.2) is invoked. First, the RTT value of the flow is determined (line 1) via a mechanism detailed in Chapter 6. In the example code, we will tentatively assume that RTT information is collected and maintained along with routing information (thus **rttlookup**() requires source, and destination information about the particular flow). In addition, the first packet arrival sample within the current RTT window is recorded (line 2). Because the demand is yet unknown, it is assumed to be unbounded (“greedy” flow), so, technically, it is sufficient to be set as high as all the available bandwidth of the class (line 3). The introduction of the new flow requires the re-calculation of the shares not just for flow *i*, but the entire share vector for all long TCP flows (line 4). A grace period of at least one RTT is given (lines 5 and 6) for the dropping of packets—since we need at least one measurement, i.e., one RTT period before we can gauge an approximation of the true demands of the flow. The last step is to invoke the **tick** function to periodically check the demands of the flow.

tick():

```
1.  while (1)
2.       $m \leftarrow \text{count}_i / \text{rtt}_i$ ;
3.       $\text{count}_i \leftarrow 0$ ;
4.      if (  $\text{share}_i < \text{demand}_i$  and  $m > \text{avg2peak}(\text{share}_i)$  ) then
5.           $\text{flag}_i \leftarrow \text{TRUE}$ ;
6.      else
7.           $\text{flag}_i \leftarrow \text{FALSE}$ ;
8.      endif
9.       $\text{demand}_i \leftarrow m \cdot \beta + \text{demand}_i \cdot (1 - \beta)$ 
10.     if (  $\sum_i \text{demand}_i > C_i^{\text{L-TCP}}$  ) then
11.          $\text{share} \leftarrow \text{maxmin}(\text{demand}, C_i^{\text{L-TCP}})$  ;
12.     endif
13.     sleep( $\text{rtt}_i$ ) ;
14. endwhile
```

Figure 3.3: The tick() function.

Essentially, in order to determine the demands of a flow, we first note that the demands can be limited already due to an upstream bottleneck (at a remote router) or because the flow does not have enough data to send. The purpose of tick (Figure 3.3) is to perform the observation of the rate during the last RTT (line 2), thus resetting the counter (line 3). If the share allocated to the flow is less than its demand, then the flow needs to be regulated by introducing losses, but this is necessary only when the current rate (translated into its window value) reached the corresponding maximum value possible as per the TCP window model of subsection 3.1 (line 4). If this is the case, we signal to the next arrival that it has to be discarded (line 5). The demand is subsequently calculated (line 9) and if the demands over all the flows cannot be satisfied with the bandwidth available to the class (line 10). The next observation will occur an RTT later (line 13). the new shares are calculated (line 11)

Furthermore, function `upon_packet_arrival` is the substance of operations taking place when a packet of flow i arrives. The only decision taken is whether the packet ought to be enqueued or dropped. Dropping the packet is a decision based on (a) whether the tick function has determined it is time to do so, and (b) if enough time has elapsed since the last drop, because, otherwise, repeated losses within an RTT time would likely force the TCP flow to timeout and its throughput to deteriorate substantially.

`upon_packet_arrival(p)`:

```
1.  now ← time();
2.  if ( flagi and dropeventi+rtti < now ) then
3.    dropeventi ← now;
4.    drop(p);
5.  else
6.    counti ← counti + p.size;
7.    enqueue(p);
8.  endif
```

Figure 3.4: The `upon_packet_arrival()` function.

A relatively minor detail is how connections are terminated. Evidently, this can be performed using two techniques (both are required in fact to compensate for the case of flows being rerouted). One is to identify the FIN/FIN-ACK packet exchange, and we will discuss this in detail at Chapter 6. The second is to add a timer to alert of possible termination after a sufficiently long period of inactivity. In both cases, it is not enough to deallocate the data structure of the terminated flow. It is essential that the `maxmin` function be re-invoked to reassign the share of the terminated flow to the remaining flows. Finally, the control loop is closed because the observed average rate influences the demand, which influences the allocation (via the fairness algorithm), which influences the losses (via the calculated share), which influence the average rate (via the scheduled losses). However, an additional feature is to be able to identify allocated rates that match the corresponding flow demand because the demand is sufficiently low, e.g., if the flow is bottlenecked at a remote node. In this case, we avoid enforcing losses on the flow because it is already constrained at another point in the network. By the same token, unused bandwidth of a flow's allocation, due to observed low demand, is (re)allocated to the rest of the competing flows on the basis of the fairness criterion used (in our case: max-min).

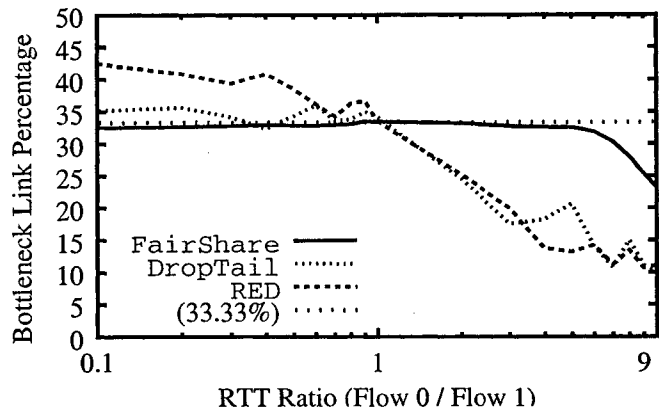
3.3 Simulation Study

The proof-of-concept simulations presented here are based on the ns-2 [6] simulator. The configurations examined are consistent with the ones used in the majority of the rest of the literature on the topic. Specifically, we focus on a single bottleneck link with a link

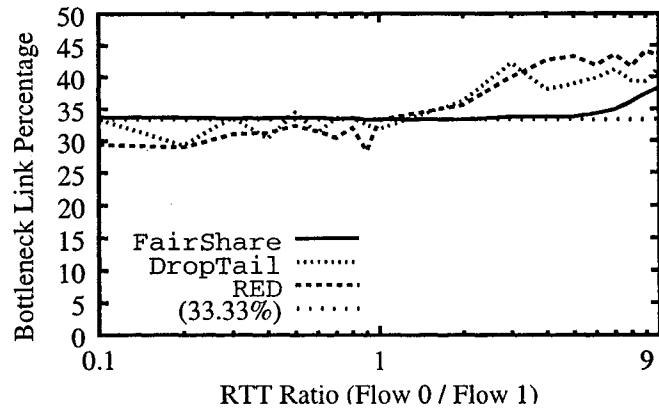
rate of 100 packets per second. Three flows traverse the link. We explore the dynamics and effectiveness of the algorithm by looking at the results of this toy example. The TCP version used is TCP-Reno (for a discussion of Tahoe's behavior see the next section). Three simulation studies were carried out using FairShare, targeting the scheme's essential feature (whether fairness is achieved) and whether the flows are correctly identified as being greedy, or not, or bottlenecked someplace else along their path to a destination.

The results of the first set are presented in Figure 3.5. Three flows are simulated. The x axis presents, in logarithmic scale, the ratio of the RTT of flow 0 over that of flow 1. Flows 1 and 2 have an end-to-end RTT of 0.2 seconds. RTT values for flow 0 range from 0.1 times to 10 times that of Flow 1 (that is, from 0.02 sec to 2 sec). The buffer size we used is 30 in all experiments concerning DropTail and RED policy. The simulations run for 300 seconds which are sufficient for qualifying long-lived flows. The plots are obtained from the average of 10 experiments. Due to small variance in our FairShare experiments, we believe that the plots of averages are good enough for presenting the performance. The rest of the RED parameters are as in the example at Chapter 1, namely, $max_p = 0.02$, $min_{th} = 5$, $max_{th} = 15$, $w_q = 0.002$. The queue size in FairShare experiments is 300. In reality, most of this large queue is not used in high occupation. The point of introducing large buffer in FairShare is that FairShare requires that no packet loss will be caused by overflowing buffer.

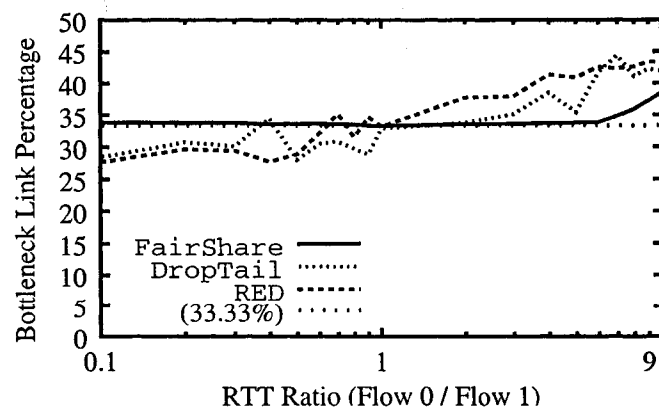
What can be concluded from Figure 3.5 is that FairShare outperforms DropTail and RED, in that, under FairShare the three competing flows receive each a third of the bottleneck link bandwidth (33%) *regardless* of the RTT of each flow. The interesting behavior at the extreme values of the RTT ratio (above a ratio of 5) is for a completely unrelated reason! Namely, the simulations were carried out with a receiver advertised window of 48 packets. What this suggests, is that even if a large bandwidth-delay product is available, a flow could not keep in the pipe more than 48 packets. Hence, once the bandwidth-delay product of flow 0 exceeds 48 packets (near the ratio of 5), it simply could not increase its window further. Naturally, the other two flows, operating at much smaller bandwidth-delay products than 48, can capitalize on flow0's inability to "fill the pipe". The results therefore suggest correct operation of the max-min fair allocation for any RTT. Another striking feature of the first set of experiments is that one cannot trust RED to always result in better fairness than DropTail



(a) Flow 0



(b) Flow 1



(c) Flow 2

Figure 3.5: Throughput achieved by FairShare vs. that by DropTail and RED.

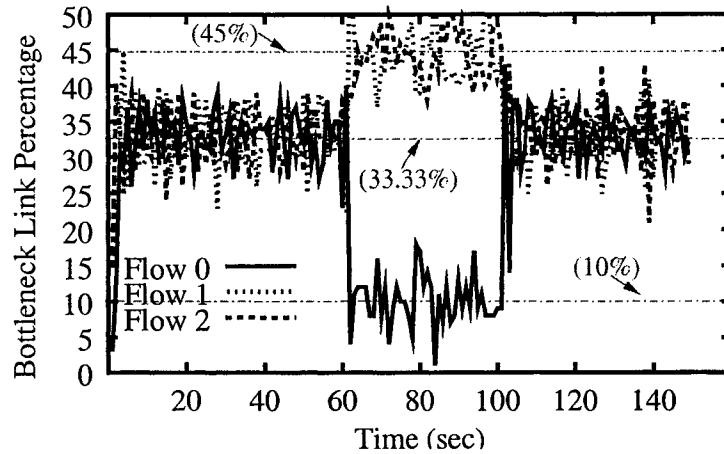


Figure 3.6: Throughput of the three competing flows under the FairShare scheme.

as the RTT-dependent behavior of RED reveals.

The next question addressed is whether the behavior of TCP flows can be correctly detected when they suddenly become non-greedy. An example of such a case are connections where long bursty data transfers alternate with periods of moderate (or no) traffic activity. An example is persistent HTTP connections that maintain a single TCP connection for multiple page transfers spaced apart in time. Moreover, applications of high performance computing where a single connection is used for occasional large transfers also fit the description. The intention of the algorithm is to correctly infer the fact that a TCP flow cannot fully utilize the bandwidth assigned by FairShare, and to subsequently, by reducing the measured demand of the flow, free up bandwidth, i.e. reduce the share of the flow and increase the share of the remaining flows. The baseline example is as before (i.e., three flows) initially bottlenecked at the link, with all RTTs equal to 0.2 seconds. The behavior of flows 1 and 2 remains greedy throughout the simulation, i.e., they can exploit as much bandwidth as is allocated to them. Flow 0 is initially greedy but at time 60 reduces its traffic intensity drops to 10% of the link's bandwidth (that is, 10 packets per second), only to return to its greedy behavior at time 100. As Figure 3.6 reveals (throughput experienced as percentage of bottleneck link), the initial fair splitting of the link bandwidth (all flows receiving 33.33% in the long run) is adjusted to the new values at time 60. That is 10% for flow 0, and, as per max-min fairness, 45% for flows 1 and 2. What really has happened is that the reduced demand of flow 0 is calculated at line 9 of `tick()`, followed by the updated `sharei` value. Consequently, an `rtti` later, line 4 of `tick()` will avoid completely setting `flagi` to

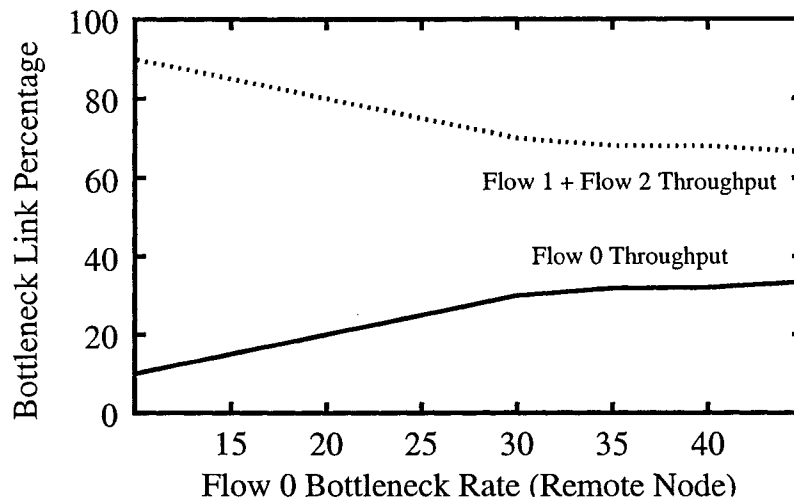


Figure 3.7: Throughput of flow 0 bottlenecked at a remote node.

TRUE, thus ensuring that, as long as the demand does not increase, subsequent invocations of `upon_packet_arrival()` will result in normal enqueueing (no loss) of the flow's arriving packet. What the behavior of flow 0 demonstrates, which is not visible in the throughput plot of Figure 3.6, is that FairShare does not inflict losses on a flow which does not exploit its potentially large window value. That is, greedy flows send as much data as their window will allow, while flows like flow 0 that are non-greedy, may not exploit their window because of the simple fact that they have insufficient volume of data to transmit. If this (smaller) bandwidth demand can be satisfied by the allocated (as per the fairness algorithm) bandwidth, no action of regulation, i.e., loss, needs to be imposed on them.

Finally, Figure 3.7 illustrates how FairShare detects that a flow (in this case flow 0) is bottlenecked at a remote node, and therefore distributes the unusable bandwidth of flow 0 to the other greedy TCP flows. In the example simulations, the remote bottleneck of flow 0 was varied from 10 to 45 packets per second. As soon as the remote bottleneck is more than 33.33, it essentially ceases to be a bottleneck, and the actual bottleneck for flow 0 becomes the current link. Hence, the throughput received by flow 0 reaches a plateau as soon as the remote bottleneck reaches the value of 33.33. At all times, the remaining greedy flows exploit the bandwidth not allocated to flow 0 and sharing it (not shown in Figure 3.7) equally.

3.4 Discussion

While FairShare appears to work as designed, one can observe that, in contrast to DropTail and RED, we make no mention of the necessary buffer size. Because the losses are scheduled, rather than being the result of overflows, we wanted to test (by suitably allocating large buffer sizes) how much buffer is really necessary. We note that because of the regular behavior of a TCP flow, for a flow with rtt_i and currently receiving a share of share_i , the ideal long-term window size will be $\text{share}_i \cdot \text{rtt}_i$, assuming the router is serving the flows providing per-flow allocation of share_i . Since the window size per RTT is measured, it can be inferred that the difference between the bandwidth-delay product and the actual measured window represents packets residing at the queue of the bottleneck node. Clearly, if the current node is not the bottleneck, then the corresponding buffer allocation for the flow is close to zero (in-flow is smaller than out-flow). If it is the bottleneck (the flow is being regulated by injecting losses) then any measured window above $\text{share}_i \cdot \text{rtt}_i$ suggests accumulation at the queue. The peak buffer size necessary is therefore $\sum_{W > w_j > \text{share}_i \cdot \text{rtt}_i} (w_j - \text{share}_i \cdot \text{rtt}_i)$ where W is the maximum window size (as the notation of subsection 4.2.1). For example,

Flow id	Current Win_Size	Peak Win_Size (W)	RTT	Share-RTT	Backlogged
1	40	48	1 (s)	36	4
2	20	24	0.5 (s)	18	2
					Total: 6

However, we have indications that this approach of per-flow allocation is a far cry from the actual allocation necessary, because the total allocated buffer can be better utilized if shared, as a common resource, among all long-lived flows. However, in this case, N , the number of flows, can significantly impact the transient demands of buffer space. Another detail is the selection of β , the parameter for calculating the exponentially averaged congestion window size. In the simulations it was assigned to 0.8. That is, we put more weight to the current window size to respond relatively quickly to window size adjustments. From a set of experiments, we found that reasonable values of β are in the 0.6 ~ 0.9 region. Our scheme works on TCP flows on both Tahoe version and Reno version. In this chapter, all simulations are on TCP-Reno because of its almost universal use and simplicity. The difference of how to deal with Reno or Tahoe lies only in the function $\text{avg2peak}(\text{share}_i)$. For Reno, the returned value is $\frac{4\text{share}_i}{3}$, consistent with the relation of average and peak window of TCP-Reno in subsection 3.1. avg2peak for TCP-Tahoe is more complicated

and requires a lookup table. A more interesting problem is how to determine if the flow is Tahoe or Reno. A possibility is to, exceptionally, monitor the behavior of the flow, upon the first inflicted loss. If the window size in the following RTT is reduced abruptly (less than half its previous value) it can be considered to be a candidate TCP-Tahoe flow. While this can work in principle, it entails the risk that the flow was TCP-Reno that happened to become non-greedy. Therefore, we can assume the default behavior is TCP-Reno, with the understanding that if after a number of induced losses it exhibits consistently the reduction of TCP-Tahoe, it will be reclassified as Tahoe. Another issue is dealing with non-cooperating flows. Our scheme works under the assumption that the sender responds to the packet losses, and backs off its congestion window, either according to Tahoe or Reno. In order to deal with non-cooperating end-users, more policing functionality is expected to be needed. It is an open issue whether *any* proposed AQM scheme, including RED, can police effectively the compliance of flows to a certain TCP implementation. We only note that our FairShare AQM gateways already maintain the state for each flow, and non-cooperating flows could be singled out on the basis of a measured demand which is consistently non-compliant.

Chapter 4

AQM and Global Fairness Objectives

4.1 Introduction

The topic of this chapter is how the particular AQM scheme that we proposed in the previous chapter, FairShare [43], which achieves *max-min* fairness on a single link, can be extended, using elements of an earlier work by Anna Charny [47], to achieve *global max-min fairness*. In particular, this chapter provides results that allow us to appreciate the efficiency and stability of a network composed of nodes that implement the FairShare scheme. In this chapter, we are still looking into the fairness among long lived flows. Policies for short-lived flows will be examined later in the thesis.

We tackle the problem from a simulation point of view, using “traditional” as well as less common topologies. In general, the main challenge of the analysis is the sheer complexity that is caused by the interaction of different components of the network, the incomplete information and the heterogeneity of the system. To the already complicated behavior of the end-to-end dynamics, AQM introduces a non-negligible impact from the router policies that is different from traditional First-Come-First-Served (FCFS). To cope with the complexity, we plan to extend the techniques used in [46, 47], which were used to demonstrate convergence of a network-wide policy to the global optimal value regardless of initial conditions. The optimal values considered here capture the max-min fair allocation. The basic idea is to explicitly calculate the optimal value of rates for each long-lived flow and to determine if FairShare does indeed achieve this desired target. The remainder of this chapter is organized as follows. Section 4.2 describes the global fairness algorithm and formal proof of convergence. Section 4.3 presents the experiments we have conducted on studying the global

behavior of FairShare. Conclusions are given in Section 4.4.

4.2 Distributed Global Fairness

Rigorous analysis of global fairness and relevant distributed algorithms to achieve it were extensively studied in [47]. For a large network configuration with multiple bottleneck links and multiple flows, a *feasible* set of flow rate allocation satisfies two conditions: First, the allocated bandwidth for any flow is a non-negative number. Second, for any link in the network and the flows passing through link l , the sum of allocated bandwidth for these flows is smaller than the capacity of C_l . A global max-min fair allocation is a feasible allocation that maximizes the smallest element(s)/allocation(s) subject to the capacity constraints. Naturally, certain exogenous constraints are also present, namely the ones on which path is used between each source-destination pair. However, the calculation of the paths is left outside the scope of this thesis, and it is normally expected to be the task of a separate routing algorithm.

4.2.1 Synchronized Algorithm

In [47], a procedure for determining global max-min rates was presented (Figure 4.1). The algorithm is better understood in its centralized form. It involves the identification of the most congested link (first level bottleneck), and application of the max-min fairness on the flows crossing this link. The demand of all flows crossing the first link is then limited (bounded) by the share they receive in this most congested link. Given the new bounds for some of the flows, the residual capacity in the remaining links is re-calculated and this, in turn, changes the share received by other flows. The process is repeated until all flows have been marked by a certain rate. Note that all flows are assumed to be greedy, but it is trivial to force the realistic assumption of bounded flows by assuming that each traffic flow is introduced via an access link, which is of course bounded, and possibly congested. It is shown in [47] that the process terminates after a limited number of iterations. First we present three definitions that will be used in the later part.

Definition 4.1 *A network-wide feasible set of rate allocations $\eta = (\eta_1, \dots, \eta_n)$, \forall link l that satisfies*

find_global_optimal_rates:

1. Determine the first level bottleneck link set L , that is, the most congested link because of the small bandwidth or more flows passing through or both.
2. Apply local max-min fairness criteria on the flows passing through L .
3. Mark the flows passing through L as flows with limited demands calculated in the last step.
4. Adjust the capacity of links which are traversed by such flows.
The new available capacity is the remaining after taking out the already allocated from the original capacity.
5. Repeat from step 1 until all flows are marked with some rates.

Figure 4.1: The **find_global_optimal_rates** procedure.

- $\eta_i \geq 0$
- $\sum_{i \in f_l} \eta_i \leq C_l$.

Definition 4.2 Consider vector $a = (a_1, \dots, a_n)$, let $\hat{a} = (\hat{a}_1, \dots, \hat{a}_n)$ be a permutation of a such that if $i < j$, $\hat{a}_i < \hat{a}_j$. Vector b is said to be **lexicographically greater than a** if either $\hat{a}_1 < \hat{b}_1$, or $\exists 1 \leq K \leq n$, so that $\hat{a}_i = \hat{b}_i \forall 1 \leq i < K$ and $\hat{a}_K < \hat{b}_K$.

Definition 4.3 Vector $\eta = (\eta_1, \dots, \eta_n)$ is called **maximum optimal allocation for network N** if

- it is a feasible set
- it is lexicographically greater than any other feasible solution

[47] also presented a formal description of the procedure of approaching global optimal rate. For the sake of brevity, we present here the simplified version of the procedure in which the backward traffic is not considered. In the description below, we use the following notation:

- Network $N(\Lambda, \Theta)$ is composed of Λ , a set of links and Θ , a set of flows.
- C_l is the capacity of link l , where $l \in \Lambda$.
- $f_l = \{s \in \Theta : s \text{ traverses } l\}$. The number of the flows in set f_l is $|f_l|$.

- $\widehat{L} = \{l \in \Lambda : |f_l| > 0\}$. That is, the set of all links $l \in \Lambda$ which satisfy that at least one flow in Θ traverses l . In other words, the subset of Λ is used by at least one flow.
- $L = \{l \in \widehat{L} : \frac{c_l}{|f_l|} = \min_{j \in \widehat{L}} \frac{c_j}{|f_j|}\}$ That is, the set of bottleneck links.
- $r_l = \frac{c_l}{|f_l|}$, where $l \in L$. That is, the fair rate for flows crossing the bottleneck link l .
- $S = \{s \in \Theta : \exists l \in L, s \text{ traverses } l\}$. That is, the set of flows which cross at least one bottleneck link in L .

The procedure might need several iterations to reach global fairness. Thus, for each iteration i , we have a set of the above variable to denote the status of the algorithm and we will use subscripts and superscripts between iterations. Specifically:

- $L(i)$, the set of bottleneck links in iterations i .
- $S(i)$, the set of flows which use at least one bottleneck link in iteration i .
- $f_l(i)$, the set of flows in the bottleneck link in iteration i .
- $r(i)$, the fair rate of flows crossing bottlenecked link in iteration i .

From the definition of the `find_global_optimal_rates` procedure, [47] derived the following two theorems about the characteristic of the procedure.

Theorem 4.1 *The `find_global_optimal_rates` procedure terminates in a finite number of iterations*

Proof

For each iteration, at least one bottleneck link will be selected from the reduced network, that is the remaining network after excluding the sessions appropriately allocated, and the links already properly shared. All flows crossing this most congested bottleneck link(s) will be added to $\tilde{S}(i)$, that is, $\tilde{S}(i) = \bigcup_{j=1}^i S(j)$ or the set allocated sessions after iteration i . Thus, at the end of each iteration, the number of flows in $\tilde{S}(i)$ increases at least by one. The procedure will terminate when all flows in Θ are in $\tilde{S}(i)$ (signifying that i is the last iteration). So, at most $|\Theta|$ iterations are needed to terminate. **End of Proof**

find_global_optimal_rates:

While ($|S(i)| < |\Theta|$) :

$i = i + 1$

GIVEN

$L(1), L(2), \dots, L(i-1)$

$S(1), S(2), \dots, S(i-1)$

$f_l(1), f_l(2), \dots, f_l(i-1)$

$r(1), r(2), \dots, r(i-1)$

DEFINE

$\tilde{S}(i-1) = S(1) \cup S(2) \cup \dots \cup S(i-1),$

$\tilde{L}(i-1) = L(1) \cup L(2) \cup \dots \cup L(i-1),$

The reduced network $N(i)$ consists of link set $\Lambda - \tilde{L}(i-1)$ and flow set $\Theta - \tilde{S}(i-1)$

$f_l(i) = \{s \in \Theta - \tilde{S}(i-1) : l \in \Lambda - \tilde{L}(i-1), s \text{ traverses } l\}$

$\hat{L}(i) = \{l \in \Lambda - \tilde{L}(i-1) : |f_l(i)| > 0\}.$

$L(i) = \{l \in \hat{L}(i) : \frac{c_l - \sum_{j=1}^{i-1} (r(j) \cdot |f_l(j)|)}{|f_l(i)| + \sum_{j=1}^{i-1} |f_l(j)|} = \min_{q \in \hat{L}(i)} \frac{c_q - \sum_{j=1}^{i-1} (r(j) \cdot |f_q(j)|)}{|f_q(i)| + \sum_{j=1}^{i-1} |f_q(j)|}\}$

$r(i) = \frac{c_l - \sum_{j=1}^{i-1} (r(j) \cdot |f_l(j)|)}{|f_l(i)| + \sum_{j=1}^{i-1} |f_l(j)|}, l \in L(i)$

$\tilde{S}(i) = S(i) \cup \tilde{S}(i-1)$

$\tilde{L}(i) = L(i) \cup \tilde{L}(i-1)$

Done

Table 4.1: The `find_global_optimal_rates` procedure.

Theorem 4.2 *When the `find_global_optimal_rates` procedure terminates, all flows are assigned their globally optimal rates.*

Proof:

First we note that $r(1) < r(2) < \dots < r(m)$.

Let n_i be the number of flows in S_i . Then the rate allocation obtained by the `find_global_optimal_rate` procedure can be written as $\beta = (\beta(1), \beta(2), \dots, \beta(|\Theta|)) = (\underbrace{r(1), \dots, r(1)}_{n_1}, \dots, \underbrace{r(m), \dots, r(m)}_{n_m})$.

Let $\alpha = (\alpha(1), \alpha(2), \dots, \alpha(|\Theta|))$ be another feasible rate allocation.

Let a be the permutation of α , so $a(1) \leq a(2) \leq \dots \leq a(|\Theta|)$.

Suppose $a(1) > r(1)$,

Consider any link $l \in L(1)$, $r(1) = \frac{c_l}{|f_l(1)|}$,

The total throughput across l is

$$a(1) \cdot |f_1| > r(1) \cdot |f_1(1)| = C_1.$$

The feasibility is violated. Thus, $a(1) \leq r(1)$.

If $a(1) < r(1)$, then β is lexicographically greater than α .

Suppose, $a(1) = r(1)$,

or in more general form, we suppose $a(i) = r(i)$, when $\forall 1 \leq i < K$ and $a(i) > r(i)$ otherwise. For the sake of simplicity and without loss of generality, we consider only the case of $n_1 = n_2 = \dots = n_m = 1$. Thus, the total throughput across l is

$$\begin{aligned} \sum_{j=1}^{K-1} (a(j) \cdot |f_l(j)|) &= \sum_{j=1}^{K-1} (r(j) \cdot |f_l(j)|) + \sum_{j=K}^m (a(j) \cdot |f_l(j)|) \\ &\geq \sum_{j=1}^{K-1} (r(j) \cdot |f_l(j)|) + a(j) \cdot \sum_{j=K}^m (|f_l(j)|) \\ &> \sum_{j=1}^{K-1} (r(j) \cdot |f_l(j)|) + r(j) \cdot \sum_{j=K}^m (|f_l(j)|) \geq C_l \end{aligned}$$

The feasibility is violated again. So, for $K \leq i \leq m$, it must be $a(i) \leq r(i)$. If the inequality is strict, then β is lexicographically greater than α . Therefore, β is lexicographically greater than any other feasible rate allocation vector α . **End of Proof**

4.2.2 Distributed Algorithm and Convergence

The distributed version of [47] of the above `find_global_optimal_rates` procedure resembles the synchronized one, except the procedure has to collect the information about the network step by step. The distributed approximation procedure includes a method to acquire the necessary bottleneck rate information via feedback to the traffic sources. It is observed that in order to calculate the fair allocation, a link needs to know only about the rates of the flows traversing it. Also, each link asynchronously calculates, and independently maintains the estimated rate for each flow an *advertised* rate, according to the local max-min fairness criteria. Using the feedback mechanism mentioned above, *advertised* rates for the same flow at multiple links along the path of the flow are summarized. The source node injects data at the *stamped* rate, which is the *minimum* of all the advertised rates it received on

the path. The link that corresponds to the minimum advertised rate is the *bottleneck* for the particular flow. In addition, the link maintains a *recorded* rate for each flow, which is the measured rate within the recent past. For all flows passing through a particular link, the recorded rate should be close to the advertised rate. This is the concept called *M-consistency* in [47]. Anytime after M-consistency is violated, the advertised rate for all flows should be re-calculated. In other words, *M-consistency* requires that source nodes conform to instructions from the gateways before the global optimal rate is achieved. [47] proved that such a distributed algorithm converges to global optimum within bounded time.

Definition 4.4 *The marked flows at any time must satisfy the following conditions, referred as M-Consistency:*

- *If any flow is marked, its recorded rate is less than or equal to the advertised rate.*
- *The advertised rate is calculated according to (Equation 4.1).*

distributed_global_optimal_rates:

Packet p :

u_p , “up-bit”, it is used to indicate that the flow has the potential to increase its rate
 ρ_p , packet’s stamped rate.

Source s :

ρ_s , stamped rate of the last feedback packet received,
 u_s , “up-bit” in source

Destination d :

ρ_d , stamped rate of the last packet received,
 u_d , “up-bit” of the last packet received.

Link l :

C_l , Capacity of the link l ,
 f_l , the set of (known) flows at the link l ,
 G_l , the set of all marked flows (known) at link l .

For any flow $i \in G_l$,

ξ_i^l , recorded rate of the flow.

μ_l , advertised rate of the link, calculated by (Equation 4.1) ,

Table 4.2: The `distributed_global_optimal_rates` procedure.

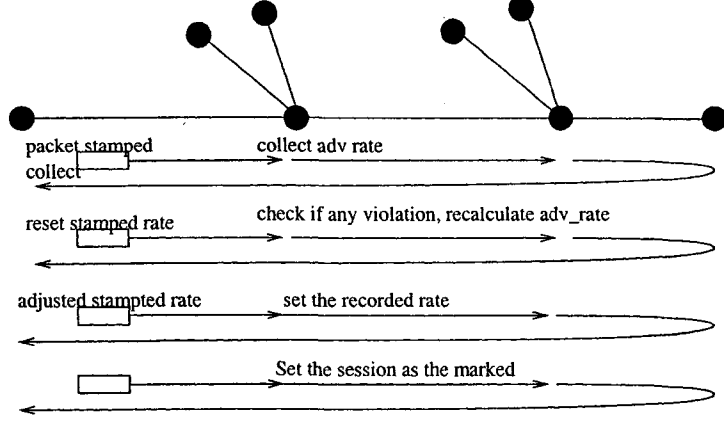


Figure 4.2: Illustration of the `distributed_global_optimal_rates` procedure.

$$\mu_l = \begin{cases} C_l, & \text{if } |f_l| = 0; \\ C_l - \sum_{j \in G_l} (\xi_j^l) + \max_{j \in G_l} (\xi_j^l), & \text{if } |f_l| = |G_l| \\ C_l = \frac{C_l - \sum_{j \in G_l} (\xi_j^l)}{|f_l| - |G_l|}, & \text{otherwise} \end{cases} \quad (4.1)$$

[47] proved that the algorithm 4.2 converges to the global optimal rate within a bounded time. The proof is as follows.

Lemma 4.1 *After any link updates its advertised rate, the marked flows are M -consistent.*

Proof Lets consider any link update.

Y : the set of flows marked at the beginning of update

μ_l : the previous advertised rate calculated by (Equation 4.1).

Z : the subset of Y with stamped rate greater than u_l .

After the previous updating, all flows in Z will be unmarked, therefore, the final advertized rate will be

$$\mu = \frac{C_l - \sum_{i \in (Y-Z)} (\xi_i)}{|f_l| - |Y-Z|} = \frac{C_l - \sum_{i \in Y} (\xi_i) + \sum_{i \in Z} (\xi_i)}{|f_l| - |Y| + |Z|} \geq \frac{C_l - \sum_{i \in Y} (\xi_i) + \sum_{i \in Z} (u_l)}{|f_l| - |Y| + |Z|} = \mu_l \quad (4.2)$$

End of Proof

Lemma 4.2 *After the flow number stabilizes at time t_0 and all these flows are known to all links of the network, for all links l at $t \geq t_0$,*

$$\mu_l(t) \geq \frac{C_l}{|f_l|}$$

Proof Let set Y as the set of flows marked, and μ is the final updated advertized rate, by the Lemma 4.1, all flows marked are M-consistent, thus

$$\forall i \in Y, \xi_i^l \leq \mu$$

, Then,

$$\mu = \frac{C_l - \sum_{i \in Y} (\xi_i)}{|f_l| - |Y|} \geq \frac{C_l - \sum_{i \in Y} (\mu)}{|f_l| - |Y|} > \frac{C_l}{|f_l|}$$

End of Proof

Lemma 4.3 *Let $r(i)$ denote the optimal rate of flows in $S(i)$, where $S(i)$ is the set of flows whose optimal rate were assigned at iteration i of the `find_global_optimal_rates` procedure (Table 4.1), and $L(i)$ is the set of bottleneck links of this iteration, for any $t > t_0$,*

$$\begin{cases} u_l(t) < r(1), & \forall l \in \Lambda - L(1); \\ u_l(t) \geq r(1), & \forall l \in L(1); \end{cases} \quad (4.3)$$

Proof By Lemma 4.2, $\mu \geq \frac{C_l}{|f_l|}$.

By Theorem 4.1,

$$\begin{cases} r(1) = \frac{C_l}{|f_l|}, & \text{if } l \in L(1), \\ r(1) < \frac{C_l}{|f_l|}, & \text{if } l \in \Lambda - L(1), \end{cases} \quad (4.4)$$

Thus, the statement of this Lemma holds. **End of Proof**

Lemma 4.4 *After some time, all flows in $S(i)$ will reach its optimal rate $r(1)$.*

Proof Lets use RTT denote the flows with the largest round trip time. By the definition of `distributed_global_optimal_rates` procedure the packet *stamped rate* ρ_p will record the

smallest advertised rate of links along the path it traversed. Therefore, by the end of the first *RTT*, all of the advertised rate are stored in packet *stamped rate*. Specifically, by the definition of the $S(1)$, which is the set of flows experiencing the first level bottleneck, all packets of any flow in $S(1)$ will have *stamped rate* at least as high as $r(1)$. And any packets of flows $\Theta - S(1)$, the packet *stamped rate* should be strictly greater than r_1 . At the beginning of the second *RTT*, the sources set their *stamped rate* ρ_s to be the same as the packet *stamped rate* ρ_p collected in the last *RTT*. At each link, the *recorded rate* ξ of flows are determined by the latest *stamped rate* of flows. By the end of the second *RTT*, all of the links have updated the *recorded rate* ξ s. In the third *RTT*, the links re-calculate *advertised rate* with the new *recorded rate* ξ . The marked flows (or already assigned flows) which has *recorded rate* greater than the first round *advertised rate* will be unmarked. For $l \in L(1)$, all of flows crossing l are in $S(1)$, and suppose all flows in $S(1)$ are marked, thus,

$$\mu_l = \frac{C_l - \sum_{i \in S(1)} \xi_i}{|f_l| - |S(1)|} \quad (4.5)$$

By Theorem 4.1, $r(1) = \frac{C_l}{|f_l|}$, Thus,

$$\mu_l = \frac{r(1) \cdot |f_l| - \sum_{i \in S(1)} \xi_i}{|f_l| - |S(1)|}$$

ξ is the μ_l of previous *RTT*. Due to Lemma 4.3, we have $\mu_l \geq r(1)$ at any time, thus, $\xi \geq r(1)$. So,

$$\mu_l \leq \frac{r(1) \cdot |f_l| - r(1) \cdot |S(1)|}{|f_l| - |S(1)|} = r(1)$$

From Lemma 4.3, $\mu_l = r(1)$. So, at the end of the third *RTT*, the *advertised rate* for $l \in S(1)$ is the optimal rate $r(1)$. At the end of the third *RTT*, all of sources of flows crossing l are assigned with a *stamped rate*, which is equal to the optimal rate. At the fourth *RTT*, all of flows in $S(1)$ will be marked as soon as the first packet with new *stamped rate* arrives link l , and will remain marked as long as the set of flows unchanged. So, 4**RTT* is the upper bound time to reach global optimum rate for iteration i . **End of Proof**

Lemma 4.5 *In the following iterations, all flows in $S(i)$ will reach its optimal rate after some time.*

Proof Suppose all flows in $S(j)$, where $1 \leq j \leq i$, have reached their optimal rate already. Lets consider a reduced network $N(L, S)$, $L = \Lambda - \tilde{L}(i)$, $S = \Theta - \tilde{S}(i)$, where

$$\tilde{L}(i) = L(1) \cup L(2) \cup \dots \cup L(i).$$

$$\tilde{S}(i) = S(1) \cup S(2) \cup \dots \cup S(i).$$

$L(i+1)$, is the set of congested link and $S(i+1)$ is the set of all flows crossing at least one link in $L(i+1)$ in this iteration, iteration $i+1$. From the same argument in Lemma 4.4, we could prove that the iteration $S(i+1)$ will converge to $r(i+1)$ after at most 4^*RTT .

End of Proof

With Lemma 4.4 as the induction base and Lemma 4.5 as the induction process, the statement of the following Theorem 4.3 holds.

Theorem 4.3 *Given arbitrary initial conditions on the states of the link in the network, states of sources, destinations, the algorithm `distributed_global_optimal_rates` procedure (Table 4.2) will converge to the global optimal rates, as long as the set of flows, their demands and routes do not change.*

The proof presented above does have its limitation. In the proof, the RTT of each flow is assumed to be constant which does not capture the dynamic of queueing delays. The limitation comes from the mathematical induction methodology used in the proof. In order to use mathematical induction, the evolution of the system has to be discretized. The problem in the proof shares the same problem as those used in the proof of correctness of software. To my best knowledge, this is the best that we can expect so far.

4.2.3 FairShare Algorithm

FairShare implements, indirectly, the same actions as the distributed global fairness algorithm of [47]. The minimum rate over all “advertised” rates is implicitly enforced because the throughput of a TCP flow is only controlled (via scheduled losses) at the link in which FairShare has assigned the smallest rate for the given flow over all the links that the flow traverses. This is because FairShare does not victimize a flow multiple times (no multiple loss points) given its ability to identify bottlenecks and the behavior of non-greedy flows. The

process of determining the share allotment to a particular flow is as per the max-min fairness criterion, which is exactly the same criterion used in [47] at any bottleneck node. Hence, with the exception of replacing the explicit rate adjustment with implicit model-based rate control at a single point (link) for any given flow, the rest of a FairShare system behaves identically to the one described in [47].

Lemma 4.6 *FairShare is an alternative implementation of the `distributed_global_optimal_rates` procedure. (Table 4.2).*

Proof

Stamped Rate *Stamped Rate* at the source is not needed except at the first router along the path, since the minimum of the advertised rate along the path is the allocated rate of the source. FairShare collects the same information via measurement. Because an upstream bottleneck link may reduce the rate of a flow, a router measures the minimum of the rates available by the flow crossing through the upstream routers. Hence, `demandi` collects the same information that is explicitly collected and made available by the source in the second *RTT* iteration (See the proof of Lemma 4.4).

Advertised Rate The advertised rate, as seen in (Equation 4.5) is also calculated in the FairShare algorithm (Figure 3.3, line 11).

M-Consistency `flagi` encodes the outcome of the comparison between the `sharei` and `demandi`. Because we lack the ability for explicit information to be sent to the source, the reduction of the rate which would have been experienced by the fact that a flow belongs to Z (See the proof of Lemma 4.1) is now encoded by `flagi` (TRUE if it violates M-consistency, FALSE if not). Subsequent to this a loss will be inflicted to the TCP flows and the rate is reduced.

Thus, FairShare has implement all of the mechanisms in the `find_global_optimal_rates` procedure. It can be considered as an alternative implementation, without modifying packet headers. **End of Proof**

FairShare possesses a similar convergence feature as that of the `find_global_optimal_rates` procedure. A major difference between FairShare and `distributed_global_optimal_rates` procedure is that with the lack of explicit information, reducing of rate is enforced via losses

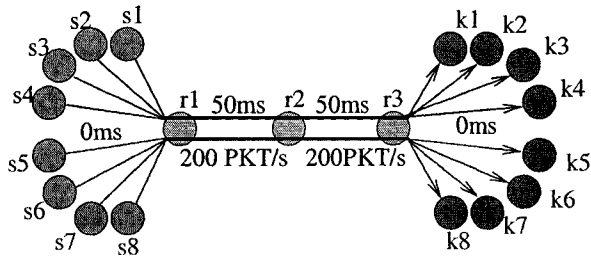
while an increase is not immediate (as it would be in `distributed_global_optimal_rates` procedure) but relies on the window inflation logic of a TCP flow under no losses. Reduction in TCP via losses does not steer the flow to a specific rate, so the window increase dynamics are essential to move to the intended long term average. Because of this, `FairShare` needs multiple RTT times before the desired allocation are reached. So one can replace the specific timing of Theorem 4.3 with claim that “eventually” the desired allocation is achieved (which can take more than $4 * RTT$).

4.3 Simulation Study

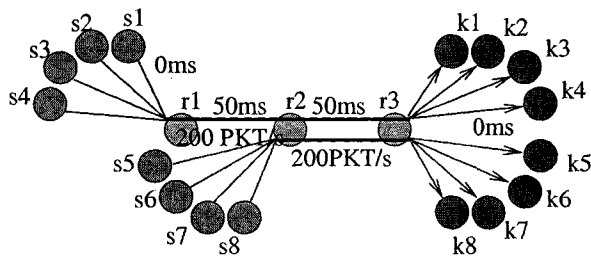
Early performance results of `FairShare` were presented in Chapter 4. In this section we demonstrate that `FairShare` quickly adapts to changes of network load caused by the arrival of new (and termination of old) flows and by variations in the demands of flows. Different network topologies are considered. The simulation experiments were conducted using ns-2 [6].

4.3.1 Experiment 1

In the first set of experiments, we investigate the impact of the bottleneck location on the performance of `FairShare`, assuming linear topology with multiple hops. The topology is depicted in Figure 4.3(a), which consists of eight source nodes, eight sink nodes and three router nodes. Eight TCP Reno flows are initiated from eight pairs of source and sink nodes. The bandwidth of link between router nodes is 200 packets/sec with delay of 50ms. In the first scenario, all eight flows are congested at the link r1-r2 (Figure 4.3(a)). In the second scenario all flows are congested at the link r2-r3 (Figure 4.3(b)). In both scenarios, the max-min rates of eight flows is expected to be 25. One aspect that is examined in these two topologies is that, in the second scenario, `FairShare` on r1-r2 identifies correctly that the first four flows are bottlenecked elsewhere (on r2-r3) and are not victimized with more losses at r1-r2. Indeed, in all cases, the goodput achieved by each connection was 25 packets per second, as an eight-way fair sharing would suggest. The smallest goodput value measured was 24.92 and the largest 25.08. That is, the desired behavior was achieved within a very tight range.



(a)



(b)

Figure 4.3: Topologies in, (a) scenario 1, (b) scenario 2.

4.3.2 Experiment 2

Our second set of experiments were conducted in the loop topology depicted in Figure 4.4(a). Four router nodes, r_1 , r_2 , r_3 and r_4 , form a loop. The links between router nodes are all of capacity equal to 100 packets/sec and of propagation delay equal to 10msec. Four TCP Reno flows, are initiated between the corresponding pair of source and sink nodes. The detailed paths for each flow are given in Table 4.3.

Flow	Path
1	$s_1 \rightarrow r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow k_1$
2	$s_2 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4 \rightarrow k_2$
3	$s_3 \rightarrow r_3 \rightarrow r_4 \rightarrow r_1 \rightarrow k_3$
4	$s_4 \rightarrow r_4 \rightarrow r_1 \rightarrow r_2 \rightarrow k_4$

Table 4.3: The paths of flows in Experiment 2.

In the experiments, we investigate the performance of our proposed FairShare policy in a dynamic environment. That is, the demands of some flows change in the process of the

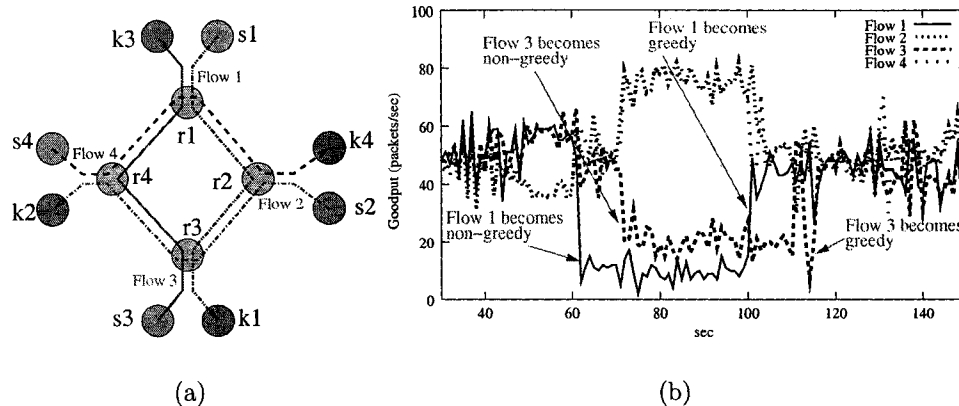


Figure 4.4: Experiment 2, (a) topology and (b) goodput.

experiments. The unit of time is seconds. Specifically, Flow 2 and 4 are always greedy, that is, they have infinite demands during the whole period of the simulation. Flow 1 is greedy from 0 to 60, then switches to having a limited demand equal to 10 packets/sec, and remains limited until time 100, at which point it resumes being greedy. In a similar vein, Flow 3 is greedy from 0 to 70, becomes non-greedy with a limited rate of 20 packets/sec, until time 110, at which point it becomes greedy again. All flows have the same RTT propagation delay (40msec).

The goodput of each flow is depicted in Figure 4.4(b). From 0 to 60 sec (Phase I), due to the uniform bottlenecks, every flow receives the same bandwidth, that is, 50 packets/sec. At time 60 and until time 70 (Phase II) the demand of Flow 1 drops to 10 packets/sec, link r4-r1 and link r3-r4 become first level bottleneck links. Consequently, Flow 3 equally share the capacity on r4-r1 and on r3-r4 with Flow 4 and Flow 2 respectively, at 50 packets/sec. The max-min rates of Flow [1-4] are 10, 50, 50, and 50, respectively.

Then at time 70 and until time 100 (Phase III), the demand of Flow 3 drops to 20 packets/s and the deadlock is broken. Bounded by r4-r1 and r3-r4, the throughput of Flow 4 and Flow 1 switch 80 packets/sec. The max-min rates of Flow [1-4] become 10, 80, 20, and 80, respectively. At time 100, and until time 110 (Phase IV) Flow 1 resumes to being greedy. The first level bottlenecks are now r1-r2 and r2-r3. The max-min rates of Flow [1-4] are 50, 50, 20, and 50, respectively. Finally Flow 3 resumes greedy at time 110 (Phase V). The evolution of the max-min rates is illustrated in Table 4.4. Figure 4.4(b) demonstrates that with a FairShare policy, flows determine and stabilize at the max-min rates in a dynamic

environment in a loop topology.

Flow	Phase				
	I	II	III	IV	V
1	50	10	10	50	50
2	50	50	80	50	50
3	50	50	20	20	50
4	50	50	80	50	50

Table 4.4: Per-flow bandwidth share in Experiment 2.

4.3.3 Experiment 3

In the third set of experiments, we investigate the impact of an interfering flow on the performance of FairShare. The topology (Figure 4.5(a)) is similar to the one in the previous experiment (Figure 2.1), except an interfering flow (Flow 5) shares link r2-r3 with Flow 1 and 2. With the presence of the Flow 5, link r2-r3 becomes the first level bottleneck, while link r4-r1 is the second level bottleneck. In the experiments, the demands of Flow 5 increase from 0 to 50. As it is shown in Figure 4.5(b), when the demand of Flow 5 is smaller than 33.33, the goodput of Flow 5 achieves its demands, while Flow 1 and 2 split the remaining capacity equally. As the demand of flow 5 is larger than 33.33, Flow 1,2 and 5 share the capacity of link r2-r3 equally. In both scenarios, Flow 3 and 4 are bounded by the second level bottleneck, on link r4-r1. In summary, with FairShare policy, flows detect the bottlenecks of different levels and achieve the max-min rates consistently.

4.3.4 Experiment 4

In the last experiment we conducted simulations with the most sophisticated configurations studied in [47]. The topology and link capacities are shown in Figure 4.6(a). Similarly to the study in [47], the object of this experiment is to investigate the response of our scheme to dynamic changes in a distributed non-trivial environment. The applications associated with flows are, again, of ftp style, that is, the applications always have data to deliver as long as the flows are active.

In this experiment, three levels of bottleneck links are presented. In the simulation, all five flows start at time 0 (Phase I). The transient portion of the results (0-30) have been

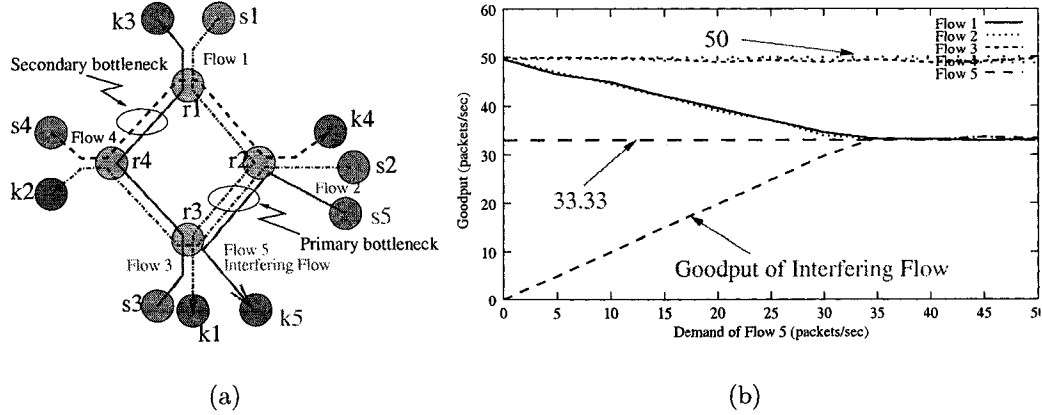


Figure 4.5: Experiment 3 (a) topology and (b) goodput.

removed. In Phase I, Flow 2, 3 and 4 share the first-level bottleneck link of 5 (r4-r6). Flow 1 is bottlenecked at link 1 (r1-r2). Flow 5 is bottlenecked at link 4 (s5-r4). Thus, the expected rates of Flow [1-5] by global max-min fairness standard are as follows: 40, 20,20,20, and 60, respectively.

Subsequently, Flow 3 terminates at time 100 (Phase II). Flow 2 and 4 share bottleneck 5 (r4-r6) equally. As in the previous phase, Flow 1 is bounded by link 1 (r1-r2), and flow 5 is bounded by link 4 (s5-r4). As a result, the expected max-min rates of Flow [1-5] are as follows: 30, 30,0,30, and 60, respectively.

At time 150, Flow 1 and 2 also terminate (Phase III). In this phase, only two flows, 4 and 5, are active. Consequently, Flow 4 is bounded by link 5 (r4-r6) and Flow 5 is bounded by link 4 (s5-r4). The max-min rates for flow[1-5] are 0, 0, 0, 60, and 60, respectively. Finally, Flow 1 resumes at time 200 (Phase IV). In this phase, the bottleneck links are link 3 (r3-r4) shared by Flow 1 and 4, and link 6 (r4-r5) shared by Flow 1 and 5. Consequently, the max-min rates for flow[1-5] are 50, 0, 0, 50, and 50, respectively.

In Table 4.5, the status of flows, the expected (analytically) max-mix rates of flows, and the measured goodputs from the simulation are presented. Figure 4.6(b) shows that with our FairShare, all flows quickly determine and stabilize at their max-min share.

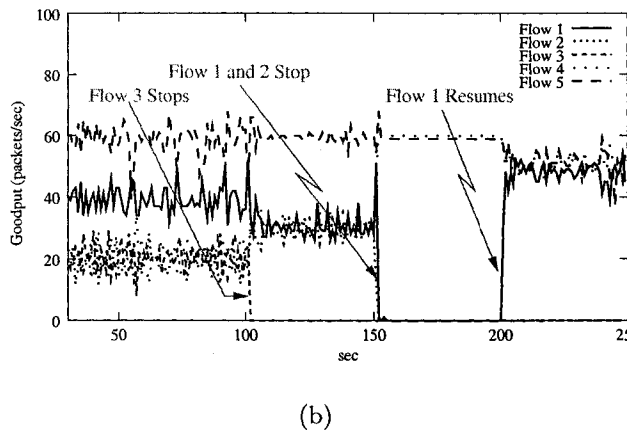
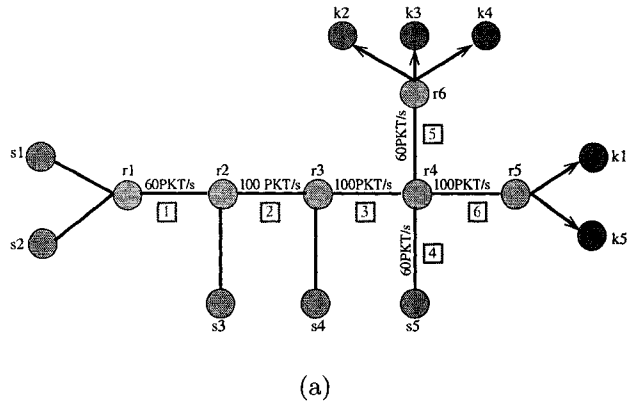


Figure 4.6: Experiment 4 (a) topology and (b) goodput.

4.4 Conclusion

What should be emphasized about the use of FairShare is the fact that, together with the totally distributed control for max-min fairness, it does not require complicated parameter tuning. Therefore, unlike schemes like RED whose ability to properly control flows lies in their accurate parameterization, the presented scheme controls all aspects of its behavior via self-contained measurement and control without user/operator intervention. To our knowledge, it is the first time that an AQM scheme is both capable of reaching a global objective as far as the TCP flow performance is concerned, and avoids the need for external parameterization. We have presented evidence of what we believe is the first time that an AQM scheme demonstrates capabilities to predictably force the entire network into a fairness objective. The solution is the synthesis of a per-link mechanism, in the guise of the

FairShare algorithm, and a totally distributed rate allocation which is an adaptation of [47]. Through simulation experiments, we have shown that the proposed AQM scheme converges to the globally max-min fair rates for various network configurations and in dynamic load environments. The FairShare policy appears to be self-stabilizing in the presence of dynamic network changes. The result of our investigation is crucial towards the practical applicability of global control algorithms for TCP flows in realistic environments.

Flow	Active?	Expected Goodput	Measured Goodput
Phase I (30–100sec)			
1	yes	40	39.40
2	yes	20	19.80
3	yes	20	20.20
4	yes	20	20.11
5	yes	60	58.98
Phase II (100–150sec)			
1	yes	30	29.19
2	yes	30	30.77
3	no	-	-
4	yes	30	29.10
5	yes	60	59.99
Phase III (150–200sec)			
1	no	-	-
2	no	-	-
3	no	-	-
4	yes	60	60.00
5	yes	60	60.00
Phase IV (200–250sec)			
1	yes	50	49.19
2	no	-	-
3	no	-	-
4	yes	50	50.70
5	yes	50	50.82

Table 4.5: Measured vs. predicted bandwidth distribution.

Chapter 5

TCP Lifetime & RTT Classification

5.1 Introduction

In the previous chapters we examined schemes that deal with long-lived flows. In the current chapter we will look into the separation of short- and long-lived flows in general. We will provide statistical study of real traffic in support of classification. Moreover, we will investigate classification schemes and ways of identifying the bandwidth demands of classes.

The rest of this chapter is organized as follows. In section 5.2, we will describe our study on real traffic to support lifetime classification. In section 5.3, the detailed scheme of identifying long-lived flows and their demands are presented. Section 5.4 presents evidence of supporting RTT classification. A RTT estimation method is given in section 5.5. In section 5.6, we present bandwidth allocation schemes based on simple rules and a more elaborate control scheme called DAS.

5.2 Lifetime Classification

The first aspect of the policies studied in this chapter is the reliance on a mechanism that is able to classify flows depending on their anticipated lifetime. We use a simple mechanism whereby all flows are assumed, when they start, to be short-lived until the timepoint when they exceed a certain number of transmitted packets, subsequent to which they are “upgraded” from short to long-lived. Clearly, considering a long-lived flow as short-lived for a short time period in the beginning of a connection is not as important to the outcome of the mechanisms presented here because it is unlikely that the TCP dynamics of the particular flow have reached equilibrium. We are concerned however with the potential of character-

izing a flow as long lived and then terminating shortly thereafter, that is, “false positive” problem. In this case, the control overhead of the resource allocation associated with the upgrade from short to long-lived would have been invoked in a wasteful manner. We note that previous studies [55, 56] did not show what constitutes a threshold between short- and long-lived flows or if it exists in all cases. Fortunately, the “false positive” problem is amortized by the nature of the heavy-tailed connection lifetime distribution. Once the short-lived flows are isolated subject to a “reasonable” threshold (e.g., a threshold larger than the average number of packets representing typical short-lived web transfers), the long-lived flows still possess a heavy-tailed behavior. Thus, a few of connections which carries the majority of the traffic is always identified as long-lived and stay in the system for a long period of time in the future. Previous studies, e.g., [50] as well as our analysis of traces of WAN traffic collected by the University of Auckland [53] indicate that the connection lifetimes, especially once short-lived outliers are isolated, are well modeled by a Pareto distribution.

The cumulative density function (CDF) for a Pareto distribution is

$$Pr\{Z < x\} = 1 - \left(\frac{\beta}{x}\right)^\alpha$$

Thus, we have

$$Pr\{Z > x\} = \left(\frac{\beta}{x}\right)^\alpha$$

and

$$Pr\{x_1 > Z > x_2\} = \left(\frac{\beta}{x_1}\right)^\alpha - \left(\frac{\beta}{x_2}\right)^\alpha$$

if $x_1 = x_2 + \epsilon$, ϵ expressing our tolerance to “false positive” if x_2 is the threshold, then $Pr\{x_1 > Z > x_2\} \rightarrow 0$ as $x_2 \rightarrow 0$. That is, the larger the threshold the less likely the “false positives”. Statistical studies of Internet traffic, e.g., [13], show that most TCP connections are short-lived, but a small number of connections have lifetimes spanning hours or even days and carry a high proportion (typically 50% to 60%) of the total carried bytes. We use five traces, the first two traces collected on December 8th 1999, starting at 12:58:38, the next two collected on January 25th 2000, starting at 14:36:40, and the last one collected on January 28th 2000, starting at 16:04:41 (Table 5.1).

In our statistical study, we use the SYN and FIN flags in the TCP handshake protocol (Figure 5.2) as indications of start and end of a particular TCP connection. The studied

Trace	Duration	Packets	Long-Lived Classification Threshold								
			50 packets			100 packets			200 packets		
			α	β	Short	α	β	Short	α	β	Short
1	2:18:59	2,727,535	1.74	3.00	50.79%	1.59	2.88	56.56%	1.51	2.81	61.21%
2	2:18:59	16,191,749	1.77	3.49	56.80%	1.86	4.01	61.94%	1.59	3.37	65.30%
3	3:11:28	3,626,938	1.99	4.00	55.65%	1.82	4.02	62.42%	1.54	3.38	67.98%
4	3:11:28	7,278,421	1.76	3.31	58.46%	1.64	3.24	64.08%	1.57	3.20	68.43%
5	24:01:22	10,554,879	1.79	3.50	55.85%	1.86	4.01	61.71%	1.86	4.01	66.65%

Table 5.1: Pareto distribution parameters (α, β) capturing the empirical distribution.

traces might include HTTP traffic, ftp traffic and other type of traffic as well, but only TCP connections are extracted and studied in our analysis.

As can be seen in (Figure 5.1), the empirical traces fit Pareto distribution closely. In the plots, x-axis represents the lifetime of the observed TCP connections in packets, and y-axis stands for the frequency of the TCP connections of the specific lifetime over the total observations. The empirical trace fits well with Pareto distribution in all three threshold values: 50, 100 and 200, respectively. Here, a threshold is the criteria of differentiating long- from short-lived TCP flows. A TCP flow with packets exceeding the threshold is considered a long-lived TCP flows.

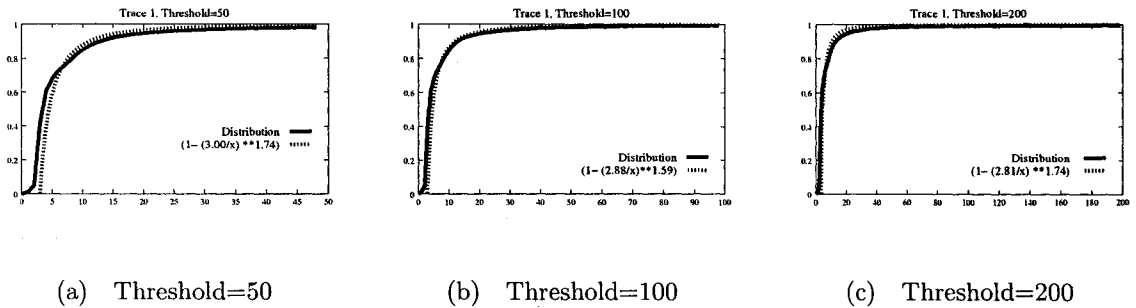


Figure 5.1: Pareto Distribution fit on Trace 1 for various thresholds.

The parameters of the specific Pareto-distribution that fit the empirical distribution of the lifetime of TCP connections are obtained by Chi square test ¹. By the definition of the

¹Chi square test is used to analyze the difference between the hypothesized distribution and the actual observed data. The test takes the form of $\chi^2 = \sum_{i=1}^K \frac{(O_i - E_i)^2}{E_i}$, where O_i is the frequency of the observed events in the sample data, E_i is the frequency of expected events and K is the number of categories.

Pareto distribution, the portion $x \in [0, b)$ is not included in the fitting processes. As can be seen in the plots (Figure 5.1.c), the Pareto distribution fits better with TCP connections of relatively longer lifetime. Since our interest is drawing a line between long-lived TCP connections and short-lived connections, fitness of short-lived TCP connections has only limited impact on our schemes and could be ignored.

For example, the CDF of load/lifetime of Trace 1 (Table 5.1) can be modeled by Pareto distribution

$$Pr\{Z \leq x\} = 1 - \left(\frac{3}{x}\right)^{1.74}$$

The percentage of flows classified as long-lived with threshold as 50, 100 and 200 are 0.748%, 0.224% and 0.067%, respectively. Consequently, the percentage of TCP flows last between 50 and 100, 100 and 200 are 0.524% and 0.157%, respectively. Thus, due to the nature of Pareto distribution, the “false positive” problem is trivial if we set the threshold value large enough. And larger threshold value could minimize the “false problem” further.

However, minimizing “false positive” is not the only factor affecting our choice of threshold value. The currently suggested threshold value should be several times of average traffic load of HTTP sessions, because HTTP constitutes the majority of short-lived traffic. We therefore argue that a reasonable threshold for separating short from long-lived flows is 50 packets. Note that according to this threshold, the qualified long-lived connections are typically less than 1% of the total TCP connections. For example, in the case of trace 1, this corresponds to 1746 long-lived flows during 8000 seconds, or about one new long-lived connection every 4 seconds, a reasonable rate at which to perform special control processing.

5.3 Identifying Long-Lived TCP Flows

Let us assume that a TCP flow starting to send data through the router is initially a short-lived and competes with the rest of the short-lived flows in the bandwidth pool C_s . After a period of time, if the flow is still active, it is upgraded to long-lived and enter into competition with the rest of the long lived flows, in the bandwidth pool C_l . The threshold of dividing long- and short-live flows is set to a small multiple of a typical delay of a short Web document transfer, i.e., on the order of a few seconds. Specifically, we consider a simple mechanism for determining long-lived flows. Incoming TCP packets are hashed on the values of source-destination addresses and ports. Upon encountering the first packet of a new flow, the

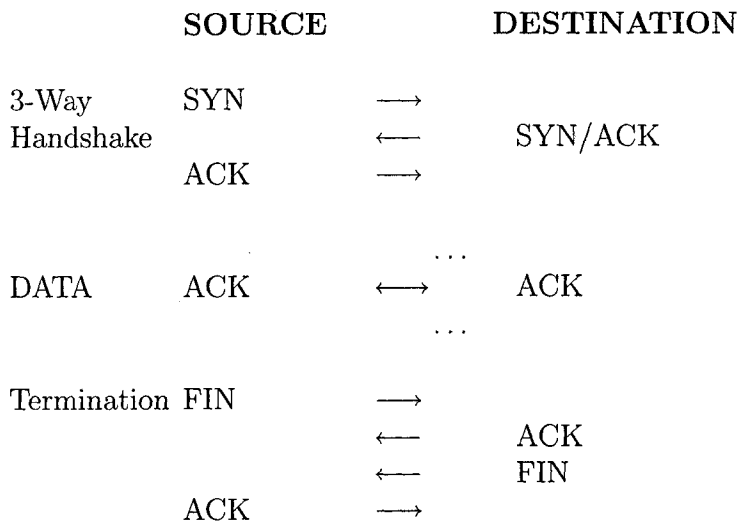


Figure 5.2: The TCP Handshake Exchanges

corresponding hash table entry stores the current time. Periodically, e.g., once every five seconds, the entries are checked for any flow that has been active for more than the threshold amount of time. This flow becomes a long-lived TCP flow. In order to avoid legitimizing as long-lived flows those flows that have sent no data in the recent past. A scheme such as the one proposed here was initially proposed by Morris [38]. According to this scheme, upon arrival of a packet, a bit flag (associated with the flow's hash entry) is set. A (separate) periodic process clears the flags periodically. In this case, the upgrade to long-lived flow can occur only if both the test of sufficient time since the start of the flow has elapsed *and* if the bit flag indicates that the connection was recently active.

5.4 RTT Classification

The second aspect of our classification schemes involves identifying the RTT of flows. The RTT measurements are also used as input to the FairShare scheme in order to help determine the per-flow target window size for a flow to be able to claim its fair share of the link's bandwidth. We decided to group TCP flows of *roughly* similar RTTs together in separate classes in order to suppress the negative interaction of TCP flows with drastically different RTTs. Such RTT-based classification scheme is supported by a previous study [57]. Our

study of the same traces as the ones we used in the previous section shows that RTT values of TCP flows are indeed widely distributed over several orders of magnitude (Figure 5.3). Likewise, an accurate model and simulation study should opt for several orders of magnitude of RTT values when synthesizing TCP traffic workloads. Such simulations have already been provided in earlier chapters and similarly, we will use traffic with wide spectrum of RTT in later chapters.

5.5 RTT Estimation

A necessary ingredient of FairShare is information about the RTT of individual flows. In the most proposed schemes, the RTT estimation is performed with both direction traffics at the end point. Such mechanisms includes the method of measuring RTT utilizing the standard ping utility to collect long-term statistics and the approach used in XCP [49] to introduce extra TCP headers, with which the source node is forced to disclose its local estimate of RTT. These approaches are not particularly appealing for three reasons. First, we do not wish to produce estimates that follow the queue dynamics, that is, we are interested in the inflexible propagation time component. Second, we do not wish to impose a continuous overhead of monitoring the RTT to the router. Finally, we do not expect that the endpoints will modify their implementations of TCP to include RTT information in the headers. The approach we used is to extract RTT information by sampling the intervals between the first few packets of a new TCP flow. Such an approach can perform the RTT estimation with one-way traffic, that is, it does not require traffic of both directions passing the same router. Recall that when a TCP connection starts, a *three way handshake* (Figure 5.2) takes place. First, the client sends a packet with SYN flag set. The server responds with a packet with ACK and SYN set. Then the client responds with an ACK packet. Data packets are to be transferred after the handshake. To terminate a flow, the client sends a FIN packet. The server sends an ACK followed by a FIN packet. The client responds with a FIN packet. We sample RTT three times. The first sample is the interval between SYN and ACK of three-way handshake. The second sample is the interval of last packet of three-way handshake and the first data packet. The third sample is the interval between the first data packet and the second data packet. We take the minimum (to isolate startup and queueing effects as much as possible) of the samples and use it as the RTT for the particular flow. The technique is similar to one

proposed in [48]. [48] also verifies the accuracy of such RTT estimation technique. Thus, we conclude that the instantaneous RTT we measured is a low cost estimation with decent accuracy.

5.6 Classification Schemes

5.6.1 Classification Schemes Based on Simple Rules

It has been observed that short-lived flows are at a disadvantage when competing against long-lived flows. Furthermore, TCP's unfairness is also due to different RTT values of different TCP flows. It is natural to investigate whether the classification can be performed based on a simple rules, and how it can be used in terms of separating the service received by each class. It is reasonable that the two causes of unfair performance between competing TCP flows, that is, connection lifetime and RTT differences should be dealt with explicitly in order to produce a well-regulated network. Lifetime-based classification schemes have been advocated [32, 15, 16] to protect short-lived TCP flows from the negative impact of long-lived TCP flows. However, they have not been combined with RTT classification schemes. The first set of classification schemes is composed of lifetime classification scheme and RTT classification scheme. The classification mechanisms are essentially (a) a threshold-based classification between short- and long-lived flows and (b) a rough separation of flows in a few classes of similar RTT values.

We are also interested to compare the classification schemes to the performance of DropTail and RED. We propose six combinations of classification scheme based on lifetime and RTT and queue management policy like DropTail and RED. We defer the evaluation experiments to next chapter.

	Classification	Queue Management
LFT-DT	Life time	DropTail
LFT-RED	Life time	RED
RTT-DT	RTT	DropTail
RTT-RED	RTT	RED
SIX-DT	Lifetime + RTT	DropTail
SIX-RED	Lifetime + RTT	RED

5.6.2 Dynamic Bandwidth Allocation for Lifetime-Based TCP Classification

Previously, bandwidth allocation schemes of TCP flows have been studied extensively, e.g., in [60, 61, 62]. Lifetime-based classification schemes advocated [15, 16, 32] the protection of short-lived TCP flows from the negative impact of long-lived TCP flows. However, the synthesis of lifetime-based classification and bandwidth allocation are, to the best of our knowledge, considered together for the first time. We also consider the impact of RTT in the unfairness between TCP flows. However, it is of secondary importance because it is used only in our FairShare policy to improve the fairness between long-lived flows, a feature that can be added but is not crucial to the short- vs. long-lived flows separation. Another reason for proposing dynamic allocation is the nature of the setup process of short-lived TCP flows, which can be characterized as a stochastic process with fairly well defined arrival rate which is independent of control policies at the intermediate routers [50]. A control policy based on the number of active TCP flows (“call” level policy) is likely to better match the medium to long term load fluctuations compared to a policy based on the packet (“packet” level) arrival process. In fact, the latter has been repeatedly shown to be accurately modeled as a self-similar process, which exhibits, among other things, infinite variance, complicating efforts for long term predictability. With the proposed call/connection-level dynamic allocation policy, an intermediate router measures the number of active TCP flows traversing each link, and dynamically adjusts the fraction of bandwidth for the class of short lived flows every time a significant change in the number of TCP flows traversing the links is detected. With the lifetime-based classification in place, the next step is to provide a differentiated service scheme whereby the available link bandwidth is allocated between the two classes: short and long-lived flows. The per-class bandwidth allocation is to be dynamically adjusted, on a periodic basis. The adjustment period is denoted by t_a . Therefore, it is assumed that the link scheduler is at least capable of providing two service classes with distinct bandwidth allocations and per-class queues. A Weighted Fair Queueing implementation with two (one per class) DropTail queues fits our assumptions, and it is widely found in modern router equipment. Within the bandwidth pool allocated for the long-lived flows we can consider the application of additional schemes to provide fairness between multiple long-lived flows because, as we have argued, the fairness criteria make sense for long-lived flows more than

they do for short-lived flows. In particular, our performance study considers the application of the FairShare [43] scheme on the long lived flows. We defer any further discussion about FairShare since it has been covered elsewhere [43] and pay more attention to a dynamic allocation scheme (DAS) of bandwidth between the two classes. The operation of DAS considers the *aggregate* traffic demands of the short-lived flows. We distinguish two basic sources of traffic dynamics: the dynamics of the demand for new connections (number of active connections, i.e., “call-level” demands) and the dynamics of the arriving packet stream that belong to various TCP flows and obey the congestion control dynamics of TCP (packet arrival rate, i.e., “packet-level” demands). DAS, depending on how it is implemented (see following subsections) may need to monitor both dynamics. However, the changes to the bandwidth allocation are performed in response to changes of the number of TCP connections. That is, the intention of DAS is to follow the call-level dynamics of TCP flows. The traffic load, measured in connections per unit of time (minus the small fraction of long-lived flows instantiated per unit of time) provides a potent metric for the anticipated load because it comes close to representing the user behavior. That is, the number of connections set up per unit of time are predominantly due to the user activity, and are thus an external demand parameter imposed on the network. DAS is composed of three elementary mechanisms. The first is keeping track of the number of flows traversing a link, either by approximating their instantaneous count, or by keeping track of the rate at which new ones are established and old ones terminated. The second mechanism translates flow level dynamics to packet traffic dynamics, by utilizing off-line measurements that characterize the length of the “typical” short-lived TCP flow and which, incidentally, closely matches the “typical” short HTTP transfer. The third, and final stage is deciding whether the measured load should trigger a re-allocation (and by how much) of the bandwidth allotted to the short and long-lived flows. That is, it translates a demand measurement to (re-) allocation action. For the rest of the document we assume that if the bandwidth allotted to short-lived flows is C_s and the link bandwidth is C , then the residual, $C - C_s$, is entirely allocated to the long-lived flows. We note that this is a technical assumption we decide to make to simplify presentation of our scheme. It is conceivable that $C - C_s$ should not be allowed to be arbitrarily small, in order to avoid starvation of the class of long-lived flows. Moreover, additional classes may need to be defined to handle other traffic flows, e.g.,

UDP traffic. We will consider three alternatives to implementing DAS. Each one represents a different way of keeping track of the connection load and a different means of interpreting load demands to bandwidth allocation. More importantly, each one represents a different overhead in terms of state information and a different degree of inaccuracy introduced via the load measurements. However, as the evaluation section illustrates, all three provide an improvement over DropTail or RED. While it is true that the improvement relies on being able to exploit a-priori knowledge of what is a “typical” load by a short-lived TCP flow, our claim is that the statistics of TCP flows are now much better understood. Clear indication to this end is the abundance of traffic studies for Internet traffic and the development of ever more accurate measurement techniques. Moreover, network operators are capable of collecting such statistics and to revise their estimates on live systems. We will present our experiment results in the next chapter.

5.6.3 DAS-BV

The first version of DAS we consider is based on the Bit Vector (thus DAS-BV) technique of determining the number of active flows crossing a network link. The technique was proposed by Morris [29]. According to this scheme, every packet going through the link is parsed. Flow identification (`flowid`) is uniquely generated via a hash function applied on the source and destination addresses and ports. Packets with the same `flowid` are considered to be of the same flow even though the hash is not perfect and hash collision could occur. The state of each flow is captured by a single bit. It is an indicator of whether the flow was “active” in the recent past. A bit vector with thousands (to tens of thousands) of entries is maintained for this reason. The hash function value is in fact the index value to the bit vector, used to access the bit associated with a flow. If a flow has sent a packet in the recent past, its corresponding bit is set. The number of 1s is read out and the bit vector is reset (all entries cleared to zero) every t_c time units. We note that the value t_c , should be long enough to capture all of the competing flows even if they are not active in the short run, but also sufficiently short to allow quick adaptation to the changing dynamics. The basic rule of thumb followed in our work is that t_c should be of the order of the worst case round-trip time delay we expect to find in the network, i.e., in the order of a few seconds. We observe trends of demand changes, via an exponential averaging scheme. If `flowCount` denotes the

estimator for the number of flows, then at the adjustment instant i , it is calculated as:

$$\text{flowCount}_i = \gamma \times \text{count}_i + (1 - \gamma) \times \text{flowCount}_{i-1}$$

where count_i is the most recent measurement as per the bit vector scheme. In order to reflect the relative error-prone nature of the bit vector counting scheme, γ is to be set to a small value, to smooth spikes (in particular downward spikes due to some flows not being active for some RTTs) of the count samples. The adjustment of bandwidth allocation is performed at an interval much larger than t_c , i.e., $t_a \gg t_c$. The gist of the scheme is to start with a bootstrap value provided by off-line measurements, which reflects the typical average load due to short lived flows and track the changes around this average value. Consequently, the traffic demand, demand_i , is determined at adjustment instant i by:

$$\text{demand}_i = \text{demand}_{i-1} \times \frac{\text{flowCount}_i}{\text{flowCount}_{i-1}}$$

whereby demand is bootstrapped to the average demand (in packets per second), and flowCount_{i-1} represents the estimate of flowCount at the previous adjustment instant (t_a time units ago). The pseudocode of the algorithm executed by DAS-BB upon packet arrival is shown in Figure 5.4. In summary DAS-BV is based on the premise that the total traffic demands are proportional to the number of active flows. Thus, tracking the number of active flows, provides the information we need to adjust the bandwidth allocation. Clearly, the task is both sensitive to how the demand values are bootstrapped to begin with, as well as the inherent problem of how new long-term trends make necessary the adjustment of the anticipated average load on a medium term basis. The study contained herein should be seen as a validation of the short-term adaptability of DAS-BV and not its long-run (weeks to months) efficacy.

5.6.4 DAS-ED

In the second alternative implementation of DAS, we determine the bandwidth allocation based on the Explicitly estimated flow Demands (ED). That is, instead of proportional allocation relative to a bootstapped value, we directly measure the arrival rate (in packets per second) from the aggregation of all short-lived flows. The measurement includes a small error factor due to flows that, in retrospect, will be upgraded to long-lived flows. We then interpret the measured arrival rate as demand. However, we stop short of increasing the

bandwidth allocation when the packet arrival rate increases. The reason is that TCP, by its very nature, attempts to congest the links it traverses if its demands is greater than the available bandwidth. Thus, instead of providing more bandwidth immediately, we take into consideration whether the number of flows starting or terminating has changed in the meantime. In short, an increase of packet arrival rate, accompanied by no change in the number of flows active on the link, can be safely attributed to the TCP dynamics and no reallocation of bandwidth takes place. Bandwidth re-allocation takes place when it is followed by an increase of the rate at which new flows are being set up. Conversely, we do not decrease the allocation of bandwidth when the packet arrival rate drops, unless it is followed by a decrease in the number of flows. In other words, we attempt to closely track the connection-level dynamics by monitoring flow setup and termination events. We subsequently change the bandwidth allocation if the direction of the change (increase, or decrease) is in agreement with the derivative of the flowcount process (positive, or negative). Otherwise, we maintain the current allocation.

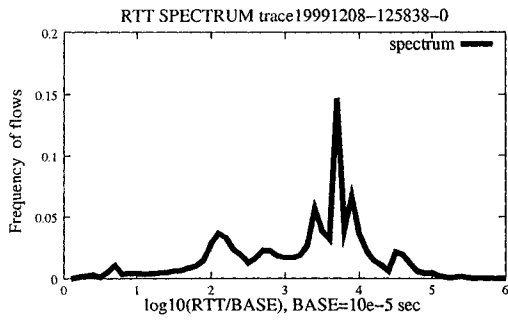
Technically, small rate fluctuation of the packet arrival rate process are inconsequential and can very well be the result of flows exploiting their TCP dynamics but bottlenecked at links on upstream routers. We ignore such fluctuations, and consider only packet rate changes that are above a threshold (in the experiments: +/-10%) relative to the current allocation. Moreover, the reliance on a fairly accurate estimator for the `flowCount` derivative, renders the inaccurate Bit Vector technique unattractive. Instead, the change in the number of active flows within an adjustment interval is derived directly by monitoring the start and termination of TCP connection, which is extracted from the flow identification number (`flowid`), and the flags, SYN and FIN. If, within an adjustment interval, the newly started flows outnumber recently terminated flows, we draw the conclusion that the number of active flows increases and the router will allocate more bandwidth if indeed the measured packet rate demand has increased. The inverse occurs when the number of flows increases and the packet arrival rate decreases. Figure 5.5 provides the outline of DAS-ED. Note that the demand now closely follows the `observedRate`, subject to the conditions stated above. That is: $\text{demand}_i = \text{observedRate}_i$. The cost of implementing the SYN and FIN identification is somewhat higher than the Bit Vector, since packets with SYN or FIN set need to be treated as special and the case of SYN and FIN retransmissions for the same flow needs

to be considered. However, the same hash-based approach can be used. We will assume that, a new SYN (or SYN/ACK), except for re-transmissions of the same, increments the `#newStart` counter. Likewise, a new FIN (or FIN/ACK), except for re-transmissions of the same, increments the `#newTerm` counter. The counters, after they are consulted, are zeroed at each adjustment instant. Similarly, the `arrivals` counter counts the packets (more accurately, the bits) arriving in the short-lived flows within the t_a period.

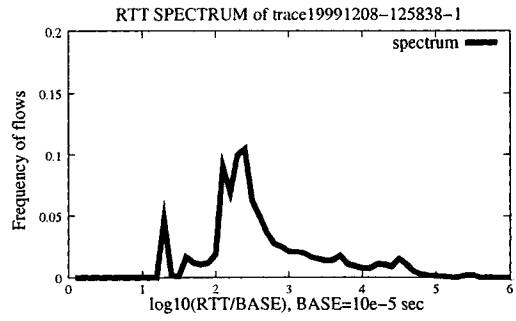
5.6.5 DAS-EL

In the third alternative, we design an Explicit-Load (EL) scheme, which tackles the allocation problem in terms of expected packet rate. The expected rate of packets is estimated by the product of `flowCount` and the average load of short-lived TCP flows `AVGLoad`. In this case, the assumption is that `AVGLoad` is obtained via independent off-line studies on empirical traces. `AVGLoad` essentially represents the average number of packets per flow traversing the link per adjustment interval, t_a . Since our scope is limited to short flows, the rate of packet arrivals is not dependent on the link capacity, because the flows never get close to expanding their window to high enough values. Rather, the value of `AVGLoad` is dependent on the average number of new connections established per unit of time and the average RTT of the mixed short-lived flows. In the DAS-EL scheme (Figure 5.6), the algorithm of detecting the number of active flow is similar to that of DAS-BV scheme (Figure 5.4), but with a different parameter value for γ (closer to 1.0). The allocated bandwidth is adjusted based on the expected packet rate. We note that the scheme presents the shortcomings of DAS-BV in that both `flowCount` is inaccurately estimated, and the fact that `AVGLoad` needs to be determined off-line with sufficient accuracy. Still, it is capable of directly expressing the load demands as a fraction of the link bandwidth, C . In two of the above scheme, δ is introduced as a sensitivity factor controlling what is the level of change of rate or flow count that is deemed “significant” to render bandwidth allocation adjustment necessary. In the following evaluation section, we have also added two other factors. First, the bandwidth cannot be allocated at arbitrarily small increments, but in tenths of the link bandwidth, C . Moreover, to avoid starvation of the long-lived flows, $0.1 \times C$ is reserved at all times for long-lived flows. That is, the bandwidth demand of the short-lived flows is at all times bounded above by $0.9 \times C$. That is, we allocate demand’ bandwidth for the short-lived flows,

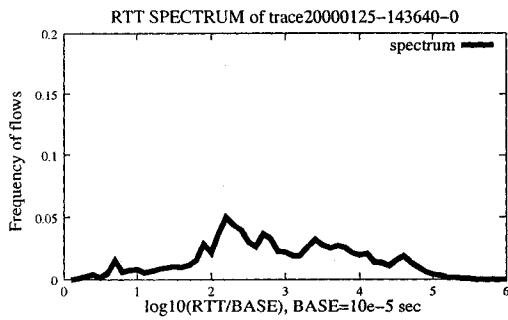
where $\text{demand}' = \min(0.9 \times C, \text{demand})$.



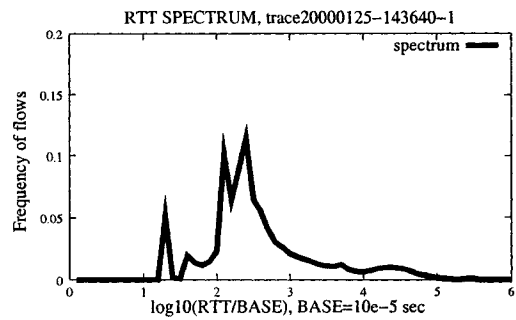
(a) Trace 1



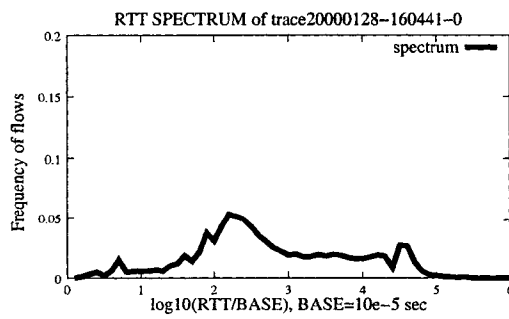
(b) Trace 2



(c) Trace 3



(d) Trace 4



(e) Trace 5

Figure 5.3: RTT Spectrum: Y axis, Frequency, X axis, $\log_{10}(\text{RTT}/10^{-5})$

`upon_packet_arrival(p):`

```
1. now ← time();
2. if (now > nextAdjustment) then
3.   if ((flowCount > old_flowCount×(1+δ)) OR (flowCount < old_flowCount×(1-δ))) then
4.     demand = demand ×  $\frac{\text{flowCount}}{\text{old\_flowCount}}$ ;
5.   endif
6.   old_flowCount = flowCount;
7.   nextAdjustment=nextAdjustment + ta;
8. endif
9. if (now > nextUpdate) then
10.  count=sumBitVector();
11.  resetBitVector();
12.  flowCount = count×γ + flowCount×(1-γ);
13.  nextUpdate = nextUpdate + tc;
14. endif
```

Figure 5.4: DAS-BV `upon_packet_arrival()`.

`upon_packet_arrival(p):`

```
1. now ← time();
2. arrivals++;
3. if (p.SYN) then #newStart++; endif
4. if (p.FIN) then #newTerm++; endif
5. if (now > nextAdjustment) then
6.   observedRate = arrivals / ta;
7.   if ((observedRate>demand×(1 + δ)) AND (#newStart>#newTerm)) then
8.     demand = observedRate;
9.   endif
10.  if ((observedRate<demand×(1 - δ)) AND (#newStart<#newTerm)) then
11.    demand = observedRate;
12.  endif
13.  #newStart = #newTerm = arrivals = 0;
14.  nextAdjustment = nextAdjustment + ta;
15. endif
```

Figure 5.5: DAS-ED `upon_packet_arrival()`.

upon_packet_arrival(*p*):

1. *now* ← **time**();
2. **if** (*now* > nextAdjustment) **then**
3. demand = flowCount × $\frac{AVGLoad}{C}$;
7. nextAdjustment = nextAdjustment + *t_a*;
8. **endif**
9. **if** (*now* > nextUpdate) **then**
10. count = sumBitVector();
11. resetBitVector();
12. flowCount = count × γ + flowCount × (1 - γ);
13. nextUpdate = nextUpdate + *t_e*;
14. **endif**

Figure 5.6: DAS-EL upon_packet_arrival().

Chapter 6

Performance Investigation of Classification Schemes

6.1 Introduction

This chapter is centered around experiments that investigate the proposed classification schemes described in the previous chapter. The first classification scheme we investigated is a dynamic bandwidth allocation scheme which we term DAS (simply standing for Dynamic Allocation Scheme). Allocating bandwidth between the two classes is performed in such a way that it reflects the demands of short-lived flows. The load demands of short-lived flows are inferences based on measurements of the number of newly starting (SYN/SYN-ACK setup phase) TCP connections multiplied by the typical TCP connection length (in terms of bits) which can be extracted either off-line or, hardware permitting, on-line over longer periods of time. The investigation of other classification schemes based on simple rules is presented as well.

6.2 Evaluation of DAS

6.2.1 Simulation Setup

In this section, we present the proof-of-concept simulation based on the `ns-2` [6] simulator. To reflect the heterogeneity of Internet traffic, traffic generated in our simulations is a mixture of TCP flows with various lifetime and RTT values. The topology used in our simulations is the familiar single-bottleneck “dumbbell” topology with the bottleneck link servicing 100 packets/sec. Source nodes are classified into three sets with different propagation delay,

Class_I, Class_II and Class_III. The average propagation delay of each class is denoted as RTT_i . The disparity of the RTTs is described by a single parameter called RTT_Ratio which defines the ratio of the average propagation delay of the adjacent classes; the relationship of propagation delays of different classes can be described as $RTT_Ratio = \frac{RTT_{III}}{RTT_{II}} = \frac{RTT_{II}}{RTT_I}$. The propagation delay of a particular connection in the simulation is randomly generated, which is between -10% and +10% of the average RTT of the Class it belongs to. For example, when we set up a connection between the Class_II node s_i and k_0 , the RTT (propagation delay) of the connection is a random number between $0.9 \times RTT_{II}$ and $1.1 \times RTT_{II}$. Both short and long-lived TCP flows are simulated. Long-lived flows start from the very beginning of simulation and can last forever unless they are shut off by the competing TCP flows due to multiple timeouts. Nine long-lived flows are initiated in total, three for each Class. In our simulation, short-lived TCP flows are emulated by ftp connections with a pre-configured small load. The value of the load is a random number with an average of 25 packets per flow. Short-lived flows stay in in the network until they finish transmission. Short-lived flows start randomly during the entire simulation period uniformly. The load of short-lived flows in the simulation is indicated by a parameter, which is referred to as the number of short-lived flows started per second. For example, if we run a simulation of 100 seconds with 2 short-lived TCP flows starting per second, we have 200 short-live flows and 9 long-lived flows over the entire simulation period. In our study, any flow will not restart once it is shut off by other flows. All TCP flows in this study are TCP Reno.

In our experiments, the buffer is split equally between the two classes equally (21 packets per class). When a DropTail or RED configuration is simulated, the queue size is the sum of the queue sizes of the two classes (for a total of 42 packets). TCP flows are divided into three groups according to their round trip propagation delays. We varied the RTT_Ratio from 2 to 10. The two classes are scheduled by the WFQ policy, according to weights that reflect the demand calculated, as outlined in the previous section. Adjustments of the bandwidth allocation are restricted in multiples of 10% of the link bandwidth. At a link capacity of 100 packets per second, and 25 packets per flow, the link reaches close to saturation when the rate of new short-lived connections initiated per unit of time is close to 4 new flows per second.

In order to allow for a reasonable fraction of long lived flow traffic, the rate at which

new short-lived flows arrive is set in the simulations to 1.2 new flows per second. To create a response to a transient overload which will be used to evaluate the responsiveness of the proposed schemes, we increase the arrival rate of new short-lived flows to 4.2 new flows per second on the 100th second of the simulation. The arrival rate of flows remains at this overload value until the 120th second of the simulation. Subsequently, the arrival rate drops back to 1.2 new short lived flows per second. Each of the three DAS schemes, as well as DropTail and RED were compared. In the case of DAS-BV, the parameters were set to: $t_a = 5$ sec., $t_c = 0.5$ sec., $\alpha = 0.05$, and $\beta = 0.1$. In the case of DAS-EL, $\alpha = 0.9$ indicating that in the exponential averaging of flowCount, the more recent measurements are considered more valuable in determining the mean. This is allowable because in DAS-EL it is not the magnitude of the difference of flowCount versus its previous estimate that matters, but rather its most recent absolute value.

6.2.2 Experiment Results

The results that summarize our findings in the best way are shown on Tables 6.1 and 6.2. They correspond to the same scenario (with the overload between 100 and 120 seconds) but at two different RTT_Ratio values, representing (for RTT_Ratio=2) the RTT mix of traffic at a router in a metropolitan area network and (for RTT_Ratio=10) the RTT mix of the traffic crossing a continental backbone router. The improvement in the average response time of short-lived TCP flows in comparison to that over DropTail can be clearly seen in Table 6.1. The response time for a short-lived flow is calculated as the interval from the timepoint that the first packet of the flow is transmitted, until the last packet of the flow is delivered to the destination. What we present in the tables is the average response time over all the short-lived flows during the entire simulation (except for the first 20 seconds because they constitute our simulator warmup stage).

It should be clear that some of the short-lived flows are initiated when the link is saturated by the rate at which new short-lived flows arrive. Under DropTail or RED we would expect that under such peak load the response time of the flows will increase as they attempt to cross a congested link. Indeed both the DAS schemes as well as DropTail and RED demonstrate such behavior. The difference lies in the fact that DAS adapts to the increased load by increasing the bandwidth allocated to these flows. Hence, it quickly exploits bandwidth

Scheme	Short Lived Flow Repines Time		Long Lived Flow Goodput	
	Average	Variance	Average	Variance
DAS-BV	3.42	18.01	34.58	526.97
DAS-ED	5.29	35.72	34.74	488.59
DAS-EL	3.74	22.53	34.64	492.61
DropTail	7.66	80.91	34.53	340.60
RED	5.99	49.58	34.48	376.12

Table 6.1: Short lived flow response times in seconds (RTT_Ratio=2).

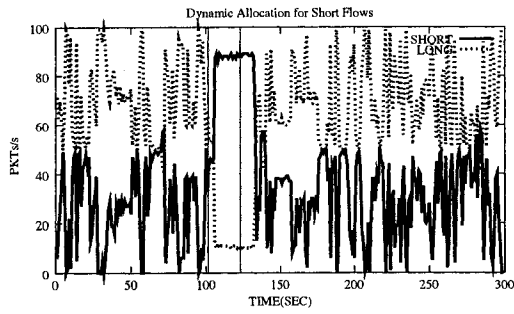
Scheme	Short Lived Flow Response Time		Long Lived Flow Goodput	
	Average	Variance	Average	Variance
DAS-BV	5.55	41.61	34.44	465.22
DAS-ED	7.57	69.60	34.48	412.86
DAS-EL	5.95	51.03	34.31	405.99
DropTail	14.35	296.37	33.93	290.64
RED	9.31	132.86	33.99	365.36

Table 6.2: Short lived flow response times in seconds (RTT_Ratio=10).

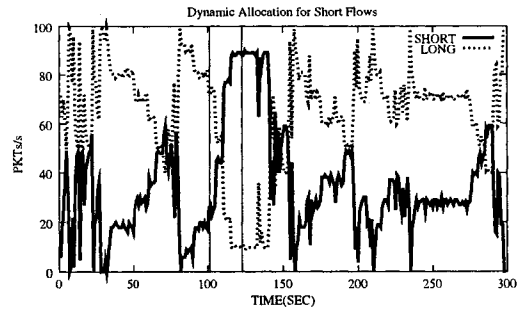
reserves by restricting the bandwidth allocated to long-lived flows. Instead, in `DropTail` and `RED`, under intense load, the long-lived flows, having reached large window size values, are not accommodating the new short-lived flows, but instead, by occupying the bulk of the available queue space, force them to losses, timeouts and degradation of the average response time. `RED` improves matters a lot compared to `DropTail` with respect to the average response time. However, by not being able to separate the short from the long-lived flows, it still occasionally victimized short lived flows unwittingly, causing some of them to experience significantly inflated response times compared to the average response time. That is why, even though the average response time in `RED` is improved, its variance is at a fairly high value (esp. at higher `RTT_Ratio` values, as Table 6.2 reveals). An interesting observation is that even though DAS “steals” away bandwidth from long-lived flows, the impact of such action on the long-term goodput of the long-lived flows is minimal. At the same time, `DropTail` and `RED` while not “stealing” away bandwidth from the long-lived flows, they do not insulate the long-lived from losses either. In fact, the increased packet loss rate of `RED` in overload (under large average window occupancy that is) frequently victimizes the long-lived flows. That is, the reduction of goodput of long-lived flows is unavoidable in order to accommodate the increased short-lived flow demands. Whereas the reduction of long-lived goodput is

accomplished via increased packet losses in the case of `DropTail` and `RED`, it is accomplished by limiting the bandwidth allocation in the case of `DAS`. The cost is a somewhat increased goodput variance (calculated over short intervals of less than second) in the case of the `DAS` schemes. Nevertheless, the most pressing performance concern of a long-lived flow is the long term throughput (and fairness) and not the short-term throughput variability. We thus claim that the increase of goodput variance over small timescales is a small price to pay for the increased response time of the short-lived flows.

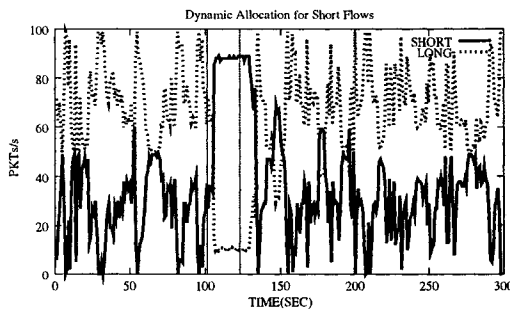
To further appreciate the ability of `DAS` schemes to adapt to the changing demands of the short-lived flows, we provide Figures 6.1, 6.2, and 6.3. The overload period is indicated by the interval between the two vertical blue lines. The re-allocation decisions are made at every 5 seconds. From the simulation results of a typical experiment (Figure 6.1(a)), the surge is detected by time 105. Likewise, the decrease of the flow demands is detected by time 125. We note that a correctly bootstrapped `DAS-BV` reacts in the same way as `DAS-EL` since both derive the load demands (in packets per second) from the number of active flows. Unlike `DAS-EL` which depends on the external independent parameter `AVGLoad`, `DAS-BV` depends on the measurement of the previous interval (and by implication, on its initial bootstrap value). When the previous measurement is not accurate, it will affect the performance of subsequent intervals. The `DAS-ED` scheme uses a different way for obtaining the expected load although the variation of `flowCount` still plays a crucial role. The expected load is based on the packet arrivals of the past adjustment interval. Because all three versions of `DAS` outperform both `DropTail` and `RED` the selection for the best among them needs to be performed on the basis of criteria beyond just the improvement of the response time of short lived flows. In particular we would like to indicate that if the load of a system is not well known, and hence both a good bootstrap value for demand (as used by `DAS-BV`) and a good estimate for `AVGLoad` (as used by `DAS-EL`) are not available, a good choice is `DAS-ED` despite the fact that it is the worst performing among the three. As more precise information about the load induced by short-lived flows becomes known, an operator could choose to transition from `DAS-ED` to `DAS-EL` or `DAS-ED`.



(a) DAS-BV

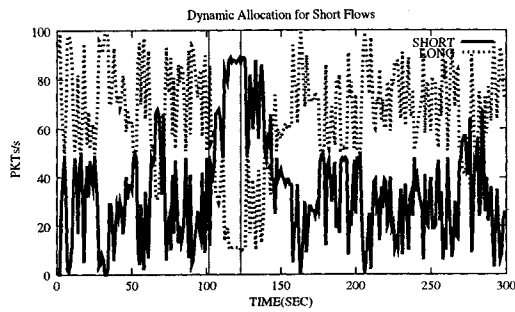


(b) DAS-ED

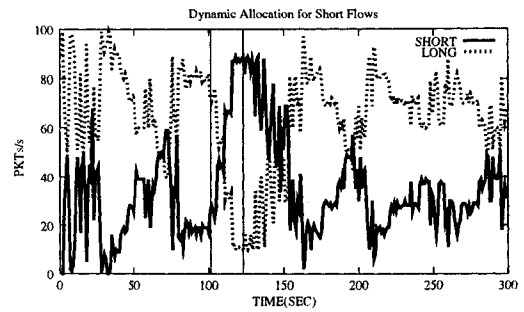


(c) DAS-EL

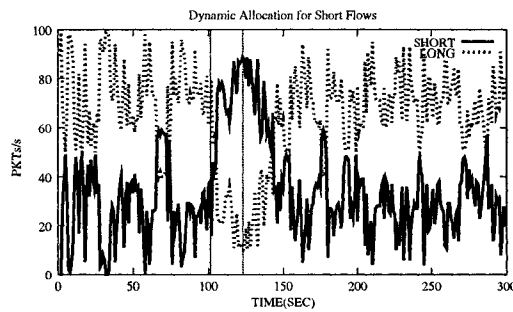
Figure 6.1: The goodput of long and short-lived flows under the DAS schemes (RTT_Ratio=2).



(a) DAS-BV

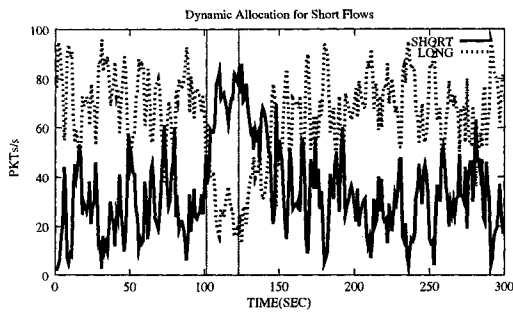


(b) DAS-ED

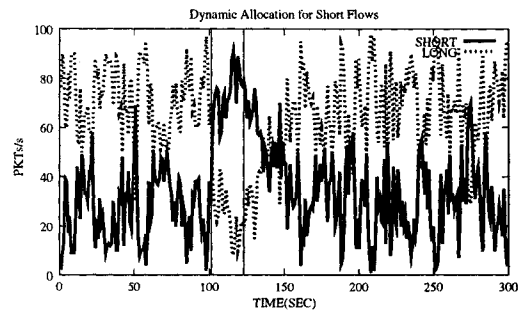


(c) DAS-EL

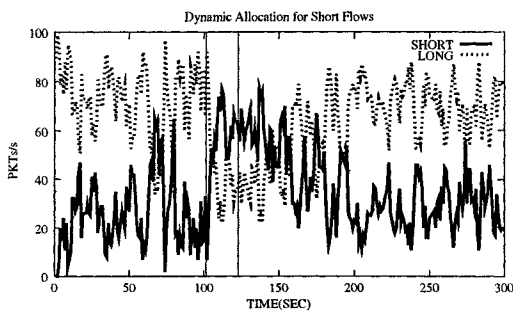
Figure 6.2: The goodput of long and short-lived flows under the DAS schemes $RTT_Ratio=10$.



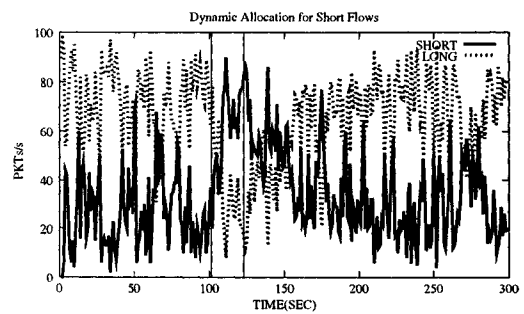
(a) DropTail (RTT_Ratio=2)



(b) RED (RTT_Ratio=2)



(c) DropTail (RTT_Ratio=10)



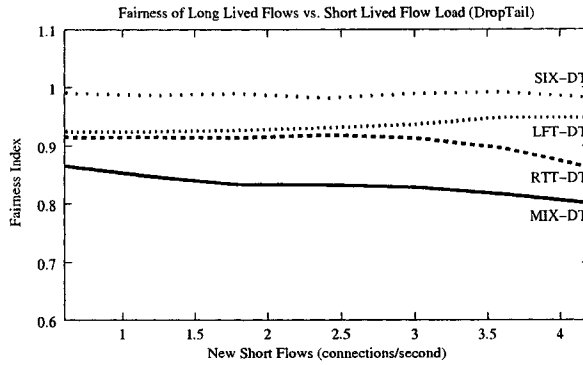
(d) RED (RTT_Ratio=10)

Figure 6.3: The goodput of long and short-lived flows for DropTail and RED.

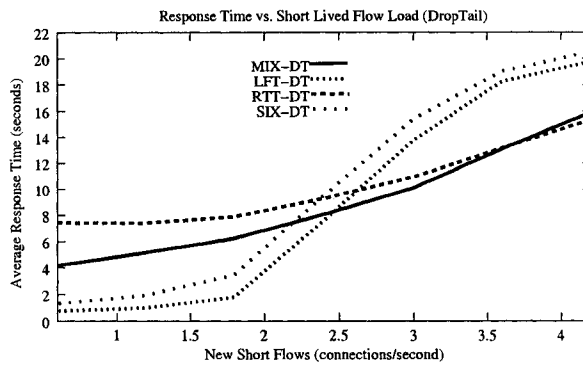
6.3 Investigation of Classification Based on Simple Rules

6.3.1 Simulation Setup

As the simulation of DAS, the *ns* [6] simulator was also used as the simulation platform for studying classification based on simple rules. The other aspects of the simulation, such as topology, parameters and traffic combinations, are identical with those in simulations in the previous section. Note that the load introduced by each new short-lived connection implies that the link will saturate close to an arrival rate of 2.4 short-lived connection requests per second. When DropTail or RED is simulated without classification schemes, all flows are mixed together and share the common pool of resources. When classification is used, the single physical queue is divided into multiple logical sub-queues of equal size, one per class. The bandwidth is also shared, using a WFQ scheduler capable to allocate bandwidth in a weighted fashion to each class. Whenever RTT classification is used, three RTT classes are implemented, corresponding to the three RTT selection ranges of the generated traffic. The RTT classes share bandwidth equally among them. Lifetime-based classes share bandwidth in a weighted fashion which is either static or dynamic. A reasonable value for a fixed allocation obtained from our study of traces suggests an aggregate short-lived flow demand in the 50% to 70% of the total available link capacity. Although the fraction varies with the definition of long-lived flows, we argue that 60% is a reasonable approximation for the total short-lived traffic demand [13]. However, we also study cases where the allocation between the two classes is dynamic, in which case, the load of short-lived flows is determined by measurement of how many new connections are started per unit of time. Eight schemes were investigated via simulations: (a) MIX-DT: DropTail with no classification, (b) MIX-RED: RED with no classification, (c) LFT-DT: lifetime classification scheme with per-queue DropTail, (d) RTT-DT: RTT-based classification scheme with per-queue DropTail, (e) SIX-DT: extensive classification into six classes (lifetime and RTT classification) with per-queue DropTail, (f) LFT-RED: lifetime classification scheme with per-queue RED, (g) RTT-RED: RTT-based classification with per-queue RED, (h) SIX-RED: the combination of extensive classification with per-queue RED. Each simulation represents 150 seconds of simulated time and the plotted results are averages of 24 runs.



(a)



(b)

Figure 6.4: Short-lived flow load on (a) fairness of long-lived flows, and (b) the response time of short-lived flows, with or without classification schemes, using DropTail.

6.3.2 Evaluation Results

We first investigate the effect of the load caused by short-lived TCP flows. The metrics of interest are the fairness among long-lived TCP flows, the response time of short-lived TCP flows, as well as the overall goodput. The fairness across the goodput of long-lived flows is defined in the usual way, $F(x) = \frac{(\sum x_i)^2}{n(\sum x_i^2)}$ [54], with a value of one representing the ideal (fair) sharing. Without separation of long and short-lived TCP flows, an increase in the load due to short-lived TCP flows may accentuate, but only slightly, the unfairness among long-lived TCP flows (Figure 6.4(a)). What is more striking is the difference in fairness across different classification schemes.

When no classification (MIX-DT) is used, some long-lived TCP flows are actually shut off

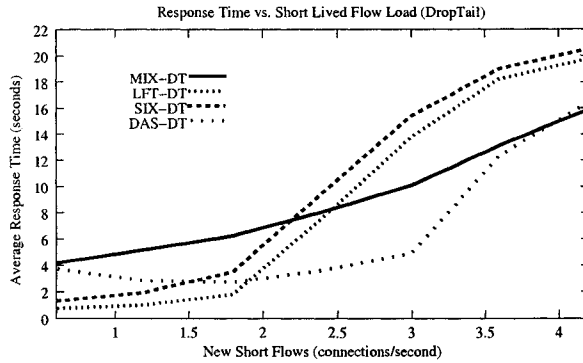


Figure 6.5: Short-lived flow load on response time for fixed vs. dynamic (DAS) bandwidth allocation.

completely when the traffic of short-lived TCP flows is extremely high. Lifetime classification schemes (LFT-DT) provide isolation between long and short-lived TCP flows. Long lived TCP flows are no longer affected by the intensified competition caused by the larger volume of short-lived flows. On the other hand, RTT classification does not separate long from short-lived flows, and hence the fairness between long-lived flows decreases as the number of short-lived TCP flows increases despite using multiple RTT classes. Finally, the scheme that uses more extensive classification, SIX-DT, combines the advantages of LFT-DT and RTT-DT providing the best fairness performance among all the schemes studied here. The impact on response delay is shown in Figure 6.4(b). In the schemes without isolation between long and short-lived flows, like MIX-DT and RTT-DT, an upper bound on the bandwidth allocated to short-lived flows is not enforced. Thus, short-lived flows might acquire, in the short run, more than 60% of the total bandwidth and therefore experience better response time. Figure 6.4(b) shows that lifetime classification schemes improve the response time of short-lived TCP flows when the allocated bandwidth is more than the total demand of short-lived TCP flows.

Instead of fixed allocation to short-lived flows, or no allocation at all, it certainly makes sense to track the traffic demand of short-lived flows (e.g., by inspecting the SYNs arriving per unit of time) and allocate bandwidth in response to the measured load. To avoid totally pushing our long-lived flows, a minimum of the link capacity can be reserved at all times for the exclusive use of long-lived flows. The performance of DAS is captured in Figure 6.5 whereby 10% is left reserved at all times for long-lived flows. DAS possesses the virtues of both fixed threshold lifetime classification schemes and schemes without classification. In terms

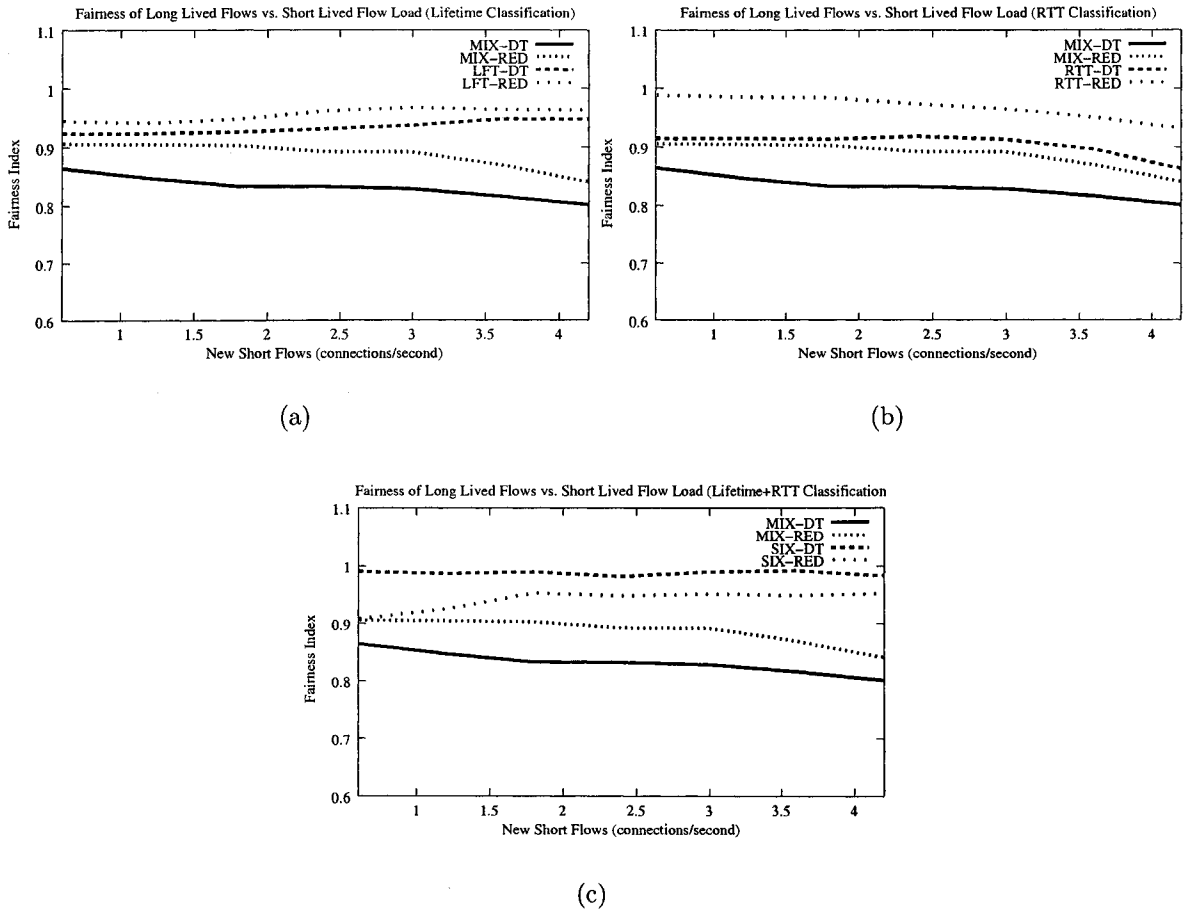
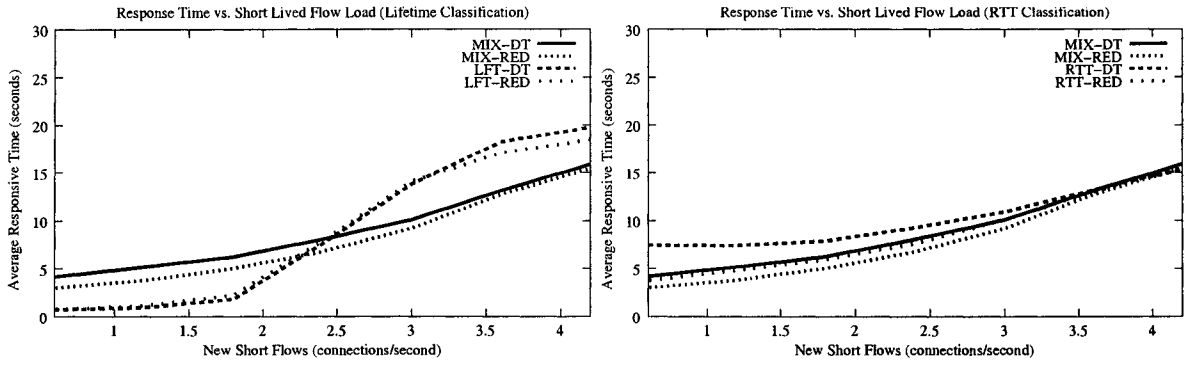


Figure 6.6: Short-lived flow load on fairness among long-lived flows for (a) lifetime-based, (b) RTT-based, and (c) combined, or no classification scheme (MIX-DT, MIX-RED).

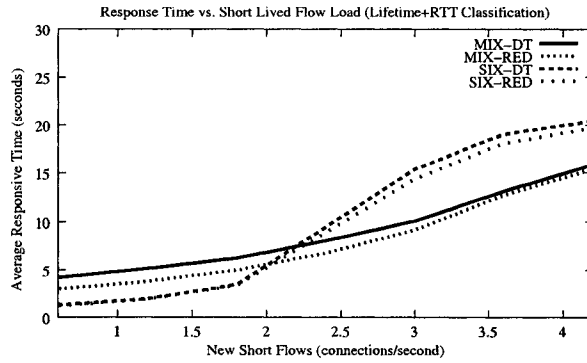
of fairness among long-lived flows, DAS is comparable to the simple lifetime classification scheme with fixed threshold and outperforms MIX-DT, while in terms of total throughput of short-lived flows, DAS is similar to the simple DropTail scheme. The more important feature however is that, in terms of the average response time for short-lived flows, DAS is a reasonable compromise (Figure 6.5). When the short-lived flow load is low, DAS performs like fixed threshold lifetime classification scheme. When the short-lived flow load is near or above saturation, it performs similarly or better than DropTail.

The comparisons of classification schemes with DropTail and classification schemes with RED, are shown in Figures 6.6, and 6.7. In terms of fairness among long-lived TCP flows, the simple lifetime classification scheme LFT-DT is a better scheme than simple MIX-RED.



(a)

(b)



(c)

Figure 6.7: Short-lived flow load on the average response time of short-lived flows for (a) lifetime-based, (b) RTT-based, and, (c) combined, or no classification scheme (MIX-DT, MIX-RED).

The reason for the improvement by simple MIX-RED over MIX-DT is due to the random nature of selecting victims in case of congestion. In MIX-RED, flows that have more packets stored in the buffer at the moment of congestion are more likely to be victimized. Thus, to some extent, the disparity of throughput between TCP flows is balanced. However, since the long and short-lived TCP flows are not actually separated, the negative impact of increased short-lived flows will exceed the control of MIX-RED policy as the traffic of short-lived flows grows. Thus, when the number of newly arriving short-lived TCP flows is large enough, the fairness index decreases. On the other hand, in LFT-DT, long-lived flows are isolated from short-lived flows, and therefore the fairness index is almost unchanged. Moreover, the benefit of lifetime classification coupled with dynamic allocation of bandwidth between short and long-lived flows (a-la DAS) results in response times that are generally insensitive to the fluctuations (admittedly minor) of the fraction of connections that turn out to be long-lived. In DAS, the bandwidth of the short-lived class is allocated based on the demands of short-lived flows. Consequently, the impact of the volume of long-lived flows on the response times is restricted. That is, it is possible, without sacrificing the fairness among long-lived flows, to reduce and equalize (lower variance) the response time of short-lived flows. The comparison of RTT-based classification schemes and RED is described in Figure 6.6(b). In general RTT-based results are always better or comparable to the corresponding RED scheme. Long and short-lived TCP flows are not separated in either type of scheme. RED uses a random approach to selecting “winners” when losses are inflicted, while RTT schemes penalize the flows in an equal fashion within their RTT class but possibly differently across RTT classes. The balancing capability of simple RED is affected by the RTT Ratio significantly. RED works better only in the case of small RTT discrepancies between the flows. On the contrary, the performance of RTT classification is not affected by the RTT Ratio (Figure 6.8(b)). In addition, our experiment shows that the combination of any classification scheme and RED outperforms the original simple classification scheme or simple RED. The reason for the improvement is due to the relative homogeneity brought by the classification scheme for the losses experienced by each class of flows (each RTT “group”). RED is not capable of handling the extremely RTT-heterogeneous flows but apparently works fine regulating groups of flows with similar RTT values. The improvement in response delay from deploying RED is minimal (Figure 6.7). The major effect of RED is in equalizing the throughput of

competing TCP flows. With unchanged resources and number of competitors, the average bandwidth allocated to each TCP flow is not expected to change a lot. Thus, the average and the variance of response delay of short-lived TCP flows do not improve significantly. In addition, the simple RTT classification scheme MIX-RTT brings no significant improvement either. As we have observed before, lifetime classification schemes including LFT and SIX perform well when the allocated bandwidth exceeds the demand. RTT classification cannot, on its own, provide any significant improvement in this sense. Consequently, in terms of response delay of short-lived flows, the only reasonable solution is allocating more bandwidth, in a dynamic fashion, tracking the rate of newly arriving short-lived flows. The impact of RTT disparity on the performance of the schemes is shown in Figure 6.8. The fairness of long-lived flows decreases without RTT-based classification. The observation is actually not a surprise. The increase of the RTT ratio will increase goodput disparity among TCP flows competing for the same congested link. It can be seen that fairness can be provided without expensive per-flow control using RTT-based classification (Figure 6.8). In fact, with RTT-based classification, the fairness remains unchanged as the RTT ratio (and hence the RTT disparity) increases.

In terms of fairness between long-lived flows, MIX-RED performs better than MIX-DT, but not better than the LFT-DT scheme (Figure 6.8(a)). The improvement of MIX-RED over MIX-DT is caused by random dropping at times of congestion—flows with more packets stored in the buffer space are more likely to experience packet drops. Thus, some extent of fairness is achieved. However, the performance of MIX-RED is afflicted by instantaneous bursts in the traffic. The congestion is essentially detected by the exponential average of the queue size, which is not fast enough to detect and respond to large bursts of traffic. In such cases, MIX-RED performs similarly to MIX-DT. In RED the dropping decision is based on the instantaneous occupancy status of the queue. When the traffic is highly bursty, the instantaneous queue occupancy does not necessarily reflect the long-term throughput correctly. The scheme may not make a correct decision in selecting a flow to victimize. In contrast to RED, LFT-DT is a simple scheme with only one parameter, the bandwidth allocation between long and short-lived flows. Moreover, in RED when RTTs get even more dispersed, the fairness index decreases. On the contrary, RTT based classification schemes like RTT and combined lifetime+RTT, group the flows of similar RTTs into the same class.

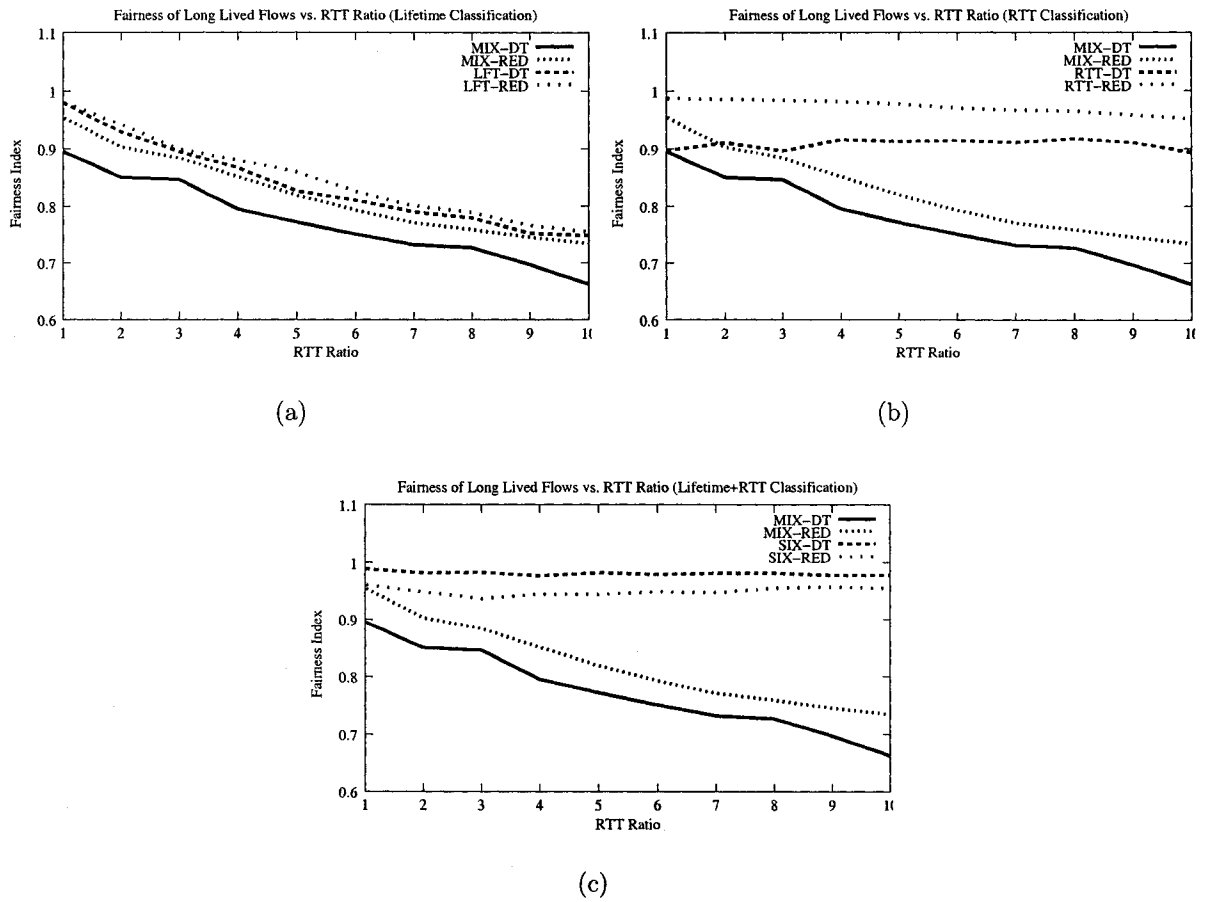


Figure 6.8: Fairness of long-lived flows for (a) lifetime classification, (b) RTT classification, and, (c) combined, or no classification scheme (MIX-DT, MIX-RED).

Thus, the fairness of long-lived flows is not affected by the increases of RTT ratio (Figure 6.8(b) and 6.8(c)) and easily surpasses that of the RED policy especially when the RTT ratio is large.

6.4 Conclusions

In Section 6.2 we explore the idea of providing improved response time to TCP flows that have been identified as being short-lived, i.e., have a few packets to send. We argue that such a separation is valuable on the basis of being a natural match to the performance needs of short lived flows, that are by far the by-product of interactive applications, e.g., web surfing. At the same time that we wish to provide an advantage to short-lived flows, we also seek to not increase dramatically the complexity of routers in order to support such a separation. Hence, we look into three schemes of different degrees of implementation complexity that are straightforward to accommodate in today's advanced routers (in particular: requiring the existence of a WFQ scheduler and a flow classifier module). The second point we make is that we are increasingly knowledgeable of the traffic characteristics of TCP flows, as a plethora of statistical studies have confirmed the heavy tailed nature of connection lifetimes, and, another set of results, provide high fidelity models for the transfer sizes of HTTP requests/responses. Our ambition is that, similar to the very good understanding that we have developed over decades for what is the "average phone call duration", a good understanding for the connection-level statistics is starting to emerge for TCP traffic as well. We therefore take into account some very basic statistical characterization of short-lived TCP flows, and exploit it in the context of dynamic bandwidth allocation. We have found that the proposed schemes can outperform DropTail and RED. This is not really surprising, given that both DropTail and RED make no explicit attempt to adapt to the changing load (expressed as new connections set up per unit of time). What appears interesting however is the observation that whereas DropTail and RED use the "sledgehammer" approach of inflicting losses, as a means of control, an alternative of controlling the bandwidth allocated to subset of short-lived flows, can result in better response times, while at the same time avoiding the counterproductive practice of intensifying losses as a means to control the TCP source behavior. Our study in Section 6.3 shows that we need to track the demands of the short-lived TCP flows if we would like to provide reasonable average response time for

short connections (as one would expect e.g. in the case of web traffic). We note that such a demand-based allocation need not be performed to the detriment of fairness among long-lived connection. In particular what these points suggest is that a lifetime-based classification scheme can be used in conjunction with a bandwidth allocation scheme that tracks the short-lived flow demands. We note that for long-lived flows, fairness might be a much more important property than response time since the user is already accepting the fact that a large transfer of data will take a long time. On the other hand, RTT-based classification schemes, although not relevant to the performance separation of short and long-lived flows, are crucial in the implementation of fairness within the corresponding classes, especially when the traffic is so diverse that RED's potential of achieving a certain level of "fairness" is exceeded. Thus, classifying all flows into the Cartesian product of RTT and lifetime classes as well as adopting a dynamic bandwidth allocation (with possible operator intervention for guaranteeing certain minimum long-lived bandwidth performance) appears to combine advantages that result in a combined scheme that seems to be capable of outperforming RED and/or overcoming the parameterization problems of RED.

Chapter 7

Some Recent Results

7.1 Introduction

The area of studying TCP performance is extremely active. In the last year (2002-2003) new proposed schemes have been reported. In particular, XCP [31] introduced a control theory approach in the study of congestion control. New models are also proposed in the study of lifetime classification schemes [14, 15, 16]. Unlike the models we have described and evaluated in Chapter 2, these models are relatively new and have not been fully understood. In this chapter, we describe our study of these new models associated with experimental results.

7.2 Evaluation of XCP

The design rationale of XCP [31] comes from observations on inherent weakness of current TCP congestion control design. Firstly, TCP has no specific and explicit congestion signal. Packet loss is interpreted as a congestion signal. This interpretation is based on the assumption that congestion causes far more packet loss than an unreliable underlying network. In fact, this assumption does not always hold in all environments, such as in wireless networks. The imprecise interpretation on packet loss causes unnecessary throttling of the sending rate and a waste of precious bandwidth. Secondly, loss as a congestion signal only reflects the worst congestion scenario. With current congestion signalling, the sender is notified only after the buffers are overflowed, which is the worst outcome of congestion. Thus, senders only provide reaction on severe congestion, rather than act proactively and eliminate congestion at an early stage. Thirdly, current congestion control is slow. TCP receives only implicit con-

gestion signals. Detected timeouts and persistent packet reordering infer congestion signals. Both mechanisms need more time than explicit congestion signalling. Fourthly, the current congestion signal is a binary signal. It can only indicate whether congestion occurs or not. It cannot provide information on the degree of congestion. With such imprecise information, TCP senders have to react conservatively. Instead of sending at a precise available rate, TCP senders back off drastically and recover cautiously in the presence congestion. This protocol design leads to oscillatory behavior of individual TCP flows, which is a particularly undesirable feature when the protocol is used for delivering real time traffic. Lastly, the loss congestion signal is unreliable. The current mechanism cannot guarantee that all TCP senders receive congestion signals. The congestion signal does not provide information on how senders should react to the congestion. Thus, the reaction on congestion is unbalanced; some aggressive TCP flows might get significantly more bandwidth than their competitors.

7.2.1 XCP Congestion Header and Efficient Control

XCP accounts for the weakness of TCP by providing explicit congestion signalling, precise congestion information, decoupled efficiency control (EC) and fairness control (FC), and robust design [31]. XCP proposes a specific congestion header as an extension of the TCP header (optional fields) to carry necessary information for congestion control, the state of each flow, and feedback from routers. Every packet carries this congestion header. The definition of congestion header is shown in the following figure. Senders maintain the current congestion window size and current RTT and transfer them to intermediate routers via the packet header information. With the information from senders, the capacity and link utilization status, intermediate routers instruct senders by stipulating feedback information in the congestion header of each packet. The information in congestion headers will be copied in the acknowledgments (ACKs) and finally delivered back to senders.

<i>cwnd</i> , sender's current congestion window size
<i>rtt</i> , sender's RTT value
<i>Feedback</i> , Instruction from the routers

XCP senders update their congestion window by adding the feedback to the current congestion window (Equation 7.1). The feedback contains the control information from routers. It is either positive or negative, which means increasing or decreasing congestion

window size, respectively. The units of feedback and window size are bits. Although feedback is mainly used as instruction supplied by the router, senders initialize the feedback at the start indicating their demands on bandwidth. The XCP sender algorithm is described in the following formula. The s in the formula (Equation 7.1) is the length of a packet in bits. It stands for the minimum unit of updating congestion window, which is one packet.

$$cwnd = \max(cwnd + Feedback, s) \quad (7.1)$$

A unique design feature of XCP is the decoupling efficiency control (EC) and fairness control (FC) in the router policy. The objective of EC is to maximize the link utilization and minimize the packet dropping and persistence queue size, which is opposed to the transient queue size due to the burst nature of window-based mechanism. The algorithm is described in (Equation 7.2)

$$\phi = \alpha \times d \times S - \beta \times Q \quad (7.2)$$

ϕ stands for the total feedback generated in the router within the control epoch d , which is the average RTT of all flows. S is the difference between the sum of demands of all flows and the link capacity. Q stands for persistent queue size that is exponentially smoothed from transient queue size. And α and β are operational constants. Basically, the total feedback is proportional to the difference between expected demands of bandwidth of all flows and the actual bandwidth. To be more accurate, the feedback is also adjusted to account for the packets stored in the buffer.

7.2.2 XCP Fairness Control

Fairness Control (FC) (Equation 7.3) is the algorithm for apportioning the total feedback, that is obtained in EC, to each individual flows. The basic idea of FC can be described as follows: In case of positive feedback, that is, the expected total demand is smaller than actual capacity, the surplus bandwidth is apportioned to all flows equally. The increase of sending rate is the same regardless of their RTT. In case of negative feedback, that is, the expected total demand exceeds the actual capacity; all flows reduce their sending rate by the same proportion. Feedback apportioned to the individual flow is uniformly distributed to all packets from that particular flow. The detailed derivation of the algorithm is described as follows.

$$Feedback = p_i - n_i \quad (7.3)$$

Derivation of Positive Feedback

The derivation of FC policy in the original paper is not explicitly laid out [31]. In order to analyze the theoretical foundation of XCP, we derived the XCP FC policy in detail. For the sake of simplicity and no loss of generality, we only analyze the essential part of FC policy where the effect of bandwidth shuffling [31] is not included. ϕ is the total feedback (in bits) determined by EC policy (Equation 7.2). When $\phi > 0$, it means that the link is under utilization, and thus we have spare bandwidth to distribute. In XCP, such spare bandwidth is distributed in the following way: the spare bandwidth should be allocated equally among the active flows. That is, the rate increase of each flow should be the same. We have the following relations:

$$\Delta r = \Delta r_1 + \Delta r_2 + \dots + \Delta r_n$$

$$\Delta r_1 = \Delta r_2 = \dots = \Delta r_n$$

Here, Δr is the total increase rate (spare bandwidth, $\frac{\phi}{d}$) in one epoch, and Δr_i stands for the increased rate for flow i . n stands for the number of active flows. From the above two equations, we also have

$$\Delta r_i = \frac{\Delta r}{n} = \frac{\phi}{nd}$$

We can also transform the rate increase into the congestion window size increase for particular flow i for a single rtt_i .

$$\Delta cwnd_i = \Delta r_i * rtt_i$$

p_i , the feedback distributed in each packet of flow i is then

$$p_i = \frac{\text{feedbacks for flow } i}{\text{the number of packets from flow } i}$$

$$p_i = \frac{\Delta cwnd_i}{\text{current rate} * d / \text{packet size}}$$

$$p_i = \frac{\Delta cwnd_i}{\frac{cwnd_i}{rtt_i} * d / \text{packet size}}$$

$$p_i = \frac{\Delta r_i * rtt_i}{\frac{cwnd_i}{rtt_i} * d / \text{packet size}} = \frac{\frac{\phi}{nd} * rtt_i}{\frac{cwnd_i}{rtt_i} * d / \text{packet size}}$$

$$p_i = \frac{\phi * packetsize}{n * d^2} * \frac{rtt_i^2}{cwnd_i}$$

When we set $\xi_p = \frac{\phi * packetsize}{n * d^2}$, We have

$$p_i = \xi_p * \frac{rtt_i^2}{cwnd_i}$$

We also have

$$\Delta r = \Delta r_1 + \Delta r_2 + \dots + \Delta r_n$$

that is, the total increase rate is the sum of increase rate of all individual flows,

$$\frac{\phi}{d} = \sum_{flow1} \frac{p1_i}{rtt1} + \sum_{flow2} \frac{p2_i}{rtt2} + \dots + \sum_{flown} \frac{pn_i}{rttn}$$

The increase rate of each individual flow is the sum of feedback carried by each flow within one rtt ,

$$\frac{\phi}{d} = \sum_{all\ packets\ in\ epoch\ d} \frac{p_i}{rtt_i} = \xi_p * \sum_{all\ packets\ in\ epoch\ d} \frac{rtt_i}{cwnd_i}$$

Thus,

$$\xi_p = \frac{\phi}{d * \sum_{all\ packets\ in\ epoch\ d} \frac{rtt_i}{cwnd_i}}$$

Derivation of Negative Feedback

When $\phi < 0$, congestion has occurred. That is, the total demands for bandwidth exceeds the link capacity. In XCP FC policy, every flow should reduce its rate by a same fraction. Suppose f is the common reduction factor, we translate the policy into the following formulas:

$$\Delta r = \Delta r_1 + \Delta r_2 + \dots + \Delta r_n$$

$$\Delta r_i = r_i * f = \frac{cwnd_i}{rtt_i} * f$$

$$\Delta cwnd_i = \Delta r_i * rtt_i = cwnd_i * f$$

$$n_i = \frac{\Delta cwnd_i}{\# \text{ of packets within } d}$$

$$n_i = \frac{cwnd_i * f}{\frac{cwnd_i}{rtt_i} * d / packetsize} = \frac{f * packetsize}{d} * rtt_i$$

If we set $\xi_n = \frac{f * packetsize}{d}$, we have

$$n_i = \xi_n * rtt_i$$

With similar reasoning as in the above section,

$$\begin{aligned}\Delta r &= \Delta r_1 + \Delta r_2 + \dots + \Delta r_n \\ \frac{\phi}{d} &= \sum_{flow1} \frac{n1_i}{rtt_1} + \sum_{flow2} \frac{n2_i}{rtt_2} + \dots + \sum_{flown} \frac{nn_i}{rtt_n} \\ \frac{\phi}{d} &= \sum_{all\ packets\ in\ epoch\ d} \frac{n_i}{rtt_i} = \xi_n * (\#packets\ in\ d) \\ \xi_n &= \frac{\phi}{d * (\#packets\ in\ d)}\end{aligned}$$

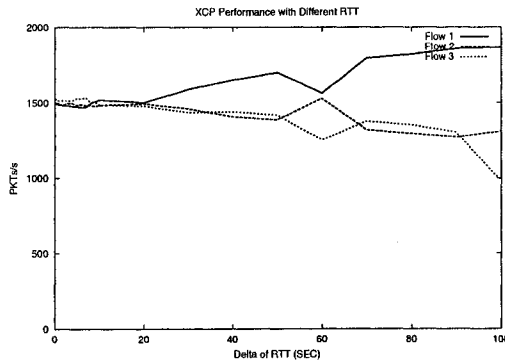
7.2.3 XCP Performance Under Mixture of RTTs

The idea underlying XCP is interesting. However, it has some major drawbacks. Firstly, it requires special congestion headers that mean a major re-design of packet header. Such modification needs universal efforts which in not necessarily non-trivial. Secondly, XCP does need global knowledge of each RTTs and the total number of flows N to obtain d, the average RTT, which is a critical parameter in XCP. Thirdly, although d is the average RTT, it might not be safe to assume d as a constant, given the large variance on RTT. Thus, the robustness of XCP under a mix of widely varying RTTs is questionable. In order to explore the performance of XCP in large RTT variance, we conducted simulations with the code provided by [31] under ns2 version 2.1b9 [6]. The simulation topology is the traditional one used in the paper [31], one congested link with several side links. We vary the difference between RTTs from 0 to 100ms. Simulation result shows that XCP deteriorates as the difference between RTTs increases.

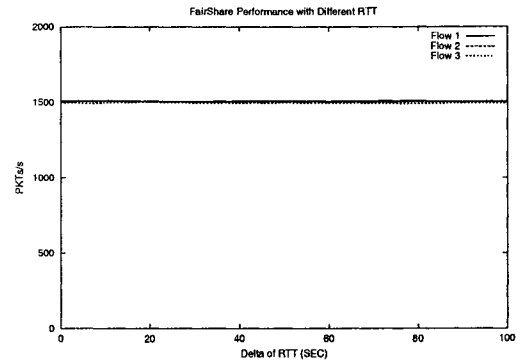
As we see from the comparison (Figure 7.1), our FairShare policy performs much better than XCP for environment with large RTT variance, in terms of fairness. Moreover, our FairShare policy does not need any modification on packet header. The reason of the robustness of our policy is that our policy is based on a solid, proved deterministic model.

7.2.4 XCP Performance Under Mixture of Lifetime and Dynamic Conditions

The other weakness of XCP is that it is designed only for long-lived flows. Only long-lived flows can receive instructions from intermediate routers and adjust their sending rate accordingly. Short-lived flows are too short for any feedback instructions; they might terminate



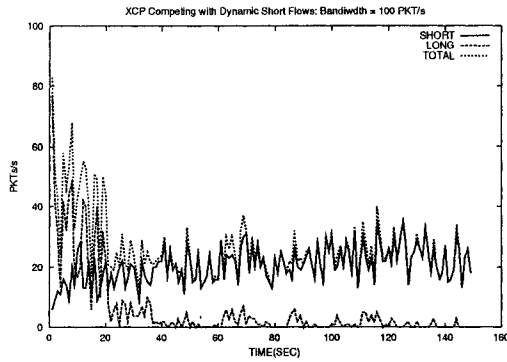
(a) XCP



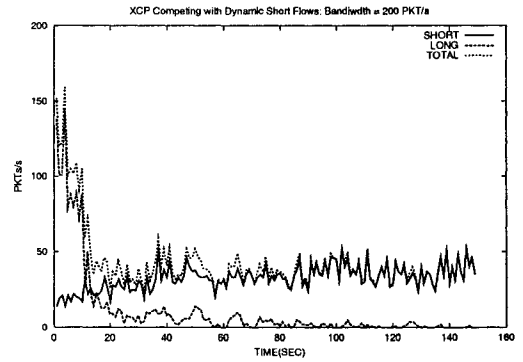
(b) FairShare

Figure 7.1: Fairness Under Different RTTs, (a) XCP, (b) FairShare.

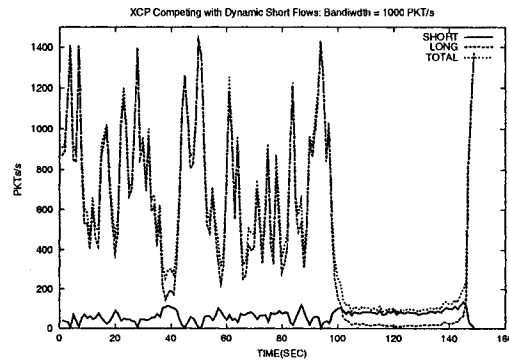
their transmission before any available feedback. The performance of XCP under a mix of lifetime is also questionable. We study XCP's performance in experiment of a mix of long-lived flows and a dynamic number of short flows via simulations. The configuration of experiments is as following: 9 long-lived flows last the whole duration, while a random number of short-lived flows compete. The load of short-lived flows is a random number uniformly between 10 and 20. The starting times of short-lived flows are also uniformly distributed in the simulation duration. The arrival rate of short-lived flows are 2 flows per second while it reaches 5 short-flows per second between 100 and 120 of simulation period. The arrival rate short-lived flow spike reflects the inhomogeneous feature of real network traffic. We have conducted three experiments with different capacity of congested link, 100, 200 and 1000 packet/second, respectively. As we can see from the figure (Figure 7.2), XCP is extremely unstable in such a dynamic environment. Firstly, the apportioned ration of bandwidth does not reflect the spike of short-lived flows. Moreover, the congested link was in low utilization in such dynamic environment. The reason for the poor performance is that the design of XCP does not account for the specific feature of short-lived flows.



(a) 100



(b) 200



(c) 1000

Figure 7.2: Performance of XCP In a Dynamic Environment, (a), 100 PKT/s , (b), 200PKT/s, (c), 1000PKT/s.

7.3 An Alternate Lifetime-Based Scheme

Recent measurement of Internet traffic [17] shows that the length TCP flows in terms of lifetime follows a heavy-tailed distribution. That is, a small fraction of long-lived TCP flows carries a large fraction of total traffic. This unbalance in TCP lifetime calls for a new design for of the network policy. Short-lived TCP flows are at a disadvantage when they compete with long-lived TCP flows. Firstly, due to the conservative of TCP congestion control and small number of transferred packets, short-lived TCP flows usually operate at the small congestion windows exponential growth stage (slow start). Any packet drop in this stage has a drastic impact on the sending rate. Packet drops have the largest impact on TCP flows during exponential growth than during congestion avoidance growth. Secondly, short-lived TCP flows have only a small number of packets to transfer. They do not have enough packets for detecting packet loss via the fast retransmission algorithm. On the other hand, they have to rely on the timeout mechanism, which is slower to respond than fast retransmission mechanism. Thirdly, short-lived TCP flows have no time to probe its actual RTT. Thus, its timeout threshold uses the default value, which is much more conservative than the timeout threshold obtained after several measurements of the RTT. Matta et al proposed isolation of TCP flows based on lifetime (or size) to protect short TCP flows from long TCP flows [14, 15, 16]. They proposed to give preferential treatment to short-lived TCP flows. In their scheme, packets from short-lived TCP flows are processed separately from those of long-lived TCP flows at intermediate routers, while edge routers classify TCP flows by their lifetime. Three classification schemes were proposed in [16]. The first one is a pure random scheme that assigns flows to different classes with equal probability. The second one makes classification decision on the traffic volume of each flow. The third scheme classifies flows based on the lifetime of flows. That is, only flows lasting longer than a particular threshold is classified as long-lived flows.

Matta et al's research was based on an analytic model of TCP flows [15]. TCP flows are modeled as Poisson processes, where burst of packets (window) arrives at the Poisson input rate of $1/RTT$. They assumed that all TCP flows have the same burst arrival rate. The burst size is static; short-lived TCP flows are modeled as processes emitting bursts of size 4, whereas long-lived TCP flows are those with burst size greater than 4.

The number of packets buffered is modeled as a Markov Chain. By using the station-

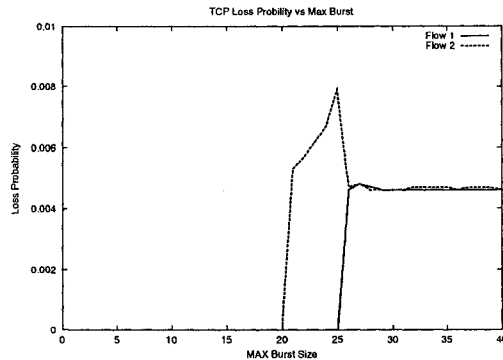


Figure 7.3: Loss Probability vs. Maximum Burst Size

ary distribution of the total number of packets in the queue, the packet loss probability is described as follows:

$$P_{DropTail}(TCP) = \pi(K) + \pi(K-1)\frac{B-1}{B} + \dots + \pi(K-B+1)\frac{1}{B} \quad (7.4)$$

Basically, Equation 7.4 suggests that the packet loss probability is equal to the sum of all cases when the sum of stationary queue size and burst size exceeds the total buffer size. For example, when the stationary queue size is $K-i$ where ($i < B$), $B-i$ packets out of B packets will be lost in each burst. Although the argument of Matta et al research is intuitively reasonable, the analytic model they constructed was not realistic. The assumption on static window size does not capture the dynamic nature of TCP flows. In order to validate their model, we conducted our experiments on ns [6]. In the experiments, we adopt the dynamic window size; that is, the window size is controlled by the congestion control mechanism. The only thing we do is enforce an upper bound on the congestion window size (as is expected to happen in a real network as well). From the experiment 7.4 results (Figure 7.3), we see with the increase of burst size, the packet loss probability increases accordingly.

The conclusion derived by Matta et al. is against our understanding of TCP's mechanism. TCP senders adjust their window size according to the packet drop they detected. TCP senders will react to the packet losses by decreasing their window size. It is unlikely that packet loss rate increases to about 80% while the sender congestion windows still increase. Our experiments show, that the loss probability at first increases with the upper

bound of window size. Then, it remains around a stable value regardless of how large of the upper bound of window size. Although Matta's classification policy is important, it is far from complete. The first weakness of Matta's policy is that the classification scheme relies on a static threshold. With this static threshold, the total bandwidth is apportioned to long- and short-lived flows. It does not account for the fact that the number of flows in realistic environment is dynamic. The second weakness is that their policy did not specify how the long-lived flows are to be controlled. As we have already seen, separation of long-lived flows from short-lived flows alone did not solve the unfairness against flows with relatively larger RTT values (see Chapter 6).

Chapter 8

Summary and Future Work

8.1 Summary

Despite years of progress, a deep theoretical understanding of the existing Internet protocols is extremely challenging. Selecting the most suitable framework of theory and making appropriate simplifications to make the problem mathematically tractable while retaining the essential features of the network system are still difficult problems. Most of the theoretical studies of computer networks come from the legacy of previous studies of the telephone communication network. The prevailing belief in the early 1990s was that certain open-looped Markovian processes might be adequate to model the Internet traffic. However, the inherent feedback feature stemming from the congestion avoidance algorithm was misinterpreted; although the probing mechanism is fully accounted for in the studies of the individual TCP flow, the mechanism's impact on the equilibrium of competing TCP flows is not fully understood and has been oversimplified. In addition, although a revolutionary understanding of self-similarity in LAN traffic, the phenomenon's physical cause and its implications for network engineering are still inconclusive. Moreover, the community is divided even on the benefits of RED, one of the earliest intelligent AQM policies, which has been already implemented in Cisco's products and recommended as a standard by IETF.

My first effort with understanding the theoretical foundation of Internet protocols involved validating the proposed models of TCP flows based on classical stochastic process. Independent Markovian processes had been proposed to modify the traffic of individual TCP flows, and the arrival processes of competing TCP flows were assumed to be independent. Later, more sophisticated models that accounted for the congestion avoidance mechanism

were proposed, and the researcher believed that the equilibrium goodput of an individual TCP flow was the function of its independent loss process. After conducting extensive simulations with a wide spectrum of parameters, We realized that the modeling work on TCP flows is still in an early stage. The oversimplified assumptions about independent arrival process and loss process do not capture the essential feature of TCP flows. More rigorous theoretical work has to be done on the suggested equilibrium state, for the mathematical proof of the existence and uniqueness of the equilibrium point is critical but still unknown. Although my early work on the theoretical foundation of protocols did not lead to mathematical solutions, we developed numerous insights into designing protocols of provisioning resources. Instead of designing protocols on the invalidated analytical results, we focused on the *DiffServ*-like large grain control algorithms. Most of my thesis work was conducted to study classification schemes at edge routers. My statistical studies on passively sampling the real life Internet traces confirm the power law distributions of the file size or the lifetime of TCP flows. Long-lived flows, which consist of less than 5% of the total active flows, carry more than 50% of the total data. This phenomenon is referred to as “the mice and elephants” or “dragonflies and tortoises” in some literature. My study confirms that propagation delays of real life TCP flows spread in a wide spectrum, and this discovery validates RTT-based classification for dealing with the well known unfairness against the TCP flows with relatively long RTT. We believe that classification schemes can be better than policies with complicated parameter tuning such as RED. In my lifetime classification scheme, We classify TCP flows at the edge routers. TCP flows that last beyond the pre-configured threshold are classified as long-lived flows. Because of the observed power law, the exact value of the threshold is not critical. We proposed a policy called **FairShare** to regulate the long-lived flows at the edge routers by using scheduled packet loss on the per-flows basis. With the explicit resource allocation, maxmin fairness among long-lived TCP flows is guaranteed. As for the short-lived flows, We believe that, instead of complicated management policies, the most efficient strategy to improve QoS of TCP short-lived flows is simply to allocate more bandwidth whenever necessarily. We proposed a control policy (**DAS**) based on the number of active TCP flows: an intermediate router detects the number of active TCP flows going through and dynamically adjusts the fraction of bandwidth for the class of short-lived TCP flows in case of any significant change in the number of active short-lived TCP flows.

8.2 Future Work

As part of future work, the following ideas could be explored:

- Our ultimate goal is to design a set of policies that can provide QoS assurance for TCP flows of different RTTs in the Internet. To do this, we have to account for several factors like the large and heterogeneous topology, the dynamic traffic within the Internet, and the difficulty of deployment. Through simulation experiments, we have shown that our proposed FairShare scheme converges to the globally max-min fair rates for various network configurations and in dynamic load environments. The FairShare policy appears to be self-stabilizing in the presence of dynamic network changes. However, we still need to conduct rigorous study on the parameterization issue in the future.
- Design a set of policies that can satisfy both long- and short-lived TCP flows is challenging. The congestion scheme in TCP sources presents itself in different ways on long-lived and short-lived flows. DAS paves a new perspective on policies providing QoS assurance for short-lived flows. Although DAS outperforms traditional AQM policies, the work on DAS is not complete. We lack a better understanding of how the proposed schemes behave in a multi-hop multiple bottleneck path environments.
- We have traded the accurate control of TCP flows with the complexity of the policies. In order to perform our policies, we have to maintain several state information for each of the flows and classes. In general, the efficiency of regulation algorithms depends on the quality of information required. Excessive information that can not be obtained accurately affects the efficiency of the algorithms negatively and the opportunity of deployment due to complexity. It is interesting to investigate the optimum design alternatives which achieve the best performance with appropriate quantity of information.
- We are also interested in various ways of incremental deployment techniques. We are nevertheless hopeful that the classification, and subsequent differentiation between two classes, is something within the reach of most modern internetworking equipment. In fact, including a classification scheme based on lifetime properties appears to also be

within the capabilities of MPLS, thus opening the possibility of controlling not only the bandwidth allocated to short-lived flows, but also their routing paths.

Bibliography

- [1] N. Cardwell and S. Savage, and T. Anderson, *Modeling the performance of short TCP connections*, Technical Report, Computer Science Department, Washington University, November 1998.
- [2] N. Cardwell and S. Savage, and T. Anderson, *Modeling TCP Latency*, In Proc. of INFOCOM '00, pp.1742–1751.
- [3] K. Thompson, G. J. Miller, and R. Wilder, *Wide Area Internet Traffic Patterns ,and Characteristics*, IEEE Network, Vol. 11, No. 6, pp.10–23, November /December 1997
- [4] Transmission Control Protocol, DARPA Internet Program, Protocol Specification, September 1981, RFC 793
- [5] V. Jacobson, *Congestion Avoidance and Control*, ACM Computer Communication Review, In Proc. of SIGCOMM'88, Stanford, CA, August, 1988, Vol.18 No.4, pp.314–329, 1988.
- [6] UCB/LBNL/VINT Network simulator - ns (version 2), <http://www-mash.cs.berkeley.edu/ns/>.
- [7] W. Steven, *TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithm*, RFC 2001, 1997
- [8] V. Jacobson, *Modified TCP Congestion Avoidance Algorithm*, end2end-interest mailing list, April 30, 1990, <ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail>.
- [9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, *TCP Selective Acknowledgment Options*, RFC 2018, 1996

- [10] D. Chiu ,and R. Jain, *Analysis of the increase and decrease algorithms for congestion avoidance in computer networks*, Computer Networks and ISDN Systems, pp.1–14, 1989.
- [11] S. Floyd ,and V. Jacobson, *On Traffic Phase Effects in Packet-Switched Gateways*, Journal of Internetworking: Practice and Experience, Vol.3 No.3, pp. 115–156, September 1992.
- [12] S. Floyd ,and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, Vol.1 No.4, pp. 397–413, August 1993.
- [13] N. Brownlee ,and K. Claffy, *Understanding Internet Traffic Streams: Dragonflies and Tortoises*, <http://www.caida.org/outreach/papers/2002/Dragonflies/>
- [14] S. Yilmaz ,and I. Matta, *On Class-based Isolation of UDP, Short-lived and Long-lived TCP Flows*, In Proc. of MASCOTS '01, Cincinnati, OH, August 2001.
- [15] L. Guo ,and I. Matta, *The War between Mice and Elephants*, In Proc. of ICNP '01, CA, November 2001.
- [16] I. Matta ,and Lw.iang Guo, *Differentiated Predictive Fair Service for TCP Flows*, In Proc. of ICNP '00, Osaka, Japan, October 2000.
- [17] A. Shaikh, J. Rexford ,and K. Shin, *Load-Sensitive Routing of Long-Lived IP Flows*, In Proc. of SIGCOMM '99, Boston, MA, Spetember, 1999.
- [18] S. Shenker, L. Zhang ,and D. Clark, *Observations on the Dynamic of a Congestion Control Algorithm*, ACM SIGCOMM Computer Communication Review, Vol.20 No.5, pp. 137–143, October 1990.
- [19] L. Qiu, Y. Zhang, and S. Keshav. *On Individual and Aggregate TCP Performance*, In Proc. of ICNP'99, pp. 203, Toronto, Canada, 1999.
- [20] M. Mathis, J. Semke, J. Mahdavi ,and T. Ott, *The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm*, Computer Communication Review, Vol.27, No.3, pp. 67–82, July 1997.

- [21] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, *Modeling TCP Throughput: A Simple Model and its Empirical Validation*, In Proc. of SIGCOMM '98, pp. 303–314, Vancouver, CA, September 1998.
- [22] T. Bu, and D. Towsley, *Fixed Point Approximation for TCP behavior in an AQM Network*, In Proc. of SIGMETRICS '01.
- [23] S. Floyd, M. Handley, J. Padhye, and J. Widmer, *Equation-Based Congestion Control for Unicast Applications*, In Proc. of SIGCOMM '00, pp.43–56.
- [24] V. Misra, W. Gong, and D. Towsley, *Stochastic Differential Equation Modeling and Analysis of TCP Window Size Behavior*, In Proc. of Performance '99.
- [25] J. Padhye, V. Firoiu, and D. Towsley, *A Stochastic Model of TCP Reno Congestion Avoidance and Control*, UMASS CMPSCI Technical Report 99-02, February 1999.
- [26] M. Yajnik, S.B. Moon, J. Kurose, and D. Towsley, *Measurement and Modeling of the Temporal Dependence in Packet Loss*, In Proc. of INFOCOM '99, pp.345–352.
- [27] E. Altman, K. Avrachenkov, and C. Barakat, *A Stochastic Model of TCP/IP with Stationary Random Loss*, In Proc. of SIGCOMM '00.
- [28] R. Morris, *Scalable TCP Congestion Control*, In Proc. of INFOCOM '00, Tel Aviv, March 2000.
- [29] R. Morris, *Scalable TCP Congestion Control*, PhD thesis, January 1999.
- [30] R. Morris, *TCP Behavior with Many Flows*, In Proc. of ICNP '97, October 1997, Atlanta, USA
- [31] D. Katabi, M. Handley, and C. Rohrs, *Internet Congestion Control for High Bandwidth-Delay Product Networks*, In Proc. of SIGCOMM '02, Pittsburgh, August, 2002. <http://ana.lcs.mit.edu/dina/>.
- [32] V. Paxson, and S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, Vol.3 No.3 pp.226–244, 1995.

- [33] M. Crovella ,and A. Bestavros, *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, IEEE/ACM Transactions on Networking, Vol.5 No.6 pp.835–846, 1997.
- [34] W. Willinger, M. Taqqu, R. Sherman, and D. V. Wilson, *Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level*, In Proc. of SIGCOMM '95, Cambridge, MA, August 1995, pp. 100–113.
- [35] K. Fall and S. Floyd, *Simulation-based Comparisons of Tahoe, Reno, and SACK TCP*, Computer Communication Review, Vol. 26 No.3 pp. 5–21, 1996.
- [36] J. Mo and J. Walrand, *Fair End-to-End Window-Based Congestion Control*, IEEE/ACM Trans. on Networking, Vol.8, No.5, pp.556–567, 2000.
- [37] A. Mayer, Y. Ofek, and M. Yung, *Approximating Max-Min Fair Rates via Distributed Local Scheduling with Partial Information*, In Proc. of INFOCOM '96, San Francisco, CA, March 1996, pp. 928–936.
- [38] R. Morris, *Scalable TCP Congestion Control*, In Proc. of INFOCOM '00, Tel Aviv, Israel, March 2000, pp. 1176–1183.
- [39] N. Cardwell, S. Savage, and T. Anderson, *Modeling TCP Latency*, In Proc. of INFOCOM '00, Tel Aviv, Israel, March 2000, pp. 1742–1751.
- [40] C. Villamizar, and C. Song, *High Performance TCP in ANSNET*, Computer Communication Review, Vol.24 No.5, 1994.
- [41] D. Clark, and W. Fang, *Explicit Allocation of Best-Effort Packet Delivery Service*, IEEE/ACM Trans. on Networking, Vol.6 No.4, pp.362–373, 1998.
- [42] G. Lo Monaco, A. Feroz, S. Kalyanaraman, and Y. Xia, *TCP-Friendly Marking for Scalable Best-Effort Services on the Internet*, Computer Communication Reviews, Vol.31, No.5, pp.11–19, 2001.
- [43] X. Wu, and I. Nikolaidis, *Sociable Elephants: Fairness Among Long Lived TCP Flows*, In Proc. of SPECTS '02, pp. 502–506, July 2002.

- [44] X. Wu ,and I. Nikolaidis, *On the Advantages of Lifetime and RTT Classification Schemes for TCP Flows*, In Proc. of IPCCC '03, April 2003.
- [45] M. May, J. Bolot, C. Diot, and B. Lyles, *Reasons not to deploy RED*, In Proc. of the IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), June 1999
- [46] K. Ramakrishnan, R. Jain, and D. Chiu, *Congestion Avoidance in Computer Networks with a Connectionless Network Layer. Part IV: A Selective Binary Feedback Scheme for General Topologies Methodology*, Technical Report DEC-TR-510, Digital Equipment Corporation, 1987.
- [47] A. Charny, *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback*, Master Thesis for EECS&EE, MIT, 1994.
- [48] H. Jiang ,and C. Dovrolis, *Passive Estimation of TCP Round-Trip Times*, ACM Computer Communication Review, Vol.32 No.3, July 2002.
- [49] D. Katabi, M. Handley, and C. Rohrs, *Internet Congestion Control for Future High Bandwidth-Delay Product Environments*, In Proc. of SIGCOMM '02, August 2002.
- [50] V. Paxson, *Empirically-Derived Analytic Models of Wide-Area TCP Connections*, IEEE/ACM Trans. on Networking, Vol.2 No.4, pp. 316–336, 1994.
- [51] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith, *Tuning RED for Web Traffic*, IEEE/ACM Trans. on Networking, Vol.9, No.3, pp. 249–264, 2001.
- [52] T. Ye ,and S. Kalyanaraman, *Adaptive Tuning of RED Using On-line Simulation* In Proc. of GLOBECOM '02, Taipei, Taiwan, November 2002.
- [53] *Auckland-II trace data - illustrated*, <http://wand.cs.waikato.ac.nz/wand/wits/auck/2/>.
- [54] D. Chiu ,and R. Jain, *Analysis of Increase Decrease Algorithms for Congestion Avoidance in Computer Network*, Computer Networks and ISDN Systems, pp.1–14, 1989.
- [55] S. Lin ,and N. McKeown, *A Simulation Study of IP Switching*, In Proc. of SIGCOMM '97, Cannes, France, September 1997.

- [56] P. Newman, T. Lyon ,and G. Minshall, *Flow Labeled IP: A Connectionless Approach to ATM*, In Proc. of INFOCOM '96,pp.544–557, San Francisco, CA, March 1996.
- [57] M. Allman, *A Web Server's View of the Transport Layer*, ACM Computer Communication Review, Vol. 30, No. 5, October 2000.
- [58] B. Suter, T. V. Lakshman, D. Stiliadis ,and A. Choudhury *Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing*, IEEE Journals in Selected Areas in Communications, August 1999.
- [59] N. Cardwell, S. Savage ,and T. Anderson, *Modeling TCP Latency*, In Proc. of INFOCOM '00, pp.1742–1751, Tel Aviv, Israel, March 2000.
- [60] I. Yeom ,and A.L. N. Reddy, *Realizing throughput guarantees in a differentiated services network*, In Proc. of ICMCS '99, Florence, Italy, June 1999.
- [61] N. Seddigh, B. Nandy ,and P. Piedad, *Bandwidth Assurance Issues for TCP flows in a Differentiated Service Network*, In Proc. of GLOBECOM '99, Rio De Janeiro, December 1999.
- [62] B. Nandy, J. Ethridge, A. Lakas ,and A. Chapman, *Aggregate Flow Control: Improving Assurances for Differentiated Services Network*, In Proc. of INFOCOM '01, pp. 1340-1349, Anchorage, Alaska, April 2001.