NEW ARCHITECTURES FOR HANDLING FAIRNESS AND CONGESTION RESPONSE CONFORMANCE IN THE INTERNET

By

Kartikeya Chandrayana

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: E.C.S.E.

Approved by the Examining Committee:

Prof. Shivkumar Kalyanaraman, Thesis Adviser

Prof. Christopher D. Carothers, Member

Prof. Koushik Kar, Member

Prof. Biplab Sikdar, Member

Prof. John T. Wen, Member

Rensselaer Polytechnic Institute Troy, New York

May 2003

NEW ARCHITECTURES FOR HANDLING FAIRNESS AND CONGESTION RESPONSE CONFORMANCE IN THE INTERNET

By

Kartikeya Chandrayana

An Abstract of a Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY Major Subject: E.C.S.E.

The original of the complete thesis is on file in the Rensselaer Polytechnic Institute Library

Examining Committee:

Prof. Shivkumar Kalyanaraman, Thesis AdviserProf. Christopher D. Carothers, MemberProf. Koushik Kar, MemberProf. Biplab Sikdar, MemberProf. John T. Wen, Member

Rensselaer Polytechnic Institute Troy, New York

May 2003

© Copyright 2003 by Kartikeya Chandrayana All Rights Reserved

CONTENTS

LI	ST O	F TAB	LES	vii	
LI	ST O	F FIGU	JRES	viii	
AI	BSTR	ACT		xi	
1.	Intro	ntroduction			
	1.1	Solutions for Preventing Congestion Collapse in the Network			
		1.1.1	End System Based Approaches	2	
		1.1.2	Network Based Approaches	4	
		1.1.3	Summary	5	
	1.2	Fairne	ss and Compliance in the Network	6	
		1.2.1	Fairness	6	
		1.2.2	Protocol Conformance	7	
		1.2.3	Summary	9	
	1.3	Contri	bution of this Thesis	10	
		1.3.1	Randomized TCP: An End-System Based Solution for Improving Fairness in a Network of Drop Tail Queues	10	
		1.3.2	Edge-Based Re-marking for Providing Fairness and Compli- ance in Network	11	
	1.4	Organ	ization of the Thesis	13	
2.	Back	ground		14	
	2.1	End S	vstem Based Mechanisms for Preventing Congestion Collapse .	15	
		2.1.1	TCP and its Variants: Congestion Avoidance and Control	15	
		2.1.2	TCP and Drop Tail Queues	17	
		2.1.3	Rate Based Proposals for End-System Based Congestion Control	18	
		2.1.4	TCP Pacing: Solution For Reducing Burstiness of TCP	19	
	2.2	Netwo	rk Based Mechanisms for Congestion Avoidance	21	
	2.3	2.3 Fairness \ldots \ldots \ldots \ldots \ldots \ldots			
		2.3.1	Oblivious Schemes for Providing Fairness in the Network	25	
		2.3.2	Non-Oblivious Schemes for Providing Fairness in the Network	26	
	2.4	Compl	iance in the Network	29	
		2.4.1	End-System Based Schemes for Compliance in the Network	30	

		2.4.2	Network Based Schemes for Compliance in the Network	31
	2.5	Optim	ization: Flow Control, Fairness and TCP Compatibility	31
3.	Rand	lomized	TCP: End System Based Mechanism for Improving Fairness	
	in a	Networ	k of Drop Tail Queues	35
	3.1	Introd	uction	35
	3.2	Rando	mized TCP	36
	3.3	Rando	mized TCP Pseudo-code	38
	3.4	Analyt TCP	cical Characterization of Increase Parameter for Randomized	39
	3.5	Queue ized T	ing Analysis to Show Reduction in Burst Losses with Random- CP	41
	36	Impler	nentation and Simulation Setup	12
	$\frac{0.0}{2.7}$	Impler	nentation on the Linux Kernel	14
	ວ.7 ງ ດ	nipier		44
	3.8	Param	Different Dettlered Der deridth	45
		3.8.1 2.0.0	Different Buffer Sizes	45
		3.0.2	Different BTT	40
	2.0	Throughput Logg Timeoutz Foirmag and Latence		
	5.9	2 0 1	Bull Data Transfor	49
		0.9.1	3 9 1 1 Same BTT	49 49
			3.9.1.2 Different BTT	49
		3.9.2	Short Web Like Transfers	51
		3.9.3	Interaction of Randomized TCP with TCP Reno	52
		3.9.4	Summary	53
	3.10	Bias A	Against Long Flows, Phase Effects, Synchronization and Burst	
		Losses	· · · · · · · · · · · · · · · · · · ·	54
		3.10.1	Bias Against Long Flows	54
			3.10.1.1 Single Bottleneck \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	55
			3.10.1.2 Multiple Bottleneck	57
		3.10.2	Phase Effects	58
		3.10.3	Synchronization	59
			3.10.3.1 Synchronization in Bulk Data Transfer	59
			3.10.3.2 Synchronization with Short Web Transfers	62
		3.10.4	Burst Losses	64

		3.10.5	Summary	. 64
	3.11	Binom	ial Congestion Control Algorithms	. 66
	3.12	Conclu	usions	. 67
4.	Selfis	sh Flow	vs: Characterization and Performance on Drop Tail Queues	. 69
	4.1	Introd	uction	. 69
	4.2	Classe	s of Selfish Flows	. 70
	4.3	Selfish	Rate Control Schemes and their Utility Functions	. 72
	4.4	Aggressive Rate Control Scheme: Control Parameters are Time De- pendent		
		4.4.1	Modifying the decrease parameter β	. 75
			4.4.1.1 Global Stability	. 78
		4.4.2	Modifying the Window Scaling Parameters, k, 1	. 78
	4 5		4.4.2.1 Global Stability	. 79
	4.5	Selfish	Flows and Drop Tail Queues	. 79
	4.6	Summ	ary	. 81
5.	Non for P	Cooper Protocol	rative Flow Control: An Edge-Based Re-marking Framework l Conformance in the Network	. 84
	5.1	Introd	uction	. 84
	5.2	Re-Ma	arking Framework for Managing Non-Conformant Users	. 87
		5.2.1	Network Model, Definitions and Assumptions	. 87
		5.2.2	$Motivation \ldots \ldots$. 89
	5.3	Re-ma	rking Framework for Managing Non-Conformant Users	. 91
	5.4	Impler	mentation \ldots	. 98
		5.4.1	Estimating the Utility Function	. 100
		5.4.2	Estimating the Loss Rate	. 102
	5.5	Simula	ation Results	. 103
		5.5.1	Evaluation of Edge Based Re-Marking Framework on an ECN Enabled Network	. 105
			5.5.1.1 Single Bottleneck \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	. 105
			5.5.1.2 Multi Bottleneck Topology	. 106
			5.5.1.3 Background Traffic	. 107
			5.5.1.4 Cross Traffic \ldots \ldots \ldots \ldots \ldots \ldots	. 109
		5.5.2	Evaluation of Edge Based Re-Marking Framework on a Non ECN Capable Network	. 110

	5.5.2.1	Single Bottleneck
	5.5.2.2	Multi-Bottleneck
	5.5.2.3	Background Traffic
	5.5.2.4	Cross Traffic $\ldots \ldots 114$
5.5.3	Estimat	ing the Utility Function
5.5.4	Sensitiv	ity Analysis of Framework
	5.5.4.1	Effect of Inaccurate RTT Estimate
	5.5.4.2	Effect of Inaccurate Utility Function Estimate 116
5.6 Sumr	nary	
6. Conclusion	ıs	
7. Future Wo	rk	

LIST OF TABLES

3.1	Comparison of Throughput (in pkts/sec), losses and timeouts for TCP Reno Vs (Paced, Random)	53
3.2	Bias Against Longer Flows: Distribution of Throughput in proportion of RTTs with Randomized TCP shows reduction of bias in contrast to TCP Reno	56
3.3	Comparison of Throughput (in Kbps) for different configuration of competing 5 Long flows (RTT=80ms) and 5 Short Flows (RTT=60ms)	56
3.4	Comparison of Throughput (in Kbps) for Multi-Bottleneck Topology	58
3.5	Comparison of Covariance Coefficient of Congestion Window for two flows for TCP Reno, Paced and Randomized. (Value around 0 is Good.)	61
3.6	Comparison of Covariance Coefficient of Congestion Windows for 3 flows for TCP Reno, Paced and Randomized. (Value around 0 is Good.)	61
3.7	Comparison of average number of burst losses in Reno and Randomized TCP (RTCP).	65
5.1	Cross Traffic (Dropping) : Comparison of throughput (packets/sec) for network with DropTail and RED queues with and without re-marking.	114

LIST OF FIGURES

3.1	Packet Sent Times with Randomized TCP $\hfill 1$	42
3.2	Topology used in the simulation	43
3.3	Loss Rate, Throughput and Timeouts for 50 flows as a function of ran- domization interval, for different values of bottleneck bandwidth	46
3.4	Loss Rate, Throughput and Timeouts for 30 flows as a function of ran- domization interval, for different values of bottleneck buffer size	47
3.5	Loss Rate, Throughput and Timeouts for 30, 50 flows as a function of randomization interval, for varying RTT. The RTT varies from 80ms-120ms, the bottleneck bandwidth is 4Mbps and the buffer size is 25 packets	48
3.6	Loss Rate, Throughput, Timeouts and fairness with Bulk Data transfer , each flow having same RTT.	50
3.7	Throughput, Fairness, Loss Rates and Timeouts for a set of flows where each flow has a different RTT.	51
3.8	Latencies for Reno, Paced and Randomized for short and moderate Web like Workloads	52
3.9	Multi Bottleneck Topology used in the simulation.	57
3.10	Single bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.	59
3.11	Phase Effects	60
3.12	Multiple bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.	61
3.13	10 flows Covar. coeff. of Congestion Window for (a) Reno, Paced & Randomized (b) Reno & Randomized, (c) Paced & Randomized	62
3.14	Covar. coeff. of Congestion Window for (a) Reno & Randomized, (b) Paced & Randomized	63
3.15	Covariance coefficients for Paced and Randomized TCP for a transfer of 2500 packets. (Value around 0 is good.)	65
3.16	Performance of Binomial Congestion Control Algorithms with Random- ization	67

4.1	Classification of Selfish Behavior in the Network. Our region of interest is the Selfish Responsive Flows
4.2	Topologies used in the Simulations
4.3	Single Bottleneck : Throughputs (in pkts/sec) for two competing flows, one is TCP while the other is Mis-behaving $(k=0,l=0.5)$ 80
4.4	Multi Bottleneck: Throughputs (in pkts/sec) for 2 competing flows on a network of Drop Tail queues. One flow is TCP while the other is Misbehaving ($k=0,l=0.5$)
5.1	Mapping a non-conformant user to a conformant space
5.2	Model for policing non-conformant users through Penalty Transformation. 86
5.3	Example1: Two competing set of flows through one bottleneck 89
5.4	Example: Three competing set of flows through two bottlenecks 90
5.5	Topologies used in the Simulations
5.6	Single Bottleneck (Marking): Throughputs (in pkts/sec) for two competing flows, one is TCP Friendly while the other is non-conformant with and without Re-Marking
5.7	Single Bottleneck (Marking): Throughputs (in pkts/sec) for ten competing flows, where seven flows are TCP Friendly while three are non-conformant ($k=0$, $l=0.5$) with and without Re-Marking 103
5.8	Single Bottleneck (Marking): Throughputs (in pkts/sec) for three competing flows, where one flow is TCP Friendly while the other two are non-conformant with ($k=0$, $l=0.5$) and ($k=0$, $l=0.2$) resp., with and without Re-Marking
5.9	Multi Bottleneck (Marking): Throughputs for 2, 10 competing flows.106
5.10	Background Traffic (Marking) : Throughputs (in pkts/sec) for two competing flows in a single bottleneck topology, where one flow is TCP Friendly while the other is Non-Conformant with ($k=0$, $l=0.5$). Also there is web-traffic in the background
5.11	Background Traffic (Marking): Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 20% noise

5.12	Background Traffic (Marking): Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 65% noise.	. 109
5.13	Cross Traffic (Marking) : Throughputs (in pkts/sec) for 10 competing flows in a multi-bottleneck topology, where on each bottleneck there are 5 TCP Friendly flows and 5 Non-Conformant with ($k=0$, $l=0.5$), with two-way traffic.	. 110
5.14	Single Bottleneck (Dropping): Throughputs (in pkts/sec) for 2 competing flows on a network of DropTail and RED queues with and without Re-Marking. One flow is TCP Friendly while the other is Non-Conformant ($k=0,l=0.5$).	. 111
5.15	Multi Bottleneck (Dropping): Throughputs (in pkts/sec) for 2 competing flows on a network of DropTail and RED queues with and without Re-Marking. One flow is TCP Friendly while the other is Non-Conformant ($k=0,l=0.5$).	. 112
5.16	Background Traffic (Dropping): Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 65% noise.	. 113
5.17	Estimation of Utility Function for 2 competing flows in a single bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with $(k=0, l=0.5)$.	. 115
5.18	Inaccurate RTT Estimates : Throughputs (in pkts/sec) for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with ($k=0, l=0.5$), when network has inaccurate RTT estimates	. 117
5.19	Inaccurate Utility Function Estimates, 20% Estimation Errors: Throughputs for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate estimates of source's utility function.	. 118
5.20	Inaccurate Utility Function Estimates, 5% Estimation Errors: Throughputs (in pkts/sec) for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate estimates of source's utility function.	. 119

ABSTRACT

One of the reasons for the spectacular growth of Internet has been its ability to support different applications. Though TCP has been the favored transport protocol, increasingly these applications have been using rate control schemes other than TCP. This flexibility of applications to choose their rate control schemes presents us with problem of *protocol non-compliance and fairness*. Traditionally, the network design has coupled the solution of these issues with AQM schemes. However, these AQM schemes are beset with configuration problems and consequently have not been deployed on the Internet. In this thesis we present deployable end-system and edge-system based solutions for protocol compliance and fairness. Specifically, we propose *Randomized TCP* an end-system based solution for improving fairness in the network and an *Edge based Re-marking framework* for providing protocol compliance and broad range of fairness objectives in the network.

Though TCP has served the Internet community well it is known to suffer from a number of phenomena which limit its effectiveness on a network of Drop Tail queues. The main problems which degrade TCP and network performance are: synchronization of congestion windows, phase effects, bias against bursty traffic and unfairness to flows with higher RTTs. In this thesis we propose an *end-system based* solution to some of these problems. Specifically, we propose to introduce randomization into the network by randomizing the sending times of packets in TCP and other similar window based transport protocols. For the TCP case, we call it *Randomized TCP*. In the proposed model, packets of window are sent after an interval of RTT(1 + x)/cwnd, where *cwnd* is the congestion window in packets and x is a random number drawn from an Uniform distribution on [-1, 1]. Our results show that this randomization breaks the synchronization of flows in the network and consequently improves fairness, reduces burst losses and phase effects.

Though the above proposal works well, its performance is constrained in presence of selfish flows in the network. Specifically, the area of concern is protocol conformance, for e.g. TCP-Friendliness. In the present Internet, the users are free to choose a rate control scheme and act unilaterally to maximize their network usage. The network on the other hand wants to allocate resources impartially or according to some specified criteria (e.g. price, service differentiation etc). However, the selfish behavior of the users might be in direct conflict with the objectives of the network. Though AQM schemes have been proposed to achieve conformance amongst users, they do not *always* protect flows from selfish behavior and are constrained by the range of fairness criteria they can provide.

In this thesis we look at "fairness" from the network's perspective and focus on managing the distribution of rate allocations. Specifically, we show that by a penalty transformation we can map a user's utility function to any target utility function. These penalty transformation agents can be implemented on the network edges and work with either marking or dropping. This transformation helps enforce protocol conformance and provides flexibility to provide a broad range of fairness criteria on the network. Another advantage of this framework is that it is independent of the buffer management policy deployed on the network, i.e. it works even on a network of Drop Tail queues.

In summary this thesis proposes novel end-system and edge-system based solutions for improving fairness, enforcing protocol conformance and managing selfish behavior in the Internet.

CHAPTER 1 Introduction

Over the years as the Internet has evolved TCP has formed the backbone of its stability. However, a decade ago the present Internet suffered from a severe congestion control problem, which was called the "Internet meltdown". To prevent such a situation Jacobson [41] proposed the congestion avoidance and control mechanisms for TCP which has subsequently become the de-facto transport protocol for the Internet.

However as the application needs changed newer rate control schemes were proposed. As such we now have an Internet which operates with a spectrum of congestion control schemes, even though TCP remains the most widely used transport protocol. In [30] the authors have argued that these new congestion control schemes can lead to a new congestion collapse and pose the problem of protocol conformance (wherein selfish schemes get an unfavorable share of bandwidth in comparison to TCP).

Though end-system based congestion control mechanisms have helped prevent Internet meltdown they are not sufficient to provide good service under all circumstances. Specifically network and end-user performance may degrade in presence of Drop Tail queues and different rate control schemes. Also end system based solutions constrain the choices of flow control protocols which might be available to any application. Towards addressing these issue, router (or network) based schemes like Active Queue Management (or AQM) have been proposed as a complement to end-system based congestion control schemes.

However these AQM proposals are beset with configuration problems and also require upgradation of the network (i.e. each bottleneck must have the AQM enabled). As a result of these implementation drawbacks, the Internet still operates with Drop Tail queues. Considering that AQM schemes are still to be widely deployed on the Internet and presence of different congestion control schemes, in this thesis we look at *deployable* end system and network based solutions to improve fairness and protocol conformance in the network.

In the following sections we first present the various solutions for preventing congestion collapse, providing fairness and protocol conformance in the network. Specificially we look at the network and end point solutions and discuss their current deployment in the network. Thereafter we present our proposals for protocol conformance and improving fairness.

1.1 Solutions for Preventing Congestion Collapse in the Network

The Internet protocol architecture is based on a connectionless end-to-end packet service using the IP protocol. The advantages of its connectionless design, flexibility and robustness have been amply demonstrated. However, these advantages are not without cost: careful design is required to provide good service under heavy load. In fact, lack of attention to the dynamics of packet forwarding can result in severe service degradation or "Internet meltdown".

As a result of this meltdown considerable research has been done on Internet dynamics and many solutions have been suggested to avoid it. These proposals can be broadly classified into two categories a) end-system based solutions, e.g. TCP and other congestion control schemes and b) network based solution. In this section we briefly discuss these proposals, their advantages and disadvantages and their current deployment in the Internet.

1.1.1 End System Based Approaches

The end-system based solutions consists of source or receiver based congestion control schemes. These schemes try to avoid congestion in the network by cutting down their transmission rate, whenever congestion is detected. The original fix for the congestion collapse (or Internet meltdown) proposed by Jacobson in 1988 [41] is one such scheme. In particular, Jacobson proposed the congestion avoidance and control features in TCP and since then TCP has been the mainstay of the Internet.

These end-system based solutions can operate with and without network support. In absence of network support the network employs simple queuing at the routers in which the packets are admitted till the queue has space. This queuing policy is called Drop Tail. Though simple to implement, Drop Tail queue do not try to manage congestion in the network, infact it is left to the end-system based application.

Though TCP has served the Internet community well it is known to suffer from a number of phenomena which limits its effectiveness when operated over a network of Drop Tail queues. The main problem which degrades TCP and network performance are: synchronization of congestion windows (or correspondingly the loss instances) causing alternate overloading and under-loading of the bottleneck [58, 72, 83]; phase effects wherein a certain section of flows face recurrent losses [32]; unfairness to flows with higher RTTs [27]; bias against bursty traffic [34] ; delays and losses due to the bursty nature of TCP traffic [83, 4].

The tail-drop discipline allows queues to maintain a full status for long periods of time. This is because Drop Tail signals congestion only when the queue has become full. If the queue is full, an arriving burst will cause multiple packets (from same or different flows) to be dropped causing global synchronization [34]. This synchronization can be attributed to two reasons: (1) the sliding window flow control of the TCP, which produces bursts of packets and (2) the Drop Tail queue at the bottleneck, which drops all packets when the buffer is full [37]. *Synchronization* of windows and loss events for flows sharing common links causes alternating periods of overload and under-load thereby leading to inefficient resource utilization. In some situations Drop Tail queuing allows a single connection or a few flows to monopolize queue space, preventing other connections from getting room in the queue. This "lock-out" phenomenon is often the result of synchronization [34, 13].

Phase effects refer to conditions where in the bandwidth-delay product of the path of a flow is not an integral multiple of the packet size [32]. Phase effects cause a specific section of competing flows to experience recurrent drops causing unfair distribution of bandwidth and increased latency. Phase effects are manifested in the network preferentially dropping packets from a specific subset of flows thereby reducing their throughput.

Drop Tail queues suffer from a problem called, "full queues", which implies

that Drop Tail queuing maintains sustained long or full queues. The sliding window protocol of TCP and the persistent full queues often results in burst losses. These burst losses causes Drop Tail queues to differentiate against TCP like schemes [34]. It has been widely shown that TCP can recover well from a single packet loss but with burst losses it often times out [75]. Consequently, these burst losses also increase the delays. It has also been reported that these burst losses are the primary reason for the bias against flows with longer RTT [3].

Drop Tail queues also do not protect flows. As noted earlier because of synchronization Drop Tail queues can let some flows monopolize the buffer space. Also, given that there are various congestion control schemes in the network, by not differentiating amongst flows, Drop Tail queues allow aggressive sources to get more bandwidth. Flows which do not react to congestion indications will push the responsive flows out of the queue and will always take up bandwidth worth their transmission rate [48]. Thus by introducing burst losses and by not protecting flows, Drop Tail queues aggravate the problem of unfair equilibrium rate allocations in the network.

1.1.2 Network Based Approaches

Though the end-system based congestion avoidance and control mechanisms are necessary and powerful, they are not sufficient to provide good service under all circumstances. Primarily there is a limit to how much control that can be accomplished from the end of the network. Specifically these problems were highlighted in the previous section and can be chiefly attributed to the full queues and lock-out behavior of the Drop Tail queues. Thus some mechanisms are needed in the routers to complement the endpoint congestion avoidance mechanisms.

Active Queue Management (AQM) was suggested as a pro-active way of managing queue at the bottleneck router. The pro-activeness was defined to be able to drop few packets before the queue gets full thereby signaling sources to cut their rates on account of impending congestion. This in turn help solved the problem of full queues. The solution to the full queues problem implied that there would be space in the queue to enque packets which consequently solved the lock-out problem of Drop Tail queues.

Random Early Drop (or RED) [34] was one such AQM proposal wherein the authors suggested to probabilistically drop packet when the queue size gets above a certain threshold. This probabilistic dropping distributed losses over time thus making them appear independent. It also introduced randomization at the bottleneck which in turn broke synchronization amongst flows and improved network performance. Also, by sending early congestion signal (by dropping a packet before the queue actually gets full) helped manage queues efficiently and also provided space to accommodate bursts. Thus RED avoids burst losses, synchronization, reduces the bias against long RTT flows and prevents timeouts.

However it's been almost a decade since the RED proposal but the Internet still operates with Drop Tail queues. This can be explained by lack of guidelines to set RED. Studies have shown that if not properly configured the performance of TCP with RED queues may even be worse than those with Drop Tail queues. Specifically, in [55, 19] the authors show that the probability of consecutive drops increases with RED queues. Though there have been some studies on how to configure RED these works attempt to configure only one or a set of parameters and as such have not found much favor with the network operators [16, 38]. Besides RED there have been other AQM proposals which have fewer parameters to configure and crisper guidelines for setting them [47, 7, 24, 76, 25]. But in-spite of numerous AQM propositions the network still operates with Drop Tail queues and consequently the problems of TCP and Drop Tail queues exist to this day.

1.1.3 Summary

The policies outlined for preventing congestion collapse require either endsystem support in form of congestion control scheme or router based schemes like AQMs. However there is a limit to the control which can be achieved by using the end-point congestion schemes. In absence of network control we have seen that the flows are subjected to burst losses and can get synchronized which in turn limits the effectiveness of TCP. Further, end-point congestion control scheme do not protect flows, on the contrary they allow some flows to monopolize the buffer space. Network control for preventing congestion collapse was envisioned in form of Active Queue Management. RED was one such proposal which absorbed bursts by probabilistically enqueing packets. This introduced randomness at the bottleneck and helped avoid synchronization of flows. However RED's performance is highly sensitive to its parameter configuration, so much so that at times the performance of Drop Tail queues might be better than RED queues. This problem is further compounded by the lack of guidelines for setting these parameters. As such the network still operates with Drop Tail queues.

To summarize, due to configuration and implementation problems with AQM the Internet still operates with Drop Tail queues. As a result, the problems of bursts losses, flow synchronization, bias against flows with longer RTT and manipulation of buffer space by selfish and unresponsive flows persist. These problems in turn limit the effectiveness of TCP and degrade the performance of the network.

1.2 Fairness and Compliance in the Network

1.2.1 Fairness

Fairness can be defined in a number of ways but its essence in each of these definitions is that it is some measure of the distribution of the allocated rate amongst users. Fairness is related not only to the network but also to the end-system's congestion control scheme. Often the end-system's objective is to be fair to the other competing user's while the network's objective is that it does not arbitrarily penalize or differentiate amongst various competing user.

Traditionally, the Internet has relied on the "end-to-end" congestion control model like TCP (or alternate transport protocols) where end users choose a rate control scheme, and the network merely drops or marks packets during congestion as a method to convey the penalty or price [44, 46, 50]. One implication of this model is that end-systems are free to choose *any* rate control scheme (which in the Kelly's framework [44] means that they can pick any desired utility function). Kelly, Low etal have shown that a particular class of fairness is associated with every utility function.

The network on the other hand seldom differentiates between flows. It drops

(or marks) packets obliviously, i.e. drops packets whenever there is no space in the router queue or if the router queue length crosses a certain threshold. Drop Tail queuing, RED and many of RED's variants can be classified as oblivious queuing disciplines. This oblivious dropping coupled with the flexibility to end system to chose a rate control scheme makes the problem of providing fair rate allocations to all users hard.

The "fair" equilibrium allocations in an oblivious network therefore depend upon the utility functions chosen freely by users. These equilibrium allocations, though fair under Kelly's framework, might be unfair from network perspective.

Moreover the fair rate allocation problem is further compounded by the fact that there is no single definition of fairness. The two most common definitions of fairness are max-min [12, 57] and proportional fairness [44]. In max-min fairness criteria the objective is to maximize the minimum unsatisfied rate allocations. Thus given the same network conditions, two competing flows should get equal share of the bottleneck. On the other hand, in proportional fairness the rate allocations are in proportion to the network resources being used. But all the same, a more general definition of fairness, (p,α) fairness is defined in [57]. Thus irrespective of user's rate control schemes it is for the network provider to decide the criteria for allocating resources amongst users.

Over the years equal allocations and Max-Min fairness [57] have formed the network's view of fair allocations. As such, AQM schemes and schedulers deployed at every bottleneck have been used to enforce conformance with these definitions, by penalizing misbehaving users [52, 26, 64, 48, 73, 39]. For example, CHOKe [64] tries to enforce Max-Min fairness [57] across the network. Similarly fair queuing and it's variant have also been used to provide Max-Min fairness. Finally to summarize, any arbitrary fairness objective cannot be achieved by AQM schemes though they could be arrived at by use of schedulers throughout the network.

1.2.2 Protocol Conformance

The congestion control scheme in TCP has been the focus of numerous studies and consequently gone through lots of changes. These changes were also motivated by varying needs of the applications using the Internet. As such, even though TCP remains the most widely used protocol, we have now a spectrum of congestion control schemes. In [30] the authors show that absence of end-to-end congestion control schemes or presence of selfish users could not only lead to TCP being beaten down but also may even result in congestion collapse.

This thus represents the problem of protocol conformance. In absence of compliance to a set of protocols, for example TCP, we might be faced with the problem of TCP flows being singled out and gets rates which are (significantly) less than their fair share. This problem is further highlighted in presence of unresponsive flows. (Flows which do not cut down their rates upon receipt of congestion indication are called unresponsive flows.) These unresponsive flows can shut-out TCP because on occurrence of congestion, TCP will cut its rate and the unresponsive flows will step in to take the available bandwidth. A similar problem is posed by responsive selfish flows (i.e. flows who react to congestion indication but are selfish as compared to TCP) in the network. Specifically, these flows could have a rate increase policy which is faster than TCP and some flows could have a rate decrease policy slower than that of TCP.

Given that TCP is the most widely used transport protocol, Floyd etal proposed the guidelines for managing and designing new congestion control schemes such that they were *friendly* to TCP. A flow is deemed TCP-Friendly if its sending rate does not exceed that of a conformant TCP flow in same circumstances. This TCP-Friendly definition can further be loosened to the following relationship between the sending rate, x, and loss rate, $p: x \propto \frac{1}{\sqrt{p}}$. This TCP Friendliness can also be understood as protocol compliance, as all flows try to be conformant to TCP.

TCP-Friendliness is the criteria not only for safeguarding TCP flows but also for enforcing some kind of fairness in the network. (TCP-Friendliness ensures Minimum Potential Delay Fairness across the network [46]) Further, we could easily expand TCP Friendliness definition to encompass a larger range of rate control scheme which can be done by relaxing the relationship between the sending and loss rates. However, enforcing TCP Friendliness on the network remains a challenging question. In [30] the authors argue that router-based mechanisms are needed to administer TCP-Friendliness. Other than these router based schemes, another way of enforcing TCP Friendliness could be design of end-point congestion control algorithms which are TCP-Friendly.

In [9, 82, 42, 70] authors have proposed a general class of TCP-Friendly congestion control schemes. Though these proposals are encouraging they solve only a part of the TCP-Friendliness problem because the end-users may willfully choose to ignore these TCP-Friendly guidelines. As such it becomes imperative to have router-based mechanisms for enforcing TCP-Friendliness. In [30] the authors point out using per-flow scheduling or pricing mechanisms for enforcing TCP-Friendliness. However, till date to the best of our knowledge, no such per-flow scheduling or pricing mechanisms have been proposed or deployed to achieve TCP-friendliness on the network.

1.2.3 Summary

From the above discussion it follows that protocol conformance and fairness are very closely related. Protocol conformance guarantees a certain kind of fairness, for example TCP-Friendliness will result in a minimum potential delay fairness across the network [46]. Similarly any fairness definition can always be translated to another protocol conformance.

Traditionally Max-Min fairness has formed the network's definition of fairness. This definition aims to provide equal allocations to different flows. However TCP allocates rate in proportion to the the round-trip times (RTT) of the flows and loss rate. This then stands in contradiction to the network's traditional fairness goals. Thus AQM and scheduling disciplines which enforce Max-Min fairness do nothing to enforce TCP-Friendliness.

Also these different schemes for providing network-wide fairness have their own drawbacks:

- We would need AQM or scheduler support throughout the network. This implies that we will have to make changes in the core.
- TCP-Friendly criteria constrains the choice of end-point congestion control schemes for users. This might also infringe with the requirements of different

protocols as these needs might not always be satisfied with TCP.

• Both AQM/Schedulers and TCP-Friendliness cannot provide a broad range of fairness criteria in the network.

1.3 Contribution of this Thesis

The goal of this research is to look at ways to improve fairness and compliance in the network. In this regard we propose two *deployable* solutions, one end-system based for improving fairness in a network of Drop Tail queues and the second an edge-based scheme for providing compliance and a broad range of fairness objective in the network.

1.3.1 Randomized TCP: An End-System Based Solution for Improving Fairness in a Network of Drop Tail Queues

In this thesis we look at an end-based solution to some of the problems of TCP and Drop Tail queues. Specifically, we propose to introduce randomization into the network by randomizing the sending times of packets in TCP and other similar window based transport protocols. For TCP we call this solution, *Randomized TCP*. In Randomized TCP, instead of sending back to back packets, the packet sending times are randomized. In particular, successive packets of a window are sent after an interval of RTT(1 + x)/cwnd, where *cwnd* is the congestion window in packets and x is a random number drawn from an Uniform distribution on [-1,1]. This solution is distributed, can be implemented at the end systems and thus is very attractive from an implementation perspective.

Our results show that Randomized TCP reduces phase effects and synchronization. Also it substantially reduces burst losses and removes the bias against longer RTT flows. In addition, the benefits of randomization can be reaped even when it is partially deployed. Randomized TCP performs better than or as well as TCP Reno, independent of the capacity and buffer size at the bottleneck and for both short and long flows. The performance improvements can be seen in throughput, fairness, loss rates, timeouts and latency of the flows. In summary *our proposal can emulate* the beneficial effects of RED in a distributed manner without the complexities and unfavorable aspects of parameter tuning of RED.

Randomized TCP is an end-based solution for improving fairness in the network and reducing the phenomena which limit the effectiveness of TCP when operating with Drop Tail queues. However, it requires a presence of Randomized flow at every bottleneck to break the synchronization at that router and improve network performance. Since Randomized TCP is end-system based solution it might not proliferate the network well enough to improve network performance. Moreover end-based systems do not protect flows and neither perform congestion avoidance. To remedy these problems we would need to monitor flows inside the network and by implication need AQM. In this section we outline a network based solution in order to improve network performance.

1.3.2 Edge-Based Re-marking for Providing Fairness and Compliance in Network

One of the aims of this research is to look at effects of selfishness of users in a network; specifically, to study in what ways and to what extent a selfish user can *deliberately* degrade the performance of other users (in the network) in order to improve his performance. First, we will look at ways to define mis-behavior of users and then follow up with analyzing the effect of selfishness on equilibrium rate distribution in a network. The next objective is to suggest scalable ways to identify the mis-behaving users in the network. We will also evaluate the stateless architectures as a means of identifying and penalizing selfish users. Towards this end we suggest an optimization model for managing the selfish behavior in the Internet. Ultimately, the aim of this thesis is to come up with a *deployable* architecture which will enable network providers to restrict and manage the selfish behavior in their network.

From the earlier discussion it follows that end-system congestion control schemes are not sufficient to provide fairness and protocol compliance in the network and need some form of network support. However, AQM schemes can not provide a broad range of fairness objective in the network and by implication cannot enforce protocol conformance. This is because most of them are oblivious to the competing flows and thus impose same penalties on all sources. We illustrate this with the help of some simulation scenarios. Using these results we first classify flows according to their response to congestion indication. From this classification we then define conformance and selfish behavior. We then explore the selfish behavior of protocols, specifically we derive the conditions under which new selfish protocols can be obtained while keeping the network stable. Through these definitions of selfishness and conformance we show that rate allocations in the network can be unfair and more importantly do not always comply with TCP-Friendliness or any other protocol conformance objective. This unfair sharing of the bottleneck is then our motivating factor for studying ways to achieve conformance and fairness in the network.

In this thesis we look at "fairness" from the network's perspective (rather than the end user's perspective) and focus on managing the distribution of rate allocations. We achieve this by transparently managing the effective range of user's utility functions. More specifically, users may choose arbitrary utility functions, but the edge of the network can re-map these utility functions into a target range of utility functions. Interestingly this re-mapping is a simple consequence of the duality framework of Low et. al. [50] and can be easily implemented at the edge of the network. Internal routers of the network function as usual, i.e. they may mark, or drop packets using any AQM scheme (including drop-tail policy). Broadly this thesis also suggests that management of mis-behaving or non-conformant flows need not be coupled with AQM design, and can be simply viewed as an edge network based policing question. Our mechanisms may also be thought of as a new class of "traffic conditioning" techniques [23], where the "conditioning" is achieved by manipulation of the feedback stream rather than manipulation of the packet stream.

The re-marking framework presented in this thesis can also be extended to provide service differentiation. Rather than mapping utility function the utility function of the sources to a single objective utility function, we could instead map it to a range of target utility functions and thereby differentiate between flows. This solution is attractive because it can be achieved irrespective of the congestion control scheme employed by the user and works independently of AQM scheme deployed on the network. Moreover, it can work with Drop-Tail queues also.

1.4 Organization of the Thesis

This thesis looks at the question of improving fairness and protocol compliance in the network through use of end-system and network based algorithms. In Chapter 2 we review the existing work on congestion control and AQM. Chapter 3 first outlines the problems with Drop Tail queues and TCP and then presents an endsystem based algorithm, called Randomized TCP, for emulating AQM behavior on a network of Drop Tail Gateways. In Chapters 4 and 5 we study the impact of selfish behavior on the network. We first define selfish behavior in Chapter 4 and outline ways in which selfish rate control schemes can be generated. Then, we use these selfish schemes use to show that Randomized TCP and other end system based schemes are insufficient to provide a fair service to all users in the network. In Chapter 5 we present an edge system based re-marking framework for managing selfish flows in the network and providing protocol conformance. Finally in Chapters 6 and 7 we present the conclusions and the future work respectively.

CHAPTER 2 Background

In this chapter we review the need of congestion control in the Internet and its objectives. Thereupon we will discuss the end-system based proposals for congestion control (e.g. TCP and its variants) as well as network based proposals i.e. AQM. This discussion on congestion control brings us to the question of how resources are shared between users, i.e. the problem of fairness. For these purposes we first review the definitions of fairness and then the various schemes for achieveing fairness in the network. Finally we discuss the question of protocol compliance and flow control optimization framework.

The rest of the chapeter is organized as follows:

- We begin with a review of end point schemes in Section 2.1 for congestion avoidance and control. In particular we review sliding window based protocols like TCP and it's variants in Section 2.1.1 and it's performance on a network of Drop Tail queues in Section 2.1.2, rate based proposals in Section 2.1.3 and finally we discuss Paced TCP a flow control proposals which is hybrid of rate and window based schemes in Section 2.1.4.
- In Section 2.2 we survey the network based mechanisms for preventing congestion collpase.
- Section 2.3 reviews the distribution of rates in a network or in other words, *fair-ness*. Section 2.3.1 and 2.3.2 discuss the oblivious and non-oblivious network based proposals for achieving fair rate distribution in the network, respectively.
- Protocol Compliace and mechanisms for achieving it are discussed in Section 2.4 and 2.4.2 respectively.
- Finally in Section 2.5 we review the optimization framework proposed for flow control. Also within the context of flow optimization we also review the definition of fairness and protocol compliance.

2.1 End System Based Mechanisms for Preventing Congestion Collapse

In 1980s lack of attention to the dynamics of packet forwarding on the Internet resulted in severe service degradation or *congestion collapse*. Since this congestion collapse considerable research has been done on Internet dynamics and many solutions have been suggested to avoid it. The original fix for congestion collapse was provided by Van Jacobson in 1988 as some modifications to TCP [41]. Ever since, TCP has been the backbone of the modern Internet.

2.1.1 TCP and its Variants: Congestion Avoidance and Control

TCP is a sliding window based transport protocol where the window is increased upon successful reception of acknowledgments (Ack). This ensures that TCP gradually probes and takes all the available bandwidth. However, this probing will result in a situation where the sender's sending rate exceeds the network capacity and at that point the network will drop the excess packets. These packet losses are construed as sign of congestion by the TCP and it reacts to it by cutting its rate (or decreasing window).

In TCP, the sender maintains a congestion window, *cwnd* which represents the number of packets outstanding in the network, i.e. packets which have not been acknowledged. Upon setup of a connection the *cwnd* is set to 1 and TCP sends out one packet. Subsequently on receipt of every acknowledgments TCP sends an extra packet into the network. This window increase phase is called Slow Start and is characterized by the exponential increase in window size. However this window increase will soon exhaust the network's capacity and excess packet(s) will be dropped. When TCP detects this packet loss (in Slow Start) it is construed as the end of the Slow Start and the TCP re-transmits the lost packets and halves its congestion window. Thereafter it enters the congestion avoidance phase where the window increase is much slower (as compared to Slow Start). In congestion avoidance phase TCP puts an extra packet only when a window worth of packets have been acknowledged. Thus during congestion avoidance the window increase is linear. Jacobson proposed that in the event of loss, a timer at TCP sender will timeout while waiting for the Ack. On the expiry of this timer TCP will retransmit the lost packet and halves the congestion window, *cwnd* and stores it in a variable called *ssthresh*. TCP then resets its *cwnd* to 2 and does a Slow Start till its *cwnd* is equal to *ssthresh*, thereupon it enters the Congestion Avoidance phase.

This reaction of TCP to congestion was found to be severe in most cases and as such there were proposals to remedy these significant window cuts. Fast Retransmit and Recovery (FRR) was proposed as a means to eliminate timeouts for retransmitting a packet [77]. In FRR, when the receiver gets out of order packets it sends out duplicate Acks for the first in sequence packet. When the sender receives three duplicate Acks, it detects that a packet is lost and it retransmits the first in sequence packet. It also sets the *ssthresh* to half of the *cwnd* value and then resets the *cwnd* to *ssthresh* + 3. There after for every duplicate Ack it receives it increments the congestion window by 1. Also, if the number of outstanding packets in the network is less than the congestion window, the sender sends new packets in to the network. Finally when the sender receives the Ack for the retransmitted packet it resets it *cwnd* to *ssthresh* and continues in the Congestion Avoidance phase. Thus FRR prevents TCP into timing out for every lost packet.

There have been other proposals to optimize the TCP and the most notable amongst those has been TCP SACK. Selective Acknowledgment (SACK) is a strategy wherein the receiver can inform the sender about all segments that have arrived successfully, so the sender needs to retransmit only the segments that have actually been lost.

Another proposal which merits mention is TCP Vegas [14]. Unlike other TCP proposals which use packet loss or marking as a congestion notification TCP Vegas uses queueing delays to decipher congestion. TCP Vegas relies on the fact that during congestion the queues will build up at the bottlenecks and as such the queueing delay will increase. This increase in queueing delay is construed as a sign of congestion and TCP Vegas decreases its window by one packet, otherwise it increases its window linearly. Since the window increase and decrease in TCP Vegas is small it is most likely to converge to the optimal bandwidth.

2.1.2 TCP and Drop Tail Queues

TCP and other similar congestion control schemes result in bursty traffic. This burstiness can be attributed to three main reasons. One, when the window is increased the last two packets are sent back to back. Two, in presence of congestion on the reverse path the Acks arrive back to back and as such the packets sent are back to back. And finally, when an Ack for a retransmitted packet arrives it might result in release of a previously stalled window which might in turn lead to back to back transmissions. In order to improve the performance of the network buffers are provided at the links to absorb these bursts.

The traditional technique for managing buffers at routers has been to set a maximum length (in terms of packets) for the buffer, accept packets for the buffer until the maximum length is reached, then reject (drop) subsequent incoming packets until the queue decreases (because a packet from the queue has been transmitted). This technique is known as "Drop Tail", since the packet that arrived most recently (i.e., the one on the tail of the queue) is dropped when the buffer is full.

However this simplistic buffer management has many problems which limit the effectiveness of end-to-end congestion control algorithms. Drop Tail queueing in some situations allows a single connection or a few flows to monopolize queue space, preventing other connections from getting room in the queue. This "lock-out" phenomenon is often the result of synchronization or other timing effects [34, 13].

The Drop Tail discipline allows queues to maintain a full (or, almost full) status for long periods of time, since Drop Tail signals congestion (via a packet drop) only when the queue has become full. It is important to reduce the steady-state queue size, and this is perhaps queue management's most important goal. However, this does not take into account the critical role that packet bursts play in Internet performance. If the queue is full or almost full, an arriving burst will cause multiple packets to be dropped. This can result in a global synchronization of flows throttling back, followed by a sustained period of lowered link utilization, reducing overall throughput [34]. Further these burst losses and phase effects also cause a bias against longer round trip time flows [34, 33].

Phase effects refer to conditions where in the bandwidth-delay product of the

path of a flow is not an integral multiple of the packet size [33]. Phase effects cause a specific section of competing flows to experience recurrent drops causing unfair distribution of bandwidth and increased latency. Phase effects are manifested in the network preferentially dropping packets from a specific subset of flows thereby reducing their throughput.

Another drawback of Drop Tail queues is that they don't protect flows, i.e. it allows for a section of flows to monopolize the entire bandwidth [13]. This is especially important in the current Internet where the end system has a flexibility of choosing its congestion control scheme. This then raises the question of unfair sharing of the bottlenecks as the aggressive flows might corner a larger share of the bandwidth.

2.1.3 Rate Based Proposals for End-System Based Congestion Control

Traditionally flow control algorithm were envisioned on sliding window protocol. However these sliding window protocols resulted in bursty traffic which brought with it host of other problems. Also there were lot of algorithms for which TCP's rate cut was considered too drastic and they needed smoother rate control protocol. For these purposes rate based flow control protocol were proposed [40, 68, 29, 69]. In this section we will review some of these proposals.

Jacobs [40] presents a scheme that uses the congestion control mechanisms of TCP, however, without retransmitting lost packets. In his scheme, the sender maintains a transmission window that is advanced based on the acknowledgments of the receiver, which are sent for each received packet. The sender then uses the window to calculate the appropriate transmission rate. Rejaie et al. present in [68] an adaptation scheme called Rate Adaptation Protocol (RAP). Just as with TCP, every packet sent is acknowledged by the receivers and these acknowledgment the sender estimates the round trip delay. If no losses are detected, the sender periodically increase its transmission rate additively as a function of the estimated round trip delay. Upon detection of a loss the rate is reduced by half in a similar manner to TCP. However, this approach as well as the one presented in [40] do not consider the cases of severe losses that might lead to long recovery periods for TCP connections. Hence, the fairness of such an approach is not always guaranteed.

In [61] the authors proposed a analytical model for calculating the average goodput of a TCP connection. Using this model Padhye et al. [29] present a scheme in which the sender estimates the round trip delay and losses based on the receiver's acknowledgments. In the case of losses, the sender restricts its transmission rate to the equivalent TCP rate calculated using TCP's throughput formula proposed in [61] otherwise the transmission rate is doubled. While the scheme behaves in a TCP-friendly manner during loss phases, its increase behavior during underload situations is rather arbitrary. Specifically, it might result in severe unfairness as the adapting end system might increase its transmission rate much faster than a competing TCP connection.

TCP Emulation At Receivers (TEAR) [69] is a combination of window and rate based congestion control. It features a TCP-like window emulation algorithm at the receivers, but the window is not used to directly control transmission. Instead, the average window size is calculated and transformed into a smoothed sending rate, which is used by the sender to space out data packets.

2.1.4 TCP Pacing: Solution For Reducing Burstiness of TCP

Sliding window based protocols like TCP often send packets in burst. As such the performance of sliding window protocols suffers on a network of Drop Tail queues. On the contrary, rate based schemes send out packets at regular intervals thus avoiding burst transmissions. However, since rate based schemes loosely observe the packet conservation principle they at times can be less responsive to network congestion. TCP Pacing [83] is a hybrid approach between window based schemes and rate based schemes. In pacing, packets to be sent in a window are spaced by $\Delta = RTT/cwnd$. This spacing of packets avoids back to back transmissions and hence removes the burstiness of TCP.

Pacing was first suggested in [83] as a correction for the compression of acks due to cross traffic. Since then the concept of pacing has been applied to slow-start, after a packet loss and after an idling time in case of web traffic [8, 62, 35, 6, 80]. In order to speed up web connections the authors in [62] suggest using pacing during the slow start as means for Ack clocking. Similar results have been reported in [6] where the authors show that performance of slow start can be improved by use of pacing. Pacing has also been suggested for improving TCP performance with asymmetry [8] and on high bandwidth delay product links [65]. In [65] the authors evaluate pacing over the entire lifetime of TCP as a means for reducing queueing bottlenecks in wireless, high bandwidth delay networks. In [35] the authors have proposed a fast web protocol, WebTP which uses pacing during congestion avoidance phase.

Rate Based Pacing has been suggested in [80] to improve startup after idling. Slow-start restart occurs when bursty data is periodically sent over a TCP connection. TCP depends on ACK clocking for flow control. Idle periods in the connection cause this clocking mechanism to break down. In [80] the authors propose a rate-based pacing (RBP), an intermediate approach to data transmission after an idle period. RBP paces outgoing packets at a certain rate until the ACK clock is restarted. Thus RBP attempts to provide a compromise between the extremes of sending back-to-back bursts and restarting with slow start.

In [4] the authors have done an exhaustive study of pacing with different operating characteristics. They show that with long flows pacing removes synchronization, improves fairness over TCP Reno and achieves the same throughput as TCP Reno. Through simulations they show that Pacing gets synchronized during the slow start, but in the congestion avoidance phase it has a de-synchronizing effects leading to slightly higher throughput. Even in presence of flows (sufficiently long) with different round-trip times pacing was shown to increase fairness with the same throughput as of TCP Reno. However, in presence of short flows the authors show that Pacing gets synchronized causing larger latencies. They contend that by evenly spacing the packets, pacing delays the congestion point, thus allowing the sources to ramp up rates, and finally on onset of congestion causing synchronized drops. This results in lower throughput and higher latencies. Also, the authors show that when Paced TCP is competing against TCP Reno it gets beaten down.

A modified version of pacing is also evaluated in [43]. In [43] the spacing interval is defined as $\frac{RTT}{cwnd+V}$, where V is the tunable parameter, which controls the aggressiveness of the pacing. However, the effect of this scheme on the synchronization of flows, phase-effects, bias against long RTT flows etc is not investigated. They observe that with bulk data transfer the modified pacing shows results similar to TCP Reno. However, for a web-like load model, the modified paced TCP exhibits lower packet loss than TCP and also the average transfer latencies are lower. However the proposal [43] do not discuss the parameter setting for V and it's effect on the pacing scheme. Also, they do not consider the case where TCP Reno and Paced TCP are multiplexed on the same link.

2.2 Network Based Mechanisms for Congestion Avoidance

From the discussion in previous section it is clear that the TCP congestion avoidance mechanisms, while necessary and powerful, are not sufficient to provide good service in all circumstances. However, there is a limit to how much control can be accomplished from the edges of the network. Some mechanisms are needed in the routers to complement the endpoint congestion avoidance mechanisms.

Active Queue Management (AQM) was proposed to complement the endsystem based congestion avoidance mechanism. The AQM proposal involved a proactive management of the bottleneck queue. Specifically it was proposed that by dropping some packets before the queue gets full is an early enough indication for the sources of impending congestion. As such these sources will react to these packet losses by cutting down their rates and as a result the queue build up at the routers won't be large. This in turn implied that burst losses and synchronization of loss events could be avoided.

By keeping the average queue size small AQM will provide greater capacity to absorb naturally occurring bursts without dropping packets. Also the small queue size reduces the delays seen by flows. This is particularly important for interactive applications such as short Web transfers, Telnet traffic, or interactive audio-video sessions. AQM can also prevent synchronization of loss events by ensuring that there will almost always be a buffer available for an incoming packet. For the same reason, active queue management can prevent a router bias against low bandwidth but highly bursty flows.

Random Early Drop (RED) [34] was the first significant AQM proposal which

advocated enqueing packets probabilistically. RED achieves this by comparing the time averaged value of queue length to a threshold and then probabilistically deciding whether to enque the packet or not. Further the probability of enqueing a packet decreases as the average queue length increases. Thus, if the queue has been mostly empty in the recent past, RED won't tend to drop packets. On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped.

RED operates with 5 control parameters and they are: the two thresholds, the minimum threshold min_{th} and the maximum threshold max_{th} , the queue averaging parameter w_q , the length of the buffer, B and the maximum dropping probability, max_p . When the average queue is in between the two thresholds packets are enqueued probabilistically, where the dropping probability increases linearly as a function of max_p and the average queue length (the dropping probability is 0 at min_{th} and 1 at max_{th}). However if the average queue crosses the maximum threshold all incoming packets are dropped.

Though RED solves many problems of drop tail queues it is not without its share of problems. The biggest concern with RED is it's configuration. RED has 5 operational control parameters and there are no fixed guidelines for tuning them. Further it has been shown that if RED is not properly configured can result in performance degradation, so much so that it is even worse than Drop Tail queues [55, 19]. Though recently there have been some proposals for configuring RED they are limited by the ability to configuring only a set of these control parameters [16, 38]. Thus in absence of strict guidelines RED has not found much favor with network operators.

Besides the problem of configurations, RED also does not protect flows [48]. When hit with a mixture of responsive and unresponsive sources, RED allows unfair bandwidth sharing. This is because RED enforces equal loss rates on each flow, irrespective of their bandwidth. As such if there are flows which do not respond to congestion then they will eventually corner bandwidth worth their sending rates and in the process beat down TCP or other responsive flows. A similar situation can be expected if there are flows who are more aggressive than TCP but are responsive. Given these deficiencies with RED there have been several other AQM proposals which attempt to solve some of these problems [48, 47, 7, 24, 76, 25]. The notable mentions amongst these schemes are Adaptive Virtual Queue (AVQ) [47] and Random Exponential Marking (REM) [7]. AVQ uses a virtual queue to enque packets in the network and the size of the virtual queue depends on the arrival rate of the traffic and the utilization desired at the bottleneck. If the virtual queue is full then the packet is not enqued. REM on the other hand tries to match the input rate to the bottleneck capacity. Though AVQ and REM have fewer parameters to configure and give crisper guidelines for deriving those parameters these schemes still do not protect flows. This is because these proposals do not differentiate between flows. However there are some schemes which take into account flow arrival rate to allocate marks (or losses) and thus protect flows [48, 59, 26, 52]. We will discuss these schemes in detail in the following section.

2.3 Fairness

Though there are many definitions of fairness but its meaning in all definitions is that it represents the distribution of rates between users. Fairness is one of the most important considerations before network providers. This is because it represents how a network distributes rates between users such that the network does not penalize any user and more importantly can also use it to provide service differentiation.

Kelly et al. in [44] showed that every rate control scheme is associated with a particular kind of fairness. Specifically they show that if all users use same congestion control scheme then the subsequent rate distribution in the network is associated with a unique fairness criteria. In Kelly's framework every rate control algorithm is associated with a *Utility function*, U(x), which is a function of its rate allocation, x on the network. The end user's objective is to maximize its utility function with respect to rate. Kelly, Mo and Walrand showed in their work that the equilibrium distribution of rates or fairness with a utility function, U(x) is given as

$$\sum_{i} p_i U_i'(x_i - x_i^*)$$
where x_i is the rate allocated to the user *i* and x_i^* is the fair rate for the user *i* [44, 57]. The most interesting fact about this formulation is that it can used to represent any fairness criteria. For example *proportional fairness*, where the rates are allocated in proportion to the network resources being used in given by

$$\sum_{i} \frac{x_i - x_i^*}{x_i}$$

Recent works in flow optimization have used this definition to relate different rate control schemes to corresponding fairness criteria. In [49, 46, 12] the authors show that TCP Reno is associated with *minimum potential delay* fairness, i.e. it tries to minimize the total delay in a file transfer. Similarly, TCP Vegas achieves proportional fairness [49]. Mo and Walrand have also proposed a range of congestion control algorithm which can achieve weighted proportional fairness [57].

Though a network can allocate bandwidth according to a range of fairness criteria, traditionally equal allocations and Max-Min fairness have formed network's criteria for providing fair service to all users. However both these criteria are significantly different from the inherent fairness provided by TCP, i.e. minimum potential delay fairness. Therefore any fairness objective in the network, other than minimum potential delay fairness, might penalize some TCP flows. As such, any fairness objective in the network also has to take into account that it does not penalize TCP flows in the network, especially when most of the network traffic is carried by TCP.

The fairness issue also assumes importance because of proliferation of different rate control schemes in the network. This is because there could exists schemes which do not react to congestion indication or their response is different from TCP's response. As such, in [30, 53, 13] the authors argue that these schemes pose twin problems of being unfair to TCP's flows and importantly congestion collapse.

Different network based mechanisms have been proposed to manage these selfish schemes to prevent congestion collapse and to provide fair service to TCP flows. These schemes can be broadly classified into two categories, oblivious and nonoblivious schemes. Oblivious schemes allocate equal marks (or loss rate) to all flows and therefore do not protect TCP flows from other selfish and non-responsive flows. RED, REM, AVQ, CHOKe [47, 7, 25, 24] are some examples of such schemes. However CHOKe [64] is one notable exception amongst these oblivious schemes as it tries to punish aggressive flows. Non-Oblivious schemes on the other hand differentiate between flows mostly on their arrival rate and thus attempt to protect flows. Flow RED (FRED) [48], Stabilized RED (SRED) [59], Stochastic Fair BLUE (SFB) [26] , RED with Proportional Dropping (RED-PD) [52] are examples of AQM schemes which look at some flow characteristic before deciding to enque them. In this section we discuss some of these proposals.

2.3.1 Oblivious Schemes for Providing Fairness in the Network

End based techniques are insufficient to protect flows in the network and thereby provide fairness. Towards achieving these objective use of AQM at routers was proposed. RED [34] was the first significant AQM proposal. However as discussed in Section 2.2 RED cannot protect flows, especially when TCP flows in the cases where they compete with unresponsive flows [48]. Moreover since RED's control parameter are statically configured, i.e. the configuration does not change with time, RED's penalty function can be severe under low loads and insufficient with large multiplexing of flows [25, 24]. This further constrains the fairness objectives which RED can achieve.

Taking into the account these configuration issues Feng et al. proposed an AQM scheme, Adaptive RED (or ARED) [17]. ARED presents an on-line algorithm for dynamically changing the values of \max_p or the maximum dropping probability, according to the observed traffic. Therefore depending on whether the queue has been full or empty the maximum dropping value is increased or decreased. BLUE is another fundamentally different AQM algorithm which uses packet loss and link idle events to manage congestion [24]. BLUE maintains a single probability, which it uses to mark (or drop) packets when they are required. If the queue is continually dropping packets due to buffer overflow, BLUE increments the marking probability, thus increasing the rate at which it sends at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability.

Though both ARED and BLUE offer dynamic configuration of RED, they still do not protect TCP flows from misbehaving users. This is because these flows consider aggregate arrival rate to the bottleneck and thus do not differentiate between flows. Also these schemes allocate equal marks to all flows and thus misbehaving blows still corner a large share of the bottleneck. This is one of the drawbacks of the oblivious schemes.

However there is one significantly different oblivious AQM proposal which does protect flows from misbehaving users. This proposal called, CHOKe [64] takes into account the number of packets queued for a flow before deciding to enque them. When a packet arrives at the bottleneck, CHOKe randomly picks a packets already enqueued in the buffer and compares the flow identifier for both these packets. If a match is found then both the packets are dropped otherwise the incoming packet in enqued. This rule of deciding to enque a packet thus punishes aggressive flows as they are more likely to have more packets enqued and thus more probable to be dropped. The authors show that CHOKe tries to achieve Max-Min fair distribution across the network [64].

In summary barring CHOKe all oblivious schemes cannot protect TCP from misbehaving flows. However, all oblivious schemes are limited by the range of fairness criteria they can provide.

2.3.2 Non-Oblivious Schemes for Providing Fairness in the Network

From discussion in the previous section it is clear that in order to protect flows from misbehaving users we will need to assign marks not only on the basis of aggregate arrival rate to the bottleneck queue but also on individual flow arrival rates. Thus if we are monitoring individual flow rates to assign marks (or drops), the subsequent schemes are called *non-oblivious* schemes. Different ways have been suggested for monitoring individual flow's share in the bottleneck. One of these is explicit rate monitoring at every bottleneck, another method involves monitoring at one bottleneck and then sending the rate information through some means (either in packet header or through specific control packets) or deciphering the rate through number of packets enqueued in the bottleneck queue. In this section we will discuss some non-oblivious network based schemes which use one of these methods for providing fairness in the network.

The first significant non-oblivious AQM proposal was Flow RED (or FRED) [48]. FRED provides better protection than RED for responsive flows by isolating non-responsive greedy flows more effectively. Instead of indicating congestion to randomly chosen connections by dropping packets proportionally, FRED generates selective feedback to a filtered set of connections which have large number of packets queued. To achieve this, FRED estimates average per-flow buffer count, *avgcq*; flows with fewer than avgcq packets buffered are favored. Also FRED maintains a count, *strike* of number of times a flow has failed to respond to congestion notification. Any flow which has a higher strike value is more likely to be dropped. However one of the main drawbacks of FRED is that it has to maintain per-flow states, i.e. states for responsive as well as non-responsive flows, and might also increase average transfer delays.

A differential dropping scheme to manage fair bandwidth allocation at the router in presence of malicious users is presented in [63]. The scheme presented is similar to FRED in the sense that it maintains information about flows to decide which packet to drop. The authors propose the use of a shadow buffer where the count of packets is stored. A packet is dropped (or marked) if the packet count for the flow in the shadow buffer exceeds its fair rate (in terms of packets).

Probabilistic Aggregate Marking (or PAM) uses RED type thresholding on the token bucket contents to mark a packet (from a traffic aggregate) [20]. If the token bucket contents fall below min_{th} packets are marked with a lower priority. If the token bucket contents are in between min_{th} and max_{th} then packets are marked according to a linear function. The authors argue that PAM offer proportional marking though this argument is not backed by any analysis.

The authors in [20] also propose a scheme similar to CSFQ (Core Stateless Fair Queuing) and call it Stateless Aggregate Fair Marker (or SAFM). The edge marks packets based on the information present in the header. In SAFM the CSFQ header contents are replaced by token bucket size, token bucket rate and (1 - token allocation probability). The ingress calculates these values while the egress uses them to construct the fair rate share vector and then using it to mark packets. The model involves per-flow calculations at the ingress router.

In [51] the authors present a model to control high bandwidth aggregates in the network which uses packet history to drop/mark aggressive flows. Also once these misbehaving sources have been identified and punished this information can be pushed back to the downstream routers which can further rate limit these flows.

Another per-flow differential dropping called RED-PD (RED- Preferential Dropping) is proposed in [52]. Therein a malicious user is detected using drop history, which implies maintaining state. However the authors claim that this state is minimal because they store information only about the aggressive flows. Nevertheless the scheme requires storing states at all the routers in the network. The authors propose the use of congestion epochs to detect aggressive users. The idea is that a flow ideally sees one drop in one congestion epoch and this information can be leveraged to detect malicious flows. A per-flow filter is applied to these malicious flows which either drops a packet probabilistically or admits the packet. The probabilistic dropping is based on ARED.

In [36] the authors present a strategy to detect unresponsive flows using Diff-Serv. They propose shaping the unresponsive flows at the edges using congestion information from the core. As such the core is required to maintain information about every dropped packet and sends this information periodically to edge routers. The scheme proposed by the authors cannot be deployed in the Internet as it requires modifying the core and also requires to maintain state inside the core which raises questions of scalability.

Yet another network based scheme to achieve fairness in the network is Core Stateless Fair Queuing or CSFQ [79, 78]. In CSFQ a flow's arrival rate is monitored at the network edges and stored in the packet header for use by other routers. Thereafter each router updates this flow arrival rate and uses it to enque the packet probabilistically. The authors contend that CSFQ achieves fair queueing in the network.

Besides these AQM based proposals another popular way of providing fairness has been the use of schedulers in the network [21, 10, 74]. Fair queueing, Weighted FQ, Deficit Round Robin (DRR) are some scheduling proposals which can be used to provide fair distribution of rates in the network. However, these schemes are constrained by the need of placement throughout the network and moreover need coordination between each scheduler.

2.4 Compliance in the Network

Over the years the Internet growth has been well supplemented by various applications who have varying needs, especially for transport protocols. Initially all these applications relied on TCP but as the requirements of the applications changed TCP was no longer the only favored transport protocols and therefore a variety of flow control algorithms were proposed. These flow control algorithms can be mainly divided into three main classes a) TCP-Compatible flows b) unresponsive flows, i.e., flows that do not slow down when congestion occurs, and (c) flows that are responsive but are not TCP-compatible [28]. Yet another class of flow control algorithm use a mix of responsiveness and unresponsiveness. Specifically these algorithms decrease their rate on receiving a congestion indication but they also have a lower limit on transmission rate, i.e. they do not react to congestion indications when the sending rate is below this limit.

As a result of this proliferation of different congestion control algorithms we may reach a stage where there is no congestion avoidance mechanisms in the network. This would bring us back to the congestion collapse problem of 1980s [28]. These different class of flow control algorithm either responsive or unresponsive also pose a problem of protocol compliance. Floyd et. al formally defined this problem of protocol conformance in [30] wherein a conformant flow was called TCP-Friendly or TCP-Compatible.

RFC 2309 defines TCP-compatibility as, "A TCP-compatible flow is responsive to congestion notification, and in steady-state uses no more bandwidth than a conformant TCP running under comparable conditions (drop rate, RTT, MTU, etc.) " [13]. Floyd et al. in [30] proposed mechanisms for protocol conformance. These mechanisms can be broadly classified into two categories a) End-System and b) Network Based mechanisms. They proposed guidelines for developing end-system based flow control protocols which have the same reaction to loss as TCP. Since then there have been many proposals for TCP- Friendly flow control algorithms [9, 82, 42, 29, 70]. However, as we have previously seen there is a limit to how much control can be exerted from just the use of end-system. As such Floyd et al. also proposed network based mechanisms in form of schedulers and pricing mechanisms to ensure TCP-Friendliness on the network. We will now discuss these proposals briefly.

2.4.1 End-System Based Schemes for Compliance in the Network

Though conformance and fairness to TCP is significant it however should not constrain the choices of end-to-end congestion control algorithms. In [9] the authors propose a class of non-linear TCP compatible congestion control schemes called Binomial Congestion Control Schemes (BCCS). AIMD, can be considered as one of congestion control schemes in the subset of TCP Compatible BCCS. Formally, the Binomial Congestion Control scheme can be defined as:

$$W_{t+R} \leftarrow W_t + \alpha / W_t^k \quad if \ no \ loss \tag{2.1}$$

$$W_{t+\delta t} \leftarrow W_t - \beta W_t^l \quad if \ loss \tag{2.2}$$

where k and l are window scaling factors for increase and decrease respectively and α and β are increase the decrease proportionality constants. For any given values of α and β TCP Compatible BCCS can be defined by $k+l = 1 : k \ge 0, l \ge 0$.

Another interesting set of TCP Compatible congestion control algorithms has been presented in [70]. The proposal called Choose Your Response Function (or CYRF) has a general increase function f and a decrease function g which together constitute the congestion control policy. Formally the TCP-Compatibility is defined by the following constraints on the these two function f, g as:

$$f(x)g(x) \propto x$$

There have also been other interesting proposals for TCP Compatible window based protocols [42, 82] but covering all of them is beyond the scope of the thesis. Besides these window based proposals there have been suggestions for TCP compatible rate control scheme. The most popular rate based scheme is called TCP-Friendly Rate Control (or TFRC) [29]. Since we have already discussed TFRC in Section 2.1.3 we do not elaborate on it any further.

Finally this section can be summarized as, "The concern expressed in [RFC2357] about fairness with TCP places a significant though not crippling constraint on the range of viable end-to-end congestion control mechanisms for best-effort traffic." [28].

2.4.2 Network Based Schemes for Compliance in the Network

Though there exists a range of end-system based TCP Compatible congestion control scheme they might still not meet the needs of various applications. Moreover there exists a possibility that end users may intentionally not use these algorithms. Therefore network based solutions are needed to enforce protocol compliance.

The network based support has been envisioned in two primary forms: a) schedulers and b) pricing mechanisms [30, 28]. Per-flow scheduling in form of Class Based Queueing, Priority Scheduling or Weighted Round Robin etc can be used to isolate flows, restrict bandwidth of misbehaving flows and thus provide TCP Compatibility. Similarly pricing mechanisms can also be used for differentiating against misbehaving flows by communicating them higher price and thus ensuring TCP Compatibility in the network. However in order to achieve TCP Compatibility for the current Internet environment where flows compete in a FIFO queue all these mechanisms require tight coordination between all routers.

2.5 Optimization: Flow Control, Fairness and TCP Compatibility

Recently congestion control schemes have been evaluated and proposed using optimization frameworks [44, 46, 50, 57]. In these papers, the resource allocation problem is proposed as 1) individual users maximizing their utility functions and 2) network maximizing every user's utility function given the network capacity constraints. In these optimization models a user s, is described with the help of it's rate, x_s , a utility function U_s and the set of links which he uses, L(s). It is further assumed that the rates bounded i.e., $m_s \leq x_s \leq M_s$. It is assumed that the utility functions are *increasing* with rates and *strictly concave*. The network is identified with links l of capacity C_l . The set of users using a link, l, is given by S(l).

The optimization problem can then be defined as user's trying to maximize their individual utility functions and the network trying to maximize the resource allocation subject to link capacity constraints. Thus the primal problem can be defined as:

$$maximize \sum_{s \in S} U_s(x_s) \tag{2.3}$$

subject to
$$\sum_{s \in S(l)} x_s \leq C_l, \quad \forall l$$
 (2.4)

for all $x_s \ge 0$. The dual formulation, D(p), for the above problem was defined by Low in [50] as:

$$D(p) = \min_{p \ge 0} \sum_{s \in S} (U_s(x_s) - \sum_l p_l x_s) + \sum_l p_l C_l$$
(2.5)

The authors in [50] show that using the Karush Kuhn Tucker (KKT) conditions and gradient projection algorithm the dual yields the following update algorithm

$$x_s(t) = U_s^{\prime-1}(\sum_l p_l)$$
(2.6)

$$p_l(t+1) = [p_l(t) + \gamma(\sum_{s \in S(l)} x_s - C_l)]^+$$
 (2.7)

Since the primal is strictly concave and the constraints are linear, there is no duality gap and hence dual optimal is also primal optimal. Further the strict concavity entails an unique global optimum, (x_s^*, p^{s*}) where $p^s = \sum_l p_l$. Also though the primal optimal, x_s^* is unique, we may not have a unique dual optimal p_l^* but instead we have a unique optimum end-to-end loss probability for every source, p^{s*} .

In [44] the authors analyze the stability and fairness of network under primal and dual formulation. The dual formulation has also been discussed in [50] where gradient projection method is used to solve the problem. A penalty function approach to solving the network problem has been suggested by the authors in [46]. Also, the current TCP implementations have been mapped to optimized rate control algorithm in [49] [46].

The user's rate control algorithm can be thought to be tightly coupled with a utility function. Also, if all the utility functions are of similar type then we can also associate a fairness criteria with it. Fairness is defined as the way the resources are distributed amongst competing users, eg. max-min fairness, where the goal is to maximize the minimum share. The fairness criteria for a set of users S, can be described in terms of utility function as follows:

$$\sum_{S} U'(x_i - x_i^*) < 0 \tag{2.8}$$

where x_i is the rate allocated to the user *i* and x_i^* is the fair rate for the user *i*. The max-min fair vector corresponds to $U(x) = \lim_{\alpha \to \infty} \frac{-1}{x^{\alpha}}$. If the rate allocations are in "proportion" to the resources used by a user, then such a rate is said to be proportional fair and is defined by $U(x) = \log(x)$.

Thus the equilibrium rate allocation is very closely tied with the utility function the user chooses to maximize. This association of equilibrium rate allocation with the utility function might prompt sources to choose a utility function (and hence an aggressive congestion control scheme) which yields them higher rate allocations than other competing sources. Such a choice of utility function will still optimize the network and keep it stable, though at the cost of unfair allocations amongst users.

Finally, in [5] the authors evaluate the existence and properties of Nash Equilibria for selfish TCP user. They define selfishness by allowing the user to choose (and modify) it's own increase and decrease parameters, α and β respectively. They pose the problem as game where all users try to maximize their goodputs and evaluate the Equilibria for TCP Reno, Tahoe and Sack with both Drop-Tail and RED queues. They show that efficient Nash Equilibria exists only for TCP-Reno and Drop-Tail queues and the equilibrium can be defined by either $\alpha = 1$ or any β ; 0. However the equilibrium for TCP Sack and Tahoe is defined by an arbitrarily large value of α and $\beta \to 1$. Also the authors show that when these TCP flavors are evaluated with RED gateways the Nash equilibrium is inefficient. A similar result is also reported in [22]. The authors evaluate the Nash Equilibria for stateless AQM schemes and show that RED and Drop Tail do not impose Nash Equilibria on selfish users.

CHAPTER 3

Randomized TCP: End System Based Mechanism for Improving Fairness in a Network of Drop Tail Queues¹

3.1 Introduction

As discussed in Chapter 2, Drop Tail queues substantially limit the effectiveness of end-to-end congestion control protocols. This is primarily due to failure to provide early congestion notification to the end users. To avoid this and for better queue management use of AQM has been suggested. However due to configuration problems these AQMs have not found their way to the Internet, which to this day operates with Drop Tail queues. As such the problems of congestion window and loss event synchronization, phase effects and bias against bursty and long RTT flows persist. In this chapter we look at a comprehensive solution to all these issues by randomizing the packet transmission times in TCP flows.

The rest of the chapter is organized as follows.

- We present an end system based scheme to introduce randomization in the network and thereby emulate AQM. The proposal is called Randomized TCP and is discussed in Section 3.2 and the algorithm is detailed in Section 3.3.
- In Section 3.4 we do a characterization of the increase parameter to enable Randomized TCP to compete fairly with TCP Reno. A queueing analysis is presented in Section 3.5 to show that the probability of burst losses decreases with Randomized TCP.
- We present the implementation of Randomized, simulation setup and define the performance metrics in Section 3.6.
- Paramter tuning Randomized TCP is presented in Section 3.8.
- In Section 3.9 we present the simulation results for comparative performance of TCP Reno, Paced TCP and Randomized TCP while Section 3.10 evaluates

¹This work was done jointly with Prof. Biplab Sikdar

the bias against longer RTT flows, phase effects, synchronization and burst losses for Randomized TCP and TCP Reno.

- We extend the randomization of sending times to other window based schemes, specifically Binomial schemes and present its results in Section 3.11.
- Finally we present the conclusions in Section 3.12.

3.2 Randomized TCP

From the discussion in the Chapter 2 it is clear that introducing randomization into the network can break synchronization. Also by introducing the randomization, we avoid burst losses, thereby making the loss events "distributed". This then helps in solving the problem of phase effects. Though AQMs can introduce randomization in networks to some extent, it is not widely deployed due to variety of reasons [54, 55]. As such, we propose an end-system based mechanism for emulating AQM behavior. Specifically we propose a modification to TCP, called *Randomized TCP*, as a mechanism for introducing randomization into the network by randomizing the packet sending times. This solution is distributed, can be implemented at the end systems and therefore is very attractive from an implementation perspective.

Randomized TCP is similar to Paced TCP in that it "paces" packet transmissions but instead of spacing the transmissions equally, it adds or subtracts a random interval to the packet sending times at TCP sources. Packet transmissions are scheduled at intervals of $\frac{RTT}{cwnd}(1+x)$, where x follows the Uniform Distribution on [-I, I]. Evidently, I has to be between 0 and 1. A packet is transmitted at the expiry of the timer, if the window allows a packet to be sent. If not, upon reception of an ack, we schedule the packet transmission with a random delay of $\frac{RTT}{cwnd}y$, where y is U(0,I). Setting I to 0 reduces Randomized TCP to Paced TCP. The Randomized TCP's sending time algorithm is stated in Section 3.3.

In Section 3.8, we investigate the optimal setting of the randomization interval and find that a Uniform distribution on [-1,1] is the best. This choice of Uniform distribution can be intuitively justified as; a) since the distribution is centered around θ , on an average there is "no randomization" and Randomized TCP behaves as Paced TCP, b) and a minimum value of -1 of randomization implies TCP Reno implementation. This implies that sometimes we send back-to-back packets and sometimes we send paced packets. Thus with a randomization interval value of 1, Randomized TCP keeps moving forth between TCP Reno and TCP Paced. Intuitively, this entails an early detection of congestion (when the TCP behaves as Reno) and an even distribution of losses and throughput (when TCP behaves as Paced). Thus Randomized TCP takes the best of both Reno and Paced TCP and ensures lesser drops (because of early congestion detection) and fairer throughput.

In Paced TCP packets from each source reach the bottleneck at an uniform rate which can lead to near perfect interleaving. Such situations can cause all sources to lose packets thereby resulting in all the sources decreasing their windows together, resulting in synchronization. But with randomization, the rate is not uniform at the bottleneck and packets from flows are dropped after differing times due to the extra delay incurred due to randomization. This means that sources decrease their windows at different times and hence the periods of increase and decrease are not as synchronized as in Paced TCP. So the congestion epochs for different flows get out of sync and the network utilization is higher. Another nice property that comes because of randomization is that the source which has lost packets once is less likely to lose again (this may not be the case with deterministic TCP for some parameter settings [32]), thereby ensuring that over a larger time scale the rate distribution is fair.

Randomizing the sending times also results in extra delays causing the RTT to increase artificially. This causes Randomized TCP to get beaten down when competing with TCP Reno. It is well known that TCP's throughput is directly proportional to the square root of the window increase parameter and inversely proportional to RTT [61]. To allow Randomized TCP to compete fairly with TCP Reno, we analytically characterize the increased RTT (in Section 3.4) and make the increase factor in TCP proportional to the square of the ratio of the changed RTT to the real RTT.

We also note that the probability of two packets coming nearly back to back is significant only when the window size is large. This means that the probability of multiple packet drops will be verylow if the window size is small, thereby reducing timeouts. Using a simple M/M/1/K queueing analysis, similar to that in [55], in Section 3.5 we try to get a quantitative feel of the probability of a packet getting dropped with Randomized TCP.

The increased randomization increases the entropy of the system which correspondingly reduces the queue sizes thereby improving the stability of the system [66]. Our results show that Randomized TCP reduces phase effects and synchronization even when multiplexed with TCP Reno flows. Also it substantially reduces burst losses and removes the bias against longer RTT flows. In addition, *the benefits of randomization can be reaped even when it is partially deployed*. Randomized TCP performs better than or as well as Paced TCP and TCP Reno, independent of the capacity and buffer size at the bottleneck and for both short and long flows. The performance improvements can be seen in throughput, fairness, loss rates, timeouts and latency of the flows. We also investigate the impact of randomization on a class of slowly varying congestion control schemes called Binomial schemes [9] and show that by incorporating randomization in these schemes, the fairness increases dramatically when competing with TCP flows in drop tail queues.

In other words our scheme can emulate the beneficial effects of RED in a distributed manner without the complexities and unfavorable aspects of parameter tuning of RED. However, we wish to emphasize that unlike RED which is a congestion avoidance scheme, Randomized TCP is just a congestion control scheme. Thus Randomized TCP does not emulate the congestion avoidance features of RED, at best it provides the other beneficial features of RED which were achieved by introducing randomization in the network (by dropping packets probabilistically).

3.3 Randomized TCP Pseudo-code

Define by α the original increase parameter for the TCP Reno and by R the RTT. Then the Randomized TCP's algorithm can be stated as

• Send a packet. Schedule the next packet to be sent at time $t = \frac{RTT}{cwnd}(1+x)$ where x is Uniformly distributed on [-1,1].

- Let t' be the arrival time of next ack. Then
 - If t' < t send the next packet at t.
 - Else send the next packet after $\frac{RTT}{cwnd}y$ where y is uniformly distributed on [0,1].
- At each RTT (estimate) update recalculate the new increase parameter as $\alpha_{new} = \alpha \left(\frac{RTT_{new}}{R}\right)^2$

3.4 Analytical Characterization of Increase Parameter for Randomized TCP

In this section we outline the methodology for setting the increase parameter, α for Randomized TCP so as to make it compete fairly with TCP Reno. This is required because randomizing the sending times results in extra delay and hence slows down the window growth. As such it is likely that Randomized TCP will lose to TCP Reno when competing on a single bottleneck.

Consider a Randomized TCP connection with a constant window size of w. Let the real RTT for the connection be a constant denoted by R. Each packet is sent after a time equal to R(1 + x)/w where x is a Uniform random variable between [-I, I] (The optimal value of this interval is shown to be 1 in section 3.8, but presently we treat it more generally). Let the first packet be sent at time t = 0. Then the timer for the w + 1th packet of the connection will be scheduled at time, say t_1 , such that

$$t_1 = R(1 + \frac{1}{w} \sum_{i=1}^{w} x_i)$$
(3.1)

where x_i is the random value for the i^{th} packet in the window. The x_i s are independent and identically distributed. The effective RTT of the flow is the given by the time when $(w + 1)^{th}$ packet is sent. In the absence of random variations in real RTT, the ACK for the first packet comes exactly after time R. If $\sum_{i=1}^{w} x_i \ge 0$ then $t_1 > R$ and we will send the $(w + 1)^{th}$ packet at time t_1 . Else, the $(w + 1)^{th}$ packet will be sent after a random time $\frac{RTT}{w}y$ after the ACK arrival, where y is drawn from an uniform distribution on [0,I].

Thus the effective RTT can be expressed as

$$RTT_{eff} = \begin{cases} R(1 + \frac{1}{w} \sum_{i=1}^{w} x_i) & \text{w.p. } P\{\sum_{i=1}^{w} x_i \ge 0\} \\ R(1 + \frac{y}{w}) & \text{w.p. } P\{\sum_{i=1}^{w} x_i \le 0\} \end{cases}$$
(3.2)

where w. p. is short for "with probability". Then, the mean effective RTT, \overline{RTT}_{eff} , can be expressed as

$$\overline{RTT}_{eff} = \{R(1 + \frac{1}{w}E[\sum_{i=1}^{w} x_i \mid (\sum_{i=1}^{w} x_i \ge 0)])\}$$
$$P\{\sum_{i=1}^{w} x_i \ge 0\} + \{R(1 + \frac{\bar{y}}{w})\} P\{\sum_{i=1}^{w} x_i \le 0\}$$
(3.3)

where \bar{y} is the mean of y equal to I/2. Since x_i follows an Uniform distribution around zero, its easy to see that

$$P\{\sum_{i=1}^{w} x_i \ge 0\} = P\{\sum_{i=1}^{w} x_i \le 0\} = 0.5.$$
(3.4)

Assuming that the window size is sufficiently large to invoke the the Central Limit Theorem we get

$$\sum_{i=1}^{w} x_i \sim N(0, \sigma^2), \quad \sigma^2 = w * \frac{I^2}{3}$$
(3.5)

The pdf of $\sum_{i=1}^{w} x_i$ conditioned on $\sum_{i=1}^{w} x_i \ge 0$ can be found out to be twice that of the Gaussian pdf multiplied by the Unit step function. From this we can derive the conditional mean as

$$E[\sum_{i=1}^{w} x_i \mid (\sum_{i=1}^{w} x_i \ge 0)] = \sqrt{\frac{2wI^2}{3\pi}}$$
(3.6)

Plugging these back into the equation for \overline{RTT}_{eff} , we get

$$\overline{RTT}_{eff} = R + \frac{1}{2w} \left(\sqrt{\frac{2wI^2}{3\pi}} + \frac{I}{2}\right) \tag{3.7}$$

For Randomized TCP with increase parameter α and effective mean RTT, \overline{RTT}_{eff} , the throughput is proportional to $\frac{\sqrt{\alpha}}{RTT}_{eff}$. To make the throughput same as that of TCP Reno (with $\alpha = 1$ and RTT = R), we set $\alpha = \frac{\overline{RTT}_{eff}^2}{R^2}$ for randomized TCP. In the real implementation, since window value changes with time, \overline{RTT}_{eff} changes with time and so we change the value of α also with time.

3.5 Queueing Analysis to Show Reduction in Burst Losses with Randomized TCP

Consider a M/M/1/K queueing system where the packets arrive according to a batch Poisson process; specifically, bursts (or batches) of B packets arrive according to a Poisson process of rate λ . Further, let us denote by $\pi(k)$ as the stationary distribution of k number of packets in the queue. Then using the PASTA (Poisson Arrival See Time Averages) property the probability of a packet drop in a Tail Drop router with TCP as input can be calculated as [55]:

$$P_{TD} = \pi(K) + \pi(K-1)\frac{B-1}{B} + \dots + \pi(K-B+1)\frac{1}{B}$$

Using the same model we will now calculate the probability of a packet being dropped for Randomized TCP. We first note that the size of burst, B, will now be changed because Randomized TCP paces the packets. Hence we first try to find the new burst size (given that the original burst size was B) and then calculate the packet drop probability. Figure 3.1 shows the epochs at which the packets are sent. Let us call the time instants at which the packets from a Paced TCP would have been sent as *centered epoch*. These *centered epochs* now represent the time instants around which we randomize the sending times of packets in Randomized TCP. Suppose a packet is sent at some time, x after the centered epoch (as shown in figure 3.1). Let us also define the length of the packet as L bits and the bottleneck link capacity as C bits/sec. Further, let the window size at steady state be W (B \leq W) and let RTT denote the round-trip time. Then the probability, p, of packets from a burst of B, arriving back-to-back at the bottleneck router can be calculated as

=

$$p = \int_{0}^{\frac{RTT}{W}} \left(\frac{1}{2}\frac{x}{\frac{RTT}{W}}\right) \left(\frac{1}{2}\frac{\frac{L}{C}}{\frac{RTT}{W}}\right) dx \qquad (3.8)$$

$$= \frac{L}{8C} \tag{3.9}$$

Note that the now, $B' = min(B * \frac{L}{8C}, B)$, represents the upper bound on the number



Time at which next packet must be sent for the packets to arrive back–to–back at the bottleneck queue which is less than or equal to L/C.

Figure 3.1: Packet Sent Times with Randomized TCP

of back-to-back packets which can be received at a bottleneck with Randomized TCP and a burst of size B. Also note that the above analysis holds true iff $\frac{L}{8C} \leq \frac{RTT}{W}$ which holds true for WANs and MANs.

Using the above equation, the probability that a packet gets dropped with Randomized TCP and drop tail router can be calculated as

$$P_{TDR} = \pi(K) + \pi(K-1)\frac{B'-1}{B'} + \dots + \pi(K-B'+1)\frac{1}{B'}.$$

Thus from the above observation we can conclude that the probability that a packet gets dropped with Randomized TCP and drop tail queue decreases. However, it should be noted that Poisson arrivals do not capture the exact packet arrivals in the Internet. Nevertheless, this exercise is just intended to show that the probability of burst losses are reduced with Randomized TCP and have been validated by our simulation results in Section 3.10.4.

3.6 Implementation and Simulation Setup

We have implemented Randomized TCP in the Network Simulator ns [1]. For our implementation, we used the congestion control and loss recovery mechanisms of TCP Reno and thus Randomized TCP has the usual slow-start and fast recovery and retransmit mechanisms. For the simulations reported in this chapter, we disabled



Figure 3.2: Topology used in the simulation.

the delayed acknowledgments option. Also, we used the modified window increase parameter for Randomized TCP implementation.

Figure 3.2 shows the topology used in the simulations. The access links were configured at a rate 4 times greater than that of the bottleneck link and all the links use Drop Tail queues. The maximum advertised window is set sufficiently high so that it does not constrain the actual window. We use a Maximum Segment Size of 500 bytes.

We evaluate the performance of randomized TCP for the following set of metrics: average throughput, fairness, loss rates, timeouts, latency and synchronization. We characterize fairness using the modified Jain's fairness index, [18, 4]. Jain's fairness index is defined as

$$f = \frac{(\sum_{i=1}^{n} x_i.RTT_i)^2}{n(\sum_{i=1}^{n} (x_i.RTT_i)^2)}$$
(3.10)

where x_i is the throughput of the i^{th} flow, RTT_i is the round-trip time of flow i and n is the number of flows.

To study the synchronization of flows we use the covariance between the congestion window of two competing flows. Flows would be synchronized if their windows increase and decrease simultaneously. In this case both flows' windows (say w_1 and w_2) would be above or below their mean values at any time t, i.e. $(w_1(t) - \bar{w}_1)(w_2(t) - \bar{w}_2) > 0$. So the cross-covariance coefficient of synchronized flows would be positive. In the case where the flows are totally out of sync, $(w_1(t) - \bar{w}_1)(w_2(t) - \bar{w}_2) < 0$, since when one flow has a large window, the other would have a smaller window and vice versa. So the cross-covariance coefficient of out of sync flows would be negative. This shows that the cross covariance coefficient of greater than 0 implies in-phase synchronization while less than 0 implies out-of phase synchronization. However, too large a negative value of cross-covariance denotes that synchronization effects still persist albeit in a negative sense. In [83] the authors also argue that out-of-phase synchronization is not good. Hence a value equal to or close to 0 for cross-covariance coefficient should be the optimal.

In the following sections we present the simulation results. We first observe the effect of bottleneck bandwidth, buffer sizes and RTTs on the randomization interval I in section 3.8. Using these simulations we propose a value of the interval for optimal performance.

Section 3.10 shows the performance of Randomized TCP with respect to phase effects, synchronization amongst flows and burst losses. In Section 3.9 we present the result of comparative performance of Randomized, Paced and Reno TCP for the following set of metrics: throughput, losses, timeouts, fairness and latency for both bulk-data transfer and short-web like transfers. Finally in Section 3.11 we present the results of extension of Randomization to Binomial schemes.

3.7 Implementation on the Linux Kernel

We have implemented the Randomized TCP in Linux [56]. The following components were required to implement Randomized TCP 1) a microsecond resolution timer for Linux, 2) a random number generator and 3) a packet scheduling methodology to schedule packets in future. We used UTIME [2] extension to the Linux kernel to introduce microsecond resolution. Scheduling of packets is done on the expiry of this microsecond timer. The Linux kernel provides a random number generator that returns a requested number of random bytes to the module invoked within the kernel. However, our requirement needs the random number to be genearated on the byte boundaries but rather or bit boundaries. We wrote functions to create such a random number and also to create both positive and negative random numbers. We tested the implementation with the simple dumb-bell topology.

3.8 Parameter Tuning

The randomization interval has a significant impact on the performance of Randomized TCP, and hence its characterization is of utmost importance. In this section we study the effect of change in bottleneck bandwidth, buffer size, number of flows and round-trip times on throughput, number of losses, timeouts as a function of the randomization interval. Through these simulations we obtain the optimal value of randomization interval. The default settings for this section are a bottleneck link of 1 Mbps, all the other links of bandwidth 4 Mbps, end-to-end propagation delay of 100ms and a Drop Tail queue of 25 packets at the bottleneck. Simulation settings are assumed to be default (as that mentioned in 3.6) unless specifically specified.

3.8.1 Different Bottleneck Bandwidth

Figures 3.3 (a), (b) and (c) plot the loss rates, throughput and timeouts respectively, for a setup of 50 flows as a function of randomization interval on a single bottleneck setup (figure 3.2). The bottleneck bandwidth was varied in this case from 3Mbps to 10Mbps while the buffer size was held constant at 25 packets. The end-to-end propagation delay was 100 ms. It can be seen that as the randomization interval increases to 1, the loss rates and the timeouts reduce, while the throughput increases or remains almost the same. Similar results were obtained with a larger buffer size. The impact of buffer size on the randomization interval is detailed in the following section.

3.8.2 Different Buffer Sizes

In order to evaluate the effect of buffer sizes, we vary the buffer size at the bottleneck from one-fourth of bandwidth delay product to one bandwidth delay product. The bottleneck link is of 4 Mbps and the end-to-end propagation delay is 100ms. Thus we vary the buffer size from one-fourth bandwidth delay product to one bandwidth delay product. Again, we plot the losses, throughput and timeouts for 30 flows as a function of randomization interval. Figure 3.4 show the effect of buffer size. From the Figure 3.4 it can be inferred that a randomization interval value of 1 gives us the best results vis-a-vis throughput, loss rate and the number



Figure 3.3: Loss Rate, Throughput and Timeouts for 50 flows as a function of randomization interval, for different values of bottleneck bandwidth.

of timeouts.

3.8.3 Different RTT

In this simulation setup every flow had a unique RTT in the range 80ms to 120ms. The RTT for the i^{th} , $i \in (0, ..., N-1)$ flow was 80+i*(120-80)/N where N is the total number of competing flows. In Figure 3.5 we plot the losses, throughput and timeouts for 30 and 50 flows as a function of randomization interval. From the



Figure 3.4: Loss Rate, Throughput and Timeouts for 30 flows as a function of randomization interval, for different values of bottleneck buffer size.

Figure 3.5 we can conclude that a randomization interval value of 1 suits almost all the simulation metrics.

From the above simulations it is evident that a higher value of randomization interval results in increased throughput and lower losses and timeouts. Randomization interval of 1 implies that inter-packet time intervals can lie anywhere between 0 and 2RTT/cwnd. This means that packets are randomized the most and this re-



Figure 3.5: Loss Rate, Throughput and Timeouts for 30, 50 flows as a function of randomization interval, for varying RTT. The RTT varies from 80ms-120ms, the bottleneck bandwidth is 4Mbps and the buffer size is 25 packets.

sults in increased scope for breaking the synchronization, thereby resulting in better performance.

This choice of randomization interval can be intuitively explained as following. With a randomized interval value of 1, randomized TCP keeps moving forth between TCP Reno and TCP Paced. (This is because, since randomization interval is Uniform on [-1,1] therefore when the randomized value is -1 then the packets are sent immediately after receiving an ACK akin to TCP Reno. Since the randomization interval is centered around 0, on an average Randomized TCP behaves as Paced TCP.) Intuitively, this entails an early detection of congestion (when the TCP behaves as Reno) and an even distribution of losses and throughput (when TCP behaves as Paced). Thus Randomized TCP takes the best of both Reno and Paced TCP and ensures lesser drops (because of early congestion detection) and fairer throughput.

3.9 Throughput, Loss, Timeouts, Fairness and Latency

In this section we compare the performance of Randomized TCP with TCP Reno and Paced TCP. We evaluate all these three schemes for both Bulk data transfers and small Web like transfers. Specifically, we compare the following metrics: average throughput, loss rate, timeouts for bulk data transfers and latency for small web like transfers. We also assess the interaction of Randomized TCP and TCP Reno on a single bottleneck for the metrics throughput, loss rate and timeouts.

3.9.1 Bulk Data Transfer

3.9.1.1 Same RTT

Figure 3.6 plots the throughput, loss rate, number of timeouts and fairness for Reno, Paced and Randomized TCP. Though Reno, Paced and Randomized TCP have the same throughput the losses are more for Paced. This is because in slow start, Pacing delays the congestion signal and hence looses a larger number of packets. As the number of flows increase Randomized TCP tends to do the best of the lot.

3.9.1.2 Different RTT

To study the performance of Randomized TCP with different RTT values for flows, we varied the RTT of each flow. The RTT of flows were in the range of 80ms to 120ms. The RTT for the i^{th} , $i \in (0, ..., N - 1)$ flow was 80 + i * (120 - 80)/Nwhere N is the total number of competing flows. Figure 3.7 shows the throughput, fairness, loss rates and timeouts as the number of flows are increased from 10 to 50. Randomized TCP is the most fair and also the throughput achieved is marginally



Figure 3.6: Loss Rate, Throughput, Timeouts and fairness with Bulk Data transfer, each flow having same RTT.

higher. However, it is interesting to note that Pacing also achieves almost the same performance as Randomized TCP. TCP Reno maintains its bias against flows with longer RTT (TCP throughput is inversely proportional to the RTT), which is shown by the fairness graph. Because of this bias, Reno's fairness curve is lowest. In [3], the authors contend that bias of TCP against longer RTT flows is considerably reduced with RED gateways due to uniform distribution of losses over time. The similarity of our simulation result to this indicates that randomization succeeds in distributing losses over time (to a certain extent), thereby decreasing TCP's bias



Figure 3.7: Throughput, Fairness, Loss Rates and Timeouts for a set of flows where each flow has a different RTT.

3.9.2 Short Web Like Transfers

In this section we present the performance of Randomized TCP for short flows. This is more representative of Web transfers. In this simulation we used a singlebottleneck link of 4Mbps with a round-trip time of 100ms (figure 3.2). The buffer was fixed at 25 packets product. 25 flows were always maintained in the network. As soon as any flow finishes, a new flow initiates transfers. We varied the workload from 10 packets to 2500 packets. Figure 3.8 (a and b) plots the latencies for Reno, Paced and Randomized TCP. For very short flows, i.e. for a workload of 10 packets to 200 packets, TCP Reno performs the best while Paced TCP performs the worst. Randomized TCP's performance though better than Paced TCP is not as good as Reno's. This can be attributed to the randomness which has been introduced in pacing intervals. Because of this randomization, Randomized TCP breaks ties and achieves better performance than Paced. Reno however, sends packets in bursts and is able to complete most of the transfers in the slow start. For workloads greater than 200 packets, Reno still performs the best, though the difference in the latencies for Reno and Randomized reduce as the workload increases. For Pacing, new flows starting in the slow start saturate the network. Due to late congestion signals in Pacing, many flows, even those who are in congestion avoidance, simultaneously drop packets thus severely diminishing Paced TCP's performance [4]. Reno performs better because Reno flows send packets in clusters, a burst from a particular flow in slow start has only local effect; it does not offect all flows [4].



Figure 3.8: Latencies for Reno, Paced and Randomized for short and moderate Web like Workloads

3.9.3 Interaction of Randomized TCP with TCP Reno

This section presents the result of multiplexing TCP Reno and Randomized TCP on the same link. In [4], the authors show that Paced TCP gets beaten

TCP Type	Throughput	Losses $(\%)$	Timeouts (%)
Reno	480.21	2.45	0.1
Paced	351.86	5.74	0.8
Reno	389.31	4.2	0.5
Random	408.92	5.1	0.8

Table 3.1: Comparison of Throughput (in pkts/sec), losses and timeouts for TCP Reno Vs (Paced, Random).

down by TCP Reno, when multiplexed on the same link. This is because a single paced connection is more likely to have at least one of its packets encounter severe congestion when multiplexed with a bursty connection [4]. This problem is the same as a source's packets getting synchronized with the buffer overflow event. Hence that flow faces a disproportionate number of losses and a lower throughput [32]. This effect is reproduced in our simulations as shown in Table 3.1 where the throughput is considerably lesser for Paced TCP (351.86 Kbps) as against TCP Reno (480.21 Kbps). The RTT for this experiment was 100ms, the bottleneck link's capacity was 1 Mbps and it was configured with Drop Tail with 25 packets of buffer.

However, when Randomized TCP is multiplexed with TCP Reno, the fairness improves considerably. This is seen in Table 3.1 where the throughput for the Randomized TCP is 408.92 Kbps when compared to 389.31 Kbps for TCP Reno. This is primarily due to two reasons. Firstly, by modifying the increase parameter α of Randomized TCP we account for the extra delay being introduced by randomization. Secondly, by reduction of synchronization of the source to buffer overflow events, we ensure equitable distribution of drops.

3.9.4 Summary

To summarize the observations of this section:

- For bulk data transfer Randomized TCP performs as well as or better than TCP Reno and Paced TCP in almost all scenarios.
- Specifically, for bulk data transfer with same RTT amongst different flows, with higher multiplexing (of flows) Randomized TCP performs the best by increasing the throughput and fairness, reducing losses and timeouts.

- For bulk data transfers where every flow has different RTT, Randomized TCP clearly out-performs TCP Reno and Paced TCP. This is important because this is more representative of the Internet.
- In the scenario where all flows have different RTT and a Drop Tail queue at the bottleneck, randomization reduces the TCP bias against longer RTT flows and achieves a performance similar to RED gateways as mentioned in [3].
- With short web like transfers, Reno performs better than Randomized TCP. However as the workloads start to increase Randomized TCP catches up with TCP Reno. Small workload flows complete their transaction in slow-start (or with very small windows). As such, if we randomize the windows when they are small, randomization generally delays the sending times which results in increased latency. Moreover, our re-characterization of increase parameter (Section 3.4) does not come into play because it works for congestion avoidance phase. As such, we conjecture that one should not randomize the sending times when the windows are small (less than 4) and during the slow-start. However, these inferences are at best intuitive and need to be evaluated in detail. One could also calculate the adjustment factor for slow start (just like we did for steady state).
- Randomized TCP and TCP Reno can compete fairly at a bottleneck. This
 is primarily because of the modification of the increase parameter, α, of the
 congestion window growth in Randomized TCP. However, Paced TCP loses
 out to TCP Reno as already shown in [4].

3.10 Bias Against Long Flows, Phase Effects, Synchronization and Burst Losses

3.10.1 Bias Against Long Flows

It has been widely reported that Drop Tail gateways have a bias against long flows [34]. In this section we first demonstrate this bias and its reduction with the use of Randomized TCP. We present the results with single as well as multiple bottleneck topologies.

3.10.1.1 Single Bottleneck

We performed simulations with two flows, one shorter RTT source(60 ms) and another longer RTT source (80 ms) and for differing link capacities to demonstrate the bias against long flows. We varied the bottleneck capacity but kept the buffer size constant at 25 packets with Drop Tail queues. The simulation time was 100 seconds.

If we assume that both flows see the same drop rate then the throughput for the two flows would be distributed as inversely proportional to the RTT (Throughput $\propto 1/\text{RTT}$ [61]. Thus here the throughput should be distributed as 8/14 (0.57) and 6/14 (0.43) of the bottleneck capacity, amongst the 60ms and 80ms sources respectively. Now consider the case when the bottleneck bandwidth is 2 Mbps and both the longer as well as the shorter flow use TCP Reno. The throughputs for the longer and the shorter flow in this case are 119.81 Kbps and 298.93 Kbps respectively. The share of the bottleneck for the two flows is 0.29 (long flow) and 0.71 (short flow) as against the theoretical values of 0.43 and 0.57 respectively. Therefore, we find that when both the sources use TCP Reno, bias against longer flow exist as expected. However, with the same 2 Mbps bottleneck, if we randomize one source (in this case, the shorter source), we find that bias against longer flow is considerably reduced as seen in Table 3.2. In fact the throughput for the 80ms and 60ms flows are 166.11 Kbps and 196.3 Kbps respectively. Also their share of the bottleneck are 0.46 (long flow) and 0.54 (short flow) as against the theoretical values of 0.43 and 0.57 respectively. This beneficial effect of Randomized TCP is preserved even if we randomize the longer flow.

A similar statement about the bias against longer flow can be made for the other case where the bottleneck is of 3 Mbps. There too when both the flows use TCP Reno the bottleneck is shared as 0.34 for the long flow and 0.64 for the short flow instead of 0.43 and 0.57 respectively. But when one of flows uses Randomized TCP while the other uses TCP Reno, the bottleneck is shared as 0.44 for the long flow and 0.56 for the shorter flow. These two examples elicits that the bias against longer flows are present with TCP Reno and are removed with Randomized TCP.

We investigated another simulation setup with a bottleneck of 1 Mbps, a Drop-

RTT	Type	Throughput (pkts/sec)	Losses	Timeouts
Long	Reno	119.81	5.7	176
Short	Reno	298.93	1.1	34
Long	Reno	166.11	1.9	52
Short	Random	196.3	1.8	43

Capacity: 2Mbps

Capacity: 3Mbps				
RTT	Type	Throughput (pkts/sec)	Losses	Timeouts
Long	Reno	208.05	2.7	64
Short	Reno	408.42	0.6	28
Long	Reno	241.64	1.2	43
Short	Random	300.08	1.05	29

Table 3.2: Bias Against Longer Flows: Distribution of Throughput in proportion of RTTs with Randomized TCP shows reduction of bias in contrast to TCP Reno.

RTT	5 Short Reno	5 Short Random	5 Short Reno	5 Long Reno
	5 Long Reno	5 Long Random	4 Long Reno +	4 Short Reno $+$
			1 Long Random	1 Short Random
Short	62.6	41.50	45.51	50.02
Long	33.35	33.60	35.80	34.71

Table 3.3: Comparison of Throughput (in Kbps) for different configuration of competing 5 Long flows (RTT=80ms) and 5 Short Flows (RTT=60ms)

Tail queue of 25 packets and 10 flows. In this experiment we had 5 sources each with RTTs of 60ms and 80ms. The results of this simulation are tabulated in Table 3.3. We first show the occurrence of bias against longer flows when all these sources used TCP Reno, and then we show the removal of this bias when all these sources used Randomized TCP. But more interestingly, we demonstrate a reduction in bias even when any one source uses Randomized TCP and the rest use TCP Reno. This implies that a presence of even a single Randomized TCP at a bottleneck might be helpful in reducing the bias against flows with larger RTT. Thus even an incremental deployment of Randomized TCPs would benefit the entire group of users.



Figure 3.9: Multi Bottleneck Topology used in the simulation.

3.10.1.2 Multiple Bottleneck

In this section we evaluate the performance of TCP Reno and Randomized TCP with a multiple bottleneck topology. The topology is shown in Figure 3.9 consists of two bottleneck links of capacity 1 Mbps and delay of 20ms. All the other links in the figure have a capacity of 4 Mbps and delays as shown in Figure 3.9. The long flows have end-to-end propagation delay of 120ms while the short flows have an end-to-end propagation delay of 60ms. Our simulation setup consist of 2 long flows denoted by (S1-D1) and (S2-D2) source-destination pairs and two small flows denoted by (S3-D3) and (S4-D4) source- destination pairs, as shown in figure 3.9. We investigate this topology when (S1,S2) and (S3,S4) use TCP Reno and Randomized TCP. Table 3.4 tabulates the results for different simulation setups.

We can see from the Table 3.4 that there exists bias against flow(s) with longer RTT when all the flows use TCP Reno (displayed by the considerable difference in their throughputs) and is subsequently removed when all the flows use Randomized TCP. However, an interesting observation again is that when the short flows use Randomized TCP while the long flows use TCP Reno, we see reduction in this bias. This further supports our argument that a presence of even a single randomized flow at every bottleneck is sufficient to reduce the bias against longer flow(s) and thus achieve a better fairness amongst flows. In another simulation setup where the long flows use TCP Reno, we see that the bias persists. This is intuitively true too. The long flows are the only sources of potential randomness at the bottleneck, which is visible at the first bottleneck. However, at the second bottleneck the streams arrive in phase because the randomness at the

Source	(S1,S2): Reno	(S1,S2): Random	(S1,S2): Reno	(S1:S2) Random
	(S3,S4): Reno	(S3,S4): Random	(S3,S4): Random	(S3,S4) Reno
S1	50.28	61.28	56.12	50.04
S2	44.08	55.64	59.24	55.00
S3	106.18	83.84	85.20	94.50
S4	108.16	84.36	86.00	98.92

Table 3.4: Comparison of Throughput (in Kbps) for Multi-Bottleneck Topology

first bottleneck is broken by the "departure process" of the queue. Thus at the second bottleneck there is no randomization to break the bias against longer flows. Hence the long flows get beaten down and the bias persists.

3.10.2 Phase Effects

In [32] the authors show that phase effects with drop-tail queues can cause a source's loss events to get synchronized with the full queues. Consequently it loses a large number of packets and gets a very low throughput. The authors also note that an appropriate randomization included in the delay would reduce the phase effects. In this section we show the presence of phase effects in Drop Tail Gateways with TCP Reno as first shown in [32]. Subsequently, using the same simulation setup we show reduction in phase-effects with the use of Randomized TCP. We use the same simulation setup as discusses by the authors in [32]. Since phase-effects can be shown by either dis-proportionately high number of losses or low throughput in this work we chose losses to demonstrate phase-effects. Each point in these losses-time plot corresponds to the average losses for the last 50 seconds of the simulation.

Figure 3.10 shows the setup for a single bottleneck topology for a 100 ms simulation, a bottleneck buffer of 15 packets and the packet size of 1000B. In this simulation we vary the RTT of source 1 by varying the delay between source 1 and bottleneck. In figure 3.11(a) we plot the losses of Source 2 against the ratio of RTTs of the two sources. As can be seen from the figure 3.11(a) that for most of the data points, source 2 sees almost no loss (source 1 sees all the losses) while for some particular values of the RTT ratios (between 1.85-2.05) it sees most of the losses showing the presence of phase effects. However, we see that the phase effects are



Figure 3.10: Single bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

removed if Randomized TCP is used and the source 2 never sees disproportionately higher percentage of network losses.

We also evaluated Randomized TCP's performance vis-a-vis phase effects for a multiple bottleneck topology as shown in figure 3.12. In this simulation we varied the RTT of source 1 by varying the delay between Source 1 and bottleneck 1. The packet size used for the simulation was 1000B, the buffer length at each bottleneck was 15 packets (slightly more than 1 bandwidth delay product) and the simulation time was 100 ms. In Figure 3.11(b) we plot the percentage losses (of the total losses at the second bottleneck) as seen by Source 3 against the RTT ratios of source 1 and 2. Again it can be seen that Source 3 sees almost 80% losses with TCP Reno while the losses are considerably reduced (to about 40%) when Randomized TCP is used. This further verifies the presence of phase effects in Reno and Drop Tail gateways and removal of phase effects with the use of Randomized TCP.

3.10.3 Synchronization

3.10.3.1 Synchronization in Bulk Data Transfer

We ran separate simulations with 2, 3, 10 and 25 flows of Reno, Paced and Randomized TCP and calculated pair-wise (between flows) covariance coefficients of congestion windows. We maintained the default simulation setup as described in Section 3.6 and the simulation time was 1000 seconds. The congestion window for each flow was sampled using a sample interval of 0.1 seconds, i.e., the congestion window was sampled approximately once every RTT. This sample set was then used to calculate the pairwise covariance coefficients.


(a) Single Bottleneck: Node 2 does not see disproportionate losses with Randomized TCP, Phase effects reduced.



(b) Multiple Bottleneck: Node 3 does not see disproportionate losses with Randomized TCP, Phase effects reduced.

Figure 3.11: Phase Effects

In our first simulation with 2 flows, we varied the bottleneck bandwidth from 3 Mbps to 5 Mbps while keeping the buffer fixed at 25 packets. Table 3.5 shows the covariance coefficients for each of the flows. It can be inferred that the synchronization in Reno increases as the bottleneck bandwidth increases. However Randomized TCP keeps the synchronization low while Paced TCP is out of phase synchronized. Also, it is interesting to note that while the synchronization increases in Reno with increase in bottleneck bandwidth, it decreases in Randomized.



Figure 3.12: Multiple bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

Bandwidth	Reno	Paced	Randomized
$3 { m Mbps}$	0.4254	-0.4124	0.1721
$4 \mathrm{~Mbps}$	0.3152	-0.1839	0.1604
$5 { m Mbps}$	0.6700	-0.3302	0.0799

Table 3.5: Comparison of Covariance Coefficient of Congestion Window for two flows for TCP Reno, Paced and Randomized. (Value around 0 is Good.)

In our second simulation with 3 flows, we kept the bottleneck bandwidth constant. Covariance coefficient values are tabulated in the table 3.6. Again, it is evident that Reno is the most synchronized and Paced TCP is out of phase synchronized. Also, it can be seen that both Paced and Randomized TCP lead to reduction in the synchronization.

Figures 3.13 and 3.14(a and b) plot the pairwise covariance coefficients for 10 and 25 flows. The y axis of the graph plots the covariance coefficient against the pair of flows on x axis, i.e., each unit of x axis corresponds to a pair of flows,

Flow Pair	Reno	Paced	Randomized
(1,2)	0.5183	-0.1454	0.2525
(1,3)	0.5416	-0.1537	0.1422
(1,4)	0.3492	-0.1833	0.1535

Table 3.6: Comparison of Covariance Coefficient of Congestion Windows for 3 flows for TCP Reno, Paced and Randomized. (Value around 0 is Good.)



Figure 3.13: 10 flows Covar. coeff. of Congestion Window for (a) Reno, Paced & Randomized (b) Reno & Randomized, (c) Paced & Randomized

starting in the order (1,2), (1,3), ..., (2,3) Since the graphs for 25 flows are not visible on one graph we plot it in two. Fig 3.14(a) plots the covariance for Reno and Randomized TCP and 3.14(b) plots it for Randomized TCP and Paced TCP. Both Paced TCP and Randomized TCP break synchronization while Reno is highly synchronized. Also, as the number of flows start increasing, Randomized TCP starts to get better than Paced TCP.

3.10.3.2 Synchronization with Short Web Transfers

In [4] the authors contend that one of the reasons for higher latency with Paced TCP in short web like transfers is that connections seem to get synchronized. In this simulation setup we have evaluated and verified their arguments. For the simulation we used a bottleneck link of 4Mbps, a RTT of 100 ms and a buffer of 25 packets. 25 flows were always maintained in the network. As soon as any flow finishes, a new flow initiates transfers. We varied the workload from 10 packets to 2500 packets.

Figure 3.15 plots the covariance coefficients of congestion windows for Paced and Randomized TCP. A closer look shows that the covariance for Randomized TCP is *consistently* lower than that for Paced TCP. In Paced TCP packets reach the



Figure 3.14: Covar. coeff. of Congestion Window for (a) Reno & Randomized, (b) Paced & Randomized

bottleneck at an uniform rate with near perfect interleaving. This causes all sources to lose packets, thereby resulting in all the sources cutting down their windows together, and hence higher covariance. But with randomization, the rate is not uniform at the bottleneck and packets from flows are dropped after differing times due to the extra delay incurred because of randomization. This means that sources decrease their windows at different times and hence the periods of increase and decrease are not as synchronized as in paced, resulting in a decreased covariance coefficient between the flows.

3.10.4 Burst Losses

In this section we investigate the proposition that Randomized TCP reduces the burst losses and also that the drops with Randomized TCP and Drop Tail queues are independent. For testing the first proposition, we varied the bottleneck bandwidth from 1-2 Mbps and the number of sources from 20 to 30. The endto-end propagation delay was 200ms, the bottleneck buffer was set as 25 packets. We assumed that there is no reverse path congestion and the maximum number of back-to-back packets or burst at the bottleneck will be just 2. We also verified this argument by cross checking the burst loss size with the congestion window trace file for each flow at the bottleneck.

Table 3.7 shows the results average number of burst losses for TCP Reno and Randomized TCP as the bottleneck bandwidth and the flow multiplexing is increased. It can be inferred from the table that as the number of flows increase, with the bandwidth kept constant, the number of back-to-back losses increase in TCP Reno and decrease (or remain constant) in Randomized TCP. This supports our argument that Randomized TCP reduces burst losses.

It can also be conjectured here that Randomized TCP distributes the loss over time. This is because, TCP Reno and Randomized TCP have the same congestion control policy the total number of drops are likely to be same for both. Thus, by reducing the burst losses Randomized TCP makes the losses distributed. This argument is further supported by the results in Section 3.10.1. There it was shown that Randomized TCP is successful in removing the TCP bias against longer RTT flows with Drop Tail queues. In [3] the authors show that TCP bias against long flows can be reduced by Active Queue Management which distributes losses uniformly over time, specifically RED. The similarity of our simulation results in 3.10.1 suggest that Randomized TCP does succeed in making losses independent by distributing them over time.

3.10.5 Summary

The observations of this section can be summarized as:

• Randomized TCP increases the fairness amongst competing flows of different

No. of	1 Mbps		2 Mbps	
Flows	Reno	RTCP	Reno	RTCP
20	87	23	1	27
25	119	18	100	31
30	141	15	168	28

Table 3.7: Comparison of average number of burst losses in Reno and Randomized TCD (PTCD)



Figure 3.15: Covariance coefficients for Paced and Randomized TCP for a transfer of 2500 packets. (Value around 0 is good.)

RTTs by removing the bias against the longer RTT flows (as found with TCP Reno) with Drop Tail queues.

- Presence of a "single" Randomized TCP flow at every bottleneck (Drop Tail Gateways) can reduce the bias against longer RTT flow at that bottleneck.
- Phase effects, which persist with TCP Reno with Drop Tail queues, are reduced if Randomized TCP is used.
- With bulk data transfers randomization reduces the synchronization of windows (thus loss events) as against TCP Reno. This should reduce the queue oscillations.
- Randomized TCP reduces synchronization with short web-transfers. This should lower average latency.

- Randomized TCP drastically reduces the number of burst losses. Specifically its performance increases as the number of flows increase.
- With Drop Tail queues Randomized TCP tries to distribute losses over time thus making them appear independent.

3.11 Binomial Congestion Control Algorithms

In [9] the authors propose a class of non-linear TCP compatible congestion control schemes called Binomial Congestion Control Schemes (BCCS) for audio and video applications. Formally, the Binomial Congestion Control scheme can be defined as:

$$W_{t+R} \leftarrow W_t + \alpha / W_t^k \quad if \ no \ loss \tag{3.11}$$

$$W_{t+\delta t} \leftarrow W_t - \beta W_t^l \quad if \ loss \tag{3.12}$$

where k and l are window scaling factors for increase and decrease respectively and α and β are increase the decrease proportionality constants. For any given values of α and β TCP Compatible BCCS can be defined by k+l = 1: $k \ge 0$, $l \ge 0$. Inverse Increase Additive Decrease or IIAD is one such BCCS with k=1, l=0. Similarly Square Root Increase and Square Root Decrease or SQRT is defined as k=0.5, l=0.5. We refer the reader to [9] for a more detailed description of Binomial congestion control schemes.

In [9], the authors show that these algorithms, specifically IIAD and SQRT, beat down TCP when sharing a drop-tail gateway and hence suggest the use of RED gateways to maintain fairness. This unfairness is due to unequal distribution of drops amongst these flows. This behavior is seen in figure 3.16 a) and c). When we incorporate randomization into binomial schemes as well and make it compete against randomized TCP, we see a marked improvement in fairness as in figure 3.16 b) and d), due to the by now familiar reasons of de-synchronization and more uniform distribution of losses. The end-to-end propagation delay for this experiment was 100ms, the bottleneck link's capacity was 1 Mbps and it was configured with Drop Tail queue with 25 packets of buffer.



Figure 3.16: Performance of Binomial Congestion Control Algorithms with Randomization

3.12 Conclusions

In this chapter we presented a methodology to introduce randomness in networks through end-to-end congestion control schemes. For the TCP case, we call it Randomized TCP. In this scheme, we space successive packet transmissions with a time interval $\Delta = RTT(1 + x)/cwnd$, where x is a zero mean random number drawn from an Uniform distribution. We showed that Randomized TCP, by introducing randomization in the network, reduces synchronization, phase effects and bias against bursty traffic, prevalent with current implementations of TCP and Drop Tail Gateways. We have also analytically characterized the new increase parameter for Randomized TCP to make it compete fairly with TCP. This was necessary because randomizing the sending times increases the RTT and as such the Randomized TCP losses to TCP Reno.

Randomized TCP reduces the bias against connections with larger RTTs with Drop Tail queues. The presence of a single Randomized flow at a bottleneck is sufficient to reduce the bias against longer RTT flows thereby motivating incremental deployment. Randomized TCP also reduces the burst losses and can also distribute losses over time thus emulating RED like properties. Multiplexing of Randomized TCP with TCP Reno helps in reducing synchronization and phase effects while increasing fairness. Additionally, when Randomized TCP is extended to Binomial congestion control schemes, there is a remarkable improvement in fairness, when competing with Reno. Consequently, it has high incentives for deployment.

Finally our results indicate that, *Randomized TCP can emulate the beneficial effects of RED in a distributed manner* without the complexities and unfavorable aspects of parameter tuning of RED. In addition, the benefits of randomization can be reaped even when it is partially deployed. However, we wish to emphasize that unlike RED which is a congestion avoidance scheme, Randomized TCP is just a congestion control scheme. Thus Randomized TCP does not emulate the congestion avoidance features of RED, at best it provides the other beneficial features of RED which were achieved by introducing randomization in the network. We are currently working on implementation of Randomized TCP in the Linux Kernel.

CHAPTER 4

Selfish Flows: Characterization and Performance on Drop Tail Queues

4.1 Introduction

Randomized TCP is an end-system based solution which emulates some beneficial properties of AQM. Specifically Randomized TCP achieves fairness in the network by reducing burst losses. However, since Randomized TCP does not differentiate between flows and does not manage queues in the network and it can not protect flows under all circumstances. In this Chapter we illustrate this through some simulation setups. We show that in presence of selfish behavior in the network the end-system based techniques are insufficient to provide fair service to all users. However, to show this we first define ways in which selfish behavior can be defined. Specifically we show the existence of stable rate control schemes. This is then used to show the unfair distribution of bandwidth amongst competing flows.

The rest of the Chapter is organized as follows.

- In Section 4.2 we classify the selfish behavior in the network. Specifically we discuss the different type of flow control algorithms being used in the Internet.
- In Section 4.3 we show a class selfish schemes in the network which are obtained by using different increase and decrease policies. However, these policies do not change with time.
- We relax the assumption of constant increase decrease policy in Section 4.4 and introduce schemes which change some of their control parameters with time.
- In Section 4.5 we use these selfish schemes to highlight the problem that endsystem based techniques are not sufficient to manage fair allocations in the network.
- Finally in Section 4.6 we summarize the arguments presented in this chapter.

4.2 Classes of Selfish Flows

In this section we will classify the selfish behavior of different rate control schemes. Since TCP is the most widely used transport protocol, for this classification we have used TCP as the benchmark flow, i.e. TCP flows are not considered to be selfish. Henceforth, we define selfish behavior as any rate control scheme which gets more share of the bottleneck bandwidth than TCP under same operating conditions. Though we have chosen TCP to indetify selfish behavior, we would like to point out that TCP is just one special case, we could have as well chosen some other rate control scheme to recognize the selfish behavior.

At the outset we can classify selfish flows into two broad categories: a) responsive or adaptive flow and b) un-responsive or non-adaptive flow. A flow is called un-responsive flow if it does not react to the congestion indications being fed to it by the network. On the other hand responsive flows react to congestion indications by cutting down their rates.

Constant Bit Rate (or CBR) flows and UDP are the two main un-responsive rate control schemes. These rate control schemes are increasingly becoming popular in the network, especially as TCP introduces delays because of all its reliability mechanisms. Most of the multimedia and gaming applications use error protection through coding schemes and therefore are resistant to packet losses, though worrying about the end-to-end delays. As such these schemes use UDP. Real Audio, Internet telephony, and on-line games like Quake, Half life etc are some applications which use UDP [31]. Finally, flows which always increase their rate with total disregard to congestion indication would complete the definition of un-responsive schemes.

These un-responsive flows can be modeled by any linear utility function, i.e. U(x) = ax and they are also characterized by constant marginal utility for any rate allocation. The utility function of such schemes is also given by a step function, i.e. till a particular rate these schemes have zero utility while after a particular rate these schemes have zero utility while after a particular rate these schemes have constant utility [71].

Responsive non-cooperative flows encompass a larger range of mis-behaving scenarios. However, their misbehavior can be differentiated on basis of their increase policy, (i.e. how they probe the network for available bandwidth) and their decrease



Figure 4.1: Classification of Selfish Behavior in the Network. Our region of interest is the Selfish Responsive Flows.

policy (or how they respond to congestion indication). These schemes could include strictly concave, concave or strictly increasing convex (with respect to rate) utility functions. Such a categorization of utility function leads to flows which are called greedy, i.e. they are always willing to consume any extra rate available (to them). This greediness enforces the strictly increasing condition on the utility functions.

Yet another class of non-cooperative functions manifest themselves as a mix of responsive and un-responsive flows. Some streaming application's rate control scheme falls in this category. These applications react to congestion indications till a certain limit (which could be rate or loss) and after that stops reacting to congestion indication and thus resorting to a CBR like transmission.

Finally, in figure 4.1 we show different rate control schemes. Though selfish behavior corresponds to all the sections other than TCP-Friendly flows, in this thesis *we will concentrate on managing selfish responsive flows*. Towards achieveing this, in this chapter we will outline various techniques which can be used to generate selfish behavior in the network. Later in Chapter 5 we will present an edge system based solution for managing selfish behavior.

4.3 Selfish Rate Control Schemes and their Utility Functions

During the congestion avoidance phase TCP increases it's window by 1 packet every RTT and cuts it's window by half on receipt of congestion indication. Thus, the rate control from TCP can described as:

$$I: W(t+R) \leftarrow W(t) + \alpha$$

$$D: W(t+R) \leftarrow W(t) - \beta W(t)$$

$$(4.1)$$

where R is the RTT, I,D represent the increase and decrease policy of TCP, respectively and α, β are the increase and decrease parameters. For TCP $\alpha = 1$ and $\beta = 0.5$.

Let us denote the instantaneous rate of a source by x and the packet loss probability as p. Further the relationship between window and rate is given as W = x.R. Then using the flow optimization analysis of Kelly, Low et al [44, 50, 46] and equilibrium properties of TCP [38] we can calculate the utility function of TCP as

$$U'(x) = p \tag{4.2}$$

$$\alpha (1-p)^{W(t)} = \beta W(t) (1-(1-p)^{W(t)})$$
(4.3)

$$\frac{\alpha}{W^2(t)\beta} = p \tag{4.4}$$

$$\frac{\alpha}{(xR)^2\beta} = p \tag{4.5}$$

$$U'(x) = \frac{\alpha}{(xR)^2\beta} \tag{4.6}$$

$$U(x) = \frac{-\alpha}{R^2 x \beta} \tag{4.7}$$

The equilibrium rate allocations of TCP can be found using equation 4.5. It is clear that the throughput of TCP, x, increases with increase in a value of α and a decrease in the value of β . Thus, a straightforward way of generating selfish flows, with to respect to TCP, would be to choose aggressive increase and decrease parameter. Therefore any choice $\alpha > 1$ or $0 < \beta < 0.5$ will result in aggressive rate control schemes. Akella et. al [5] use this definition of selfishness to evaluate the properties of Nash Equilibria with selfish flows and Drop Tail, RED queues.

Bansal et. al proposed non-linear increase decrease policies in form of Binomial Congestion Congestion Schemes. Formally, these schemes can be written as:

$$I: W(t+R) \leftarrow W(t) + \frac{\alpha}{W(t)^k}$$

$$D: W(t+R) \leftarrow W(t) - \beta W(t)^l$$
(4.8)

where α, β, k, l define the Binomial Algorithm. TCP-Friendly Binomial schemes are defined as all the scheme satisfying k + l = 1. Using an analysis similar to the one stated above the utility function of the binomial schemes can be calculated as:

$$p = \frac{\alpha}{\beta W(t)^{k+l+1}} \tag{4.9}$$

$$= \frac{\alpha}{\beta (xR)^{k+l+1}} \tag{4.10}$$

$$U(x) = \frac{-\alpha}{\beta (k+l)R (xR)^{k+l}}$$
(4.11)

Again, selfish schemes can be generated by changing the increase and decrease parameters, α and β respectively. However, the throughput of Binomial schemes is given as

$$x = \frac{1}{R} \left(\frac{\alpha}{\beta d} \right)^{\frac{1}{k+l+1}}$$

Since, p < 1 it is easy to see that equilibrium allocations increase with decreasing value of k + l. Thus selfish rate control schemes can be generated by choosing k + l < 1. Further the utility function of Binomial schemes is strictly concave with respect to β, x and k, l.

Sastry et. al further relaxed the increase and decrease policy to come up with the following rate update rules:

$$I: W(t+R) \leftarrow W(t) + f(W(t))$$

$$D: W(t+R) \leftarrow W(t) - g(W(t))$$

$$(4.12)$$

where f, g are some functions of window, W. The utility function of such schemes can be calculated as

$$U(x) = \int_{x} \frac{1}{Rxf(x)g(x)} dx$$
(4.13)

and the TCP-Friendly schemes are given by

$$f(x)g(x) \propto x \tag{4.14}$$

Further, Binomial congestion control schemes are special case of the above model.

From the discussions of utility function characterization of Binomial schemes and TCP, it is clear that selfish flows are given by

$$U(x) \propto \frac{-1}{x^n}, \quad n < 1 \tag{4.15}$$

Then using equation 4.15 selfish schemes can be generated by choosing f, g such that

$$f(x)g(x) < x \tag{4.16}$$

Also it can be easily shown that the strict concavity of the Utility function can be guaranteed by the following equation

$$\frac{-1}{x} > \frac{f'(x)}{f(x)} + \frac{g'(x)}{g(x)}$$
(4.17)

4.4 Aggressive Rate Control Scheme: Control Parameters are Time Dependent

In the previous section we looked at different selfish flow control schemes which were obtained by changing the increase and decrease rules. However, these rules do not change with time. In this section we consider rate control schemes where some parameters are allowed to change with time. Specifically, we will look at the timevarying Binomial schemes. We can get selfish rate control by either modifying the decrease parameter, β over time or by changing its binomial parameters k and l.

In the previous section we found the utility function for binomial schemes. It can be seen that the utility function is strictly concave with respect to x, β , k and l.

In this section we will evaluate what update rules are allowed for decrease parameter and k, l.

4.4.1 Modifying the decrease parameter β

Let us assume that the source changes it's decrease parameter, β with time and lets represent it as $\beta(t)$. Assume that the other parameters, α , k and l are kept constant. Then the network optimization

$$maximize \quad \sum_{s \in S} \quad w_s U_s(x_s) \tag{4.18}$$

subject to
$$\sum_{s \in S(l)} x_s \le C_l, \quad \forall l$$
 (4.19)

can be rewritten as

$$J(x_s, \beta_s) = \max_{x_s} \sum_s w_s U_s(x_s) - \gamma \sum_l \int_0^{\sum_j x_j} p_l(C_l, x) dx$$
(4.20)

where w_s is the weight for the utility function, U_s , of source s, x_s is the rate of source s, γ is some constant greater than 0 and p_l represents the penalty function and is given as

$$p_l(C_l, \lambda) = \frac{(\lambda - C_l)^+}{\lambda}.$$
(4.21)

where $y^+ = max(y, 0)$. Using the utility function of the binomial scheme we can rewrite equation 4.20 as

$$J(x_s,\beta_s) = \max_{x_s} \sum_s w_s \frac{-\alpha}{\beta_s(t) \ (k+l)d \ (x_s(t)d)^{k+l}} - \gamma \sum_l \int_0^{\sum_j x_j} p_l(C_l,x)dx \quad (4.22)$$

The function $J(x_s, \beta_s)$ is strictly concave and hence has a unique optimum. Therefore a sufficient condition to reach the optimal point is

$$\frac{dJ(x_s,\beta_s)}{dt} > 0 \tag{4.23}$$

To achieve this consider the following differential equation

$$\frac{dJ(x_s,\beta_s)}{dt} = \frac{\partial J(x_s,\beta_s)}{\partial x_s} \frac{\partial x_s}{\partial t} + \frac{\partial J(x_s,\beta_s)}{\partial \beta_s} \frac{\partial \beta_s}{\partial t}$$
(4.24)

which can be calculated as

$$\frac{dJ(x_s,\beta_s)}{dt} = \left(a_s \frac{U'_s(x_s)}{\beta_s} - \gamma \sum_l p_l(C_l,\sum_j x_j)\right) \dot{x_s} + \frac{b_s}{\beta_s(t)^2 x_s(t)^{k+l}} \dot{\beta_s}$$
(4.25)

where a_s, b_s are constant and can be calculated as

$$a_s = \frac{w_s \alpha}{d^{k+l+1}} \tag{4.26}$$

$$b_s = \frac{w_s \alpha}{(k+l)d^{k+l+1}} \tag{4.27}$$

Assume we choose the following rule for updating the rates

$$\dot{x_s} = \rho \left(a_s \frac{U'_s(x_s)}{\beta_s} - \gamma \sum_l p_l(C_l, \sum_j x_j) \right)$$
(4.28)

where ρ is some constant greater than 0. Such a choice of update rule also satisfies our needs as this rule corresponds to the window dynamics of binomial congestion control schemes. Substituting this update rule into equation (4.25) we get

$$\frac{dJ(x_s,\beta_s)}{dt} = \rho \left(a_s \frac{U'_s(x_s)}{\beta_s} - \gamma \sum_l p_l(C_l,\sum_j x_j) \right)^2 + \frac{b_s}{\beta_s(t)^2 x_s(t)^{k+l}} \dot{\beta}_s$$
(4.29)

From the above equation we can conclude that a **sufficient condition** for the game to reach its optimal point is that $\dot{\beta}(t) \geq 0$, i.e the decrease parameter increases with time. However in the previous section we have seen that the selfish behavior of a rate control algorithm increases with a decreasing value of β . But our present update rule for β , $\dot{\beta}(t) > 0$ will eventually make a selfish scheme TCP-friendly because eventually we will reach a value of $\beta \geq 0.5$. Thus we are interested in the update rule where the end-system is allowed to decrease its value of β with time. Assume that we choose $\dot{\beta}(t) \leq 0$ then the optimal point is reached if equation (4.23) is satisfied. This can be further expressed as

$$|\dot{\beta}(t)| < \frac{\dot{x}_s(t)^2}{\rho} \frac{\beta_s(t)^2 x_s(t)^{k+l}}{b_s}$$
(4.30)

$$< \frac{\dot{x}_s(t)^2}{\rho} \frac{\beta_s(t)^2 dW_s(t)^{k+l}}{w_s \alpha} \tag{4.31}$$

$$\dot{\beta}(t) < 0 \tag{4.32}$$

Assuming that the window size is always greater than or equal to 1, we can upper bound the decrease rate as

$$|\dot{\beta}(t)| < \frac{\dot{x}_s(t)^2}{\rho} \frac{\beta_s(t)^2 d}{w_s \alpha}$$

$$\tag{4.33}$$

Thus the minimum value of $\dot{x_s}$ can be calculated as

$$inf\{\dot{x_s}\} = inf\left(\rho\left(a_s \frac{U_s'(x_s)}{\beta_s} - \gamma \sum_l p_l(C_l, \sum_j x_j)\right)\right)$$
(4.34)

where *inf* is the infimum. Since the marginal utility is decreasing with rate while being strictly positive, i.e. $U'_s(x_s) > 0$ the first term in equation (4.34) is always positive. Thus we may rewrite the above equation as

$$inf\{\dot{x}_s\} = inf\left(\rho\left(-\gamma\sum_l p_l(C_l,\sum_j x_j)\right)\right)$$
(4.35)

Also from equation (4.21) the maximum value which $p_l(C_l, u)$ can achieve is 1. Let the maximum number of links used by any flow is upper bounded by L. Then we may write the minimum value of $\dot{x_s}$ as

$$\inf\{\dot{x_s}\} = -\rho\gamma L \tag{4.36}$$

Substituting equation (4.36) in equation (4.33) we get the upper bound on the value of $\dot{\beta}$ as

$$|\dot{\beta}(t)| < \frac{(-\rho\gamma L)^2}{\rho} \frac{\beta_s(t)^2 d}{w_s \alpha}$$
(4.37)

$$|\dot{\beta}(t)| < \rho(\gamma L)^2 \frac{\beta_s(t)^2 d}{w_s \alpha} \tag{4.38}$$

Finally, if the rate of change of decrease parameter, β , is non-negative or if negative is bounded by equation (4.38) then the system admits a unique optimal point.

4.4.1.1 Global Stability

Since the function $J(x_s, \beta_s)$ (equation 4.20) is strictly concave it has a unique maxima. Lets denote this maximum by J_{max} . Then the $J = J_{max} - J(x_s, \beta_s)$ can be thought of as Lyapunov function. It is easy to see that $J \ge 0$. Further it can be shown that J is Lipschitz continuous on $x_s \in (m_s, M_s), m_s > 0$ and $\beta > 0$. Then from equations (4.29, 4.38) we can conclude that

$$\frac{dJ}{dt} = -\frac{dJ(x_s, \beta_s)}{dt} < 0 \tag{4.39}$$

Thus from Lyapunov's stability theorem (Theorem 3.1 [45]) we have that the update rules for rate and the decrease parameter yield a stable system under equations (4.29, 4.38).

4.4.2 Modifying the Window Scaling Parameters, k, l

In the previous section we assumed that the window scaling parameters, k, l were held constant. In this section we relax this assumption however we add the assumption that the decrease parameter β is held constant. Further let us denote by n = k + l. Throughout this section we will use n for our analysis and use it to make observations about changing k, l with time. Then we may write the network optimization problem as

$$J(x_s, n_s) = \max_{x_s} \sum_s w_s \frac{-\alpha}{\beta} nd \left(x_s(t)d\right)^n - \gamma \sum_l \int_0^{\sum_j x_j} p_l(C_l, x) dx$$
(4.40)

where p_l is given by equation (4.21), and n_s represents the window scale parameter for source s. Since the above objective function is strictly concave in n and x the sufficient condition to reach the optimal point is

$$\frac{dJ(x_s, n_s)}{dt} > 0 \tag{4.41}$$

$$\frac{dJ(x_s, n_s)}{dt} = \frac{\partial J(x_s, n_s)}{\partial x_s} \frac{\partial x_s}{\partial t} + \frac{\partial J(x_s, n_s)}{\partial n_s} \frac{\partial n_s}{\partial t}$$
(4.42)

Assuming the update rule for rate is given by equation (4.28) the sufficient condition for above algorithm is met if

$$\frac{\partial J(x_s, n_s)}{\partial n_s} \frac{\partial n_s}{\partial t} \ge 0 \tag{4.43}$$

But

$$\frac{\partial J(x_s, n_s)}{\partial n_s} \frac{\partial n_s}{\partial t} = \frac{w_s \alpha}{\beta R} \left(n_s log(x_s R_s) + \frac{1}{n_s^2} \right) \dot{n_s}(t)$$
(4.44)

and since $\alpha, \beta, w_s, R, n_s$ are all positive we have that the optimal point will always be reached if

$$\dot{n_s}(t) \ge 0 \tag{4.45}$$

This update rule points that if the window scaling factors increase or stay constant with time, the optimal point will always be achieved. Further this also points to the following interesting update rule for the window scaling parameters

$$\dot{k}_s(t) = -\dot{l}_s(t) \tag{4.46}$$

4.4.2.1 Global Stability

Using an analysis similar to the one in Section 4.4.1.1 $J = J_{max} - J(x_s, n_s)$ can be thought of as the Lyapunov function. It's easy to see that J is again always positive and Lipschitz continuous on $x_s \in (m_s, M_s), m_s > 0$ and $n_s > 0$. Further from equations (4.41, 4.43) we have that $\dot{J}(t) \leq 0$. Thus J satisfies all the conditions for Lyapunov stability (Theorem 3.1 [45]) and the update rules of n_s, x_s are stable.

4.5 Selfish Flows and Drop Tail Queues

In the previous sections we showed different ways to generate mis-behaving flows in the network. In this section we will validate our claim that the Drop Tail queues do not protect TCP flows from mis-behaving flows. Further, Randomized TCP also is a marginal improvement over TCP and consequently does not protect TCP flows.



Figure 4.2: Topologies used in the Simulations.



Figure 4.3: Single Bottleneck: Throughputs (in pkts/sec) for two competing flows, one is TCP while the other is Mis-behaving (k=0,l=0.5)

Because of the simplicity of implementation and understanding, for this work we used Binomial scheme to generate misbehaving flows. We fixed the values of α, β as 1 and 0.5 respectively. TCP flows are defined by k = 0, l = 1 and as discusses previously in this Chapter, misbehaving flows are defined by k + l < 1. This is because network allocates more resources to flows which have higher marginal utility, U'_s . Henceforth, we will use the k and l values to identify misbehaving flows.

In figures 4.3 and 4.4 we plot the throughputs for flows competing on a single and multi-bottleneck topologies respectively. We first present the result with a single bottleneck (4.2 a) of 0.8Mbps and access links of 8Mbps for 2 competing flows. We evaluated the single bottleneck topology for the two cases, one when we used TCP Reno flow and a misbehaving flow (k=0, l=0.5) and in the second case we replaced the TCP Reno flow with Randomized TCP flow. In both the cases flows have

80



Figure 4.4: Multi Bottleneck: Throughputs (in pkts/sec) for 2 competing flows on a network of Drop Tail queues. One flow is TCP while the other is Misbehaving (k=0,l=0.5).

same RTT of 60ms. It can be seen from the figure 4.3 that in both the cases the misbehaving flow gets most of the bottleneck share. Moreover it beats the TCP flows comprehensively. Further, Randomized TCP only marginally improves the performance. This can be explained by the fact that network allocates equal losses to both the flows and the misbehaving flow by cutting down its window slowly as compared to TCP flows always get a larger share of the bandwidth.

Figure 4.2 b) show a multi-bottleneck topology with a TCP flow traversing both the bottlenecks while one short mis-behaving flow (k=0, l=0.5), each going through one bottleneck. It can be seen from figure 4.4 that TCP flow is almost shut out by the mis-behaving flows, who now get all the bandwidth. Not only is the TCP flow is forced into multiple timeouts (23 for this case) but these timeouts occur with very small windows and are often back to back. Similar results were obtained with Randomized TCP flow. In summary, with Drop Tail queues mis-behaving flows may get significant share of the bandwidth, almost to the extent of shutting out TCP flows.

4.6 Summary

In this chapter we first classified selfish flows. These flows were primarily characterized by their response to congestion indication. There are some flows which do not react to congestion indication. Instead either they keep sending their traffic at a specified rate on keep increasing their rates with total disregard to the state of congestion in the network. These flows are called non-responsive flows. However, there is another section of congestion control scheme, which reacts to congestion indication by cutting down the rate. But, in spite of this rate cut these flows may be selfish. This can be attributed to either their aggressive window increase (as compared to TCP) or by a smaller rate cut (than TCP). These flows are called responsive flows. The last category of the flows use both responsive and non-responsive policy. These flows, upon reception of congestion indication cut down their rates. However, beyond a certain threshold (which could either be a rate limit or a loss rate limit) these flows stop reacting to congestion signals, instead they keep sending data into the network at a constant rate.

After this classification of selfish behavior of rate control schemes we suggested ways in which these selfish responsive schemes could be implemented on the network. The simplest selfish schemes can be obtained by using aggressive increase and decrease parameter in TCP, i.e. $\alpha > 1$ or $0 < \beta < 0.5$. Another suggested method of generating selfish schemes was using Binomial schemes. These algorithms are characterized by window scaling parameters k, l besides the increase and decrease parameters. Selfish schemes can be created by using aggressive window scaling parameters, specifically by choosing k or l such that k + l < 1. These increase and decrease policy can be further generalized to include any function f(x), g(x) such that they are always positive. If f(x), g(x) are chosen such that f(x)g(x) is sub-linear then we can produce a variety of selfish rate control schemes. Besides these schemes, we also looked at schemes where the user is allowed to change his congestion control parameter with time. In this chapter we formulated guidelines under which these schemes might be stable.

Finally, in this chapter we evaluated the performance of Drop Tail queues in presence of selfish rate control schemes. Our results show that Drop Tail queues cannot protect TCP flows from selfish users. Moreover the performance of Drop Tail queues deteortes with multiple bottleneck, so much so that selfish flows can almost shut out TCP flows. Further, Randomized TCP also does not protect flows. This can be explained by the fact that network allocates equal losses to both the flows and the misbehaving flow by cutting down its window slowly as compared to TCP flows always get a larger share of the bandwidth. Therefore it is imperative that we use some network based strategies to protect flows from selfish behavior in the network.

CHAPTER 5

Non Cooperative Flow Control: An Edge-Based Re-marking Framework for Congestion Response Conformance in the Network

5.1 Introduction

One of the primary reasons for the success of the Internet has been its ability to support different applications. However, even though TCP remains the most widely used transport protocol, it does not always meet the requirements of these applications. This in turn has fueled development of numerous rate control schemes [9, 82, 70, 29, 68, 43]. As such we now have an Internet which supports spectrum of rate control schemes. However, this flexibility for the end users to choose their own rate control scheme presents us with problem of *protocol conformance and fairness*.

Protocol Conformance (or compliance) refers to a set of rate control schemes who have similar response to congestion indication [28]. In absence of compliance to a set of protocols, for example TCP, Floyd et. al [30] show that TCP flows might receive bandwidth significantly less than their fair share. This unfair sharing of the bottleneck is further highlighted in presence of unresponsive flows. In [30, 53, 13] Floyd et. al. also argue that absence of protocol conformance may lead to congestion collapse. These adverse effects of protocol non-conformance can also be explained by the flow optimization framework.

Recent work in flow control and optimization has shown that a flow's equilibrium allocation is very closely tied with the utility function (or equivalently the rate control scheme) the user chooses to maximize [44, 46, 50, 57]. Moreover, the network obliviously (i.e. does not differentiate between flows) drops (or marks) packets from different flows. Thus the unilateral action of flows to maximize their utilities (or rates) coupled with the oblivious network poses the problem of unfair bandwidth sharing and encourages selfish behavior (on part of users) in the network. In summary, protocol non-conformance pose twin problems of congestion collapse



Figure 5.1: Mapping a non-conformant user to a conformant space.

and unfair bandwidth sharing.

Towards addressing the issue of protocol conformance, especially given that TCP is the most widely used transport protocol, Floyd et. al. proposed guidelines for managing and designing new congestion control schemes such that they were *friendly* to TCP. This criteria of protocol conformance (to TCP) is called *TCP Friendliness*. A flow is deemed TCP-Friendly if its sending rate does not exceed that of a conformant TCP flow in same circumstances. This TCP-Friendly definition can further be relaxed to the following relationship between the sending rate, x, and loss rate, $p: x \propto \frac{1}{\sqrt{p}}$.

In this thesis, different rate control schemes are called conformant if they are maximizing the same utility function. Thus, in such a framework all congestion control schemes whose utility function is given by $\frac{-1}{x}$, where x is the rate, are called TCP Friendly. However, the protocol conformance need not be necessarily limited to a single utility function. Instead, it could be relaxed to include a range of utility functions, i.e. conformant space can be $[U_{obj}^1, U_{obj}^m]$. Figure 5.1 shows an example of such a conformant space, where any rate control scheme whose utility function lies in the shaded area is called conformant.

This chapter considers the question of how to enforce protocol conformance and consequently manage misbehaving flows at the edge of the network. We show that by a penalty transformation derived from the duality framework of Low et. al [50] we can map any utility function, U_s to a range of target utility functions $[U_{obj}^1, U_{obj}^m]$, or even to a single target utility function, U_{obj} . This thus helps us achieve our objective of protocol conformance. Figure 5.1 shows an example of this framework. Figure 5.1 shows a selfish user whose utility function is identified as Us(x) while the conformant space is defined by the utility functions, U1(x) and U2(x). Here the selfish user is mapped to the conformant space by a penalty transformation.

This chapter also considers "fairness" from the network's perspective (rather than the end user's perspective), and focuses on how a network can manage the distribution of rate amongst users. This can be achieved within the protocol conformance framework, by actively monitoring for selfish flows in the network and conveying them appropriate congestion indications. We also show that the proposed framework can provide broad range of fairness objectives, which to this date were not possible with existing network based solutions. Further, we show that the network may chose to convey congestion penalties to selfish users either by dropping or marking their packets. This flexibility of conveying penalty either by dropping or marking packets enables the proposed framework to work even with Drop Tail queues. However, a limitation of this model is that we *only considers mapping responsive rate control schemes*.

In summary, we propose a framework for policing non-conformant flows in the Internet by conveying a higher penalty to them. Further, these penalty transformation agents *can be placed on the network edges* and we can choose to either re-mark or drop the packets (or acks). This function can also be thought of as a generalization of ack-pacing being done by Packeteer [60] which are widely deployed in the Internet. Figure 5.2 shows the model for policing non-conformant users.

The edge based re-marking framework presented in this thesis can also be extended to provide service differentiation. This could be achieved by grouping users into different sets and then mapping these sets to different ranges of target utility functions, e.g. map U_s to $\{U_{obj}^1, U_{obj}^m\}$. Even otherwise for purposes of service differentiation flows can themselves be mapped to a range of utility functions (see figure 5.1). This solution is attractive because it can be achieved irrespective of the congestion control scheme employed by the user and also does not require any support from the core routers in the network. Moreover, it can work with Drop-Tail



Figure 5.2: Model for policing non-conformant users through Penalty Transformation.

queues or any other oblivious AQM scheme.

The rest of the chapter examines the policing of non-compliant sources in detail. The organization of the chapter is as follows:

- In Section 5.2 we present the network model, assumptions and motivation for protocol conformance.
- In Section 5.3 we present the edge based re-marking model.
- We present the implementation and simulation setup in Section 5.4 and estimation of utility function is described in Section 5.4.1.
- In Section 5.5 we present the results of the re-marking framework. The model is evaluated with both marking and dropping for single and multi-bottleneck topologies, background traffic and reverse path congestion.
- Finally we present the conclusions and limitations of the model in Section 5.6.

5.2 Re-Marking Framework for Managing Non-Conformant Users

5.2.1 Network Model, Definitions and Assumptions

Consider a user s, who is described with the help of his rate, x_s , a utility function U_s and the set of links which he uses, L(s). Let the network be identified

with links l of capacity C_l and the set of users using a link, l, be given by S(l). Further, we will assume that the rates are bounded and that the utility functions are *increasing* with rates and *strictly concave*. Formally, the assumptions are stated as

- A1: The Utility functions are continuous, strictly concave and increasing in their arguments. Further the rates are bounded by I: $[m_s, M_s]$.
- A2: The curvature of U_s are bounded away from 0 on I, i.e. $-U''_s(x_s) \ge 1/\alpha_s > 0$.

Then the flow optimization problem can be defined as users trying to maximize their individual utility functions and the network trying to maximize the resource allocation subject to link capacity constraints. The problem is formally defined as [50]:

$$maximize \sum_{s \in S} U_s(x_s) \tag{5.1}$$

subject to
$$\sum_{s \in S(l)} x_s \leq C_l, \quad \forall l$$
 (5.2)

for all $x_s \ge 0$. The solution to this problem is given by the following update rules

$$x_s(t) = U_s^{\prime -1}(\sum_l p_l)$$
(5.3)

$$p_l(t+1) = [p_l(t) + \gamma(\sum_{s \in S(l)} x_s - C_l)]^+$$
 (5.4)

where p_l are the dual variables of the problem and can be identified as penalties, price or link loss probability [50, 46, 44].

From the above update rules it follows that both the rate control algorithm and the equilibrium rate can be associated with the utility function user chooses to maximize (equation (5.3, 5.4)). However, given that the *same price* is being communicated by the network, the equilibrium rates can be different, but are still fair within Kelly's utility function framework. Thus even though the network doesn't desire to be perceived unfair, a bias in equilibrium rates can be created by choosing two different utility functions. We now illustrate this through an example.



Figure 5.3: Example1: Two competing set of flows through one bottleneck.

5.2.2 Motivation

In this section we will illustrate through examples what we mean by unequal bandwidth sharing. Specifically, we present two examples illustrating selfish behavior in a single bottleneck topology and a multi-bottleneck topology.

Example 1. Consider a bottleneck link where two set of rate control schemes compete for the bandwidth as shown in figure 5.3. The utility function for Set 1 is given by $U_s(x_s) = w_s \log(x_s)$ and that for Set 2 is given by $U_s(x_s) = -w_s x_s^{-1}$, where w_s represents the weight assigned to the flow. Let there be 50 sources each in Set 1 and Set 2. Assume that the link capacity to be 300, weights to be 1, the round-trip time (RTT) for all sources to be same. Then the throughput seen by each source can be obtained by solving the following optimization problem:

$$max \qquad \sum_{i=1}^{50} log x_i - \sum_{j=51}^{100} \frac{1}{x_j} \tag{5.5}$$

subject to
$$\sum_{i=1}^{50} x_i + \sum_{j=51}^{100} x_j \le 300$$
 (5.6)

and $x_i, x_j \ge 0 \ \forall i, j$. Solving this problem yields $x_i = 4.0, i \in \{1, ..., 50\}$ and $x_j = 2.0, j \in \{51, ..., 100\}$.

Thus even though the network is fair, the equilibrium rate depends on the rate control algorithm chosen by the sources. This differentiation in the rates is present because the network conveys the same congestion price to each competing user. (Henceforth we will call such a network as an oblivious network) and users respond differently to the congestion penalties. Another reason for rate differentiation can be attributed to how the users probe the network (or the increase policy). Thus with oblivious network, different final allocations can primarily be associated with



Figure 5.4: Example: Three competing set of flows through two bottlenecks.

users rate control schemes.

Example 2. Figure 5.4 shows a scenario where there are 50 flows in Set 1 traversing both the bottlenecks. Let all of them have the same utility function of $U_s(x_s) = -x_s^{-1}$. Then there are two other sets of flows, Set 2 and 3 which go through bottleneck 1 and 2 respectively. Sets 2 and 3 have 50 flows each and their utility function is given as $U_s(x_s) = \log(x_s)$. We will assume that all the flows have same RTT. Let the capacity of both the bottleneck links be 300 units. Then the final rates are the solution to the following optimization problem

$$max \qquad \sum_{i=1}^{50} \frac{-1}{x_i} + \sum_{i=51}^{100} \log x_i + \sum_{i=101}^{150} \log x_i \tag{5.7}$$

subject to
$$\sum_{i=1}^{50} x_i + \sum_{i=51}^{100} x_i \le 300$$
 (5.8)

$$\sum_{i=1}^{50} x_i + \sum_{i=101}^{150} x_i \le 300 \tag{5.9}$$

Solving the above optimization problem we get the equilibrium rate allocation, $x = \{1.5, 4.5, 4.5\}$ for the flows in Set 1, 2 and 3 respectively. However, if all the flows in Set 1 use a utility function, $U_s(x_s) = \log(x_s)$ then on solving the corresponding optimization problem we would get the final rate allocations for Set 1, 2 and 3 as $x = \{2, 4, 4\}$, respectively.

The above examples illustrate that if a subset of flows on the network change their utility function then the rate allocations at the bottleneck change. Thus with oblivious (i.e. which do not differentiate between flows) queue management schemes at the bottleneck e.g. RED the fairness (or the final rate allocation) in the network seems to be solely governed by its users rate control scheme. Also, it can be seen from the example 2 that by using a slightly aggressive utility function, $log(x_s)$ instead of $\frac{-1}{x_s}$, the users of Set 1 can significantly alter their rates. In this case the users in Set 1 increased their equilibrium allocations by 33%.

These examples thus illustrate that in an oblivious network, the fairness criteria is dependent upon the utility function chosen by the user. In other words the network does not control the rate distribution of the users and does not enforce any particular fairness criteria. For example, TCP Reno is associated with minimum potential delay fairness while TCP Vegas with proportional fairness [46], however when both TCP Reno and Vegas flows are competing for bandwidth, the final rate allocation is neither minimum potential delay fair nor proportionally fair. This is also illustrated in the above examples when the flows in Set 1 use the utility function $U_s(x_s) = -x_s^{-1}$ (TCP Reno) while the rest use $U_s(x_s) = log(x_s)$ (TCP Vegas), which can be verified to be neither minimum potential delay fair nor proportionally fair. But, if all the competing users deploy the same rate control scheme, e.g. use $U_s(x_s) = log(x_s)$, the final rate allocation as the bottleneck is indeed proportionally fair, as desired.

To summarize the arguments of this section, in presence of queue management schemes which do not differentiate between flows the fairness or the equilibrium rate allocations depend almost entirely on user's rate control schemes. Thus there are clear incentives for selfish behavior. Also the above arguments suggests that fairness might not entirely be network's prerogative, especially if the network does not differentiate between flows. Now we outline the re-marking framework, wherein the network by transforming the congestion penalties can make it appear as if all the users are maximizing the same utility function. Thus, the network by choosing a utility function can provide the fairness associated with that utility function throughout the network.

5.3 Re-marking Framework for Managing Non-Conformant Users

From the discussion in the previous section it follows that sources can choose rate control schemes which yield higher rate allocations. Another important point to note is that even though the sources are cooperative (i.e. they react to congestion indication) they still can be unfair. As such it is imperative to decouple the fairness criteria from the user's rate control scheme and let the network decide the fairness. That way the network not only has the flexibility of being fair, but more importantly it can choose the fairness criteria it wants to provide. Now we shall describe the re-marking framework to manage non-conformant users.

Lets assume that the users are maximizing the utility function U_s and that the network decides that the final equilibrium rate allocation should be, as if every user chose to maximize a utility function of U_{obj} . Then the rate updation algorithm (and thus equilibrium rates) of the users is given by equations (5.3,5.4). Now, if we communicate a link price $f(p_l)$, instead of p_l , then the user-rate updation algorithm will be

$$x_s(t) = U'^{-1}(\sum_{l \in L(s)} f(p_l))$$

Further, if we choose $f(p_l) : f(p_l) \ge 0$, $\forall p_l, f(0) = 0$ and the following condition holds true

$$\sum_{l \in L(s)} f(p_l) = U'_s(U'^{-1}_{obj}(\sum_{l \in L(s)} p_l)) = g(\sum_{l \in L(s)} p_l)$$
(5.10)

then the rate updation algorithm algorithm becomes

$$x_s = U_s^{\prime-1}(\sum_{l \in L(s)} f(p_l))$$
(5.11)

$$= U_{s}^{\prime-1}[U_{s}^{\prime}(U_{obj}^{\prime-1}(p^{s}))]$$
(5.12)

$$= U_{obj}^{'-1}(p^s) (5.13)$$

where $p^s = \sum_{l \in L(s)} p_l$. From the above equation it is easy to see that by communicating a different price we can transform the user's utility function from $U_s(x)$ to $U_{obj}(x)$. This transformation can be explained by the following modified dual:

$$D(p) = \min_{p \ge 0} \sum_{s \in S} U_s(x_s) - \sum_l f(p_l) (\sum_{s \in S(l)} x_s - C_l)$$
(5.14)

where $f(p_l)$ is defined by equation 5.10. Next we will show that a unique solution exists for the modified dual, but before that we prove the following proposition.

Proposition 1. Given the non-negativity constraint on x_s and p_l and strictly concave utility functions U_s and U_{obj} , the function $g(\sum_{l \in S(l)} p_l)$, $f(p_l)$ as defined in (5.10) are non-negative and strictly increasing in their argument.

Proof. Define $p^s = \sum_{l \in S(l)} p_l$. Note $g(p^s) = U'_s(U'_{obj}(p^s))$. Recognizing that $U'_{obj}(p^s)$ is just x_s from equation (4), we can rewrite $g(p^s)$ as $g(p^s) = U'_s(x_s(p^s))$. Since $U_s(x_s)$ is increasing and strictly concave in its arguments hence $U'_s(x_s) \ge 0$. Hence, $g(p^s)$ is greater than 0.

Let's define $F(p^s) = U'_{obj}(p^s)$ and it's inverse as $H(p^s) = F^{-1}(p^s)$. Therefore, $H(F(p^s)) = p^s$.

Now differentiating both sides with respect to p^s we get,

$$H'(F(p^s)) \cdot F'(p^s) = 1$$
 (5.15)

or
$$(U_{obj}^{'-1}())' = \frac{1}{U_{obj}^{''}(p^s)}.$$
 (5.16)

Now, differentiating $g(p^s)$ with respect to p^s we get

$$g'(p^{s}) = U''_{s}(U'^{-1}_{obj}(p^{s})) \cdot (U'^{-1}_{obj}(p^{s}))'$$

= $U''_{s}(\cdot)(U'^{-1}_{obj}(\cdot))'.$ (5.17)

Since U_s and U_{obj} are strictly concave therefore $U''_s(), U''_{obj}() < 0$ and from equation (5.16) we conclude that $g'(p^s)$ is greater than 0. Combining $g'(p^s) > 0$ and the definition of $f(p_l)$ (equation 5.10) we conclude $f'(p_l) > 0$.

Theorem 1. The modified dual represents a non-linear optimization problem where the objective function is as if every user is maximizing a utility function of U_{obj} subject to the capacity constraints. Moreover, if the objective utility function is strictly concave then a unique maximizer exists.

Proof. The transformation or the re-mapping function, $U'_{s}(U'^{-1}_{obj}(p))$, can also be explained as the solution to the following set of equations:

$$\sum_{s \in S(l)} x_s \leq C_l, \quad \forall l \tag{5.18}$$

$$p_l(\sum_{s \in S(l)} x_s - C_l) = 0 (5.19)$$

$$U'_{s}(x_{s}) = g(\sum_{l \in L(s)} p_{l})$$
 (5.20)

$$p, x \geq 0 \tag{5.21}$$

Then using equation 5.10 we can rewrite equation 5.20 as

$$U_{obj}'(x_s) = \sum_{l \in L(s)} p_l$$
 (5.22)

Then equations (5.18-5.22) are the KKT conditions for the following strictly concave maximization problem

$$\underbrace{\max_{x}}_{x} \quad \sum_{s \in S} \quad U_{obj}(x_s) \tag{5.23}$$

$$\sum_{s \in S(l)} x_s \leq C_l, \ \forall l \tag{5.24}$$

$$x \ge 0 \tag{5.25}$$

Then using assumption A1 we conclude that the objective function (equation 5.23) is strictly concave and hence an unique solution exists.

Then, using the KKT conditions and the gradient projection method we get the following rate and price updation rules

$$x_s = U_s^{\prime - 1}(\sum_l f(p_l))$$
(5.26)

$$p_l(t+1) = [p_l(t) + \gamma \frac{\partial}{\partial p_l} f(p_l) (\sum_{s \in S(l)} x_s - C_l)]^+$$
(5.27)

The above formulation is however difficult to implement because it requires per-flow queuing and that too inside the network. Since upgrades in the network are hard to achieve consider the following update rule.

$$p_l(t+1) = [p_l(t) + \gamma(\sum_{s \in S(l)} x_s - C_l)]^+$$
 (5.28)

$$x_s(t+1) = U_s^{\prime-1}(\sum_l f(p_l(t)))$$
(5.29)

Next we establish that the update rule presented above converges to the optimal point. But before that we prove that D(p) is lower bounded, continuously differentiable and convex and $\nabla D(p)$ is Lipschitz continuous.

Proposition 2. Under Assumptions A1 $\nabla D(p)$ is Lipschitz.

Proof. Define by A the incidence matrix where A_{ls} is 1 if source s uses link l and 0 otherwise. Further let the total number of links used by any source be bounded by L and the total number of sources by S. Then after some simplification we have

$$\frac{\partial x(p)}{\partial p} = diag\left(\frac{1}{U_{obj}''(x_s(p))}\right) A^T$$
(5.30)

Also from equation (5.14) we get $\nabla D = f'(p)(C - Ax)$. Differentiating it again with respect to p_l we get

$$\nabla^2 D = f''(p)(C - Ax) + f'(p)(-A\frac{\partial x(p)}{\partial p})$$
(5.31)

After some simplification f''(p) can be calculated as

$$f''(p) = \frac{U_s^{3'}(x(p))}{\alpha_s^2} + \frac{\alpha_s}{U_{obj}^{3'}(x(p))}$$
(5.32)

Since the utility functions are strictly increasing in their arguments hence they will be rightly skewed, i.e. $U_s^{3'}$ is bounded away from 0. Further since the rate are bounded by I (Assumption A1) the second derivative of f(p) will be bounded, let us say that this bound is F. After some simplification the bound on $f'(p)(-A\frac{\partial x(p)}{\partial p})$ can be calculated as βLS (for some $\beta > 0$ and β function of $\alpha_s(> 0)$). Then using the capacity constraint we conclude that ∇D will be Lipschitz with the following bound

$$\left\|\nabla D(q) - \nabla D(p)\right\| \le \left(FC + \beta LS\right) \left\|q - p\right\|$$

Proposition 3. Under assumption A1 D(p) is lower bounded, continuously differ-
Proof. By Assumption (A1), U_s is bounded and continuously differentiable thus U'_{s}^{-1} (and f(p)) exist and is also bounded and continuously differentiable. Therefore, D(p), as defined in equation (5.14) is also lower bounded and continuously differentiable.

Further from the assumption A1 (strictly increasing and strictly concave utility function) and equation (5) f''(p) (as defined in equation (5.32)) will be greater than 0. Using this knowledge and the capacity constraint the first term in equation (5.31) is always greater than or equal to 0. The second term of equation (5.31) is

$$f'(p)(-A \ diag\left(\frac{1}{U''_{obj}(x_s(p))}\right)A^T)$$

also strictly positive because from Proposition 1, f'(p) is always greater than 0, the incidence matrix is a 0-1 matrix and the utility functions are strictly concave. Thus we can say that equation (5.31) is greater than 0. Or $\nabla^2 D(p) \ge 0$ and D(p) is strictly convex.

Proposition 4. Given the non-negativity constraint on x_s and p_l and strictly concave utility functions U_s and U_{obj} , the new update algorithm as defined in equations (5.28, 5.29) converges to the optimal point.

Proof. Using the equation 5.14 and differentiating it with respect to time we get

$$\frac{d}{dt}D(p) = \frac{d}{dp_l}\left(f(p_l)[C_l - \sum_{s \in S(l)} x_s]\right)\frac{d}{dt}p_l$$
$$\frac{d}{dt}p_l = \gamma(\sum_{s \in S(l)} x_s - C_l), \ \gamma > 0$$
$$Thus, \ \frac{d}{dt}D(p) = -\gamma \frac{d}{dp_l}f(p_l)[C_l - \sum_{s \in S(l)} x_s]^2$$

Since, $f'(p_l) > 0$ and $\gamma > 0$ we can establish that $D(p_l(t))$ is a decreasing function in t. Also since D is strictly convex (see Proposition 3), there exists a minima, and

 $\frac{d}{dt}D(p(t)) \leq 0$ implies convergence to the optimal point.

Theorem 2. Assume that utility functions, U_s , are increasing, strictly concave and continuously differentiable, and their curvature is bounded away from 0. Then starting from any initial rates in the interior of X and prices $p(0) \ge 0$, every accumulation point (x^*, p^*) of the sequence $(\mathbf{x}(t), \mathbf{p}(t))$ generated by the above algorithm and equations (5.28,5.29) is primal dual optimal.

Proof. By Propositions 3 and 4 the dual objective function D(p) is convex, lower bounded and $\nabla D(p)$ is Lipschitz, then any accumulation point p^* of the sequence $\{p(t)\}$ generated by the gradient projection algorithm is dual optimal [11]. Moreover, the constraints are linear and the primal problem is strictly concave hence there is no duality gap. Therefore dual optimal is also primal optimal.

Thus this update rule minimizes the dual function and converges asymptotically. The above update rule also does not change the core network, as we retain the price update rule as proposed in [50]. Further, the price being communicated to the user can be updated at the edge. We now state the algorithm for the edge re-marker as

Edge Marker's Algorithm:

- For each source, receive from the network the total price for the source's traffic as $p^s(t) = \sum_{l \in S(l)} p_l(t)$.
- Recalculate (or Re-mark) the new price for the source as

$$p_{new}^s = g(\sum_{l \in S(l)} p_l(t)).$$

• Communicate this *re-marked* price to the source.

The update algorithm for the network and the source are given by equation (5.28)and (5.29) respectively.

Finally, we end this section with the following remark on convergence of the algorithm.

Proposition 5. The rate of convergence of the algorithm is given by the smallest eigen vector of ABA^t where A is the routing matrix and B is $diag(U_{obj}^{'-1}(p^*))'$ and p^* is equilibrium price.

Proof. In Theorem 1 we showed that by our penalty transformation the original optimization problem was translated into as if all the users were maximizing a utility function of U_{obj} and the update algorithm can be written as

$$\frac{\partial p(t)}{\partial t} = -\gamma (\mathbf{A}\mathbf{x} - \mathbf{C}) \tag{5.33}$$

$$\mathbf{x} = \mathbf{U_{obj}}^{\prime-1}(\mathbf{p^t}\mathbf{A}) \tag{5.34}$$

where **p** is the vector of dual variables or price in our framework. Since the objective function is strictly concave a unique maximizer exists, let this maximizer be called p^* . Let $p(t) = p^* + \xi(t)$. Then linearizing the system of equations (5.33,5.34) about p^* and after some simplification we get

$$\frac{\partial (p^* + \xi(t))}{\partial t} = -\gamma \left(\mathbf{A} \mathbf{U}_{\mathbf{obj}}'^{-1} ((\mathbf{p}^* + \xi(t))^{\mathbf{t}} \mathbf{A}) - \mathbf{C} \right)$$
(5.35)

$$\frac{\partial \xi(t)}{\partial t} = -\gamma \mathbf{A} B \mathbf{A}^t \tag{5.36}$$

$$B = diag(\mathbf{U}_{\mathbf{obj}})^{\prime - 1}(p^*))^{\prime}$$

$$(5.37)$$

Thus the rate of convergence of the algorithm is given by the smallest eigen vector of ABA^t .

5.4 Implementation

We implemented the edge based re-marker in the NS (Network Simulator). The edge based re-marker was tested for two scenarios, one when network marks packets and second when it drops packets. For the case where the network marks packets, the edge based re-marker was placed on the reverse path (i.e. on the reverse access link of the user) and re-marked the ACKs. However when the network is dropping packets, the edge based re-marker was placed on the forward path at the network egress and dropped packet. Also, in either cases the edge re-marker estimated the loss rate for each flow and subsequently used it to re-mark or drop acks or packets. We also assumed that we know the utility functions of all the flows. However, later we detail a procedure to estimate the utility functions of users at network edges.

For our simulation we used the congestion control and loss recovery mechanisms of TCP New Reno. Also, we disabled the delayed acknowledgments option. In our simulations we have assumed TCP Friendliness as the conformance criteria. Thus all rate control schemes whose utility function is given by $U_s = \frac{-1}{x}$ are called conformant (or compliant or TCP Friendly). For simulating mis-behaving (or selfish) flows we used the Binomial Congestion Control scheme (BCCS) proposed in [9]. As explained in Chapter 4 the BCCS is described by the window increase and decrease parameters: α and β respectively and the window increase and decrease scaling factors, k and l respectively. For our simulations, we fixed the values of α, β as 1 and 0.5 respectively. Also, as shown in Chapter 4 the utility function for Binomial schemes is defined as $U(x) \propto \frac{-1}{x^n}$, where k+l=n. Thus conformant or TCP Friendly flows are described by k+l = 1. Since the network allocates more resources to user's whose marginal utility are higher, non-conformant or misbehaving flows can be generated by choosing k + l < 1. Henceforth, we will use the k and l values to identify mis-behaving flows.

Figure 5.5(a) shows the single bottleneck topology used in the simulations. The access links were configured at a rate 10 times greater than that of the bottleneck link. All the links use Random Early Drop (RED) queues with min thresh and max thresh set as buffer/3 and 0.8*buffer respectively, where buffer is the total bottleneck buffer length. Further, the weight was set as 0.002 and the marking probability for RED was set to 0.1. The RTT was 60ms and the packet size 500B.

Figure 5.5(b) shows a multi-bottleneck topology used in the simulation. The bottleneck buffer was set to 25 packets. We also evaluated our framework for another multi-bottleneck setup of bottleneck link of 10 Mbps, access link of 100 Mbps and a buffer of 250 packets. The link delays were kept the same. RED minimum and maximum threshold settings were similar to those of single bottleneck. Also for all



(b) Multi-Bottleneck Topology

Figure 5.5: Topologies used in the Simulations.

the simulation setups (single or multi-bottleneck) the access link rate are always 10 times greater than that of the bottleneck link.

The maximum advertised window is set sufficiently high so that it does not constrain the actual window. The simulation time for each setup was 1500 seconds. We plot the throughput of competing flows in packets/sec, averaged over 20 roundtrip times. We assumed that all the flows have infinite data to transfer.

5.4.1 Estimating the Utility Function

The framework presented in this paper works well if the network knows the utility function of the non-cooperative flows. Thus estimating the utility function parameters is of paramount importance. In this section we briefly describe ways to estimate the utility function.

In this paper we have chosen BCCS schemes as non-cooperative users. These

schemes can generally be described by their exponent, n, by the following relationship

$$U(x) \propto \frac{-1}{x^n} \tag{5.38}$$

Thus, for describing these class of selfish flows we just need to estimate the parameter, n. For that purposes consider the following relationship between the rate and the loss probability

$$U'_s(x_s) \propto p$$
 (5.39)

$$p \propto \frac{n}{x^{n+1}} \tag{5.40}$$

$$log(p) = log(nK) - (n+1)log(x)$$
 (5.41)

where K is some constant. It is interesting to note that estimating the parameter n is nothing but a regression analysis on the equation (5.41). But for those purposes we would need to have a measure of the throughput x and the loss probability p. These can be calculated by either sampling the packet stream (at the egress) or the ackstream. If processing the packet-stream we could just count the number of packets sent and lost in a specified time. Using this time-series a Linear-Least Squared Errors (LLSE) method could be applied to estimate n. In this paper we employed LLSE to estimate the parameter n and present the results in Section results.

For a more general utility function as defined in equation (4.13) we could employ the Non-Linear Least Squared techniques to detect a power-series in x and n. We are currently working on this estimation problem.

However, for all these estimation to work we would need to identify the noncooperative users first. We leave this problem as that of future work, but would like to point out to some schemes described in SRED, Stochastic Fair Blue and RED-PD. But irrespective of how we detect these selfish users, we will have to store information (or state) about them at the routers. However, since these will be malicious users the amount of state would not be large. Moreover, it is further constrained by the fact that this state information is kept *only* at the edge routers and *nothing* needs to be stored in the core. Also, since we are trying to police selfish flows we are inadvertently doing per-flow management where per-flow is the total number of selfish flows in the Internet. Stateless schemes like CHOKe which police the selfish flows enforce a max-min fairness criteria and therefore may end up differentiating flows on RTTs and also in a multi-bottleneck scenario do not enforce the protocol compliance criteria (like TCP Friendliness).

5.4.2 Estimating the Loss Rate

The scheme proposed in this paper is sensitive to loss-estimation. If we underestimate the loss rate, then we would be not penalizing the selfish user enough and consequently there will be unfair rate allocation in the network. On the other hand over-estimation of losses leads to the scenario where selfish user's are over penalized and get a lesser share of the bandwidth, even less than the conformant users.

In [29] the authors propose two methods for estimating losses: Exponential Weighted Moving Average (EWMA) and the Weighted Average Loss Indication (WALI). In the EWMA proposal the losses are averaged infinitely over time while in WALI the authors average a fixed window of loss event with higher weights to the current loss events. However, there are no clear guidelines on how to configure EWMA and WALI weights; the values assigned are obtained by trials and some intuition regarding the importance of recent loss events. End-to-end losses in the Internet have been examined using real traffic traces in [81]. Again, the modeling aspect is limited to comparing a fixed window averaging scheme to an EWMA scheme, no attempt is made to find weights for past samples or adapt these weights. The traffic process (not the loss process) has been modeled using an AR model in [15]. The authors in [15] make no attempt to analyze the loss process and restrict the paper to fitting an AR model to traffic traces. In [67] the propose a linear prediction formulation to predict these loss rates. In order to adapt the loss rates to the samples as they arrive, the author has suggested to use the Recursive Least Squares algorithm.

For estimating the losses we have used EWMA and the WALI methods of Equation-Based Rate Control algorithm [29]. We updated these loss indications every RTT and we have assumed that the network knows the RTT of the flows. We present the results for EWMA based loss-estimator. Similar results were obtained with WALI based estimator. For EWMA based system we gave 60% weight to the history, while with the WALI based estimator we measured samples over 8 windows to estimate losses.



Figure 5.6: Single Bottleneck (Marking): Throughputs (in pkts/sec) for two competing flows, one is TCP Friendly while the other is non-conformant with and without Re-Marking.



Figure 5.7: Single Bottleneck (Marking): Throughputs (in pkts/sec) for ten competing flows, where seven flows are TCP Friendly while three are non-conformant (k=0, l=0.5) with and without Re-Marking.

5.5 Simulation Results

In the following sections we present our simulation results. Our simulation objectives can be stated as

• Validate the model with both single as well as multi-bottleneck topologies with varying degrees of (flow) multiplexing.



Figure 5.8: Single Bottleneck (Marking): Throughputs (in pkts/sec) for three competing flows, where one flow is TCP Friendly while the other two are non-conformant with (k=0, l=0.5) and (k=0, l=0.2) resp., with and without Re-Marking.

- Examine the robustness of the model in presence of background (web) traffic and reverse path congestion.
- Verify if the model works with dropping as a congestion notification mechanism. Specifically, if it can work with a network of Droptail queues only.
- Substantiate and test how to estimate utility functions.
- Test the sensitivity of the model with respect to inaccurate RTT and utility function estimates.

The result section is organized into two separate sub-sections to evaluate the framework with both marking and dropping. In Section 5.5.1 we present the results of edge-based re-marking framework. For the results in this section, the penalty transformation agents were placed on ingress node in the reverse path and re-marked the Acks. In order to evaluate the network where ECN is not enabled and dropping is used to convey congestion indication, we placed the penalty transformation at the egress nodes on the forward path. These agents conveyed appropriate penalties by dropping packets from the malicious flows. The results with dropping are reported in Section 5.5.2.

5.5.1 Evaluation of Edge Based Re-Marking Framework on an ECN Enabled Network

In this section we present the results of managing selfish behavior on an ECN enabled network. We assume that the network operates with RED queues and their parameter setting are detailed in Section 5.4. We evaluate the framework for both single and multi-bottleneck scenarios, , background traffic and with reverse path congestion.

5.5.1.1 Single Bottleneck

In figure 5.6 a) we present the throughputs of two competing flows on a single bottleneck of 0.8 Mbps with a buffer of 25 packets. Here, one of the flows is TCP, while the other is non-conformant and is defined by k = 0 and l = 0.5. As the figure 5.6 a) shows, when we do not re-mark the non-conformant flow, it garners more bandwidth than the TCP friendly flow. However, re-marking the non-conformant flow makes the two flows to share the bandwidth equitably. Figure 5.6 b) show similar results where the non-conformant user is defined by k = 0, l = 0.8.

Figure 5.7 shows the results for a set of 10 competing flows on a 10Mbps bottleneck and 150 packet buffer. The flow set comprises of 7 TCP Friendly flows while the remaining 3 flows are non-conformant and are defined by k = 0 and l = 0.5. The bandwidth is shared equitably in presence of re-marking, however in absence of re-marking mis-behaving flows easily beat the TCP Friendly flows.

We also evaluated our scheme for a scenarios where every flow has a different utility function. Figure 5.8 shows the result for one such setup for a bottleneck bandwidth of 0.8 Mbps. In the first simulation setup we have three flows, one TCP-Friendly flow and the others are defined as (k=0, l=0.5) and (k=0, l=0.2). We can see from the figure 5.8 that in the absence of re-marking, non-conformant flows beat the TCP friendly flow; however when we re-mark the non-conformant flows the bandwidth is shared fairly. These simulation results also illustrates that the edge based re-marking framework can map the utility functions of the selfish flows to that of TCP, thus making them appear TCP Friendly.

5.5.1.2 Multi Bottleneck Topology

In this section we present the results for multi-bottleneck topology (figure 5.5 b)). We define long flow as a flow which traverses both the bottleneck, whereas the short flows are defined as flows traversing only one bottleneck. In the first



Figure 5.9: Multi Bottleneck (Marking): Throughputs for 2, 10 competing flows.

simulation setup, where there were 2 flows on each bottleneck, (0.8Mbps, 25 packet buffer), we first measured the optimal rate allocations when all the flows (long and

short) are TCP friendly and plot them in 5.9 a). As expected, the short flows grab more share of the bottleneck because they have smaller RTTs and go through a single bottleneck as compared to the long flow. We then changed the short flows to be non-conformant (k=0, l=0.5) and plot the result in 5.9 b). The effect of misbehavior is more pronounced in this case as the non-conformant flows are trying to shut out the TCP friendly flow. However, when we used our model to re-mark the non-conformant flows we see that (figure 5.9 c)) the flows now share the bandwidth fairly. More importantly, we see that the result in figure 5.9 c) is very similar to 5.9 a), i.e., we have successfully mapped the utility function of the non-conformant and short flows TCP-Friendly and similar results were obtained.

In figures 5.9 d), e) and f) we plot the results for a multi-bottleneck topology (10Mbps, 250 packets buffer) where on each bottleneck there are 5 TCP Friendly flows and 5 non-conformant flows (k=0, l=0.5). Figure 5.9 (d) plots the throughput of long and short flows, if all of them were TCP Friendly. As expected the longer flows get a smaller share of the bottleneck than the shorter flows. In Figure 5.9 (e), we changed the shorter flows to act as non-conformant flows and plot the throughput, and it can be seen that the non-conformant shorter flows conveniently beat down the TCP friendly flows. However, in presence of re-marking, (Figure 5.9 (f)) the non-conformant flows are conveyed higher price by the edge-re-marker and thereupon share the bottleneck more favorably with the longer flows. Once again, we see that *re-marking tends to achieve the same performance as those as if all the flows were TCP Friendly*.

5.5.1.3 Background Traffic

In this section we evaluate the framework in presence of noise-like mice traffic. HTTP sources were added to the persistent non-conformant and conformant sources. Each http page sends a single packet request to the destination, which then replies with a file of size which was exponentially distributed with 12 1Kb packets. After a source completes this transfer it waits for a random time, which was exponentially distributed with a mean of 1 second and then repeats the process.



Figure 5.10: Background Traffic (Marking): Throughputs (in pkts/sec) for two competing flows in a single bottleneck topology, where one flow is TCP Friendly while the other is Non-Conformant with (k=0, l=0.5). Also there is web-traffic in the background.



Figure 5.11: Background Traffic (Marking): Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 20% noise.

Two sets of simulations were conducted for the single bottleneck case. In the first simulation, there were two persistent flows (one non-conformant and the other TCP Friendly) competing for a bottleneck of 0.8 Mbps. 2 and 4 http sources were added to generate 15% and 25% noise (i.e. the http sources occupied 15% and 25% of the bottleneck bandwidth). The results for this simulation are plotted in figure 5.10. Again, it can be seen that the re-marking works well in the presence of noise and the bottleneck is shared equitably. In another simulation we increased the number of competing persistent flows to 10 and of these, 7 flows were TCP Friendly while the remaining 3 where non-conformant (k=0,l=0.5). The bottleneck



Figure 5.12: Background Traffic (Marking):Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 65% noise.

bandwidth for this simulation was 10Mbps and a buffer of size 150 packets. Also in this setup we increased the noise sufficiently high to validate the robustness of the scheme in presence of many flows and noise. Figures 5.11 and 5.12 plot the results for the cases where the noise traffic is 20% (25 http sources) and 65% (80 http sources) respectively. Figure 5.12 also shows the robustness of the scheme. The re-marker manages to efficiently patrol non-conformant users even when the noise is the network is sufficiently high, (65% noise).

5.5.1.4 Cross Traffic

In this section we present the results for our penalty function transformer when two way traffic is present. We evaluate this scenario with the multi-bottleneck topology, where we have 5 TCP Friendly long flows and 5 non-conformant (k=0, l=0.5) short flows on each bottleneck. Additionally, on the reverse path, there are 5 TCP Reno flows on each bottleneck. The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 250 packets. Re-Marking, once again achieves equitable sharing of the bottleneck (as shown in Figures 5.13 (a) and (b)).



Figure 5.13: Cross Traffic (Marking): Throughputs (in pkts/sec) for 10 competing flows in a multi-bottleneck topology, where on each bottleneck there are 5 TCP Friendly flows and 5 Non-Conformant with (k=0, l=0.5), with two-way traffic.

5.5.2 Evaluation of Edge Based Re-Marking Framework on a Non ECN Capable Network

Up till now we have discussed the non-conformant framework with re-marking, i.e., we have assumed that ECN support is available in the network. In this section, we look at the alternative scenario, when drops are used to convey congestion penalties. We present the results for two cases, when one the network operates with Drop Tail queues only and then the second where the network works with RED queues. For RED parameter settings the reader is referred to Section 5.4. Again, we test the framework for single and multi-bottleneck scenarios and cross traffic and reverse path congestion.

5.5.2.1 Single Bottleneck

We present the result with a single bottleneck of 0.8Mbps and access links of 8Mbps for 2 competing flows. One of the flows is TCP-Friendly while the other is misbehaving flow (with k=0, l=0.5). Both the flows have same RTT of 60ms. For such a scenario we sampled the packet-stream at the egress router and also placed the re-marker there. The re-marker in this case conveyed the transformed penalties to the mis-behaving flows by dropping its packets. Figure 5.14 a) and b) shows the results of with and without the re-marking framework with Drop Tail and RED queues respectively. It can be seen from the figure 5.14 a) that in a network



Figure 5.14: Single Bottleneck (Dropping): Throughputs (in pkts/sec) for 2 competing flows on a network of DropTail and RED queues with and without Re-Marking. One flow is TCP Friendly while the other is Non-Conformant (k=0,l=0.5).

of Drop Tail queues and absence of re-marking the non-conformant flow gets most of the bottleneck share. Moreover it beats the TCP-Friendly flow comprehensively as against the same simulation setup with RED queues (as shown in figure 5.14 b). However, when we start re-marking the misbehaving flows this bias against the TCP-Friendly is reversed. But, it can be seen from the figure 5.14 a) that now TCP-Friendly flow gets a better share of the bottleneck. This is because unlike marking, dropping is a stricter means to convey congestion notification as it can lead to timeouts. As such the misbehaving flow suffers.

5.5.2.2 Multi-Bottleneck

Figure 5.5 b) show a multi-bottleneck topology with a TCP-Friendly flow traversing both the bottlenecks while one short mis-behaving flow (k=0, l=0.5), each going through one bottleneck. It can be seen from figure 5.15 a) TCP-Friendly is almost shut out by the mis-behaving flows, who now get all the bandwidth. Not only is the TCP-Friendly flow is forced into multiple timeouts (23 for this case) but these timeouts occur with very small windows and are often back to back. Similar results were obtained with a higher multiplexing (of flows) but are not reported here. In summary, with DropTail queues mis-behaving flows may get significant share of the bandwidth, almost to the extent of shutting out conformant flows.

Figure 5.15 b) plots the throughput when instead of DropTail queues we used



Figure 5.15: Multi Bottleneck (Dropping): Throughputs (in pkts/sec) for 2 competing flows on a network of DropTail and RED queues with and without Re-Marking. One flow is TCP Friendly while the other is Non-Conformant (k=0,l=0.5).

RED queues at the bottleneck. (The reader is referred to Section 5.4 for RED settings.) It can be concluded from the figures that though RED improves the shares of TCP-Friendly flow, the unfair rate allocations because of mis-behavior of flows persist. This is because RED is an oblivious AQM scheme and therefore allocates equal marks to all users. Then as outlined earlier, the final rate allocations are dependent on the utility function used by the user's and as such an different choices of utility function can cause unfair sharing of the bottleneck. In figure 5.15 c) and d) we plot the results with re-marking enabled in the network, with DropTail and RED queues respectively. Our results suggests that when re-marking is enabled on a network of DropTail queues we can significantly improve the sharing of the bottleneck. On a network of RED queues with re-marking enabled the results are even more appealing thus pointing to virtues of deploying RED in the network.



Figure 5.16: Background Traffic (Dropping):Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) with 65% noise.

5.5.2.3 Background Traffic

In this section we evaluate the framework in presence of noise-like mice traffic. HTTP sources were added to the persistent non-conformant and conformant sources. The details of these HTTP sources have already been outlined in Section 5.5.1.3 and as such are not reported here.

We used a single bottleneck topology and different level of flow multiplexing to evaluate the effect of background traffic on the performance of a droptail queue network with and without re-marking. However we report results for one case where there were 10 persistent and of these, 7 flows were TCP Friendly while the remaining 3 where non-conformant (k=0,l=0.5). The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 150 packets. Also in this setup we increased the noise sufficiently high to validate the robustness of the scheme in presence of many flows and noise. Figures 5.16 a) and 5.16 b) plot the results for the cases where the noise traffic is 65% (or 80 http sources), i.e. mice traffic occupied 65% of the bandwidth. Figure 5.16 b)shows the robustness of the scheme when sufficiently high (65%) noise is present in the network and the re-marker still manages to efficiently patrol non-conformant users.

Flow	DropTail		RED	
Type	No-Rem	Rem	No-Rem	Rem
TCP-Friendly	55	85	100	200
Non-Conformant	450	400	400	325

Table 5.1: Cross Traffic (Dropping): Comparison of throughput (packets/sec) for network with DropTail and RED queues with and without re-marking.

5.5.2.4 Cross Traffic

Finally we present the results for the edge based re-marker when two way traffic is present in the network. We evaluate this scenario with the multi-bottleneck topology, where we have 5 TCP Friendly long flows and 5 non-conformant (k=0, l=0.5) short flows on each bottleneck. Additionally, on the reverse path, there are 5 TCP Reno flows on each bottleneck. The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 250 packets. Re-Marking, once again achieves fair sharing of the bottleneck (as shown in Table 5.1). However, it can be seen from the results that DropTail queues perform poorly in comparison to RED queues. This further suggests that deployment of RED will help in improving overall network performance, especially in presence of non-conformant flows.

5.5.3 Estimating the Utility Function

The framework presented in this paper works well if the network knows the utility function of the non-conformant flows. Thus estimating the utility function parameters is of paramount importance. In section we outlined a LMMSE method to estimate the utility function. In this section we present the results of estimating utility function.

We have assumed that the identity of misbehaving user is revealed to us. Thereafter, we sample its packet stream at the egress node counting the number of packets sent as well as packets lost. This data set is then separated into bins of 0.5, 1.0 and 2 seconds where in each bin we measure the number of packets sent and the loss rate for that bin. Once we have constructed such a series we used LMMSE method detailed in Section 5.4.1. The results of a simulation of 2 flows, one TCP and

the other a non-conformant flow with k=0, l=0.5, competing on a single bottleneck (see 5.5 a) is showed in figure 5.17 with the bin size being 2 seconds. Further, the bottleneck is 0.8Mb, the access links of 8Mb and the bottleneck employs RED with a buffer size of 25 packets. Also, RTT of the flows is 60ms. Figure 5.17 a) and b) show the estimation results for the non-conformant and the TCP flow respectively. The slope of the graph in each case measures n+1, where n is the exponent (see Section 5.4.1). The exponents theoretical values for our simulation are 0.5 and 1.0 for the non-conformant and TCP flow respectively. For the non-conformant flow we estimate the exponent to be approximately 0.6 (the slope of the graph is 1.5). Similarly for the TCP flow we estimate the exponent to be approximately 0.8.



Figure 5.17: Estimation of Utility Function for 2 competing flows in a single bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5).

Figure 5.19 a) shows the throughput of the two competing flows when we used these exponent values to remark the flows. Figure 5.6 a) shows the throughput when we didn't mark any flow. Because of estimation errors the re-mapping of utility functions was not exact and as such we see that the non-conformant flow still gets more bandwidth. However, there is a significant improvement in the TCPfriendly flow's allocation thus suggesting that the model improves the fair share of the conformant flows at the bottleneck by penalizing misbehaving flows.

5.5.4 Sensitivity Analysis of Framework

In this section we investigate the effect of inaccurate estimation. Specifically we test the validity of the model in presence of inaccurate utility function and RTT estimates. RTT-estimation is needed for updating our congestion indication estimations (which is similar to the one presented in [29]) while utility function estimation is needed for re-mapping. Our simulation results suggests that the inaccurate RTT estimates don't have a pronounced effect on the re-mapping, at most they might slow the convergence (to the objective utility function). However, large errors in estimation of utility function may over-penalize the non-conformant sources. For the results reported in this section, we assumed that the network was ECN capable and therefore marked packets.

5.5.4.1 Effect of Inaccurate RTT Estimate

In all our previous simulations we assumed that the network knows the RTT of the flows. We used these RTT estimates to update our congestion indication estimations. For the results presented in this section we looked at two cases, one when we under-estimated the RTT and the other when we over-estimated it. We present the results with a single-bottleneck of 0.8Mbps, 25 packet buffer and 2 competing flows.

Figure 5.18 a) shows the results when the RTT was under-estimated as 0.05 (instead of 0.06). Figure 5.18 b) shows similar results when we over-estimated the RTT as 0.07. The figures suggest that inaccuracy in RTT estimates alters the convergence speed to the optimal point; a larger value of RTT will slow down the convergence while a smaller value will increase the convergence. However, from both the results its easy to see that the effect of inaccurate RTT estimation is not pronounced and the model works well. We ran simulations with higher degree of multiplexing and came to a similar conclusion. However, we do not present those results here.

5.5.4.2 Effect of Inaccurate Utility Function Estimate

Till now we have assumed that the network knows the utility function of the flows. Since utility functions are not being explicit conveyed to the network therefore



Figure 5.18: Inaccurate RTT Estimates: Throughputs (in pkts/sec) for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate RTT estimates.

we will need to estimate them. Thus we need to explore the effect of inaccurate utility function estimates. In this section we evaluate the model's sensitivity to utility functions; when the utility functions are under-estimated and second when they are over-estimated. Under-estimation here refers to the case when we estimate the utility function to be less aggressive than it really is, i.e. when k + l values are reported to be larger than the actual values. Over-estimation refers to the case where we report the flow to be more aggressive than it really is, i.e. k + l values are reported to be smaller than the actual values. We present the results with a single-bottleneck topology (figure 5.5 a)) for 2 flows.

Figure 5.19 a) shows the results when the utility function was under-estimated as 0.6 (instead of 0.5). Figure 5.19 b) shows similar results when we over-estimated it as 0.4. It can be seen from the results that the model is sensitive to inaccurate estimate of utility functions. When we under-estimated the utility function (k + l = 0.6) the model didn't penalize the mis-behaving flow much, and as such it still garners more bandwidth than the TCP flow. In the case of over-estimation (k + l = 0.4) we see that the network penalizes the mis-behaving flow more and consequently brings it share down below the TCP Friendly flow.

However, the estimation errors pointed out in the simulation are large (the error is 20% since we estimate the k+l values as 0.5 ± 0.1). We evaluated the model for two other error estimates, 10% and 5% and report the result for the 5% error case

in figure 5.20. As expected, as the estimation error decreases the model starts to get better. Further we found that for estimation errors of more than 5% the model does not penalize (or over penalizes) the mis-behaving flow much and it consequently has a larger (or smaller) share at the bottleneck. However in spite of these estimation errors, these shares are still more fair than the case when there was no re-marking present. For estimation errors of less than or equal to 5% the model worked well (figure 5.20). We evaluated the model for different simulation setting where we had 10 flows (5 mis-behaving, 5 friendly) and came to a similar conclusion.



(a) Under-estimation of Utility Function: In-sufficient Re-marking, Misbehaving Flow Still Wins

(b) Over-Estimation of Utility Function: Excessive Re-marking, Misbehaving Flow Over Penalized (Loses to TCP)

Figure 5.19: Inaccurate Utility Function Estimates, 20% Estimation Errors: Throughputs for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate estimates of source's utility function.

5.6 Summary

This chapter addresses the question of protocol conformance in the Internet. In presence of different rate control schemes in the Internet, we consider a rate control protocol to be conformant if they are obtained by maximizing same utility function. Towards understanding the effect of protocol non-conformance, we looked at the impact of non-conformant flows (or mis-behaving flows) on a network of Droptail and RED queues. Our results show that on a network of DropTail queues non-conformant flows get a large (unfair) share of the bandwidth. Further in a multi-bottleneck scenario non-conformant flows can almost shut out the conformant



(a) Under-estimation of Utility Function: Bottleneck is still shared equitably because estimation errors are small



(b) Over-Estimation of Utility Function: Bottleneck is still shared fairly because estimation errors are small.

Figure 5.20: Inaccurate Utility Function Estimates, 5% Estimation Errors: Throughputs (in pkts/sec) for 2 competing flows in a singlebottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate estimates of source's utility function.

flows by pushing them into timeouts. However, on a network of RED queues though the non-conformant still share the bottleneck unfairly but the conformant flows are not shut out. In other words the mis-behavior has a significant impact on a network of Droptail queues than RED queues thus motivating for use of RED.

In this chapter we have proposed an abstract model for modeling and managing non-conformant flows. The primary objective of this framework are to look at ways to achieve protocol conformance. However, in this chapter we also look at the fairness at network's perspective, i.e. how a network might allocate resources amongst different users. Towards addressing these issues we have proposed a framework to map a user's utility function, U_s , to any objective utility function, U_{obj} , by manipulating congestion penalties. These penalty transformation agents can be completely implemented at network edges. Further we have a flexibility of choosing either to re-mark the packets or acks. In cases where we do not have access to the packet-stream we can re-mark the ack-stream and achieve the goals of the model. Packeteer boxes, deployed widely on the Internet, already do a similar work by accessing the ack-stream and pacing the acks [60] and work well with even 20,000 flows.

This proposed utility function transformation can decouple the fairness criteria

from the user's rate control scheme. This allows the network provider to allocate bandwidth amongst user's according to a broad range of fairness criteria. This framework could also find application in pricing especially those of usage based or flat rate pricing. By having all users conform to a particular rate control scheme, the network ensures that it is fair to all users and hence makes usage based or flat rate billing more meaningful. Broadly, this chapter also suggests that management of mis-behaving or non-conformant flows need *not* be coupled with AQM design and can be simply viewed as an edge network based policing question. This framework may also be thought of as a new class of "traffic conditioning" technique, where the "conditioning" is achieved by manipulation of the feedback stream rather than manipulation of the packet stream.

We have analyzed the framework and evaluated it for various single and multibottleneck scenarios with marking and dropping policies being used for congestion notification. Further we showed model is robust and works well even in presence of high background (web) traffic and reverse path congestion. In this chapter we have also presented a scheme to estimate the utility function of the non-conformant user. We also tested the sensitivity of the framework to estimation errors (for RTT and utility function). Our results how that inaccurate RTT estimates do not have a very profound effect on the model's correctness. However, in presence of large utility function estimation errors the model does not fully correct the non-conformant flows, but still considerably improves the fairness at the bottleneck (as compared to the scenario when there was no re-marking).

However, a limitation of the proposed framework is that it only considers the mapping of selfish responsive schemes and might not work well if path asymmetry exists. In such a situation we would have to place the penalty transformation at every exit routers. Further, path asymmetry will also result in erroneous values for network losses which might make the framework either over penalize or under penalize the selfish flow. Thus when a flow may take different paths, we would need coordination between all the penalty transformation agents. However, if a single ingress (or egress) router is used by the flow then the model is immune to path-asymmetry problems of the network. Both unique ingress or egress routers is generally true in the present Internet.

CHAPTER 6 Conclusions

A lot of research has *separately* considered protocol conformance, management of selfish behavior in the network, fairness and differentiated services for the Internet and have often associated these tasks with AQM schemes. However, the AQM schemes are beset with parameter configuration problem and moreover a solution to one of the problems may not be applicable to others. This leads to implementational complexities and is one of the major reason why the proposed solutions are not implemented in real networks. In this thesis we have outlined *deployable* end-system and edge based architectures for protocol conformance and providing fairness in the Internet. The proposed frameworks can also efficiently manage selfish flows at the edge of the network and can be leveraged to provide differentiated service.

It has been widely reported that TCP's performance is severely constrained on a network of Drop Tail queues. Though AQMs have been suggested as a solution to these problems, because of configuration problems they have not found much favor with the network providers. Consequently the Internet still operates with Drop Tail queues. In this thesis we propose *Randomized TCP*, an end-system based solution, to emulate AQM behavior on a network of Drop Tail queues. Specifically, we propose to randomize the packet sending times in TCP. In Randomized TCP successive packets of a window are sent after an interval of RTT(1+x)/cwnd, where *cwnd* is the congestion window in packets and x is a random number drawn from an Uniform distribution on [-1, 1]. This solution is distributed, can be implemented at the end systems and therefore is very attractive from an implementation perspective.

Randomized TCP introduces randomization in the network which helps break flow synchronization. Loss of synchronization thereupon results in lesser burst losses, reduction of phase effects, removal of Drop Tail's bias against flows with longer RTT flows and improvement in fairness in the network. We evaluated Randomized TCP for a variety of single and multi-bottleneck topologies. Our results show that a presence of even a *single* Randomized TCP at a bottleneck is helpful in improving performance of the network. Thus even an incremental deployment of Randomized TCPs would benefit the entire group of users. Finally, we extended the randomization of sending times to other window based protocols, primarily Binomial congestion control schemes. Again, randomization of sending times, improves the fairness in the network and allows different TCP Friendly Binomial schemes to share bandwidth equitably.

Though Randomized TCP improves performance of TCP and network there is a limit to how much control that can be achieved by end-system schemes, especially in a network which operates with disparate congestion control schemes. These different rate control schemes present us with the problem of protocol conformance which manifests itself as smaller problems of fairness and management of selfish behavior in the network. As a first step towards addressing these issues, we first define selfish behavior and conformance. In this thesis, different rate control schemes are called conformant if they are maximizing the same utility function. In this thesis, we define TCP Friendly schemes as the conformant schemes. Thereafter we use this definition of conformance to define selfish end-system rate control schemes.

In this thesis we propose an edge system based re-marking framework to enforce protocol conformance on the Internet. We achieve this by transparently managing the effective range of user's utility functions. More specifically, users may choose arbitrary utility functions, but the edge of the network can re-map these utility functions into a target range of utility functions. This framework thus lets the network choose the target utility functions and thereby allows it to distribute resources amongst users according to some specified fairness criteria. Alternatively, this framework also considers providing fairness from a network's point of view and thus effectively decouples the fairness from user's utility functions.

The proposed framework can be implemented on the network edges and can work with either dropping or marking enabled network. Also the edge based remarking is independent of the buffer management policies in the network and therefore works even with a network of Drop Tail queues. Moreover, the flexibility of the framework to map any utility function to any target utility function helps it provide broad range of fairness criteria. This edge based re-marking model also suggests that the management of selfish flows in the network need not be necessarily coupled with AQM, instead it can be achieved by simple edge based modules. Finally the re-marking architecture proposed in this thesis can be thought of as a new class of traffic conditioning scheme and can be leveraged to provide service differentiation.

We have evaluated the edge based re-marking framework for a variety of single and multi bottleneck scenarios with both background web traffic and reverse path congestion. Our results show that the framework can map utility functions in all the cases, with either dropping or marking being used to convey penalties. However, a limitation of the model is that it only considers responsive users and may not work well in cases of path asymmetry.

CHAPTER 7 Future Work

The re-marking framework presented in this paper manages selfish-responsive (i.e. non-conformant flows who react to congestion indication) flows at the edge of the networks. However the model presented in the paper assumes certain informations about the flows. For our future work we plan to relax these assumptions. Specifically, we plan to puruse these issues in our thesis:

- Estimation of Utility Function: In Chapter 5 we presented a Linear Least Squared Estimation (LLSE) technique for estimating a range of utility functions. However, as pointed out in Section 5.4.1 (Chapter 5) this covers only a section of rate control schemes. In future we plan to elaborate on ways to estimate utility function which covers a broader range of rate control schemes. Expected Completion Date: Aug 2003
- Identification of Selfish Users: In this thesis we have assumed that we know the identity of the selfish users. In our future work we plan to investigate scalable ways to identify selfish users at the edge of the network. Expected Completion Date: Nov 2003
- Service Differentiation: The model presented in this thesis can also be leveraged to provide service differentiation amongst various competing users. In future, we intend to explore this possibility and show how we can achieve service differentiation.

Expected Completion Date: Aug 2003

- [1] "Network simulator," http://www.isi.edu/nsnam.
- [2] "Utime: Microsecond resoultion timers for linux," http://www.ittc.ku.edu/utime/.
- [3] A. A. Abouzeid and S. Roy, "Analytic understanding of red gateways with multiple competing tcp flows," in *IEEE Globecom*, 2000.
- [4] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of tcp pacing," in *IEEE Infocom*, 2000.
- [5] A. Akella and etal, "Selfish behavior and stability of the internet: A game theoretic analysis of tcp," in ACM Sigcomm, 2002.
- [6] M. Aron and P. Druschel, "Tcp: Improving startup dynamics by adaptive times and congestion control," 1999, tR98-318, Rice Univ. Techreport.
- [7] S. Athuraliya, S. H. Low, and D. E. Lapsley, "Random early marking," in *QofIS*, 2000.
- [8] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The effects of assymetry in tcp performance," in ACM/IEEE Mobicom, 1997.
- [9] D. Bansal and H. Balakrishnan, "Binomial congestion control scheme," in *IEEE Infocom*, 2000.
- [10] J. C. R. Bennett and H. Zhang, "Worst case fair weighted fair queueing," in *IEEE Infocom*, 1996.
- [11] D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and distributed computation: Numerical methods," 1989, prentice-Hall.
- [12] T. Bonald and L. Massoulie, "Impact of fairness on internet performance," in ACM Sigmetrics, 2001.
- [13] B. Braden and etal, "Recommendations on queue management and congestion avoidance in the internet," April 1998, iETF RFC 2309.

- [14] L. Brakmo and L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communication*, 1995.
- [15] K. C. C. You, "Time series models for internet data traffic," in 24th Conf. on Local Computer Networks, 1999.
- [16] K. Chandrayana, B. Sikdar, and S. Kalyanaraman, "Scalable configuration of red queue parameters," in *IEEE High Speed Switching and Routing*, 2000.
- [17] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "A self-configuring RED gateway," in *IEEE Infocom*, 1999.
- [18] D. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks* and ISDN, vol. 17, issue = 1, pp. 1–14, 1989.
- [19] M. Christiansen, K. Jaffay, D. Ott, and F. D. Smith, "Tuning RED for web traffic," in ACM Sigcomm, 2000.
- [20] A. Das, D. Dutta, and A. Helmy, "Fair stateless aggregate marking techniques using aqm techniques," in *IEEE/IFIP MMNS*, 2002.
- [21] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of fair queueing algorithm," in ACM Sigcomm, 1989.
- [22] D. Dutta, A. Goel, and J. Heidemann, "Oblivious aqm and nash equilibria," in *IEEE Infocom*, 2003.
- [23] S. B. et al, "An architecture for differentiated services," December 1998, iETF RFC 2476.
- [24] W. Feng and et al, "Blue: A new class of active queue management algorithms," 1999, uM CSE-TR-387-99, Univ. of Michigan Techreport.
- [25] W. C. Feng and etal, "A self-configuring red gateway," in *IEEE Infocom*, 1999.
- [26] —, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *IEEE Infocom*, 2001.

- [27] S. Floyd, "Connections with multiple congested gateways in sacket-switched networks part 1: One-way traffic," ACM Computer Communication Review, vol. 21, pp. 30–47, 1991.
- [28] —, "Congestion control principles," September 2000, iETF RFC 2914.
- [29] S. Floyd and etal, "Equation-based congestion control for unicast applications," in ACM Sigcomm, 2000.
- [30] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 458–472, 1999.
- [31] S. Floyd, M. Handley, and E. Kohler, "Problem statement of dcp," 2002.
- [32] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 115–156, 1992.
- [33] —, "On traffic phase effects in packet-switched gateways," ACM Computer Communication Review, vol. 21, 1992.
- [34] —, "Random early drop gateways for congestion avoidance," IEEE/ACM Transactions on Networking, vol. 1, pp. 397–413, 1993.
- [35] R. Gupta, M. Chen, S. McCanne, and J. Walrand, "Webtp: A receiver driven transport protocol," 1999, check it out.
- [36] A. Habib and B. Bhargava, "Unresponsive flow detection and control using the differentiated services framework," 2002.
- [37] E. Hashem, "Analysis of random drop for gateway congestion control," 1989, report LCS TR-465, MIT.
- [38] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong, "A control theoretic analysis of RED," in *IEEE Infocom*, 2001.

- [39] S. S. I. Stoica and H. Zhang, "Core-stateless fair queueing: Achieving apporximately fair bandwidth allocations in high speed networks," in ACM Sigcomm, 1998.
- [40] S. Jacobs and A. Eleftheriadis, "Providing video services over networks without quality of service guarantees," in *RTMW*, 1996.
- [41] V. Jacobson, "Congestion avoidance and control," in ACM Sigcomm, 1988.
- [42] S. Jin, L. Guo, I. Matta, and A. Bestavros, "TCP-friendly SIMD congestion control and its convergence behavior," Tech. Rep., 2001.
- [43] J. Ke and C. Williamson, "Towards a rate based tcp protocol for the web," in MASCOTS, 2000.
- [44] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.
- [45] H. K. Khalil, "Non linear systems," pp. 100–101, 1996.
- [46] S. Kunniyur and R. Srikant, "End-to-end congestion control: Utility functions, random losses and ecn marks," in *IEEE Infocom*, 2000.
- [47] —, "Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management," in ACM Sigcomm, 2001.
- [48] D. Lin and R. Morris, "Dynamics of random early detection," in ACM Sigcomm, 1997.
- [49] S. Low, "A duality model of tcp and queue management algorithms," in ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management, 2000.
- [50] S. Low and D. Lapsley, "Optimization flow control, i: Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 861–975, 1999.

- [51] R. Mahajan and etal, "Controlling high bandwidth aggregates in the network," ACM Computer Communication Review, 2002.
- [52] R. Mahajan and S. Floyd, "Red-pd: Controlling high bandwidth flows at the congested router," 2001.
- [53] A. Mankin and etal, "Ietf criteria for evaluating reliable multicast transport and application protocols," June 1998, iETF RFC 2357.
- [54] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to deploy red," in ACM IWQoS, 1999.
- [55] M. May, T. Bonald, and J.-C. Bolot, "Analytic evaluation of red performance," in *IEEE Infocom*, 2000.
- [56] M. Mehta, "On randomizing the sending times in tcp: An implementation," 2001, masters Thesis, ECSE, R.P.I.
- [57] J. Mo and J. Walrand, "Fair end-to-end window-based congestion," IEEE/ACM Transactions on Networking, vol. 8, pp. 556–567, 2000.
- [58] J. Moghul, "Observing tcp dynamics in real networks," in ACM Sigcomm, 1992.
- [59] T. J. Ott, T. V. Lakshman, and L. Wong, "Sred: Stabalized red," in *IEEE Infocom*, 1999.
- [60] Packeteer, "http://www.packeteer.com."
- [61] J. Padhye and etal, "Modeling tcp reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 133–145, 2000.
- [62] V. Padmanabhan and R. Katz, "Tcp fast start: A technique for speeding up web transfers," in *IEEE Globecom*, 1998.

- [63] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," 2001, http://www.research.att.com/simbreslau/papers/afd-techreport.ps.gz.
- [64] R. Pan, B. Prabhakar, and K. Psounis, "Choke a stateless queue management scheme for approximating fair bandwidth allocation," in *IEEE Infocom*, 2000.
- [65] C. Partridge, "Ack spacing for high bandwidth-delay paths with insufficient buffering," 1998.
- [66] N. Plotkin and P. Varaiya, "The entropy of traffic streams in atm virtual circuits," in *IEEE Infocom*, 1994.
- [67] S. Raghunath, "Modeling the tcp loss rate process for efficient congestion control," Tech. Rep., 2002.
- [68] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *IEEE Infocom*, 1999.
- [69] I. Rhee, V. Ozdemir, and Y. Yi, "Tear: Tcp emulation at receivers flow control for multimedia streaming," 2000, nCSU Technical Report.
- [70] N. R. Sastry and S. S. Lam, "CYRF: A framework for window-based unicast congestion control," in *ICNP*, 2002.
- [71] S. Shenker, "Fundamental design issues for the future internet," IEEE Journal on Selected Areas in Communication, vol. 13, 1995.
- [72] S. Shenker, L. Zhang, and D. Clark, "Some observations on the dynamics of a congestion control algorithm," ACM Computer Communication Review, vol. 20, pp. 30–39, 1990.
- [73] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in ACM Sigcomm, 1995.
- [74] M. Shreedhar and G. Verghese, "Efficient fair queueing using deficit round robin," in ACM Sigcomm, 1994.
- [75] B. Sikdar, "Network traffic modeling and transmission control protocol," Ph.D. dissertation, R. P. I., 2001.
- [76] B. Sikdar, K. Chandrayana, K. Vastola, and S. Kalyanaraman, "Queue management algorithm and traffic self similarity," in *IEEE High Speed Switching and Routing*, 2002.
- [77] W. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," 1997, iETF RFC 2001.
- [78] I. Stoica, S. Shenker, and H. Zhang, "Core stateless fair queueing: Acheiving approximately fair bandwidth allocations in a high speed networks," in ACM Sigcomm, 1998.
- [79] I. Stoica and H. Zhang, "Providing gauranteed services without per flow management," in ACM Sigcomm, 1999.
- [80] V. Visweswaraiah and J. Heidemann, "Improving restart of idle tcp connections," 1997, tech Report 97-661, Univ. of South California.
- [81] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and modeling of the temporal dependence in packet loss," in *IEEE Infocom*, 1999.
- [82] Y. R. Yang, M. S. Kim, and S. S. Lam, "Transient behaviors of TCP-friendly congestion control protocols," in *IEEE Infocom*, 2001.
- [83] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in ACM Sigcomm, 1991.